Real Time Interactive Massive Model Visualization

D. Kasik†[1] D. Manocha[2] A. Stephens‡[3] B. Bruderlin[4] P. Slusallek[5] E. Gobbetti[6] W. Correa[7] I. Quilez[8]

[1] The Boeing Company [2] University of North Carolina [3] University of Utah [4] Technical University of Ilmenau
[5] Saarland University [6] CRS4 [7] IBM [8] VRContext

Abstract

Real-time interaction with complex models has always challenged interactive computer graphics. Such models can easily contain gigabytes of data. This tutorial covers state-of- the-art techniques that remove current memory and performance constraints. This allows a fundamental change in visualization systems: users can interact with huge models in real time.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Line and Curve Generation

1. Tutorial Description

The amount of data produced by today's engineering design and scientific analysis applications often exceeds the capability of conventional interactive computer graphics. Users produce tens of gigabytes of data while designing a product or analyzing results. Techniques for examining all this data simultaneously and interactively are not readily available in today's visualization or CAD tools.

Combining specific algorithms, specialized data structures, and high performance hardware has enabled real-time visualization and is a significant research area. As a result, users can see an entire airplane instead of a subsection or full level-of-detail of a building instead of a simplified form.

This tutorial presents seven different solutions to the problem. Each instructor will focus on the practical aspects of their implementation and provide examples, either as movies or live demos. The tutorial will provide participants with the knowledge to identify trade- offs and weigh benefits. In addition, we discuss system implementation issues, the conceptual basis for the work, the impact on the user community, how to accelerate user acceptance of the technology, and methods to increase the amount of test data for the research community.

Key technical topics include: software techniques to overcome performance and memory size limitations (e.g., kd-trees, occlusion culling, multi-threaded programming, parallel processor transaction management, memory-mapped files, display lists, cache coherent layouts); computing architecture (e.g., parallel processor architectures, single and multi- GPU hardware, thin client access to rendering services, hardware occlusion culling, cell computers, multi-core CPUs); and overall system architecture (e.g., preprocessing, large user communities, model configuration management, network transfer of basic geometry).

2. Instructor Information

The instructors come from academia, start-up companies, and industry. Each has built an approach that combines one or more of the above technologies. The tutorial is organized around the instructor's technical approach, what parts have worked, and lessons learned when applying these technologies to real-world problems.

2.1. Instructor Backgrounds

Dave Kasik is the Boeing Enterprise Visualization Architect. His research interests include innovative combinations of basic 3D graphics and user interface technologies and increasing awareness of the impact of visualization technology inside and outside Boeing. Dave has a BA in Quantitative Studies from the Johns Hopkins University and an MS in

† Tutorial organizer.

‡ Tutorial notes editor.

Tutorial Schedule

8:30	9:15	Motivation and Challenges	Dave Kasik
9:15	10:05	Interactive View-Dependent Rendering and Shadow Generation in Complex Datasets	Dinesh Manocha
10:20	11:10	Ray Tracing with Multi-Core/Shared Memory Systems	Abe Stephens
11:10	12:00	Visibility-guided Rendering to Accelerate 3D Graphics Hardware Performance	Beat Bruderlin
1:00	1:50	Massive Model Visualization using Realtime Ray Tracing	Philipp Slusallek
1:50	2:40	GPU-friendly accelerated mesh-based and mesh-less techniques for output-sensitive rendering of huge complex 3D models.	Enrico Gobbetti
2:55	3:45	Interactive Out-Of-Core Visualization of Large Datasets on Commodity PCs	Wagner Correa
3:45	4:35	Putting Theory into Practice	Inigo Quilez
4:35	5:00	Panel discussion	Moderator: Dave Kasik

Computer Science from the University of Colorado. He is a member of IEEE, ACM, ACM SIGGRAPH (he has attended all SIGGRAPH conferences), and ACM SIGCHI. He is a member of the editorial board for IEEE Computer Graphics and Applications.

Dinesh Manocha is currently a professor of Computer Science at the University of North Carolina at Chapel Hill. He was selected as an Alfred P. Sloan Research Fellow. He received NSF Career Award in 1995, Office of Naval Research Young Investigator Award in 1996, Honda Research Initiation Award in 1997, and the Hettleman Prize for scholarly achievement at UNC Chapel Hill in 1998. He has also received best paper and panel awards at ACM SuperComputing, ACM Multimedia, ACM Solid Modeling, Pacific Graphics, IEEE VR, IEEE Visualization, and Eurographics. He has served on the program committees and editorial boards of leading conferences in computer graphics and geometric modeling.

Manocha has been working on large model visualization for more than 10 years. His research group at UNC Chapel Hill has published numerous papers on model simplification, visibility computations, large data management and integrating these techniques at ACM SIGGRAPH and other conferences. He has also organized SIGGRAPH courses on interactive walkthroughs, large model visualization, and GPGPU.

Abe Stephens is a PhD student at the University of Utah working in the Scientific Computing and Imaging Institute under the direction of Steven Parker. His work focuses on interactive ray tracing, especially large data visualization. He has worked closely with Silicon Graphics to improve interactive ray tracing on large parallel systems and collaborated with Intel's Microprocessor Technology Lab. Abe received a BS in Computer Science from Rensselaer Polytechnic Institute in 2003.

Beat Bruderlin is professor of Computer Science at the Technical University of Ilmenau, Germany. His work focuses on computer geometry with applications to computer aided design and engineering visualization. Other interests include new interaction techniques for 3D design. Beat Bruderlin received his M.S. degree in Physics from the University of Basel and a PhD in Computer Science from the Swiss Federal Institute of Technology (ETH) âĂŞ Zurich, Switzerland. He was a faculty member at the University of Utah, before joining TU Ilmenau. In 2004 he founded 3Dinteractiverendering software.

Philipp Slusallek is professor for computer graphics and digital media at Saarland University, Germany. Before joining Saarland University he was visiting assistant professor at Stanford University. He received a Diploma/MSc in physics from the University of TuÌĹbingen and a Doctor/PhD in computer science from the University of Erlangen. Philipp has published and taught extensively, including a SIGGRAPH05 course, about real-time ray tracing. He is the principal investigator for the OpenRT project, which aims at establishing real-time ray-tracing as an alternative technology for interactive and photorealistic 3D graphics. This work includes the development of a highly optimized ray tracing software, custom hardware for ray tracing, approaches to massive model visualization, and real-time lighting simulation algorithms. Recently he co-founded "inTrace", a spin-off company that commercializes real- time ray tracing technology.

Enrico Gobbetti is the founder and director of the Visual Computing (ViC) group at the Center for Advanced Studies, Research, and Development in Sardinia (CRS4). At CRS4, Enrico developed and managed a graphics research program supported through industrial and government grants. His research interests span many areas of computer graphics. His most recent contributions include a new breed of coarse-grained adaptive multiresolution techniques for processing and rendering large scale geometric models. Enrico holds an Engineering degree (1989) and a Ph.D. degree (1993) in Computer Science from the Swiss Federal Institute of Technology in Lausanne (EPFL). For more information, see www.crs4.it/vic

Wagner Correa is a Research Staff Member with the IBM Watson Research Center. He is part of the Visualization

Systems Group, which recently released the Deep Computing Visualization (DCV) suite of programs for immersive and remote visualization. Prior to joining IBM, Wagner was teaching Computer Graphics at the Federal University of Minas Gerais (UFMG) in Brazil. Wagner holds a B.S. degree (1994) and an M.S. degree (1996) in Computer Science from UFMG, and an M.A. degree (1998) and a Ph.D. degree (2004) in Computer Science from Princeton University.

Inigo Quilez received a degree as a Telecommunications Engineer from the University of Basque Country (Spain), with intensification in digital signal processing. He has extensively worked in real-time computer graphics, within the "demoscene" since 1998, especially in the subject of extreme procedural content creation and data compression. Well known in the fractals community, work is still needed to give aesthetics a more important role in the scientific work. Since he joined VRcontext in 2003, his work focuses on research and development of photorealistic rendering and massive model visualization techniques among others, focusing in shared memory msystems (especially the Silicon Graphics architecture).

2.2. Co-Instructors

In addition to the instructors listed above the following co-instructors contributed to the tutorial.

- Sung-Eui Yoon, Lawrence Livermore National Laboratory.
- Andreas Dietrich, Saarland University.
- Alain Hubrecht, VRContext.
- Fabio Marton, CRS4.

3. Tutorial Summary

The following sections contain tutorial notes and accompanying material.

3.1. Motivation and Challenges

The human visual system provides an extremely efficient way to communicate both context and detail. The amount of data that's being generated is actually exceeding the rate of change of Moore's law.

The domains in which data is expanding range from computer-aided design and manufacturing to arts and entertainment to intelligence analysis. The most effective way people have to comprehend and communicate the overall implications relies on computer graphics. Clear examples from specific domains show the effectiveness of interactive, real-time 3D graphics.

The overall system implications of real-time interaction are essential to ultimately make the technology implementable. The attendees will get a clear understanding of all aspects that will make or break the widespread adoption of the techniques described in the other sections of the tutorial.

Additional References:

- David J. Kasik, Strategies for Consistent Image Partitioning, IEEE MultiMedia, vol. 11, no. 1, pp. 32-41, 2004.
- David J. Kasik, William Buxton, David R. Ferguson, Ten CAD Challenges, IEEE Computer Graphics and Applications, vol. 25, no. 2, pp. 81-92, Mar/Apr, 2005.

3.2. Interactive View-Dependent Rendering and Shadow Generation in Complex Datasets

Current GPUs are progressing at a rate faster than Moore's Law. In theory, they are capable of achieving peak throughput of tens of millions of triangles per second. However, they are optimized for game-like environments and it is a major challenge to render complex models composed of tens or hundreds of millions of triangles at interactive rates. We outline a number of algorithms to overcome these problems.

We outline techniques to build good scene graph representations of complex datasets using partitioning and clustering algorithms. Furthermore, we present an optimization-based algorithm to compute cache coherent layouts for improved CPU and GPU throughputs. In order to achieve high frame rates, we only render triangles that the user can ultimately see. We present efficient and practical techniques for view-dependent simplification and occlusion culling on large models. Furthermore, we describe novel hierarchical data structures to integrate these algorithms. Finally, we present novel algorithms for shadow generation. Specifically, we present subdivided shadow maps, which can overcome perspective aliasing problems and work well on current GPUs.

Finally, we demonstrate the application of our algorithms to different types of complex models on a commodity desktop or laptop. The set of models include large scanned datasets, isosurfaces extracted from simulation data, CAD environments of powerplants, airplanes and tankers, and terrain datasets. We also outline many open problems in this area.

Additional References:

- Sung-Eui Yoon, Brian Salomon, Russell Gayle, Dinesh Manocha, Quick-VDR: Interactive View-Dependent Rendering of Massive Models, vis, pp. 131-138, 15th IEEE Visualization 2004 (VIS'04), 2004.
- Yoon, S., Lindstrom, P., Pascucci, V., and Manocha, Cache-oblivious mesh layouts. ACM Trans. Graph. 24, 3 (Jul. 2005), 886-893.
- Brandon Lloyd, Sung-Eui Yoon, David Tuft, Dinesh Manocha, Subdivided Shadow Maps. UNC Tech Report 24 (2005).
- Sung-Eui Yoon, Christian Lauterbach, and Dinesh Manocha. R-LODs: Fast LOD-Based Ray Tracing of Massive Models. UNC Tech Report (to appear)

3.3. Ray Tracing with Multi-core/Shared Memory Systems

Shared memory computer systems provide a strong platform for interactive software rendering. They combine a large number of processors, enormous amounts of memory and many shared devices across a single system.

Special hardware in these systems provides many facilities that must be implemented by hand on a cluster. Still multi-processor servers, or even multi-core workstations, behave differently from standard single processor desktops and the graphics programmer must pay attention to the system architecture in order to obtain optimal performance. The challenges encountered implementing an interactive renderer on such a system are not well addressed by existing HPC tools and programming techniques and require the programmer to interact with the system at a much lower level.

Moving beyond basic ray tracing techniques we provide detailed examples of how parallel system architecture effects renderer implementation. This section will attempt to address three basic questions: How are the user's expectations different when rendering on a moderately sized parallel system? How are parallel systems constructed and how well suited are their memory systems for interactive rendering? How does the software design effect the renderer's ability to scale on larger systems or larger problems?

3.4. Visibility-Guided Rendering to Accelerate 3D Graphics Hardware Performance

Hardware accelerators for 3D graphics (GPUs) have become ubiquitous in PCs and laptops. They are very powerful for real-time visualization of scenes up to a few million polygons at very high (almost photo-realistic) quality. GPUs have been successfully applied in computer games and engineering visualization. However, a straightforward use of GPUs, as is the current practice, can no longer deal with the ever larger datasets of fully detailed engineering models such as airplanes, industrial plants, cars, etc. This puts a severe limitation on the data explosion we currently encounter in industry. Already some large models are 100 to 1000 times larger than what can be handled by GPUs in real time.

Visibility-guided Rendering is a novel approach for real-time rendering of very large 3d datasets.

We start by identifying the main differences between sampling-based rendering (e.g. ray tracing) and rasterization-based rendering (e.g. hardware-accelerated OpenGL). By comparing the pros and cons of the two opposite approaches at a high abstraction level, we can explain some of the current limitations of OpenGL and develop ideas for overcoming these limitations.

The key advantage of VGR is to efficiently determine visibility of the vast majority of polygons before they are sent to the graphics hardware. Different culling techniques are presented which can be used in combination. Spatial data structures, as well as hardware features of the GPU can be exploited to implement the culling techniques optimally. The visibility-guided rendering approach relies heavily on efficient memory management, concurrency between CPU and GPU, as well as optimal use of low-level OS functionality to handle very large models.

In the outlook we touch on modern multi GPU hardware architecture, the use of programmable shaders in the context of VGR, as well interactive object manipulation functionality. The presentation concludes with a life demo of our software.

3.5. Massive Model Visualization using Realtime Ray Tracing

Real-time ray tracing has become an attractive alternative to rasterization based rendering, particularly for highly complex data sets including both surface and volume data. Ray tracing handles massive datasets well because of output sensitivity and the logarithmic complexity of the ray tracing computations with respect to the scene size. Ray tracing easily handles huge scenes as long as they fit into main memory. For even larger data sets, active memory management is necessary to always keep the working set in main memory and on-demand swap data in and out depending on its visibility. All these operations require spatial index structures (e.g. kd-trees) that need to be built out-of-core and often offline. We will discuss efficient techniques for this task, including approaches that allow efficient memory management at runtime. In addition, we will discuss several extensions that are necessary for efficiently ray tracing large models. An intrinsic property of ray tracing is that once a scene can be ray traced, adding advanced optical effects or lighting simulation is fairly straightforward. We will discuss how such advanced effects can be used for achieving photorealistic visualization even of highly complex models such as natural environments with environment lighting, complex shading, and efficient anti-aliasing.

Handling dynamic scenes has been a major issue with ray tracing. We outline two areas where large progress has recently been made: Designing scenes graphs for efficiently handling changes in very large models and novel index structures that allow fast updates after changes to the geometry. Finally, we will discuss trends in hardware and how they will help in making ray tracing available to a larger and larger set of applications. In particular we will briefly compare the capabilities of multi-core CPUs (including the Cell processor), GPUs, and custom hardware with respect to ray tracing of complex models.

3.6. GPU-Friendly Accelerated Mesh-based and Mesh-less Techniques for the Output Sensitive Rendering of Huge Complex 3D Models

In recent years, the large entertainment and gaming market has resulted in major investments in commodity graphics chip technology, leading to state-of-the-art programmable graphics units (GPUs) with greater complexity and computational density than current CPUs. GPUs are not only powerful, ubiquitous, and cheap, but their programmability is leading to new ways to tackle the large scale data visualization problems.

This section of the tutorial will discuss GPU friendly output sensitive techniques for harnessing the raw power and programmability features of these chips to interactively render very large complex 3D models. In this context, we will discuss and compare two different approaches: a mesh-based framework based on multi-scale geometric models (Batched Multi-Triangulation, IEEE Viz 2005), that is well suited to models with dense geometric details, and a mesh-less framework (Far Voxels, SIGGRAPH 2005), that handles datasets that combine complicated geometry and appearance with a large depth complexity by modeling model appearance rather geometry.

The tutorial section will be illustrated with practical examples of the visual inspection of very different kinds of models, including very large CAD assemblies, terrains, isosurfaces, and laser scans visualized on a laptop.

Additional References:

- Paolo Cignoni, Fabio Ganovelli, Enrico Gobbetti, Fabio Marton, Federico Ponchio, and Roberto Scopigno. Batched Multi Triangulation. In Proceedings IEEE Visualization. Pages 207-214. IEEE Computer Society Press, October 2005.
- Enrico Gobbetti and Fabio Marton. Far Voxels - A Multiresolution Framework for Interactive Rendering of Huge Complex 3D Models on Commodity Graphics Platforms. ACM Transactions on Graphics, 24(3): 878-885, August 2005. Proc. SIGGRAPH 2005.
- Paolo Cignoni, Fabio Ganovelli, Enrico Gobbetti, Fabio Marton, Federico Ponchio, and Roberto Scopigno. Adaptive TetraPuzzles - Efficient Out-of-core Construction and Visualization of Gigantic Polygonal Models. ACM Transactions on Graphics, 23(3): 796-803, August 2004. Proc. SIGGRAPH 2004.

3.7. Interactive Out-of-Core Visualization of Large Datasets on Commodity PCs

This section of the tutorial will focus on interactive visualization of large datasets on commodity PCs. Interactive visualization has applications in many areas, including computer-aided design, engineering, entertainment, and training. Traditionally, visualization of large datasets has required expensive high-end graphics workstations.

Recently, with the exponential trend of higher performance and lower cost of PC graphics cards, inexpensive PCs are becoming an attractive alternative to high-end machines. But a barrier in exploiting this potential is the small memory size of commodity PCs. To address this problem, we will present out-of-core techniques for visualizing datasets much larger than main memory.

We will start by presenting out-of-core preprocessing techniques. We will show how to build a hierarchical decomposition of the dataset using an octree, precompute coefficients used for visibility determination, and create levels of detail.

We will then present out-of-core techniques used at runtime. We will describe how to find the visible set using a fast approximate algorithm followed by a hardware-assisted conservative algorithm. We will also show how to use multiple threads to overlap visibility computation, cache management, prefetching, and rasterization.

We will finish by describing a parallel extension of the system that uses a cluster of PCs to drive a high-resolution, multi-tile screen. A thin client process manages interaction with the user, and a set of server processes render the multiple screen tiles. Large shared file systems (network or server-attached) provide storage for the complex dataset.

A system based on these techniques is a cost-effective alternative to high-end machines, and can help bring visualization of large datasets to a broader audience.

Additional References:

- W. T. Correa. New Techniques for Out-Of-Core Visualization of Large Datasets. PhD thesis, Princeton University, 2004.
- W. T. Correa, J. T. Klosowski, and C. T. Silva. Out-of-core sort-first parallel rendering for cluster-based tiled displays. In Proceedings of PGV 2002 (4th Eurographics Workshop on Parallel Graphics and Visualization), pages 89-96, 2002.
- W. T. Correa, J. T. Klosowski, and C. T. Silva. Visibility-based prefetching for interactive out-of-core rendering. In Proceedings of PVG 2003 (6th IEEE Symposium on Parallel and Large-Data Visualization and Graphics), pages 1-8, 2003.

3.8. Putting Theory into Practice

Moving academic results to a real product is not always easy. Not only is a big part of the papers normal focus on very specific cases but research experiments are also often made with different constraints than those found in production. Even when applicable to a product, many techniques have been only tested in a few well known models used in CG literature, while customers demand algorithms to work in all kind of data, from best case models to ill-formed geometry, that must not be removed from the dataset and that can degrade significantly an algorithm's performance (as accelerating data structures).

On the other hand, the kind of data used in most 3D massive datasets visualization or collision detection research papers handle high density (e.g., the happy Buddha) or medium density models (the PowerPlant or the Boeing 777). However, a product has to also deal with low density massive models. Low density models make some techniques not applicable or less efficient and add a new set of problems.

Another common source of problems is that many techniques assume that pre- computation time and effort is not important for the user. In practice, both the pre- calculation time and the complexity of the pre-compute process is of significant concern to production users. From the marketing point of view, there is also a resistance present in users to change to new technologies (even if a lot better than old ones); and lot of work must be done to give the application the look and feeling of the tools users are already used to.

Finally, we will demonstrate how a visualization and collision detection on massive model looks like in a real time application, both using OpenGL and software ray-tracing.

Boeing Technology
Information Technology

Motivation and Challenges in Real-time, Interactive Massive Model Visualization

Dave Kasik
Technical Fellow
The Boeing Company
Seattle WA USA
david.j.kasik@boeing.com
+1-425-830-4276

Section Goal

- Provide understanding of:
  - Usage scenarios for interactive, massive model visualization.
  - The technology implications of interactive, massive model visualization.
  - How the user community can assist the research community.

Section Outline

- Motivation for effort from a user's perspective, including sample use cases
- Characterization of user tasks that can be addressed by visual analysis
- General processing architecture alternatives
  - Client-based
  - Hybrid client-server
  - Server-based
- Contrast of issues between GPU and CPU-based approaches
- Additional technical challenges:
  - Network impact
  - Pre-processing
  - Version management
  - Rigid body motion
  - Collision detection
- Pragmatics of getting data released to the research community

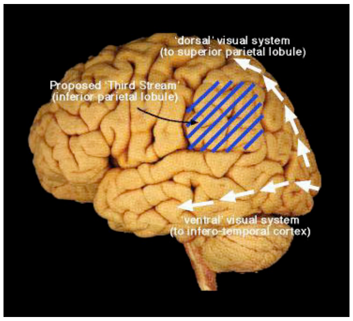
Data Explosion

- All storage media produced about 5 exabytes of new information in 2002.
  - 92% was stored on magnetic media, mostly hard disks.
- This amount of new information is about double of the amount stored in 1999.
- Information flows through electronic channels (telephone, radio, TV, and the Internet) contained ~18 exabytes of new information in 2002.
  - This is 3 1/2 times more than is stored.
  - 98% is voice and data sent telephonically via fixed lines and wireless

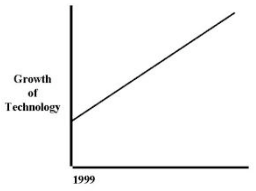
What Do These Numbers Mean?

- Kilobyte (KB) = 1,000 bytes, 10^3
  - 2 KB: Typewritten page
- Megabyte (MB) = 1,000,000 bytes, 10^6
  - Small novel
- Gigabyte (GB) = 1,000,000,000 bytes, 10^9
  - Pickup truck filled with books
- Terabyte (TB) = 1,000,000,000,000 bytes, 10^{12}
  - 50,000 trees made into paper
  - 2 TB: An academic research library
- Petabyte (PB) = 1,000,000,000,000,000 bytes, 10^{15}
  - 200 PB: All printed material
- Exabyte (EB) = 1,000,000,000,000,000,000 bytes, 10^{18}
  - 2 EB: Total volume of information generated in 1999
  - 5 EB: All words ever spoken by human beings

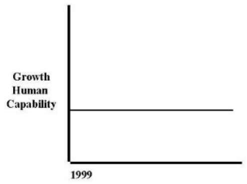
Human Visual Communication Processor


'dorsal' visual system
(to superior parietal lobule)
Proposed 'Third Stream'
(inferior parietal lobule)
'ventral' visual system
(to infero-temporal cortex)

Buxton's Conundrum



Moore's Law



God's Law

Use Cases

- What can a user do with interactive visualization alone?

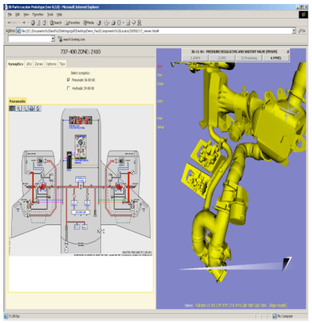
Visual Task Analysis

- Find an object in a complex scene.
- Focus on the found object to better understand surface characteristics (e.g., smoothness, roughness).
- Once the object is found, look at objects in the immediately surrounding volume.
- Visually scan the scene.
- Observe dynamics in the entire scene (conventionally by animation).
- Work with multiple versions of the same set of objects to compare the two sets.
- More detail in DJ Kasik, "Strategies for Consistent Image Partitioning", IEEE Multimedia, Jan-Mar, 2004, pp. 32-41.

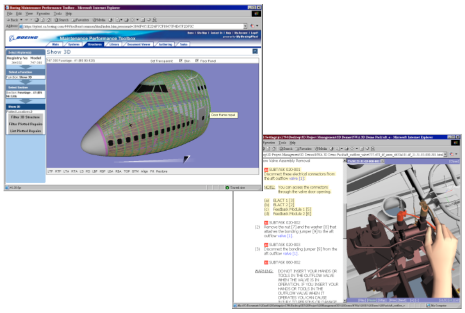
Potential Applications

- Design reviews
- Engineering analysis (loads, CFD, etc.)
- Safety
- Survivability
- Part context
- Reverse engineering from massive scans
- Quality assurance inspection
- Manufacturing instructions
- Part catalogs
- Training
- Maintenance instructions
- Sales and marketing
- Basically, any process where quick navigation is needed to go anywhere in a digital model

Concrete Example 1 – Tracing Systems



Concrete Example 2 – Maintenance Tasks



SUBTASK 020-001
Disconnect these electrical connectors from the aft outflow valve [1]:

NOTE: You can access the connectors through the valve door opening.

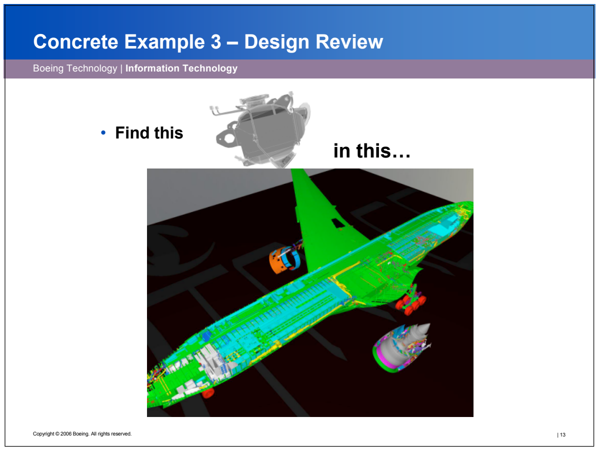
(a) ELACT 1 [3]
(b) ELACT 2 [2]
(c) Feedback Module 1 [5]
(d) Feedback Module 2 [6]

SUBTASK 020-002
(2) Remove the nut [7] and the washer [8] that attaches the bonding jumper [9] to the aft outflow valve [1].

SUBTASK 020-003
(3) Disconnect the bonding jumper [9] from the aft outflow valve [1].

SUBTASK 860-002

WARNING: DO NOT INSERT YOUR HANDS OR TOOLS IN THE OUTFLOW VALVE WHEN THE VALVE IS IN OPERATION. IF YOU INSERT YOUR HANDS OR TOOLS IN THE OUTFLOW VALVE WHEN IT OPERATES YOU CAN CAUSE

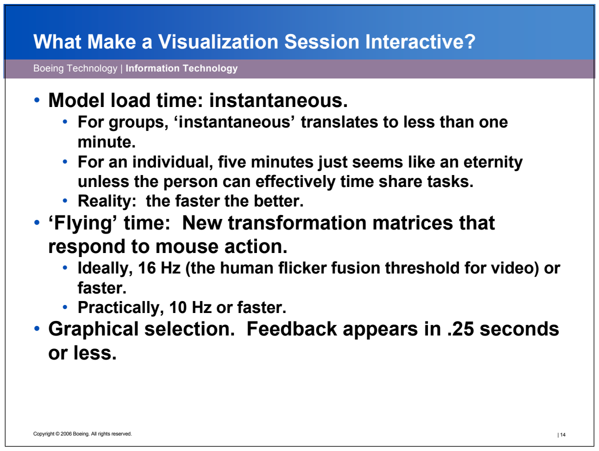


Concrete Example 3 – Design Review

Boeing Technology | **Information Technology**

- Find this

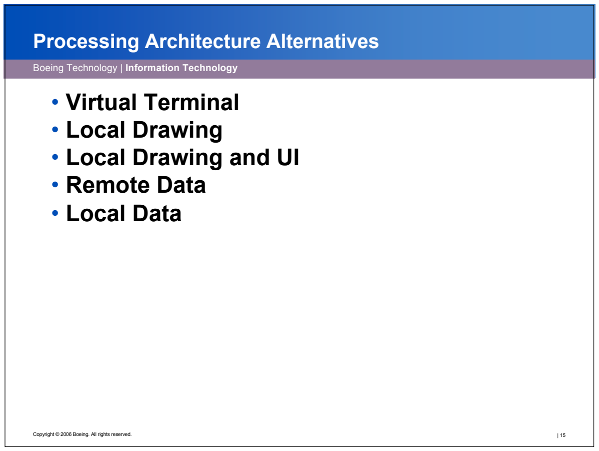
in this…



What Make a Visualization Session Interactive?

Boeing Technology | **Information Technology**

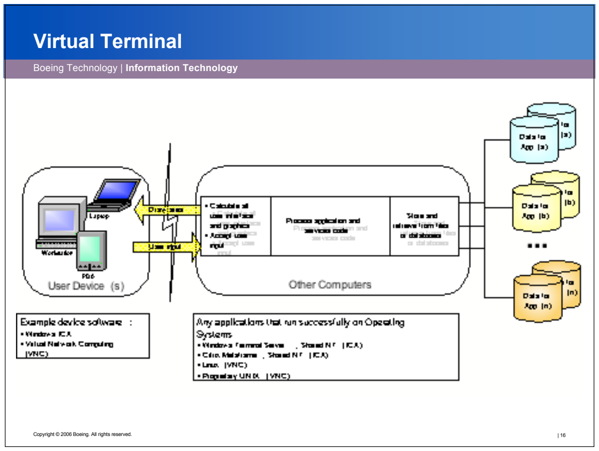
- Model load time: instantaneous.
  - For groups, 'instantaneous' translates to less than one minute.
  - For an individual, five minutes just seems like an eternity unless the person can effectively time share tasks.
  - Reality: the faster the better.
- 'Flying' time: New transformation matrices that respond to mouse action.
  - Ideally, 16 Hz (the human flicker fusion threshold for video) or faster.
  - Practically, 10 Hz or faster.
- Graphical selection. Feedback appears in .25 seconds or less.



Processing Architecture Alternatives

Boeing Technology | **Information Technology**

- Virtual Terminal
- Local Drawing
- Local Drawing and UI
- Remote Data
- Local Data

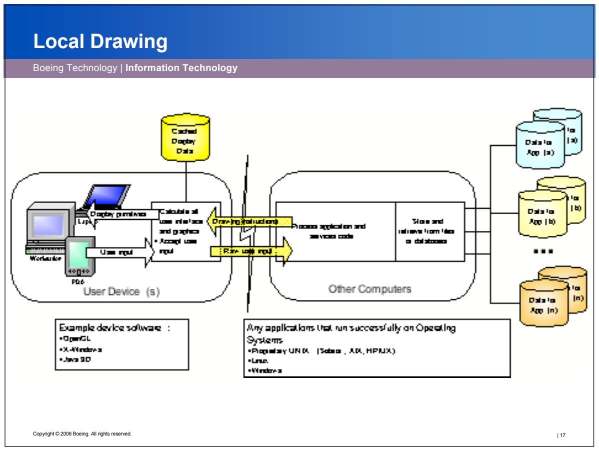


Virtual Terminal

Boeing Technology | **Information Technology**

Example device software :
- Windows ICA
- Virtual Network Computing (VNC)

Any applications that run successfully on Operating Systems
- Windows Terminal Server , Shared NT (ICA)
- Citrix Metaframe , Shared NT (ICA)
- Linux (VNC)
- Proprietary UNIX (VNC)

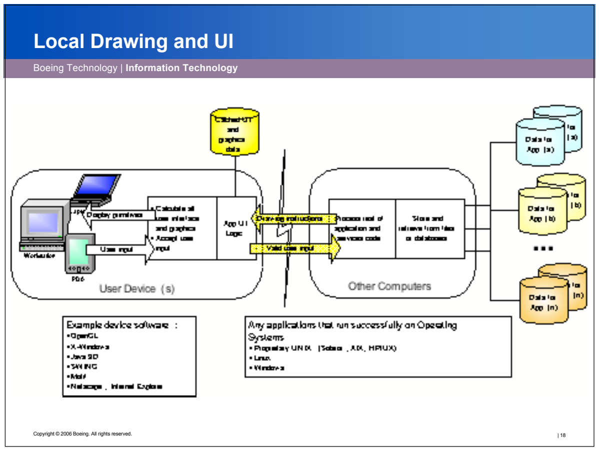


Local Drawing

Boeing Technology | **Information Technology**

Example device software :
- OpenGL
- X-Windows
- Java 3D

Any applications that run successfully on Operating Systems
- Proprietary UNIX (Solaris , AIX, HPUX)
- Linux
- Windows

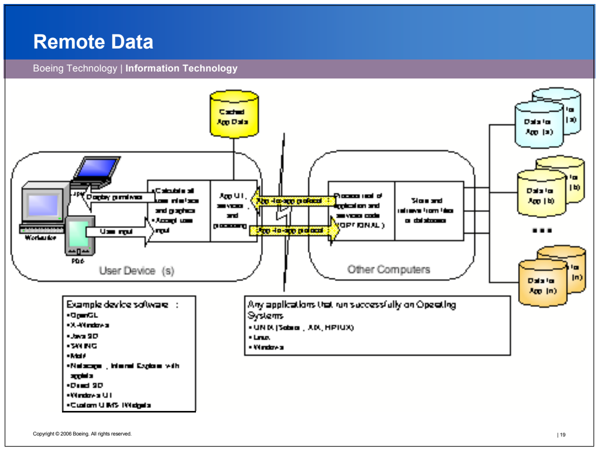


Local Drawing and UI

Boeing Technology | **Information Technology**

Example device software :
- OpenGL
- X-Windows
- Java 3D
- SWING
- Motif
- Netscape , Internet Explorer

Any applications that run successfully on Operating Systems
- Proprietary UNIX (Solaris , AIX, HPUX)
- Linux
- Windows

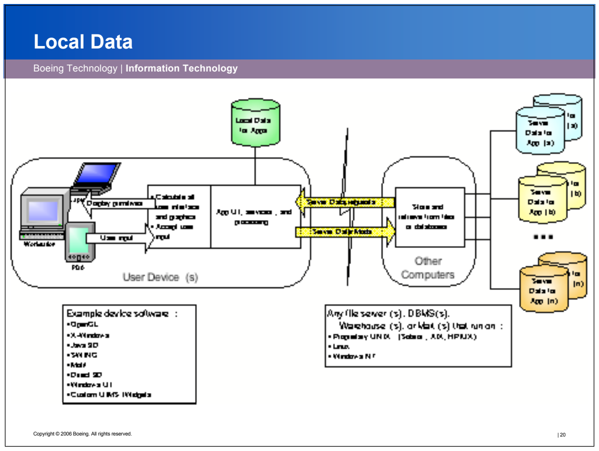


Remote Data

Boeing Technology | **Information Technology**

Example device software :
- OpenGL
- X-Windows
- Java 3D
- SWING
- Motif
- Netscape , Internet Explorer with applets
- Direct 3D
- Windows UI
- Custom UIMS Widgets

Any applications that run successfully on Operating Systems
- UNIX (Solaris , AIX, HPUX)
- Linux
- Windows

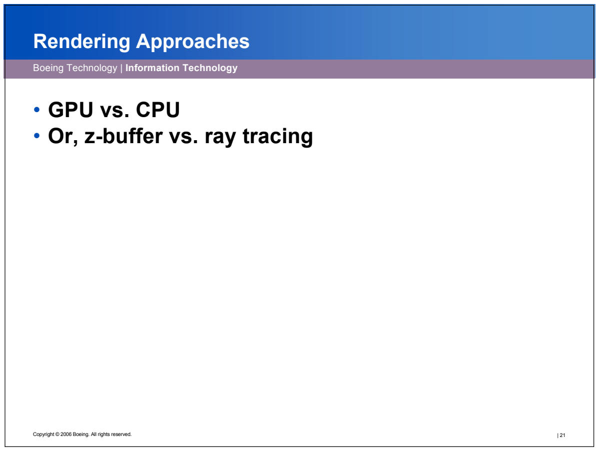


Local Data

Boeing Technology | **Information Technology**

Example device software :
- OpenGL
- X-Windows
- Java 3D
- SWING
- Motif
- Direct 3D
- Windows UI
- Custom UIMS Widgets

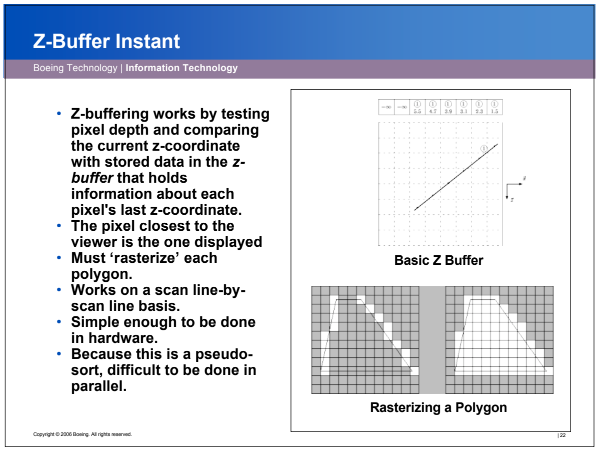
Any file server (s), DBMS(s), Warehouse (s), or Mart (s) that run on :
- Proprietary UNIX (Solaris , AIX, HPUX)
- Linux
- Windows NT



Rendering Approaches

Boeing Technology | **Information Technology**

- GPU vs. CPU
- Or, z-buffer vs. ray tracing



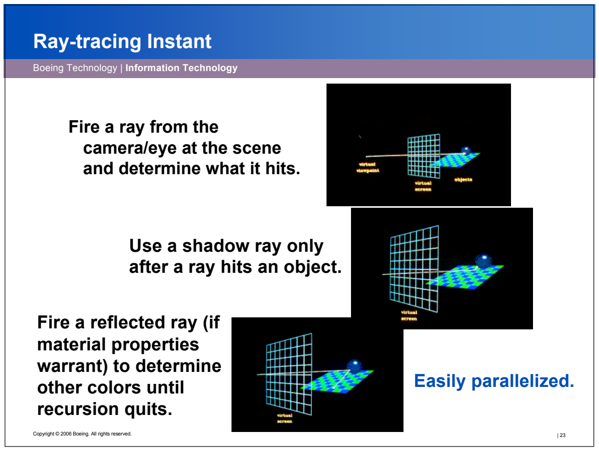
Z-Buffer Instant

Boeing Technology | **Information Technology**

- Z-buffering works by testing pixel depth and comparing the current z-coordinate with stored data in the *z-buffer* that holds information about each pixel's last z-coordinate.
- The pixel closest to the viewer is the one displayed
- Must 'rasterize' each polygon.
- Works on a scan line-by-scan line basis.
- Simple enough to be done in hardware.
- Because this is a pseudo-sort, difficult to be done in parallel.

Basic Z Buffer

Rasterizing a Polygon



Ray-tracing Instant

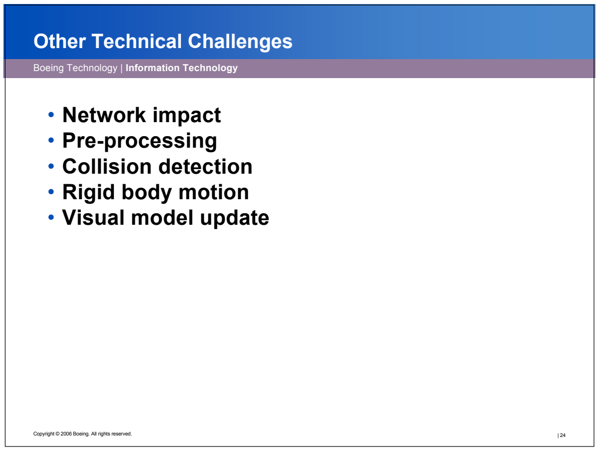
Boeing Technology | **Information Technology**

Fire a ray from the camera/eye at the scene and determine what it hits.

Use a shadow ray only after a ray hits an object.

Fire a reflected ray (if material properties warrant) to determine other colors until recursion quits.

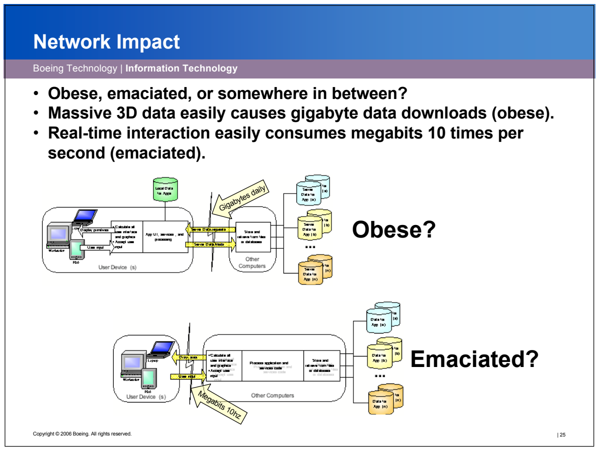
Easily parallelized.



Other Technical Challenges

Boeing Technology | **Information Technology**

- Network impact
- Pre-processing
- Collision detection
- Rigid body motion
- Visual model update

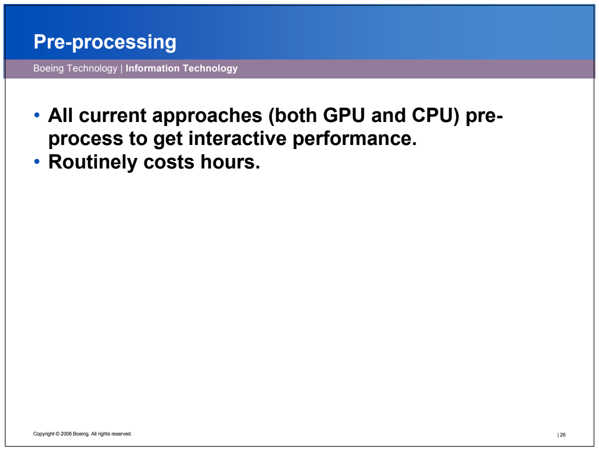


Network Impact

Boeing Technology | **Information Technology**

- Obese, emaciated, or somewhere in between?
- Massive 3D data easily causes gigabyte data downloads (obese).
- Real-time interaction easily consumes megabits 10 times per second (emaciated).

| 25

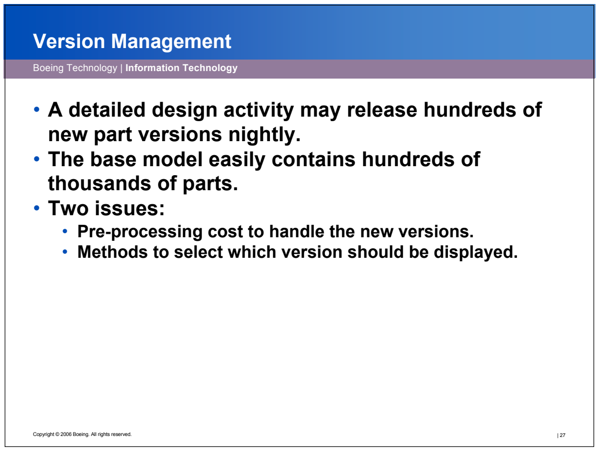


Pre-processing

Boeing Technology | **Information Technology**

- All current approaches (both GPU and CPU) pre-process to get interactive performance.
- Routinely costs hours.

| 26

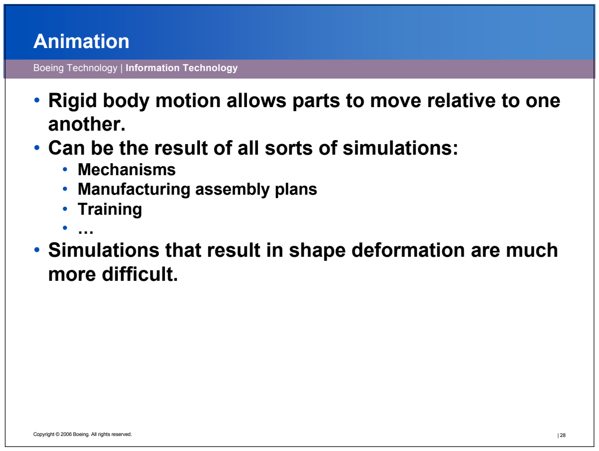


Version Management

Boeing Technology | **Information Technology**

- A detailed design activity may release hundreds of new part versions nightly.
- The base model easily contains hundreds of thousands of parts.
- Two issues:
  - Pre-processing cost to handle the new versions.
  - Methods to select which version should be displayed.

| 27

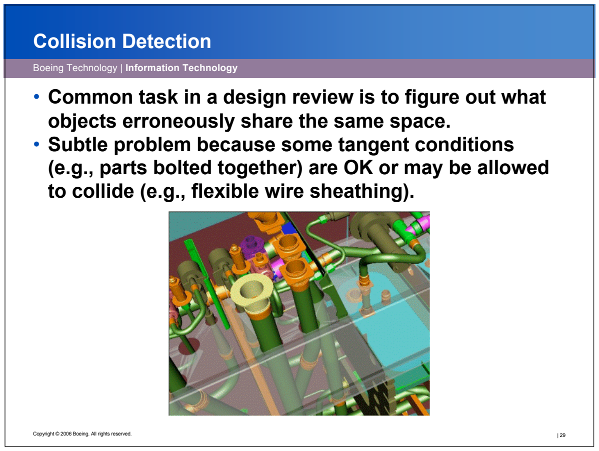


Animation

Boeing Technology | **Information Technology**

- Rigid body motion allows parts to move relative to one another.
- Can be the result of all sorts of simulations:
  - Mechanisms
  - Manufacturing assembly plans
  - Training
  - …
- Simulations that result in shape deformation are much more difficult.

| 28

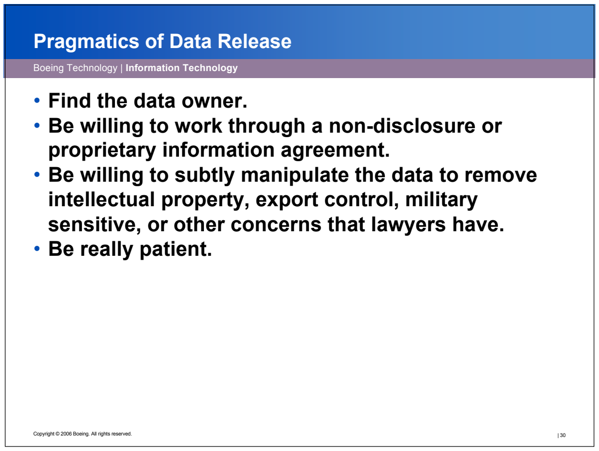


Collision Detection

Boeing Technology | **Information Technology**

- Common task in a design review is to figure out what objects erroneously share the same space.
- Subtle problem because some tangent conditions (e.g., parts bolted together) are OK or may be allowed to collide (e.g., flexible wire sheathing).

| 29

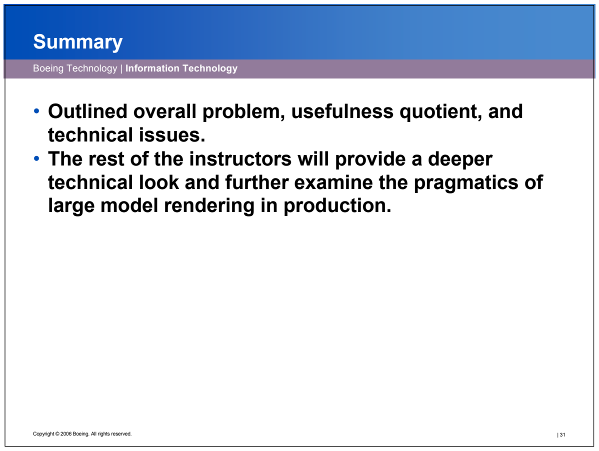


Pragmatics of Data Release

Boeing Technology | **Information Technology**

- Find the data owner.
- Be willing to work through a non-disclosure or proprietary information agreement.
- Be willing to subtly manipulate the data to remove intellectual property, export control, military sensitive, or other concerns that lawyers have.
- Be really patient.

| 30



Summary

Boeing Technology | **Information Technology**

- **Outlined overall problem, usefulness quotient, and technical issues.**
- **The rest of the instructors will provide a deeper technical look and further examine the pragmatics of large model rendering in production.**

BOEING

Strategies for Consistent Image Partitioning

David J. Kasik
The Boeing Company

Graphic display technologies have traditionally targeted devices with midsize screens. However, devices with small and large screen sizes are gaining popularity, with users increasingly attempting to access complex images using small-screen devices. To display high-quality images irrespective of screen size, new methods of visualization become necessary.

Computer graphics display technology has reached a crossroads. New devices have dramatically different display characteristics and continue to rapidly evolve. The most compelling difference is screen size, which now ranges from tiny (as in cell phones or watches) to gargantuan (as in display walls and theater screens). Such variation creates significant problems in the design of images that communicate irrespective of screen size, especially considering the wide variety of displayable images. The "Designing for Variable Screen Sizes" sidebar discusses efforts to integrate images traditionally designed for midsize screens into devices with a variety of screen sizes.

As available computing technology expands, the overall user community has a significantly larger palette of graphics devices available to them. As Figure 1 illustrates, most current research investment aims at medium-sized (10- to 20-inch) screens. Although people are now buying numerous devices with small and large screens, investment in improving the interactive experience has lagged behind.

Graphical user interfaces (GUIs) dominate user interaction with midsize screens, but they neither translate well to small screens nor take advantage of the expanded area available on large screens. Adding complexity is the growing number of people with wide-ranging skill levels, education, and cultural backgrounds using devices with a variety of screen sizes. Finally, many companies' user population spans the globe; thus designers, developers, and deliverers must collaborate to ensure timely delivery of quality products.

This article deals with a fundamental problem: How can designers or users themselves partition images to maximize communication bandwidth to a person viewing the image on a small screen?

Maximizing communication impact

Virtually any 2D image, series of images that create a movie, or 3D image can be drawn just about anywhere. Not all images are created equally, however, and graphic communication techniques vary significantly. Massironi's excellent taxonomy delineates the types of images people have used to communicate visually throughout history.[1] Every image's basic goal is similar: communicate the right information in an easily consumable format.

Although many graphic styles are available, the design, analysis, assembly, and maintenance of complex physical products such as commercial airplanes and satellites use only a few. Boeing and other industrial companies that produce physical products primarily use two types of representational drawing, both subsets of Massironi's taxonomy: *technical drawings* and *descriptive geometry*.

These two types of graphic images appear in a number of different forms. Whereas engineering drawings must be dimensionally accurate and provide the textual information necessary for construction and assembly, other technical drawings don't require such accuracy. For example, technical illustrations, which depict parts relationships for maintenance, and production illustrations, which contain instructions for assembling hydraulic and electrical systems, don't require dimensional accuracy.

Early 3D concept drawings represent notions to customers and initiate more detailed configuration and preliminary design activities. Designers use descriptive geometry techniques to generate detailed product models with 3D surfaces and solids.

Visual task analysis

People who view graphic images of models representing physical products participate in the product's life cycle. Boeing's typical internal user community includes sales, design, building, and maintenance staff. A commercial airplane's user

1070-986X/04/$20.00 © 2004 IEEE

Designing for Variable Screen Sizes

The variety of graphics display devices has significantly increased since the early 1990s. Announcements of new devices seem to appear weekly. Advances in midsize graphics devices, inexpensive memory, and computing processors have largely removed restrictions on the types of pictures users could draw (images composed only of lines or text with a limited number of colors, for example).

Midsize screens

Current devices use raster technology to provide excellent display quality in terms of color, brightness, and pixel size. Newer devices (such as IBM's MD22292A1 liquid crystal display) come close to achieving a pixel size that attains the minimum angle of resolution (between .00833° and .01667°) for an individual.[1] Vendors are developing unique configurations with midsize screens for special effects such as:

- Automatic stereo (http://www.mrl.nyu.edu/projects/autostereo/)
- Multiple parallel planes for true 3D (http://www.3dmedia.com/products.html)
- Specially treated plastic cubes for true 3D (http://3dtl.com)

Small screens

Devices, like users, are becoming increasingly mobile. Cell phones, personal digital assistants (PDAs), wearable computers with glasses or head-mounted displays, tablet computers, and notebook computers rely on small screens and less capable computers than devices with midsize screens. Small screens generally have fewer pixels and a limited set of colors.

Such limitations haven't decreased these devices' popularity nor have they discouraged designers from customizing general devices to build dedicated, special-purpose devices (onboard navigation aids and barcode scanners, for example). The components needed to build portable devices continue to improve; thus, the only factors that won't change significantly are the display size and the number of pixels devices can display.

Boeing supplies cell phones to most of its employees, and many employees also carry other devices with small screens. Assembly-line mechanics, quality-assurance inspectors, shipping and receiving personnel, and others use the small devices to access the corporate graphic images they need to effectively do their jobs.

Large screens

Large-screen devices come in a range of geometric configurations. Plasma panels use an alternate display technology. Projectors, the de facto standard, increase image size on a planar screen. The Elumens (http://elumens.com) VisionDome products use a conventional projector with a special lens for a hemispherical screen. A truncated cylinder surrounds the viewer's head in a Panoscope360 (http://panoscope360.com). A CAVE (http://www.fakespacesystems.com/workspace1.shtml) projects images onto the planar walls to form a cube. Holograms (http://www.zebraimaging.com) have specially etched large-format film to create a full-3D illusion.

Large-screen device problems differ from those of midsize screens. Because large-screen graphics drivers are based on the pixel resolution of a midsize screen, users can experience a dramatic increase in pixel size depending on their distance from the screen. This phenomenon increases the viewer's angle of

continued on p. 34

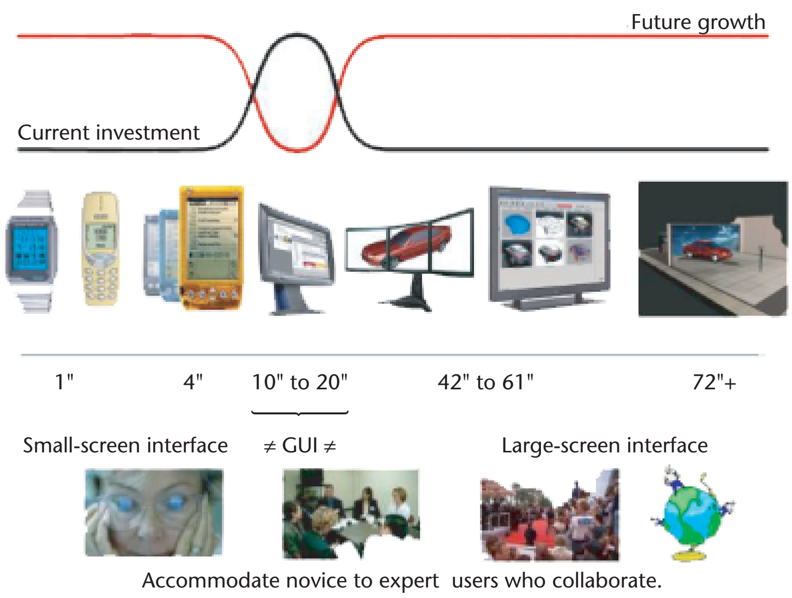


Figure 1. Users can choose among a wide array of devices with different screen sizes although most research investment still goes toward midsize screens.

continued from p. 33

resolution and reduces brightness (a measure of image crispness), making the image appear less smooth than it does on a midsize screen.[2]

User interface evolution

The standard user interface paradigm for application command and control has evolved significantly from text commands typed into glass teletypes. Although early graphics devices supported graphical selection of commands and objects with lightpens, thumbwheels, or tablets, pervasive use of graphic user interfaces (GUIs) didn't occur until raster graphics devices and mouse pointers became inexpensive in the mid-1980s. GUIs have been the basic interface standard since then.

The fundamental assumptions of a GUI for application command and control don't translate well to either small- or large-screen devices. Not only is screen area a problem on small devices, but interaction through a keyboard and mouse is also difficult.

Many large companies, startups, and research projects have suggested ways to interact with small screens. However, most of the efforts focus on effectively filling out forms and entering text rather than interacting with complex 2D or 3D graphics. For example, cell phones have developed enough thumbs-only 10-key typists to sustain text-messaging services. Other companies are introducing products that allow better interfaces for form users.

Microsoft, AT&T, IBM, and other large companies, as well as startups such as Aligo, Cysive, and Trigenix are introducing products for mobile, small-screen devices that focus on applications with forms and Web-based interfaces as opposed to applications in which the user directly interacts with graphics. Several interesting research projects are based on small-screen devices, including Geney (http://geney.juxta.com), a tool for teaching genetics using multiple cooperating PDAs.

Large screens expand the field of view and make both new interaction styles and devices possible. A GUI is well tuned to an individual's interacting with a screen. In fact, the most common technique for controlling interaction with a large-screen device is to engage a designated driver using a mouse, keyboard, and midsize screen. The large screen becomes a slave to the midsize screen, and their displays are identical.

Efforts to explore new user interface strategies for single users and large screens focus on devices (data gloves and head-mounted displays, for example) that enhance the virtual reality illusion. VR interactive tools generally offer interaction that is a GUI analogue but driven by a data glove. Two-handed input (see, for example, a list of Buxton's work at http://billbuxton.com/papers.html#anchor1442822) and tangible interfaces (such as the work by MIT's Tangible Media group, http://tangible.media.mit.edu) are interesting possibilities for extending the range of interface tools for large-screen devices beyond windows, icons, mouse, and pointer (WIMP).

Some early work (see for example, Cinematrix, http://cinematrix.com; HI-Space, http://www.hitl.washington.edu/projects/hispace/; and Roomware, http://www.ipsi.fhg.de/ambiente/english/projekte/projekte/roomware.html) investigated alternate user interfaces that work with groups of users and multiple large-screen devices.

Integrating graphics and user interaction

Current midsize devices let users efficiently and effectively control and manipulate 2D, video (that is, a collection of graphic images displayed one after another fast enough to give the viewer the illusion of motion), and 3D graphic images displaying user data in addition to user interface objects. Current large-screen devices let graphic images communicate effectively. A significant amount of work is needed to expand the interaction tools available to a single user and to make the devices more effective for colocated groups. Promising work is ongoing in both areas. The next logical extension is interactive techniques to improve user efficiency.

Similar work for small-screen devices is lagging. Basic interaction is limited to text and forms and excludes interaction with graphic images. Part of this can be attributed to the lack of interactive tools for entering text and coordinate positions. However, a more fundamental problem exists: It's difficult to understand how to shrink or partition a 2D, video, or 3D graphic image while retaining communication efficiency.

References

1. K.R. Boff, L. Kaufman, and J.P. Thomas, eds., *Handbook of Perception and Human Performance: Sensory Processes and Perception*, vol. 1, John Wiley & Sons, 1986.
2. D. Kasik et al., "Evaluating Graphics Displays for Complex 3D Models," *IEEE Computer Graphics and Applications*, vol. 22, no. 3, May/June 2002, pp. 56-64.

community extends far beyond company boundaries. For example, airline personnel buy the product, perform maintenance, and order spare parts. Partners design and analyze significant airplane components. Suppliers formulate proposals to build many of the parts. Government agencies determine if the final product performs according to published regulations and guidelines.

Although current practice still generates a lot of paper, the user communities rely on computer graphics in 2D, video, and 3D. A wide variety of visual tools developed both internally (such as FlyThru) and externally (such as Dassault Systemes' Digital Mock-up Utility) is available to users. The tools can show or hide objects, apply texture, identify spatial position, generate multiple concurrent views, and so on. None of the tools fundamentally change the style in which the image is rendered.

Once a user has an image and a set of visual tools, he or she can extract a significant amount of information using visual analysis techniques. Users can perform a wide variety of tasks using visual analysis alone. By observing users at work and consulting with different groups of users, I developed a set of typical tasks that can be performed using visual analysis.

First, find an object in a complex scene given

- the physical object,
- a picture of the object,
- a mental image of the object, or
- a verbal description of the object.

Second, focus on the found object to better understand surface characteristics (smoothness or roughness, for example). A user can determine characteristics by visually inspecting the object or by interpreting its physical characteristics (such as the effects of aerodynamics flow or stress).

Next, look at objects in the immediate surrounding volume to

- identify discrepancies in space consumption. (Do the pieces occupy too much of the same space?)
- determine interference and overlap. Whether you do this through direct visual inspection or by recognizing the results of computing overlaps in a batch mode, you must perform a visual analysis to determine whether the interference is acceptable.
- find gaps or voids between objects (that is, ensure proper clearance). A method to measure distance between objects often supplements visual gap analysis.
- trace a path that connects objects. In an airplane, these connections are long, skinny things like hydraulic tubes and wire bundles.

Next, visually scan the scene to find

- misplaced objects,
- forgotten objects (“Drat, we forgot a wing!”),
- patterns from similar objects or members of a family of parts,
- objects that might not be in a production configuration (debugging or placeholder objects, for example),
- maintenance accessibility or manufacturing assembly problems, or
- a part’s conformance to the design. For example, you must periodically examine tools that are in the field to determine needed upgrades.

The next task involves observing scene dynamics (typically by animation) to

- recognize dynamic interference conditions (such as display results from kinematics or mechanisms analysis, vibration, or tolerance buildup),
- follow system flow (fluid flow in hydraulic tubes, for example),
- detect effect of loads, aerodynamic flows, and so on over time, or
- receive instructions about assembly (for manufacturing) and disassembly and reassembly (for maintenance) sequences.

All these tasks assume a single window displaying one style. It’s also useful to compare multiple versions of the same set of objects for

- subjective preference (for example, “I like the way that car’s hood reflects the lights on the showroom floor”) or
- net version change.

Boeing’s user community has reasonable access to devices with small, midsize, and large screens. Users who need graphic information delivered directly to a job site are increasingly interested in small devices. These users perform only a subset of the visual analysis tasks listed above: they find objects in complex images, trace paths, check design conformance, follow flows, and understand assembly/disassembly/reassembly sequences and documentation.

Engineers with access to midsize and large screens, generally in an office environment, perform most other tasks during product design and analysis.

Supporting the thousands of Boeing users with small-screen devices requires (at a minimum) partitioning graphic images into meaningful chunks.

Seeing and understanding images

Substantial research examines how people actually see and perceive the physical world.[2] Other research[3] has determined how light is transmitted into the eye and what areas of the brain process the signals.

When a person starts navigating graphic images, additional brain processing occurs. Siegel and White differentiate navigation strategies (landmark, route, and survey) through landscapes[4] and show that such strategies are hierarchically related.[5] Other experiments show that navigation strategies might not be hierarchical; rather, they depend on whether the user recognizes the landscape in a scene.[6]

Scientists don't perfectly understand the cognitive processes a person uses to identify an image (for example, "Is that image an engine or a wing?"). Biederman has developed a reasonable theory[7] that describes a four-stage mental process. A person recognizes a 3D shape during the first three stages, and he introduces the notion of generic *geometric ions* (geons). The fourth stage determines the 3D shape's meaning and doesn't have a corresponding generic basis because the meaning depends on a person's culture and background.

Partitioning strategies

To develop effective partitioning strategies for small screens, a designer or image author must consider the areas discussed previously: limitations of small screens for the images used in complex products, tasks involved in visual analysis, and the fundamentals of human visual processing.

A partitioning strategy for the types of graphic images common in product development must preserve as much of the images' communication impact as possible. I applied Massironi's more general graphic communication techniques[1] to develop several principles for partitioning graphic images:

- Graphic images that don't require dimensional accuracy (such as some technical drawings) can be subtly changed. People recognize many objects even when they are subtly redrawn—for example, the 26-character alphabet is recognizable in many fonts.

- Cultural background impacts communication. Visually changing a font in the English alphabet doesn't communicate completely to people who only read Chinese or Greek.

- Users gather a substantial amount of information from an image's general context.

- A complex image, the kind routinely used in companies like Boeing, forces viewers to scan multiple zones to identify the most interesting zone. The image author can graphically enhance areas to attract initial interest.

- Graphic images have natural edges that define structure.

Designers can use any of the partitioning strategies to adapt graphic images to small screens. I haven't found a documented systematic approach to the partitioning problem.

Partitioning strategies differ from technologies that actually break an image into smaller parts. For example, a now defunct start-up company, Newstakes, used image processing to break images into multiple areas. However, Newstakes users had to know the area's size, which area to display first, and so on.

The most straightforward solution to the problem is to treat a small screen as you do any other screen: display the entire image (2D, video, or 3D) and give the user full control of viewing transformations. If adequate network bandwidth and local device memory are available, the problem becomes providing a user interface that allows extremely fast interaction with local controls that consume only a small amount of screen space.

Figure 2 shows a typical set of controls for an engineering drawing display intended for a midsize screen. When squeezed to the size of a typical PDA screen, the user controls alone dominate the display. The display gives no context, one of the key principles in graphic image communication. Other strategies subdivide the picture into discrete tiles, which helps the user overcome screen size, device processing power, and network bandwidth limits.

2D images

Millions of 2D images document essential information for existing Boeing airplanes. The vast majority are technical drawings consisting of lines, curves, and text, and were intended to be delivered on nothing smaller than letter-size paper.

A typical PDA screen is 2.25 × 3.0 inches, with a pixel resolution of 240 × 320. Both measurements affect partitioning strategies.

The most basic display technique is to draw the whole image in a single tile, as Figure 3 shows.

The graphical viewer provides navigation tools for panning and zooming. The user can figure out where to start based on the general graphical forms on the display device. Zooming can be problematic for raster images because the picture quality degrades as the image is magnified. Vector forms (such as computer graphics metafile [CGM] or scalable vector graphics [SVG]) retain better image quality because the raster image is regenerated at each step of the magnification process.

Most conventional pan tools use sliders to control *x* and *y*; zoom uses a magnifying glass or numeric value. Both are necessary for images such as Figure 3, where without them users can't read the text.

Conventional pan and zoom tools, however, rarely provide context. Bier et al. describe additional zoom techniques that led to the Magic Lens implementation.[8] Other context-preserving zoom tools are available, such as the Idelix Pliable Display Technology (http://www.idelix.com).

Partitioning strategies for 2D images. Some 2D graphic image-partitioning strategies split the initial image into a set of tiles. In multiple tiles,

- individual tiles should be of equal size for cross-tile navigation. The user initially receives a single tile at a higher zoom level than the image shown in Figure 2. Users can often understand information in the first picture, which avoids having to transmit the entire image across a low-bandwidth network to a limited-memory device.

- text, especially in technical drawings, represents a difficult challenge for any tiling scheme. The rule of thumb is that a text string with no blanks should stay on a single tile. If a string won't fit at an acceptable scale, the best strategy is to send a tile that's larger than the screen and give the user pan and zoom tools.

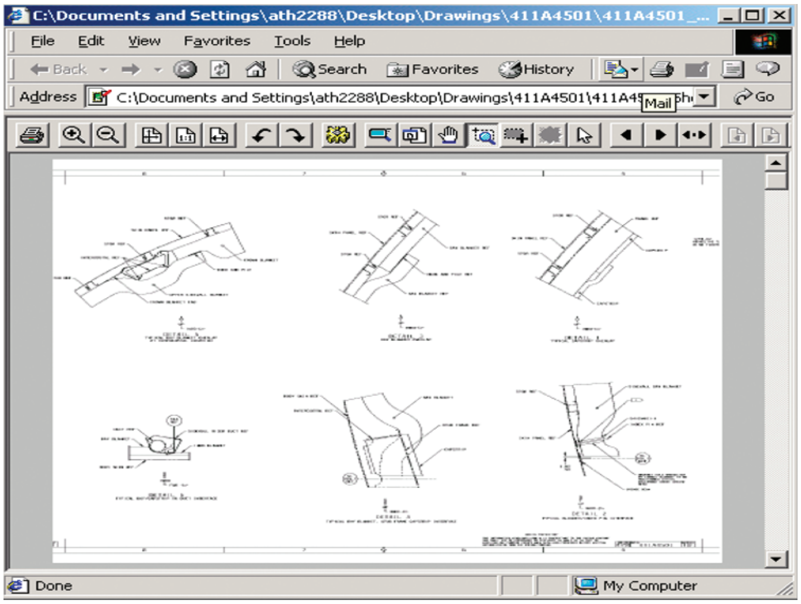

C:\Documents and Settings\ath2288\Desktop\Drawings\411A4501\411A4501_...
File
Edit
View
Favorites
Tools
Help
Back
Search
Favorites
History
Address
C:\Documents and Settings\ath2288\Desktop\Drawings\411A4501\411A45
Mail
Go
Done
My Computer

Figure 2. Typical engineering drawing and controls.

- each tile should slightly overlap its neighbors. This keeps context as the user navigates to new tiles and facilitates path following.

- any tiling scheme will occasionally fail. Reverting to a single tile containing the entire image (under either application or user control) is an acceptable risk-management technique.

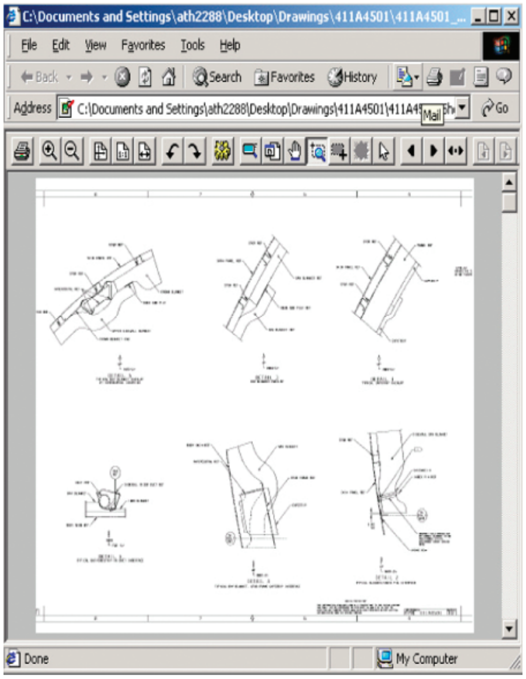

C:\Documents and Settings\ath2288\Desktop\Drawings\411A4501\411A4501_...
File
Edit
View
Favorites
Tools
Help
Back
Search
Favorites
History
Address
C:\Documents and Settings\ath2288\Desktop\Drawings\411A4501\411A45
Mail
Go
Done
My Computer

Figure 3. Engineering drawings redrawn for a 2.25" x 3.0" screen.

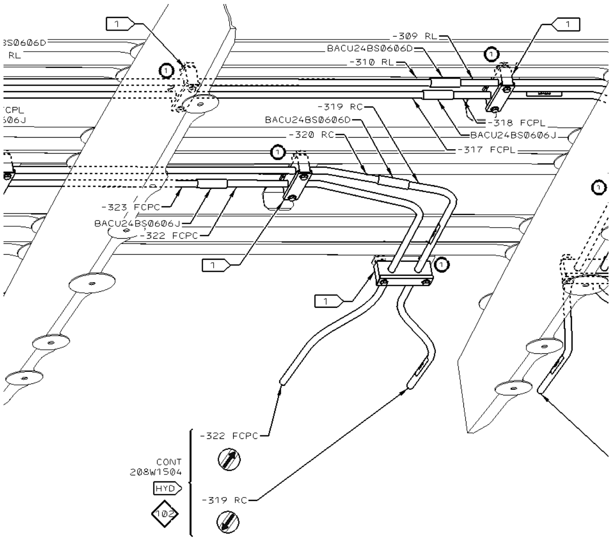


Figure 4. Set of assembly routing instructions for hydraulic tubing. The image author draws viewers' attention to several areas by marking them with a 1 in a bold-type circle.

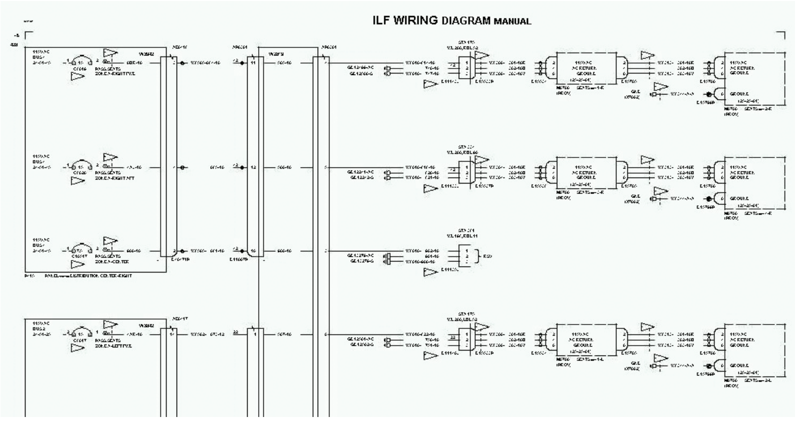


Figure 5. Sample wiring diagram. The long horizontal and vertical lines indicate the image's tile structure.

I've developed four strategies based on these principles.

Strategy 1: Author-defined visual targets. Authors of graphic images, especially complex images, typically use heavyweight lines or bold or italicized text to draw attention to a specific region, such as the bold borders around the number 1 in Figure 4.

The strategy should identify as many attention-getting landmarks as possible. If there's only one clear landmark on an image, it should be the first tile the user sees. Otherwise, the strategy should use a consistent heuristic to determine which highlight to show the user first (for example, the first tile shown contains a landmark that's above and to the left of other tiles with landmarks).

Strategy 2: Follow long horizontal and vertical lines. Not all graphic images contain attention-getting cues. In the wiring diagram in Figure 5, for example, users can infer the tile structure from the long horizontal and vertical lines.

After strategy 2 determines the basic tile size, strategy 1's results will identify the first tile to be displayed. If the image author didn't provide a specific landmark, the display strategy should use a consistent heuristic (for example, always draw the tile in the upper left corner first) to start.

Strategy 3: Follow long horizontal and vertical white space. Figure 3 shows a different technique for visually finding blocks in a complex graphic image. In this figure, long horizontal and vertical white spaces define the tile edges. White space clearly delineates interesting areas that we can use as tiles. As in strategy 2, if the image author didn't establish a landmark, the strategy should use a consistent heuristic to determine which tile to display first.

Strategy 4: Follow long, arbitrarily sloped lines. Another alternative when the author doesn't provide landmarks is to supplement the techniques from strategy 2 (long horizontal and vertical lines) with the identification of long lines with arbitrary slopes. Partitioning tiles along long lines facilitates path following. Again, the strategy should determine the first tile to display in a consistent manner.

Cross-tile navigation. These four strategies will likely result in tiles that are slightly larger than the screen to accommodate nonblank text strings.

Screen area is a significant constraint when providing context and navigation assistance for a graphic image that's split into multiple tiles. As noted previously, pan and zoom tools for an individual tile can consume a significant amount of screen space, although lens tools help in that situation.

One possible approach to cross-tile navigation lets users select from a list of tile names. This forces the user to make an extra selection, complicating path following. The key to providing some level of context-preserving navigation is to include navigation aids in each tile so a user need only take one action to move to the next tile.

A more straightforward technique is to use a set of dynamically computed navigation arrows, as Figure 6 shows. Each arrow sits at the physical edge of the screen.

Users can select the arrows, which indicate additional tiles in each of eight directions. If a computer program inserts arrows as it sends each tile to the device, absence of an arrow indicates that navigation has taken the user to an edge or corner. This technique gives the user a sense of the image's size and a limited sense of context.

Intelligent Zoom[9] is an alternate technique initially used to shrink wall-sized displays of electric power grid control systems. Intelligent Zoom overlays a map showing the tiles composing the image. When a user selects a tile, the other tiles shrink and the camera seems to enlarge the selected tile. This technique generally consumes less screen space than conventional 2D navigation techniques and provides good visual context. However, the user must make an extra selection to move between tiles.

Video

Many sources of video exist, from Flash animations to collections of camera-captured raster images. Although not required, sound generally accompanies video sequences.

Thus, video affects two senses. In terms of small screens, designers must balance the screen-size problem with the projection-rate problem. Time dominates screen resolution because the motion (both picture and sound) conveys critical information. Local device capacity and network performance drive the strategies more than maximizing communication impact.

Using strictly local device resources for storage and playback provides the most consistent video playback rate. However, device resource limitations make streaming necessary for the foreseeable future. Streaming can be successful as long as network bandwidth supports acceptable video playback rates. Given current and foreseeable variation in network rates (rates change during a playback session), there are four strategies that vary how a source sends individual frames during a single playback session. In other words, if the bandwidth is good enough, send a full resolution picture. If bandwidth degrades, use one of the following strategies to change the image quality.

Strategy 1: Make pixels bigger. In this technique, often used with videophones, the source sends fewer but larger pixels on a frame-by-frame basis. The number of pixels transmitted increases or decreases as network performance varies. For relatively unchanging pictures, some implementations transmit only deltas.

Strategy 2: Draw visual targets at high resolution. Strategy 1 for 2D images notes that authors of graphic images, especially complex images, leave hints about what's most important in the scene. The pixel budget for a single image can be spent in another way: give greater resolution to graphically important areas and less to others.

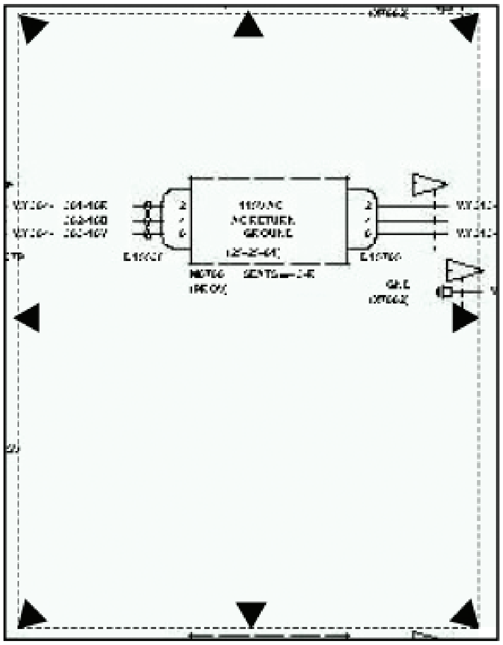


Figure 6. Navigation arrows on a tile.

Strategy 3: Show full frames at fixed intervals. Not every video frame must be visible. The easiest technique is to change the frame rate so that it shows only every *n*th frame, but at full resolution. As bandwidth changes, *n*'s value changes.

Strategy 4: Show key frames. Video relies on intraframe coherence to create the illusion of motion. This means that some frames will be highly similar to preceding frames. A big change from one frame to another indicates a new sequence. By transmitting only key frames, the images have less movement but still communicate basic information.[10]

Degrading sound can help manage variable network speeds. Although voice alone can be slowed down (think of controls on voicemail systems), music is often part of the streaming media. A general strategy dedicates enough available bandwidth to keep the audio playing at a normal rate and changes the graphic images to fit into the remainder. People can more easily adapt to image changes and still obtain most of the information from them.

3D images

Boeing defines its airplanes and many other complex physical products in three dimensions. Most of these products are physically larger than

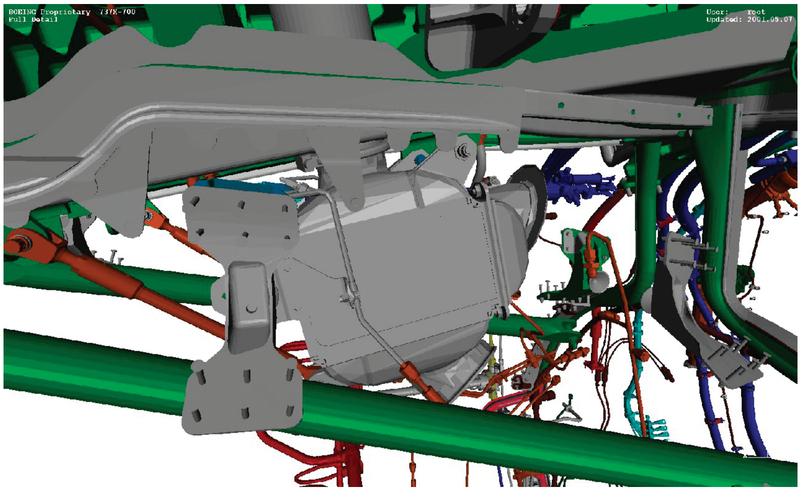


Figure 7. Typical 3D image of airplane structure.

any screen. Figure 7 contains a typical image from a Boeing commercial airplane.

These images fit in Massironi's descriptive geometry category. Computer users performing visual analysis tasks can use interactive techniques to help them understand the data's 3D nature. At a minimum, the interactive techniques give a user control of viewing transformations (scale, rotate, and translate). Computing these operations on the local device keeps performance consistent.

If implemented, the tiling strategies described for 2D images become rectangular solid subdivision strategies. Voxelization techniques can help determine the solid subdivisions. Understanding how voxels relate to their neighbors is a difficult cognitive task, however, because the neighbors are in 3D, not 2D. It's also difficult for users to navigate to a neighbor. Rather than the eight neighbors in 2D, in 3D they have 26.

Thus, the most effective way to deal with 3D on a small screen is to rely on user-controlled viewing transformations and standard techniques for showing context (displaying coordinate axes representing current orientation or thumbnail views, for example).

Implementation and challenges

When implemented, image partitioning strategies help users deal with the screen-space and resolution limitations of small screens. However, companies like Boeing currently have millions of digital graphic images, most of which were designed for midsize to large sheets of paper or display screens. In addition, authors of video sequences for training or assembly and maintenance instructions assume that frame rate will be adequate to depict motion. Manually modifying either form is impractical.

Thus, we need automated techniques to break 2D graphic images into tiles and analyze video frames to adapt to varying network performance levels. Newstakes provided a nontraditional use of image-processing algorithms to address both problems.

Automatic 2D partitioning needs methods to recognize characters (at a minimum, to break on blanks), find visual targets, and identify regions bounded by lines or white space.

A Boeing-Newstakes feasibility study sought to determine whether image processing was practical for images in the wiring diagram class shown in Figure 5. The study demonstrated feasibility but ran out of funding before applying the technology to other types of technical images, and found no general, published strategies resulting in high-quality partitions. Newstakes image-processing control algorithms were cleverly implemented and recognized the rectangular regularity of wiring diagrams. However, the study didn't find a good way to identify the first tile to be displayed.

The strategies I've presented should facilitate development of image-processing partitioning and initial position heuristics for all image classes.

Effective automation tools address a significant percentage of the legacy data problem. Individual window managers, applications, browsers, and browser plug-ins implement local pan and zoom tools inconsistently, and few provide an indication of context. This adds to user confusion, and additional work is needed to provide a more consistent set of user navigation tools. Even more variation exists for local 3D scene navigation.

Finally, I've focused on users who only look at the graphic images. Another large segment of the user community directly interacts with objects within the graphic image. Providing effective graphic selection tools for small screens might require different solutions from those currently implemented for larger screens.

Conclusion

The ability to push well beyond the perceived capabilities of a computing device characterizes all user communities. The world of graphics devices has expanded dramatically over the past few years, and all signs point to an even more dramatic expansion in the next 5 to 10 years.[11]

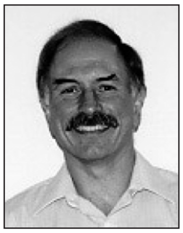
This article starts to address the difficult problem of finding acceptable methods to deal with screen-size variations. Instead of approaching the problem from a technology perspective, my techniques preserve graphic communication impact for specific visual analysis tasks without extensively modifying existing images or overly constraining authors of new images. Adapting visual content to retain graphic communication impact is essential to maximize the effectiveness of new devices.

Acknowledgments

Forrest Warner and Jim Ginn of Boeing provided invaluable assistance in building the inventory of visual analysis tasks. Bill Buxton of the University of Toronto collaborated in developing Figure 1. John Dill of Simon Fraser University and Brian Fisher of the University of British Columbia gave me pointers to neurophysiological literature. Gurminder Singh of the Naval Postgraduate School introduced me to Newstakes technology. Terry Porlier helped solidify early drafts of this article. The technical reviewers and IEEE editors gracefully pointed to areas that let me substantially improve the writing quality. **MM**

References

1. M. Massironi, *The Psychology of Graphic Images: Seeing, Drawing, Communicating*, Lawrence Erlbaum Assoc., 2002.
2. J.J. Gibson, *The Perception of the Visual World*, Houghton Mifflin, 1950.
3. D.H. Hubel and T.N. Wiesel, "Brain Mechanisms of Vision," *Scientific Am.*, vol. 241, no. 3, Sept. 1979, pp. 150-162.
4. A.W. Siegel and S.H. White, "The Development of Spatial Representations of Large-Scale Environments," *Advances in Childhood Development and Behavior*, H.W. Reese, ed., vol. 10, Academic Press, 1975, pp. 9-55.
5. P.W. Thorndyke and B. Hayes-Roth, "Differences in Spatial Knowledge Acquired from Maps and Navigation," *Cognitive Psychology*, vol. 4, 1982, pp. 560-589.
6. V. Aginsky et al., "Two Strategies for Learning a Route in a Driving Simulator," *J. Environmental Psychology*, vol. 17, 1997, pp. 317-331.
7. I. Biederman, "Recognition-by-Components: A Theory of Human Image Understanding," *Psychological Rev.*, vol. 94, no. 2, 1987, pp. 115-147.
8. E.A. Bier et al., "A Taxonomy of See through Tools," *Proc. Computer–Human Interaction* (CHI 94), ACM Press, 1994, pp. 358-364.
9. L. Bartram et al., "The Continuous Zoom: A Constrained Fisheye View Method for Visualizing and Navigating Large Information Spaces," *Proc. User Interface Software and Technology*, ACM Press, 1995.
10. V. Vasudevan, G. Singh, and M. Jesudoss, "Creating a Slide Show Presentation from Full Motion Video," US patent 09/215,004, Patent and Trademark Office, Washington, D.C., 1999.
11. S. Ortiz Jr., "New Monitor Technologies Are on Display," *Computer*, vol. 36, no. 2, Feb. 2003, pp. 13-16.



David J. Kasik is the Boeing Commercial Airplanes (BCA) Information Systems architect for software engineering, geometry and visualization, and user interface technology. He's currently working to understand the impact of large-screen devices on a large group of engineers and the usefulness of small-screen devices for untetherable factory users. Kasik has a BA in quantitative studies from The Johns Hopkins University and an MS in computer science from the University of Colorado. He's a member of the ACM, ACM Siggraph, ACM Sigchi, and the IEEE.

Readers may contact the author at Boeing Commercial Airplanes, PO Box 3707, Seattle, WA 98124; david.j.kasik@boeing.com.

Ten CAD Challenges

Editor: Frank Bliss

David J. Kasik
The Boeing Company

William Buxton
Buxton Design

David R. Ferguson
DRF Associates

We discuss the significant technical challenges facing the CAD industry and research community and present an approach to address these issues.

The world is enamored with the number 10. Perhaps it's because arithmetic is a lot easier when the number 10 is part of an operation. Or maybe David Letterman and many others have finally achieved a long term impact on society with their top and bottom 10 lists.

Ivan Sutherland used the number 10 to bring significant consistency to hidden surface algorithms for computer graphics in his classic paper.[1] It's in honor of Sutherland—developer of the original Sketchpad application, the forerunner of today's computer-aided design (CAD) applications—that we describe our 10 CAD challenges. The concept of CAD itself has expanded into computer-aided manufacturing (CAM) and computer-aided engineering (CAE). Basic CAD techniques are reapplied in a number of places, including electrical and mechanical product development, buildings, and entertainment. In this article, we focus on mechanical product development.

CAD, CAM, and CAE generate massive amounts of data that must be clearly organized and placed under strict configuration control. The CAD industry also provides support to these functions through product lifecycle management (PLM) systems. In assessing the state-of-the-art CAD, we're examining a reasonably mature, respected, and accepted form of technology. The industry's multibillion dollar yearly revenues have been reasonably flat for the past few years. For an introduction to CAD's history, see the "Brief CAD History" sidebar.

We could place our 10 challenges in a number of different categories. We've chosen three: computational geometry, interactive techniques, and scale. Given our knowledge of the state of the art, the categories we would have chosen 10 years ago would likely be different. Each of the categories presents an opportunity to expand CAD's penetration to new user communities and increase its long term impact.

Geometry represents the kernel data form that a designer uses to define a physical product in any CAD application. Correct assembly of the geometric shape, structure, and system components are the basis for figuring out how to produce the complete product (through CAM) and how the product will respond once in service (through CAE). Only a few commonly used libraries implement computational geometry algorithms, an indication of the technology's relative maturity. Our analysis identified three specific geometry challenges:

- geometry shape control;
- interoperability across CAD, CAM, and CAE applications; and
- automatically morphing geometry in a meaningful way during design optimization.

The key component of all CAD applications is the end user. Users' success or failure in interacting with CAD products ultimately governs the success of the products themselves. We extend the meaning of *interactive techniques* for this article to include more than low-level input device, display, and human factor issues. We deliberately include the way users work with the application to accomplish a work task. Using this extension lets us define three difficult challenges with interactive techniques:

- the order and flow of the tasks a user performs to accomplish something,
- the relationships among tasks in a complex design environment, and
- the manner in which old designs can effectively seed new ones.

The third category describes the challenge of scale. In many ways, the CAD market has reached a plateau just because it has not yet discovered ways of going beyond its current limits. We introduce scale challenges that we have observed as the definition of the current limits of CAD penetration. Should these challenges be addressed in a meaningful way, larger growth in CAD business will likely occur. The four scale challenges include

- finding ways to cope with vastly larger quantities of data,
- communicating key concepts to user communities traditionally outside the CAD community,
- reliably migrating data to new versions of software and hardware as product life spans increase, and
- improving productivity for geographically distributed project teams.

As we built this set of challenges, we realized that the sum of the challenges was greater than we expected. Therefore, we suggest an approach that can help others understand and manage the changes needed. We believe that the technical challenges we've identified will lead to fundamental changes in the way people work.

Geometry

Geometry lies at the core of all CAD/CAM/CAE systems and, while not the only item of interest in product design and development, it's the sine qua non of design. Geometry includes both the algorithms and mathematical forms used to create, combine, manipulate, and analyze the geometric properties of objects: points, lines (curves), surfaces, solids, and collections of objects.

Geometry begets three fundamental questions: What are the objects to be represented? What mathematical forms and approximations will be used to represent them? How will information about the representation be computed and used?

Given the power and generality of current design systems, we can imagine that all issues related to these questions had been adequately answered. However, this is not the case. We discuss three geometry challenges (shape control, interoperability, and geometry in design exploration) that still require extensive research and development.

History

To illustrate the evolution of geometry methods and usage we focus on five time periods: pre and early 19th century and early, mid, and late 20th century. In each of these periods there was a fundamental shift in the methods and usage of geometry driven by new design requirements.

Prior to the 19th century, the major use of geometry was to define and maintain line drawings for manufacturing and records keeping. The tools and methods were rudimentary: lines were constructed by ruler and compass methods and curves were traced from a draftsman's or loftsman's spline, a thin wooden or metal strip bent and held to a desired shape using weights (see Figure 1).

In the early 19th century, these methods were augmented with descriptive geometry, primarily using second degree algebraic equations, to provide more precise mathematical descriptions and increase accuracy and precision in engineering drawings. The objects of interest were still lines, but descriptive geometry required new tools (for example, slide rules and tables) to aid in calculating various curve properties (such as point location). Descriptive geometry changed from curves drawn on paper to precise mathematical formulas from which any point on a curve could be calculated accurately.

The next major advance took place early in the 20th

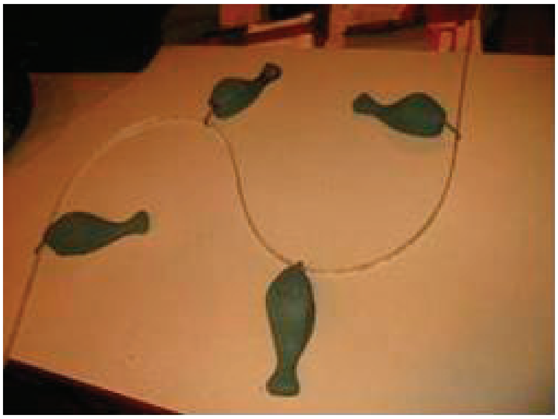
Brief CAD history

CAD was one of the first computer graphics applications in both academia and industry. Ivan Sutherland's Sketchpad at the Massachusetts Institute of Technology and the DAC-1 project at General Motors both started in the early 1960s. Industry developed its own CAD applications, delivered on multiuser mainframes, in the late 1960s and 1970s. The 1980s turnkey systems bundled hardware and software. Current CAD implementations separate hardware and software components. As a result, CAD software most often executes locally on powerful Unix or Wintel workstations with specialized 3D accelerator hardware and stores data on distributed servers.

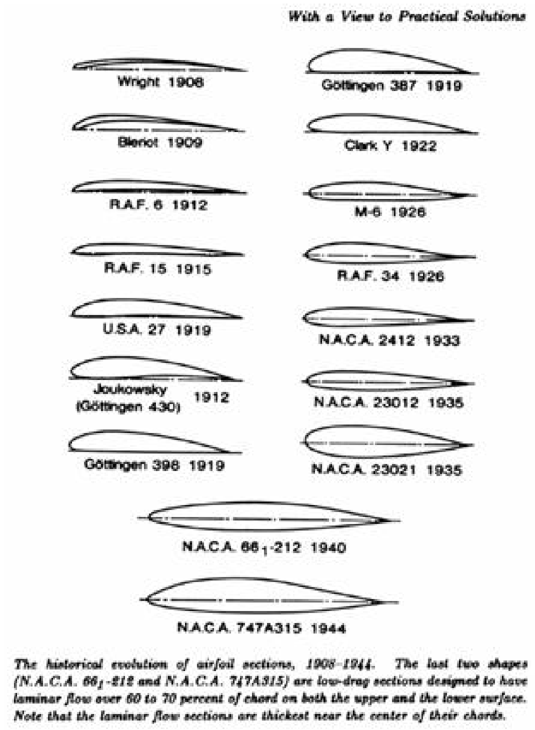
Even though engineering drawings were the dominant output of now defunct CAD vendors—for example, Lockheed's CADAM and commercial companies like AD2000, Applicon, Autotrol, ComputerVision, Gerber IDS, Intergraph, Matra Datavision, and VersaCAD—through the 1980s, early stages of design relied on a variety of unique curve and surface forms. Customized systems were used in aerospace for surface lofting (such as TX-95 at Boeing) and surface design (such as CADD at McDonnell-Douglas) and in automotive industry for surface fitting—such as Gordon surfaces (General Motors), Overhauser surfaces and Coons patches (Ford), and Bezier surfaces (Renault). Current CAD applications rely on more general geometric forms like nonuniform rational b-splines (NURBS).

Solid modeling also started in the late 1960s and early 1970s but from different roots. Larry Roberts worked at MIT to automatically identify solids from photographs. The Mathematics Application Group, Inc. used combinatorial solid geometry to define targets for nuclear incident analysis and subsequently developed ray traced rendering and solid modeling in Synthavision. Other efforts (for example, TIPS from Hokkaido University, Build-2 from Cambridge, (Part and Assembly Description Language PADL) from University of Rochester) had limited industrial impact.

Solid modeling is the 1990s preferred technique for defining 3D geometry in small through large companies, and good modeling software is readily available. The use of 3D is common in computer animation, aerospace, automotive, (and is becoming acceptable for building architecture), and 2D is also used effectively.



1 **Early meaning of spline weights. (Courtesy of Dave Ferguson)**



The historical evolution of airfoil sections, 1908–1944. The last two shapes (N.A.C.A. 66_1-212 and N.A.C.A. 747A315) are low-drag sections designed to have laminar flow over 60 to 70 percent of chord on both the upper and the lower surface. Note that the laminar flow sections are thickest near the center of their chords.

2 NACA Catalog of Airfoil Shapes. (Courtesy US Centennial of Flight Commission)

3 P51 Mustangs.

century. Catalogs describing functionality and application of particular families of curves in engineering began appearing. The catalogs were based on careful scientific analysis and engineering experimentation. For example, the National Advisory Committee for Aeronautics (NACA) generated a catalog of curves that classified airfoil shapes according to flow properties (see Figure 2 and http://www.centennialofflight.gov/essay/Evolution_of_Technology/airfoils/Tech5G1.htm and http://www.centennialofflight.gov/essay/Evolution_of_Technology/NACA/Tech1.htm). Objects became more than curves; they also had attached properties that described their appropriate use in various design activities. Design could now begin with curves known a priori to have properties needed for a viable design.

In the 1940s, Roy Liming at North American Aircraft introduced actual surface representations in his conic lofting system.[2] These were not like today's surface representations, but his system did provide for complete surface definitions rather than simply a family of lines. It was now possible to think of mathematically computing mass, aerodynamic, and hydrodynamic properties. At this point, geometry began changing from describing a physical object to becoming the base for calculating engineering characteristics. This lessened the dependence on physical models and began the push toward virtual design. Tools also began changing. With the increased emphasis on analysis, calculators and computers became required tools.

Liming's conic lofting methods proved their worth in the design of the P51 Mustang airplane (see Figure 3). Liming boasted that the Britain-based Mustangs could fly to Berlin and back because their surface contours did not deviate from the mathematical ideal.[3] However, it was not possible to visualize the geometry without a physical prototype or mock-up.

Modern graphics systems made it possible to display 3D geometry. Designers could visually inspect designs to find mismatched or ill-fitting parts and shape defects. The new display technologies required radical change in how geometry was represented: Everything in a design had to have a precise mathematical representation. Gone were the line drawings used by draftsmen. The old algebraic methods of descriptive geometry were displaced by mathematical splines, NURBS, tensor product splines, and other analytically based forms.

Having a complete mathematical description brought another fundamental change. Previously, the preferred method of constructing a curve or a surface was highly intuitive: lay out sequences of points; construct curves that pass through the points; and then construct a surface from the curves by cross-plotting, conic lofting, or some other means. Using purely mathematical algorithms to interpolate or approximate opened up new possibilities: Curves could be defined by approximating a sequence of points, surfaces could be defined by clouds of points, and geometry could be constrained directly to satisfy engineering constraints (for example, clearances and shape).

Geometric methods and objects used to support product design and definition have changed significantly. As design objectives and systems continue to respond to the ever-increasing need to design rapidly, efficiently, and virtually, geometric methods will need to meet the challenges and change accordingly.

Challenge 1: Shape control

The success that graphics had in forcing everything to have a precise mathematical representation actually increased concern over inflections. The hand-drawn and hand-constructed methods, including Liming, had implicit control of shape whereas the new, polynomial and piecewise polynomial methods (splines, B-splines, and so on) did not. Inflections in a curve passing through a sequence of data points are possible using polynomials or piecewise polynomials even though the data points do not suggest inflections. Therefore, it became important to devise algorithms that not only fit points but also allowed shape control. Shape control defines the occurrence and position of inflection points (points at which the signed curvature of a planar curve changes). The control is needed for both engineering and manufacturing optimization.

Researchers have proposed various schemes for shape control. Most failed because there were always cases for which the methods failed to properly preserve shape. Many methods exist for detecting shape anomalies after the fact, and a person must fix the anomalies by hand. Removing the person in the loop lets geometry pass directly to other applications for optimization. The challenge becomes finding algorithms that avoid anomalies in the first place.

Attempts to modify the methods used previously to account for shape control did not work in general, and it wasn't until the 1980s that we realized the basic principles of those methods were wrong.[4] Although nonlinear methods[4] have proven themselves in a variety of shape control situations, most of these methods have not made their way into commercial CAD systems.

Challenge 2: Interoperability

Current CAD systems do not integrate well with CAE analysis (such as structural mechanics, fluid dynamics, and electromagnetics).[5] For example, computational fluid dynamics (CFD) must interrogate geometry quickly and reliably. Most CFD codes construct a computational grid from the geometry. Building the grid reliably means that there should be no unintended holes in the geometry—that is, the geometry should be what CFD practitioners refer to as watertight. Real geometry from CAD systems is rarely watertight.

Geometry from one CAD system is difficult to translate reliably into another. Estimates peg the cost of interoperability in the US auto industry at $1 billion per year.[6]

Holes, translation error, and other problems arise from two major sources: floating-point arithmetic and tolerances. Floating-point arithmetic, which forces approximations in numerical calculation, is addressable theoretically but not practically. We could impose higher precision (double, triple, and so on) to drive down the resulting errors. Or we could change to a rational arithmetic system and eliminate the need for floating point. Digital floating-point arithmetic is a research area by itself.[7]

Tolerances control the accuracy of computed solutions and are a fact of life in today's CAD systems. A simple example involves calculating the curve of intersection between two surfaces. When the algebraic equations representing the geometry are simple (for example, a plane or a circle), a closed form solution for the intersection exists. However, closed form solutions generally do not exist for operations on equations of a sufficiently high degree (for example, intersecting two bicubic surfaces). Computing the intersection curve uses approximation, a problem independent of precision. Some CAD systems will recompute intersection curves if more accuracy is needed. This doesn't solve the problem, especially if the intersection curve is used to generate other geometry.

Tolerances are needed to control the approximation.[8] Too loose a tolerance can give results that are fast but incorrect. Too tight a tolerance can result in poor performance or failure to converge. Even seemingly simple surface-to-surface intersections become difficult because of choosing tolerances.

Tolerances determine the success of downstream engineering (CFD, finite element) and manufacturing (numerical control programming, quality assurance) analyses. Selecting a tolerance that guarantees a high probability of success requires that the geometry generator understand the kinds of analyses to be employed, the environment of the analyses, and even the specific software to be used a priori.

In summary, digital arithmetic and current math theory are insufficient to perform reliably for complex geometry operations and to interoperate well with downstream analysis software. The geometry must be as watertight as possible for downstream use, and algorithms cannot result in topological inconsistencies (for example, self-intersections and overlaps). The challenge is to find ways to deal with poor results. Perhaps a new math theory that has closed-form solutions for complex surface operations and supports watertight representations for downstream analysis is the way to address this challenge.

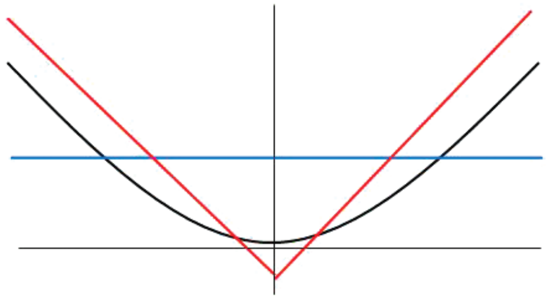
Challenge 3: Design exploration

Automated design exploration through multidisciplinary optimization presents the third challenge. Design optimization requires that geometry remains topologically valid as parameters are perturbed while preserving the designer's intent. There are two aspects to consider: how to parameterize the geometry for downstream analysis and how to structure geometry algorithms to support continuous morphing, a key to any optimization process. The former is primarily an engineering function, which we do not discuss here.

Morphing is a requirement that CAD systems do not currently support. Morphing algorithms today allow the hole in the upper block to flip into the lower block when the edges of the two blocks align. This is fine geometrically. However, this is a disaster for optimization, because the geometry does not morph continuously with the parameters.[9]

The challenge is to design and build geometry systems that ensure the continuity of morphing operations. Morphing continuity differs from geometric continuity. Geometry often has discontinuities (for example, tangents) that must be preserved during morphing. Morphing continuity means that the geometry doesn't change suddenly as parameters change.

Parameter values must be simultaneously set to reasonable values to ensure valid geometry for analysis and optimization. Automating morphing is a challenge because CAD systems have evolved as interactive systems that let users fix poor results. Design optimization needs a geometry system that automatically varies parameters without user guidance and yet maintains design integrity and intent.



4 Blue curve morphs discontinuously into red curve. (Courtesy Boeing)

Multidisciplinary design causes us to rethink the geometric design process as well as the algorithms. For example, many CAD systems use a piecewise quadratic or cubic algorithm for defining a curve through a sequence of points. These algorithms will not reproduce an embedded straight line exactly. Preserving embedded line segments forces the curve fit algorithm to be modified whenever three successive points lie on or are near (determined by some system tolerance) a straight line.

Figure 4 contains a second example of geometry that seems reasonable but causes problems during morphing. Consider the surface $H(x, y, z) = y - ax^2 = 0$ so its intersection with the $z = 0$ plane is described by the function $f(x) = ax^2$. Parameter a is a shape parameter that varies according to an optimization process. Suppose the algorithm for approximating intersection curves uses piecewise straight lines and the fewest join points possible to achieve a certain tolerance. Now suppose the tolerance is 0.5. For values of a approximately equal to 1, the approximation will alternate between a function with constant value 0.5—that is, a spline with no knots—and a piecewise linear function—that is, a spline with one interior knot. In other words, varying the parameter a from slightly less than 1 to slightly greater than 1 causes the model to change discontinuously. As a passes through the value 1, the intersection curve not only changes shape (the blue line morphs to the red line in Figure 4), but its properties (for example, arc length) change discontinuously. This algorithm design produces good static approximations but fails during morphing.

Summary

Addressing the geometry challenges outlined here will not be just a simple task of going through and improving or deleting offending algorithms. It also requires a fundamental rethinking of how geometry design systems should work.

Interactive techniques

Significant improvements in interaction are not going to be achieved by making more efficient menus or a better mouse. Rather, they are going to depend on rethinking the nature of the process. Three specific challenges result:

- changing the order of workflow,
- understanding the concept of *place* in the workplace, and
- intentional design.

History

Interaction technology has evolved slowly over the past 40 years. Input devices have not changed significantly. Physical buttons, such as keyboards, functions keys, voice, fingers, and so on, let people talk to the machine. Graphical pointers come in numerous shapes and sizes and let us move in two or three dimensions, with similar resolution to early devices. The graphical user interface has remained essentially unchanged since 1984.

Displays have gotten a lot smaller and a lot larger. Smaller displays are ubiquitous because of the phenomenal growth in the cell phone industry; larger display penetration is steadily growing. The basic resolution of display devices (number of units per square inch) is about the same as Sutherland's Sketchpad was in the early 1960s.

Improvements have been achieved largely by adding new functionality, enhanced graphic design, and better flow of control. However, systems are not easier to use, and the demands on the user today might be even higher than 20 years ago. The complexity of designs, data, and geometry are growing as fast, or faster, than the power of the tools to handle it.

Nevertheless, some things are changing, and these changes will afford the potential to break out of this situation. For example, more than 10 years ago, Nicholas Negroponte challenged people to imagine what would be possible if bandwidth was essentially free. The only thing that matched how amazing and unlikely that concept was at the time was its prescience.

The parallel challenge today would be, "Imagine what would be possible if screen real-estate were essentially free." Already, paper movie posters are being replaced by $10,000 plasma panels. Imagine the potential impact when the cost of a comparable display drops two orders of magnitude and it is cheaper to mount a 100 dpi display on your wall than it is to mount a conventional whiteboard today.

Large displays will be embedded in the architecture of our workspace. We will stroll through and interact with such spaces with agile small, portable, wireless devices. And many of the changes that are going to affect the future of CAD will emerge from the evolving behavior of both people and devices as they function within them.

Within this context, a divide-and-conquer approach will be used to address the complexity posed by today's CAD systems. While power will come in numbers, most will be relatively inexpensive and target a particular function. The isolated gadgets that first emerge as add-ons to existing systems will morph into the keystones of a new mosaic of integrated technologies that will transform the process.

To realize the potential of this, the following three challenges involve thinking about evolution in a human, not technology, centric way.

Challenge 4: Reversing engineering

Important changes do not result from doing the same things faster or for less money. They come when you flip approaches and methods on their head, that is, we must constantly ask ourselves, Do we do things the way that we do today because it is the right way, or because it's the only way that we knew how when we started?

The inertia of the status quo blinds us to the recognition that major changes, such as those due to Moore's law, open up different and desirable approaches. If we overcome that inertia, we can explore other approaches.

In the field of aerodynamics, for example, it's truly important to get designs that are efficient and safe. Government agencies such as NASA and companies such as Boeing spend millions of dollars on tools, such as wind tunnels and CFD analysis, that help them test such systems. However, these tests typically come late in the design process. Now combine this fact with one of the most basic rules of design: The later in the process that a mistake is detected, the more expensive it is to fix.[10]

Replacing this rigorous testing at the back end is not the answer, but it would be more efficient if technology existed that allowed preliminary tests on their desktop or even on their PDA. Designers could then catch bad designs much earlier in the process. This would also help them discover, explore, refine, and understand the most promising designs as early in the process as possible.

This approach to CFD could be applied to a number of other parts of the CAD workflow. Consider the ability to introduce elements such as stress testing, volume or strength calculations, and so on, earlier in the workflow. This would allow designers to consider the interconnection and interoperability of parts much earlier in the design cycle than is currently the case.

Adding simulation earlier into the design process will enable consideration of behavior, rather than just form, to play into the process. The fundamental change in workflow is the essential interactive technique that will make the geometry design exploration challenge of the previous section a useful and usable capability.

Challenge 5: Everything in its place

There was a time when someone's location in a CAD environment indicated their current job. For example, in an automotive design studio, there is often one location where people deal with interiors, another contains the full-sized clay models, and some other location is set up for exploring colors.

However, the typical modern CAD environment contains a uniform sea of anonymous cubicles or desks. In general, anyone walking through such a space will not likely be able to tell if they are in the accounting or the engineering department. Because people don't move around anyhow and are essentially anchored to their desk, this organization isn't important in some ways. Yet, in the midst of all of this, we hear an ever-louder call for collaboration. We also hear the emergence of ubiquitous computing. How do these two points relate, if at all, in terms of transforming the CAD workplace?

To begin with, ubiquitous computing will break the chain that anchors the engineer to a general-purpose workstation. This change will not just enable but necessitate the designer to move from one specialized area to another in a manner harkening back to the best of past practice.

Not only will the workspace be broken up into specialized areas with specialized tools, but these tools will also consist of a combination of private and public displays and technologies. Some will be mobile and others embedded in the environment. Examples of the latter would include large format displays that function as digital corkboards, surfaces where you can view large parts on a 1:1 scale, and areas where you can generate physical parts using a 3D printer.

The physical mobility of a person and data can greatly impact agility of thought. Mobility brings increased opportunity for collaboration and increased visibility of a particular activity. Rather than design a system that lets us send more email or documentation to the person at the other side of the studio, the studio should be designed so that we have a greater probability and opportunity to bump into and work with that person face to face. If we are going to have work-across sites, then linking designers' efforts must be linked automatically as a consequence of undertaking a particular activity.

Our notion of space is not about making things more abstract or virtual. Rather, it concerns the recognition and exploitation of the attributes and affordances of movement and location in a technology-augmented conventional architectural space. We discuss the impact of collaboration among the geographically distributed in challenge 10.

The challenge to future systems is as much about human–human interaction as it is about new forms of interaction between human and machine.

Challenge 6: What we do

CAD companies might view themselves as primarily purveyors of tools for the creation of high-quality 3D models. Consequently, if they want to grow their business, they might conclude that they should make a better, more usable modeler.

However, this stream of thought might be as wrong as it is reasonable. Consider the following questions:

- Is there a shortage of trained people to fill the existing demand for creating 3D models?
- Are there things that need to be modeled that cannot be built by existing users with their current skills and tools?
- Is there a huge untapped market for 3D models waiting for an easy-to-use modeling package that takes little training to use?

In general, the basic answer to all of these questions is "No." It's not modeling—at least in the sense that it exists in today's CAD packages—that lies behind any of the fundamental challenges outlined in this article.

In fact, it might well be that all of these questions are poorly posed, since they all assume that the intent of the user is modeling. It is not. Rather, it's getting a product or a part made. Modeling is just one way of doing so, and in many cases, not always the best way.

We are then challenged to answer this better question: What besides modeling from scratch might enable us to achieve our product design? A good answer to that question should lead to a more innovative and effective solution than today's design-from-scratch modeling approach.

Our favorite recourse in such cases is to look to the past. In this case, the clue lies in Figure 2, the NACA catalog of airfoil sections. As we discussed previously, such catalogs let designers build an aircraft by selecting components with known properties rather than working from scratch.

We suggest moving back to this approach—but with appropriate modifications. As product complexity increases, the CAD process will subdivide into those specialists who design and build components (often out of subcomponents) and those who make larger assemblies out of them. However, the components in future CAD systems will not be fixed objects made up of what computer scientists might call declarative data. Rather, they will include a strong procedural component.

The basic problem is that objects, even those as simple as an airfoil, don't easily scale. If you make exactly the same form, but larger, the plane might not fly. A simple example is the hummingbird. It can definitely fly. However, if you scaled it up in size by a factor of 10, it could not.

In this object-oriented process, designers generally do more than just select components and plug them into a larger assembly. Rather, they take components and transform them into what is needed. The components act as a starting point, something designers work from, rather than start from scratch.

Given that physical products don't scale, one key aspect of component design is the component's embedded capacity to be transformed along meaningful and desired dimensions, while maintaining specific tolerance, performance characteristics, manufacturability, maintainability, and so on.

Embedded in the component catalog, therefore, are just not the parts, but the knowledge of how parts can be transformed while maintaining essential properties. Consequently, the tools of the CAD engineer not only help the designer find the appropriate components but also support transforming them in such a way as to maintain the desired properties.

Finding, cloning, modifying, morphing, and adapting will largely replace constructing from scratch. In so doing, the assumption is that methods such as aerodynamic testing, stress analysis, and others, can occur in advance for each component and carry over into an assembly. This approach transforms the current practices of who does what, where, when, why, and how.

Summary

Emerging user-interface technology will allow us to transform how we work. Furthermore, the new technology can help us implement the transformation without any major discontinuity with respect to current practice.

Our sense and analysis of interaction needs to switch from how we interact with a specific computer or package to how we interact as people, how we interact with change, and how we interact with our materials—at what level, in what way, and to what objective.

Scale

The challenge of scale is one that scientists and engineers continue to encounter as they push the limits of macro- and nanotechnology. Moore's law governs the expansion of computing hardware's limits. Advanced hardware lets us produce larger quantities of data. Software advances tend to lag behind more powerful hardware, yet we seem to be able to consume computing hardware resources at a rate that always leaves CAD/CAM users begging for larger storage capacity, greater network bandwidth, and better performance.

Managing scale results in a delicate balancing act of knowing when "good enough" occurs. While there are numerous dimensions of scale in CAD/CAM/CAE, the primary areas of challenge today include

- Sheer data quantity that can exceed a project team's software, hardware, and cognitive capacity.
- Making critical data comprehensible to other users.
- Keeping data meaningful as product life spans become longer.
- Collaboration with a geographically distributed workforce.

The challenges of scale in this section derive from experience at Boeing, a large and varied aerospace manufacturer. While Boeing does not represent the norm, similar problems exist in automobile production, shipbuilding, and other manufacturing companies. More importantly, solving a Boeing-sized problem has often resulted in breakthroughs that have profoundly affected smaller efforts.

History

Scale has grown as the overall process of designing, building, and maintaining complex products has changed.

Humans have designed and built extraordinarily complex artifacts throughout history. The seven wonders of the ancient world are clear examples: Roman aqueducts still deliver water service to Italian cities, and the Great Wall of China still stands, acting as a tourist destination and is clearly visible from the ground and Earth orbit.

Documentation on the ancient design process is sketchy, but became more formal during the Renaissance. Leonardo da Vinci's sketches of various mechanical possibilities captured a more formal depiction and allowed designers to analyze possible configuration strengths and weaknesses before construction started. These practices continued to evolve into highly accurate renderings documented as engineering drawings in the first half of the 20th century. The evolution of computational geometry described earlier has taken us to the point we are now in the 21st century.

The challenges we discuss in the rest of this section are the direct result of change in the fundamental design-analyze-build-maintain process. Until the latter part of the 20th century, highly complex projects featured integrated design-build teams. Slave labor was the dominate force in the earliest build teams, and onsite designers oversaw every detail of construction. Large efforts took a long time and involved thousands of workers. There has been a desire to decrease construction time and the number of people ever since.

As increased specialization in individual aspects became more prevalent, principal design and assembly continued to exist within close geographic proximity. Consider the evolution of design-build at Boeing. Boeing's first airplanes were designed and built in the Red Barn (see Figure 5). As build problems occurred, engineers could walk to the factory to make on-site corrections.

As automation and the complexity of the product increased, later airplane generations continued to be designed and built in the relatively close proximity of the Puget Sound region in Washington. Boeing located the engineers designing the 737 and 757 within minutes of the Renton assembly plant; Everett housed the 747, 767, and 777. These engineers were responsible for the design of all of the major sections of all airplanes.

With the 787, its next-generation airplane, Boeing is distributing the design, manufacture, and assembly of major subsections across the world. Each major supplier will deliver preassembled sections of the airplane to final assembly, rather than delivering smaller components as occurred with previous models. This approach will dramatically reduce final assembly times. Computing and communications systems are mandatory for this to occur, and huge amounts of data must be integrated and managed. Because the design is completely digital and 3D, a larger variety of people will look at images of the data during the 787's life cycle. Time and distance complete the notion of scale: mechanical products built today have longer and longer life times, and geographic distance separates people.

Challenge 7: Understanding vast quantities of data

Design, engineering, manufacturing, and maintenance processes have routinely existed in virtual isolation from one another. Part of the scope challenge relates to data, especially as products become complex. Another part relates to people because the graphics vocabulary used in the design process is dramatically different from the one used when the product is being manufactured and assembled.

Early efforts in product design used the waterfall model. In this apprach, designers handed drawings to engineers for analysis leading to design improvements. The drawings then made their way to manufacturing planners, people making and assembling parts, and others responsible for production maintenance. The late 1980s saw a change in the designer-engineering analysis interface because of the advent of 3D modeling systems with enough capacity to generate and manage an entire commercial airplane. Downstream users continued with master definitions represented as 2D drawings.

Initial efforts to deal with issues like manufacturability and maintainability focused on integrated design-build teams. The team members used different applications and kept data in segregated areas with different configuration management schemes.

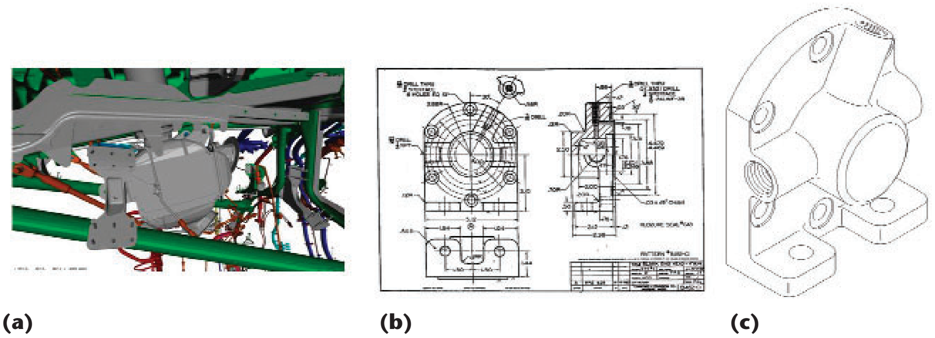
CAD/CAM software vendors are starting to address this problem with product lifecycle management (PLM) systems that extend previous product data management (PDM) systems. PLM systems let companies store customer requirements, design geometry, engineering analysis results, manufacturing plans, factory process designs, maintenance designs, and so on, in a single repository that infuses configuration management throughout.

5 **Boeing Red Barn. (Courtesy Boeing)**

PLM systems are difficult to sell. The essential benefit (managing data across key aspects of a manufacturing company) generates more cost in each individual area but decreases overall product costs through forced pre-facto integration and better configuration management and visibility. PLM systems compete with other large corporate systems for managing personnel, enterprise resource planning (ERP), supply chain management (SCM), and so on. Personnel management, PLM, ERP, and SCM systems all replicate significant portions of the same data. Finally, getting new technology sold, especially when the technology works behind the scenes, is always a difficult problem.

The real problems occur after product integration becomes institutionalized. Integrated products must be examined from a number of different views. Each group has a different purpose and wants to believe that their view is the master. For example, a commercial airplane has the following views:

- *As-designed view* (engineering bill of materials). The relationships among individual components reflect a logical organization of the data (primarily geometry) as modified to satisfy engineering improvements for aerodynamics, structural strength, weight, and so on.
- *As-ordered view* (manufacturing bill-of-materials). The relationships among individual components reflect the parts that must be ordered and the specific attributes (including geometric definition) needed to successfully acquire the parts. Concepts like alternate suppliers are important in this view.
- *As-delivered view.* This view contains information about the physical configuration turned over to a customer after the assembly process is complete. Some components might have been replaced (for example, a supplier changed) or modified during assembly.



6 Communicating data: (a) solid shaded image, (b) engineering drawing, and (c) stylized maintenance image. (Courtesy Boeing)

- *As-owned view.* The actual components in a final product change over time. Components get routinely replaced or repaired as maintenance occurs.

Each of these views is essential during a product's lifecycle and each wants to be master. Providing each view without negatively impacting the integrity of the underlying data poses a difficult challenge.

In terms of sheer storage space, the amount of data needed to faithfully represent a product and to show how decisions were reached in its design is expanding by decimal orders of magnitude. The data ranges from marketing intelligence about customers to versions of complex geometry to multiple data sets from engineering analysis runs to manufacturing planning scripts to sensors monitoring manufacturing processes, physical testing, and real-time product health.

In addition to managing multiple configurations and views, people face a challenge when they have to interpret this vast quantity of information. For example, technology is on the horizon that will allow real-time display[11,12] of an entire commercial airplane model. Humans have a finite ability to comprehend complexity, and understanding how to best display the data in a meaningful way requires extensive work.

Challenge 8: Appropriate designs for other uses and users

This challenge focuses on the large number of people who are removed from the data creation and analysis process and rely on the digital product definition for their own job tasks. These people participate in sales, fabrication, assembly, certification, maintenance, and product operation. In the case of a commercial airplane, the user community extends far beyond company boundaries. For example, airline personnel buy Boeing products, perform maintenance, order spares, train pilots and attendants, and so on. Partners design and analyze significant airplane components. Suppliers formulate proposals to fabricate parts and assemble components. Government agencies certify that the final product performs according to published regulations and guidelines.

Many of these groups have their own graphic vocabulary because different information must be communicated to different audiences. Consider the following examples.

Engineers and designers commonly use images like Figure 6a. The colors are meaningful in terms of the engineering and design functions of individual components, but not for manufacturing or maintenance. This is a significant change from practice prior to 1990, where the master communication mechanism was the engineering drawing (see Figure 6b). The authority image for certification, manufacturing, and maintenance is still the engineering drawing. Communicating via the engineering drawing to a maintenance engineer causes a radical transformation, as Figure 6c shows.

Figures 6b and 6c are clear illustrations of this challenge. The two represent exactly the same 3D geometry. However, the graphic styles are dramatically different. Tools are commonly available to work with the base 3D model—for example, make everything in a scene translucent but objects of interest; show and hide; rotate, scale, and translate; and measure. We can generate the basic 3D hidden line image for Figure 6c, do some early guesses to explode parts, and so on.

These tools fundamentally change the basic style in which the graphic image is rendered, and some might even produce engineering drawings. None contains an ability to adapt an image for specific uses or users. Therefore, the challenge lies in knowing how to draw the image to communicate to a specific user community because the communication techniques and standards vary significantly from community to community and from company to company.

As a result, companies spend significant amounts of time and labor retouching or redrawing images. Automation also offers the possibility of providing new types of images that might be more task appropriate.

Generating the initial image itself presents a significant challenge. The taxonomy of effective graphic images (Figure 6 represents a small subset) is much larger. Massironi[13] has developed an excellent taxonomy that delineates the types of images and the fundamental graphic techniques people have used to communicate visually.

Challenge 9: Retrieving data years later

Today's product specifications for tolerance, fit, reliability, and so on, are greatly different than they were 40 years ago. For example, the Boeing 707 successfully introduced commercial aviation to the jet age. Yet the 707's part fit was loose enough that it received the nickname "the flying shim." On the other hand, the first Boeing 777 fit together so precisely (largely due to the use of CAD/CAM techniques from 10 years ago) that the number of discrepancies needing redesign was substantially less than what had appeared to be an extremely optimistic early prediction. Rather than the multiple mock-ups needed for previous models, the 777 manufacturing mock-up flew as part of the flight certification process. Similar stories exist in the automotive and other industries.

One constant remains: the engineering drawing serves as the design, manufacturing, and certification authority. While we realize that the engineering drawing has its limits, it has another important attribute: It can be archived for long periods of time and still be understood. The Mylar film that Boeing uses for its permanent records lasts longer than the 50-year life span of commercial airplanes.

In contrast, consider the challenge if the archives were stored digitally. The media itself is not the problem because it can be routinely copied to new media. The real challenge is changing software versions. Users routinely expect that some percentage of their data will not migrate successfully from one version to the next. Some algorithms will be tweaked, and what worked in the last version doesn't in the next (or vice versa). A substantial version change or choosing a different software supplier means massive amounts of data conversion and rework. The net result is that companies cannot afford to change to a new system version from the same vendor until years after initial release, let alone change to a new vendor entirely. As the amount of configuration-managed data increases, the data migration challenge becomes even more pronounced.

Challenge 10: Limited by the speed of light

Challenge 5 discussed the impact of users being released from the constraints of single user desktop and laptop computers into an environment where workspaces can be shared. This challenge addresses the users working in a geographically distributed environment, where walking across the hall becomes impossible.

A continued debate rages about the internationalization of large and small businesses throughout the world. In spite of specific national interests and boundaries, companies (many of which are small and mid-sized) are acquiring design, manufacturing, assembly, maintenance, and customer support service help throughout the world. The incentives range from specific technical skills to trade offsets to cheaper labor.

Doing effective distance collaboration for 3D design, which lacks the immediacy and extensive cues of a face-to-face session, is a long-term research and cultural challenge. The key productivity aspect for improved design cycle time is collaboration across a number of different stakeholders. There are a wide variety of collaboration models and a reasonable amount of research done in computer-supported collaborative work. Researchers are starting to pay attention to collaborative task analysis.[14] However, research on how to make such efforts more effective on a global scale is in its infancy, and the cognitive effects of distance on a collaborative 3D work environment have not been addressed.

Summary

This final set of challenges indicates that the way groups of people work on design problems and the longevity of the electronic versions of their products is undergoing a dramatic change. The magnitude of the challenges these changes are causing is at least as great as getting people to adopt CAD in the first place because the challenges impact the fundamental way people work.

How to proceed: Spurn the incremental

We conclude by suggesting an approach that can address our 10 challenges. The basic CAD business model is incremental refinement, and it can only take us so far.

Incremental refinement is based on bringing out the $n + 1$st version, where the new version release has improvements. For this model to work, the incremental improvement for release$_n$, ΔI_n, must be greater than some threshold value, *VT*, which represents the minimum improvement that will still motivate a company to purchase the new version.

The cost of achieving that degree of improvement increases with each release and can be approximated by

$$\Delta I_n \geq VT \Rightarrow \$\Delta I_n = \$\Delta I_2^{\,On}$$

In other words, to keep the incremental value of release$_n$ high enough to motivate purchases, the cost of those improvements is on the order of the cost of those of the second release (that is, the first upgrade), raised to the order's *n*th power.

Two factors contribute to this cost picture. First, as systems go through successive releases, they grow in complexity. This negatively affects their malleability and increases the cost of adding value or refactoring the software or basic functionality. It's not just the number of lines of code or additional features that cause this. As a system approaches maturity, the legacy of the initial underlying architecture, technologies, and paradigms creates a straightjacket that severely affects the cost of change.

Next, as products mature, the software more or less works as intended and markets approach saturation. For most users, the current version of the software is good enough. Changes increasingly tend toward tweaks and tuning rather than major improvements. Or they are directed at more specialized functions needed by a smaller segment of the market. Consequently, there is less to motivate most customers to upgrade.

As the product reaches late maturity, the accumulated impact is that development costs increase while the size of the addressable market decreases. Software sales are no longer sufficient to cover the growing development costs. At this point, companies increasingly rely on annual support contracts, a model difficult to sustain. At best, the switch to support revenue delays the collision of technology costs and economic viability. Even when a company totally reimplements its product for a new version, the tasks the user performs are generally the same as in the previous version and often give less functionality than the last version of the old system.

The CAD industry is reaching this state. How do we get to the next level, the one that should address these challenges?

Order-of-magnitude approach

If incremental refinement won't work, perhaps it will come from some new breakthrough. Perhaps some new invention will magically appear and save the day. The National Academy of Science recently released a report that studied the genesis of a large number of technologies, including graphical user interfaces, portable communication, relational databases, and so on. For each, it looked at how long that technology took to get from discovery to a $1 billion industry. The average was about 20 years.[15] This suggests that any technologies that are going to have impact over the next 10 years are already known and have probably been known for at least 10 years.

We believe that the answer is not new technologies but new insights into technologies that are already known, whose potential is perhaps not appreciated, or which have so far not been technically or economically feasible. The engineering of significantly better products should come from fresh insights on what is already known or knowable.

By analogy, think about the marks on the door frame of the kitchen closet where parents record the growth of their children. Without that documentation, especially because you live with them day-to-day, you generally don't notice the changes. Long after it has actually happened, by measuring them again, you notice that they have passed on to the next stage of growth. This also happens in living with technology.

Our observation leads to the order-of-magnitude (OOM) rule, which says: If anything changes by an order of magnitude along any dimension, it is no longer the same thing.

OOM is a way to measure significance of the types of changes needed to meet the technical challenges we've described. Exploiting this rule forces us to notice OOM changes and understand their implications. However, it also lies in teasing out less obvious, but meaningful dimensions, along which to test for such OOM changes.

Because we have all been so intimately involved with these challenges and CAD technology, we have not taken the time to create the measures that allow us to see where profound changes have already occurred and where they need to occur. We believe that the approaches needed to address the 10 challenges are most likely to result from rethinking things from a human-centric rather than a technology-centric perspective.

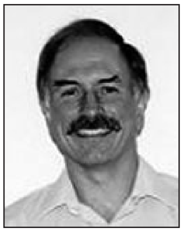
Conclusion

The next wave in CAD will come about largely through the cumulative effect of the introduction of a number of small, lightweight technologies that collectively form a synergistic mosaic, rather than due to the introduction of some monolithic new technology. The value of this approach is that it can be introduced incrementally, without requiring some disruptive discontinuity in skills and production. Thus, change will occur through radical evolution. That is, incremental evolutionary change will happen in a way that leads us to a radically new approach to working.

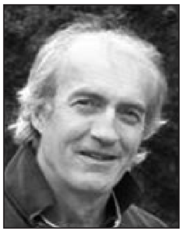
There will be changes in how we do things, and these will significantly affect the nature of CAD systems and how they function. We suggest that using OOM offers a strategy to cause revolution in an evolutionary manner. Ultimate success will happen if and only if all of the technical areas in which OOM changes occur are kept in balance. We conclude that institutionalizing radical evolution is the top CAD challenge we face in 2005, 2015, and beyond. ■

References

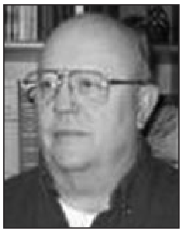
1. I. Sutherland, R. Sproull, and R. Schumacker, "A Characterization of Ten Hidden-surface Algorithms," *ACM Computing Surveys*, vol. 6, no. 1, Mar. 1974, pp. 1-55.
2. R.A. Liming, *Practical Analytic Geometry with Applications to Aircraft*, Macmillan Company, 1944.
3. P. Garrison, "How the P-51 Mustang Was Born," *Air & Space Magazine*, Aug./Sept. 1996.
4. D.R. Ferguson, P.D. Frank, and A.K. Jones, "Surface Shape Control Using Constrained Optimization on the B-spline Representation," *Computer Aided Geometric Design*, vol. 5, no. 2, 1988, pp. 87-103.
5. R.T. Farouki,"Closing the Gap between CAD Model and Downstream Application," *SIAM News*, vol. 32, no. 5, 1999, pp. 303-319.
6. Research Trianagle Inst., *Interoperability Cost Analysis of the US Automotive Supply Chain*, Nat'l Inst. of Science & Technology, March 1999; http://www.mel.nist.gov/msid/sima/interop_costs.pdf.
7. C. M. Hoffmann, *Geometric and Solid Modeling: An Introduction*, Morgan Kaufmann, 1989.
8. D.R. Ferguson et al., "Algorithmic Tolerances and Semantics in Data Exchange," *Proc. 13th ACM Symp. Computational Geometry*, ACM Press, 1997, pp. 403-405.
9. V. Shapiro, *Current Limitations of CAD Systems*; http://sal-cnc.me.wisc.edu/Research/parametric/limits.html.
10. F. Brooks, *The Mythical Man-Month: Essays on Software Engineering*, Addison-Wesley, 1975.
11. W. Baxter et al., "GigaWalk: Interactive Walkthrough of Complex Environments," *Proc. Eurographics Workshop on Rendering*, ACM Int'l Conf. Proceeding Series, 2002, pp. 203-214.
12. I. Wald, C. Benthin, and P. Slusallek, "Distributed Interactive Ray Tracing of Dynamic Scenes," *Proc. IEEE Symp. Parallel and Large-Data Visualization and Graphics* (PVG), IEEE Press, 2003.
13. M. Massironi, *The Psychology of Graphic Images: Seeing, Drawing, Communicating*, Lawrence Erlbaum Assoc., 2002.
14. D. Pinelle, C. Gutwin, and S. Greenberg, "Task Analysis for Groupware Usability Evaluation: Modeling Shared-Workspace Tasks with Mechanics of Collaboration," *ACM Trans, Computer–Human Interaction*, vol. 10, no. 4, 2003, pp. 281-311.
15. Computer Science and Telecommunications Board, "Computer Science and Telecommunications Board of the National Research Council," *Innovation in Information Technology*, National Academies Press, 2003.



***David J. Kasik** is an architect for software engineering, geometry and visualization, and user-interface technology at the Boeing Commercial Airplanes Group Information Systems. His research interests include an innovative combination of basic technologies and increasing awareness of the impact of computing innovation outside the computing community. Kasik has a BA in quantitative studies from the Johns Hopkins University and an MS in computer science from* *the University of Colorado. He is a member of ACM, ACM Siggraph, and ACM SIGCHI and received a Siggraph Special Recognition Award in 1992 for acting as Liaison to the Exhibitor Advisory Committee. Contact him at David.j.kasik@boeing.com.*



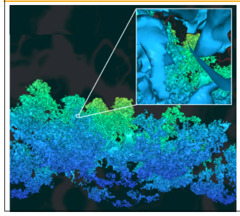
***William Buxton** is principal of his own boutique design and consulting firm, Buxton Design, where his time is split between working for clients, writing, and lecturing. He is also a chief scientist for Bruce Mau Design of Toronto, and is an associate professor in the Department of Computer Science at the University of Toronto. Buxton has a B.Mus. from Queens University and an MSc in computer science from the University of Toronto. He received the 1995 Canadian Human–Computer Communications Society Award, the 2000 New Media Visionary of the Year Award, and is a member of the CHI Academy. Contact him at bill@billbuxton.com.*



***David R. Ferguson** was Boeing's leader in geometry research and development until his retirement. His research interests include the mathematics of curves and surfaces with special emphasis on nonlinear methods for geometric construction. Ferguson has a BS in mathematics from Seattle University, and an MS and PhD in mathematics from the University of Wisconsin. He is an editor of* Computer Aided Geometric Design. *Contact him at drf.assoc@comcast.net.*

For further information on this or any other computing topic, please visit our Digital Library at http://www.computer.org/publications/dlib.

Quick-VDR: Interactive View-Dependent Rendering of Massive Models



Sung-Eui Yoon
Brian Salomon
Russell Gayle
Dinesh Manocha

University of North Carolina at Chapel Hill
http://gamma.cs.unc.edu/QVDR

Goal

- Interactive display of complex and massive models at high image fidelity
  - Models from CAD, scientific simulation, and scanning devices
  - High primitive counts
  - Irregular distribution of geometry

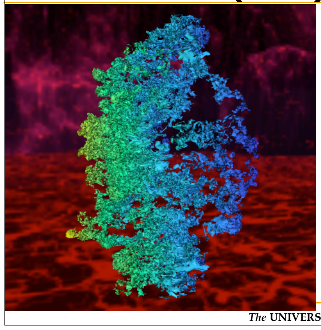
CAD Model – Power Plant

12 million triangles

Irregular distribution

Large occluders

Isosurface from Turbulence Simulation (LLNL)



100 million triangles

High depth complexity

Small occluders

Many holes

Scanned Model – St. Matthew

372 million triangles

Highly tessellated

Problems of current representation

- High cost of refining and rendering the geometry
- High memory footprint
- Complicate integration with other techniques
  - occlusion culling and out-of-core management

Main Contribution

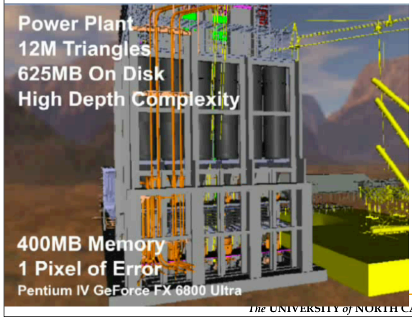
- New view-dependent rendering algorithm

Clustered Hierarchy of Progressive Meshes (CHPM)

Out-of-core construction and rendering

Integrating occlusion culling and out-of-core management

Realtime Captured Video – Power plant



Pentium IV
GeForce FX
6800 Ultra

1 Pixel of Error

Previous Work

- Geometric simplification
  - ◆ View-dependent simplification
- Out-of-core simplification
- Occlusion culling
- Hybrid algorithms

Geometric Simplification

- Static [Cohen et al. 96 and Erikson et al. 01]
- View-dependent [Hoppe 97; Luebke et al. 97; Xia and Varshney 97]
- Surveyed in *Level of Detail* book [Luebke el al. 02]

View-Dependent Simplification

- Progressive mesh [Hop96]
- Merge tree [Xia and Varshney 97]
- View-dependent refinement of progressive meshes [Hoppe 97]
- Octree-based vertex clustering [Luebke and Erikson 97]
- View-dependent tree [El-Sana and Varshney 99]
- Out-of-core approaches [Decoro and Pajarola 02; Lindstrom 03]

Out-of-core Simplification

- Static [Lindstrom and Silva 01; Shaffer and Garland 01; Cignoni et al. 03]
- Dynamic [Hoppe 98; Prince 00; El-Sana and Chiang 00; Lindstrom 03]

Occlusion Culling

- A recent survey is in [Cohen-Or et al. 03]
- Image-based occlusion representation
  - ♦[Greene et al 93, Zhang et al, 97, Klosowski and Silva 01]
- Additional graphics processors
  - ♦[Wonka et al. 01, Govindaraju et al. 03]

Hybrid Approaches

- Combine model simplification and occlusion culling techniques
  - ♦UC Berkeley Walkthrough [Funkhouser 96]
  - ♦UNC MMR system [Aliaga 99]
- Integrate occlusion culling with view dependent rendering
  - ♦[El-Sana et al. 01; Yoon et al. 03]

Outline

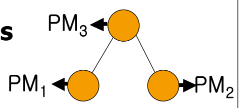
- CHPM representation
- Building a CHPM
- Interactive display
- Implementation and results
- Conclusion and future work

Outline

- CHPM representation
- Building a CHPM
- Interactive display
- Implementation and results
- Conclusion and future work

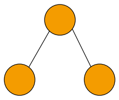
Clustered Hierarchy of Progressive Meshes (CHPM)

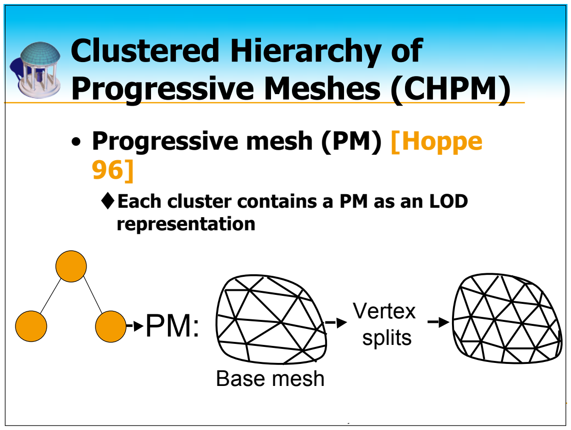
- Novel view-dependent representation
- Cluster hierarchy
- Progressive meshes



Clustered Hierarchy of Progressive Meshes (CHPM)

- Cluster hierarchy
  - ♦Clusters are spatially localized mesh regions
  - ♦Used for visibility computations and out-of-core rendering





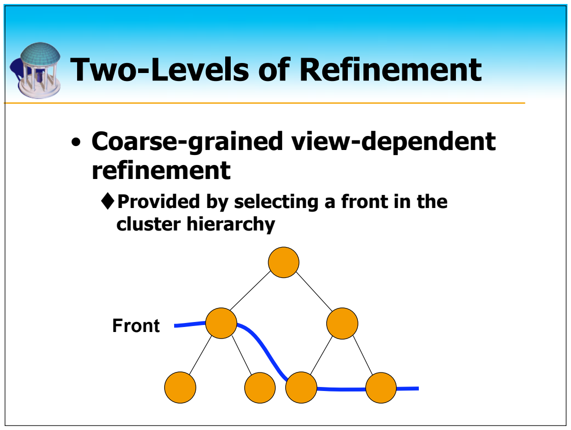
Clustered Hierarchy of Progressive Meshes (CHPM)

- Progressive mesh (PM) [Hoppe 96]
  - ♦Each cluster contains a PM as an LOD representation

PM:

Vertex splits

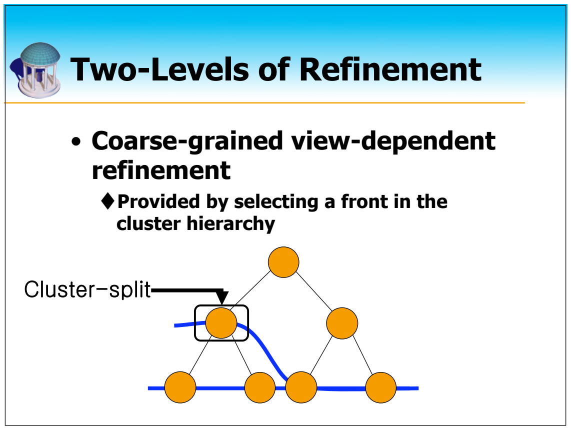
Base mesh



Two-Levels of Refinement

- Coarse-grained view-dependent refinement
  - ♦Provided by selecting a front in the cluster hierarchy

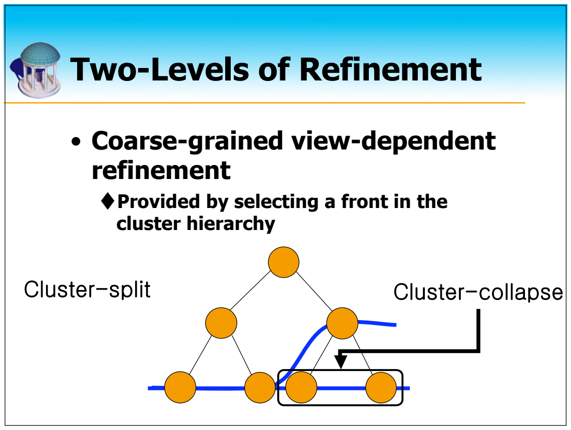
Front



Two-Levels of Refinement

- Coarse-grained view-dependent refinement
  - ♦Provided by selecting a front in the cluster hierarchy

Cluster–split

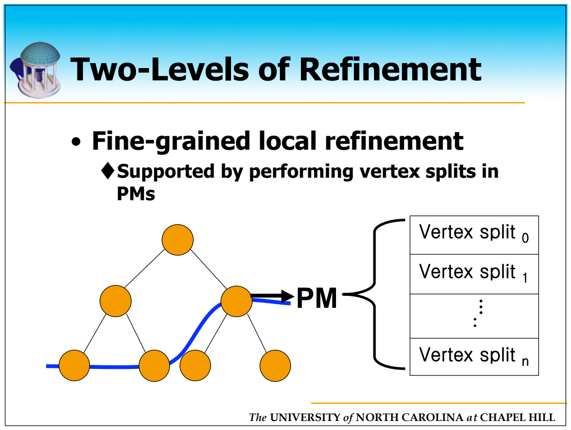


Two-Levels of Refinement

- Coarse-grained view-dependent refinement
  - ♦Provided by selecting a front in the cluster hierarchy

Cluster–split

Cluster–collapse



Two-Levels of Refinement

- Fine-grained local refinement
  - ♦Supported by performing vertex splits in PMs

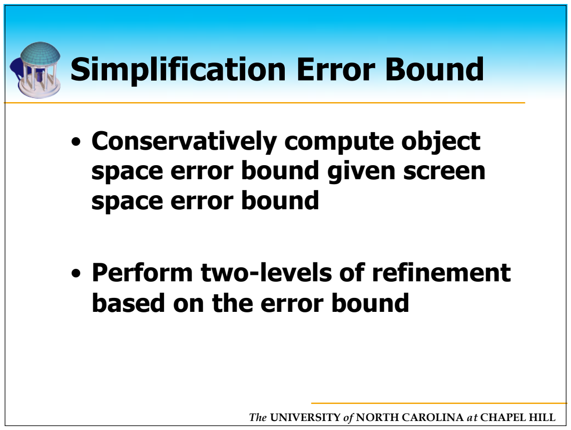
PM

Vertex split $_0$

Vertex split $_1$

⋮

Vertex split $_n$



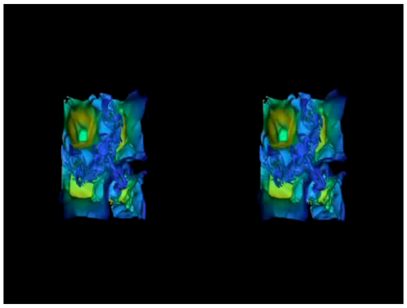
Simplification Error Bound

- Conservatively compute object space error bound given screen space error bound
- Perform two-levels of refinement based on the error bound

Main Properties of CHPM

- Low refinement cost
  - ♦1 or 2 order of magnitude lower than a vertex hierarchy
- Alleviates visual popping artifacts
  - ♦Provides smooth transition between different LODs

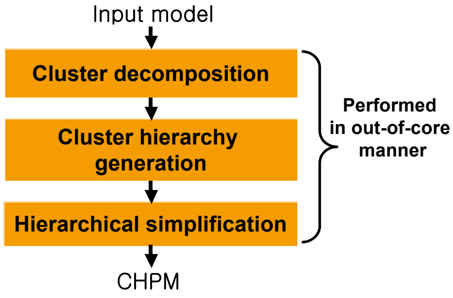
Video – Comparison CHPM with Vertex Hierarchy



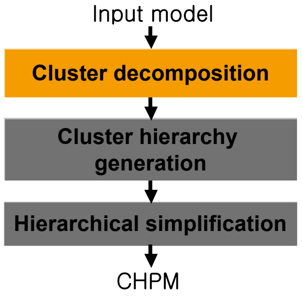
Outline

- CHPM representation
- Building a CHPM
- Interactive display
- Implementation and results
- Conclusion and future work

Overview of Building a CHPM


Input model
Cluster decomposition
Cluster hierarchy generation
Hierarchical simplification
CHPM
Performed in out-of-core manner

Overview of Building a CHPM

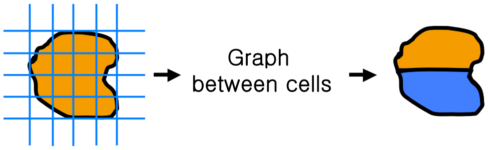


Cluster Decomposition

- Cluster
  - ♦Main unit for view-dependent refinement, occlusion culling, out-of-core management
  - ♦Spatially localized portion of the mesh
  - ♦Equally sized in terms of number of triangles

Cluster Decomposition

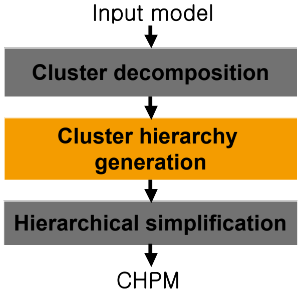
- **Similar to [Ho et al. 01; Isenburg and Gumhold 03]**



An Example of Cluster Decomposition

Each cluster contains 4K triangles

Overview of Building a CHPM



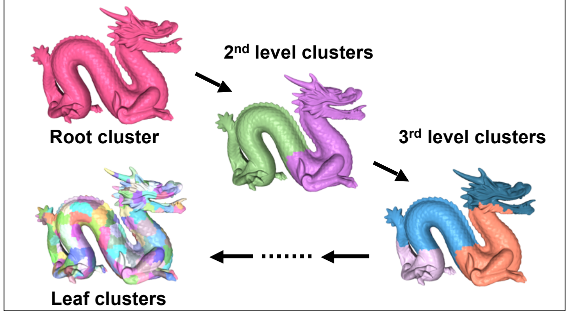
Cluster Hierarchy Generation - Properties

1. **Nearly equal cluster size**
2. **High spatial locality**
3. **Minimum shared vertices**
4. **Balanced cluster hierarchy**

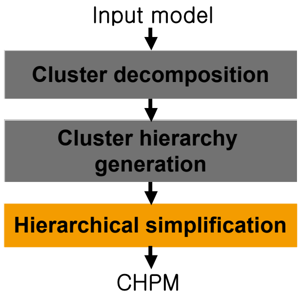
Cluster Hierarchy Generation - Algorithm

- **Node**
  - **Corresponds to a cluster**
  - **Weighted by the number of vertices**
- **Edge**
  - **Made if clusters share vertices**
  - **Weighted by the number of shared vertices**

Cluster Hierarchy Generation - Algorithm

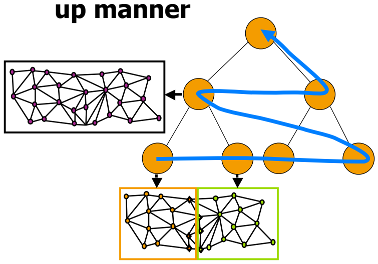


Overview of Building a CHPM



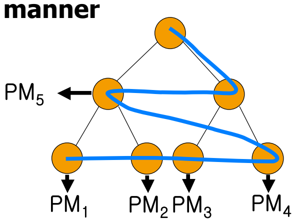
Out-of-core Hierarchical Simplification

- Simplifies clusters in a bottom-up manner

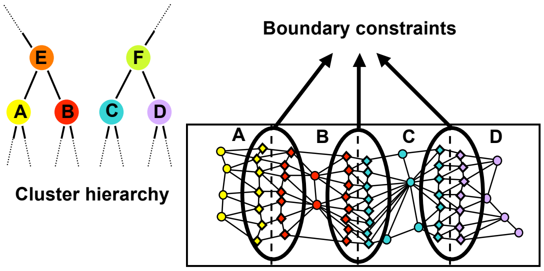


Out-of-core Hierarchical Simplification

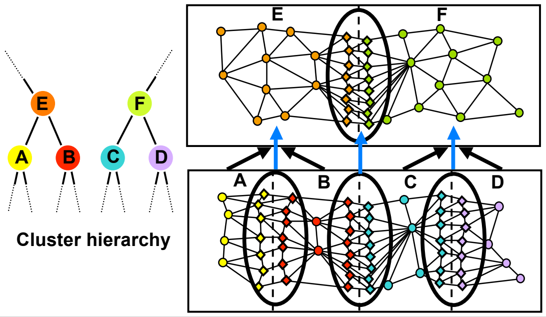
- Simplifies clusters in a bottom-up manner



Boundary Simplification



Boundary Simplification

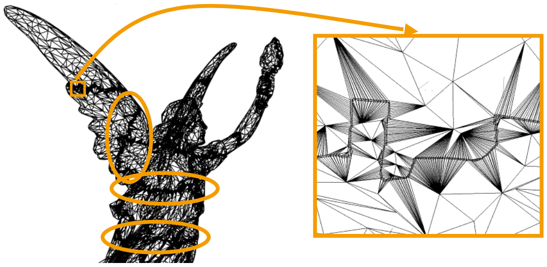


Boundary Constraints

- Common problem in many hierarchical simplification
  - ♦ [Hoppe 98; Prince 00; Govindaraju et al. 03]
  - ♦ Degrades the quality of simplification
  - ♦ Decrease rendering performance at runtime
  - ♦ Aggravated as a depth of hierarchy increases

Boundary Constraints

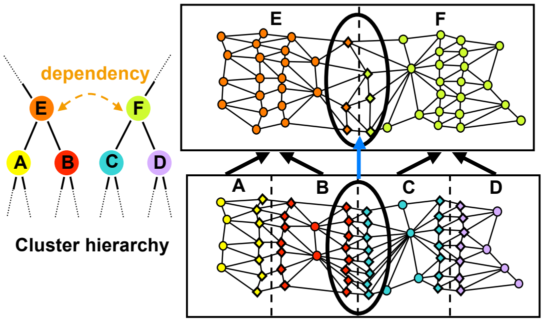
Boundary Constraints



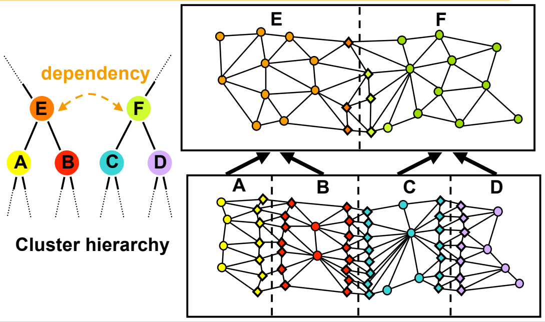
Cluster Dependencies

- **Replaces preprocessing constraints by runtime dependencies**

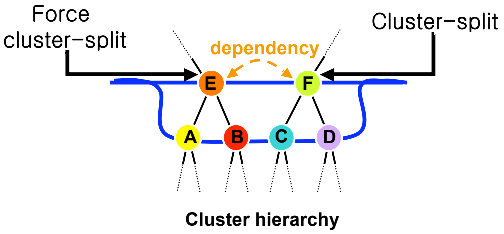
Cluster Dependencies



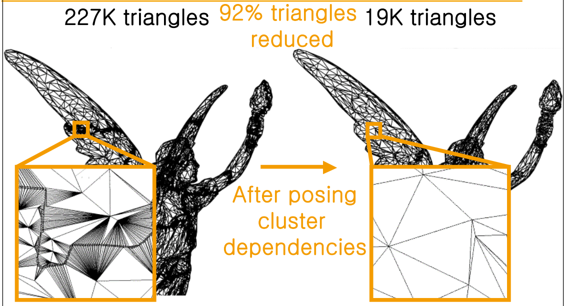
Cluster Dependencies



Cluster Dependencies at Runtime



Cluster Dependencies

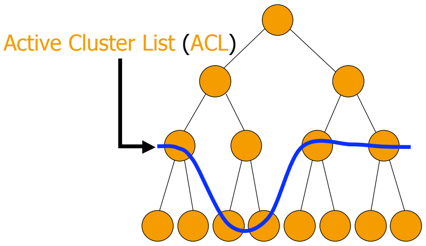


Outline

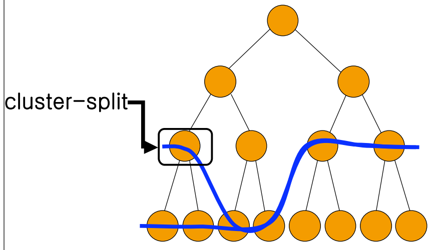
- CHPM representation
- Building a CHPM
- Interactive display
- Implementation and results
- Conclusion and future work

View-Dependent Refinement

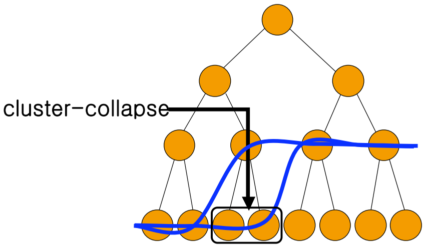
Traverse and update previous active cluster list



View-Dependent Refinement

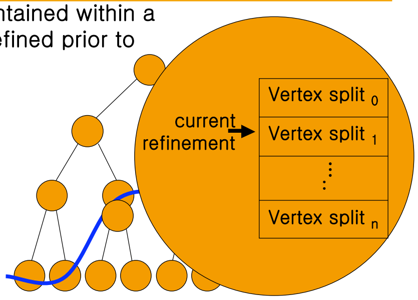


View-Dependent Refinement



View-Dependent Refinement

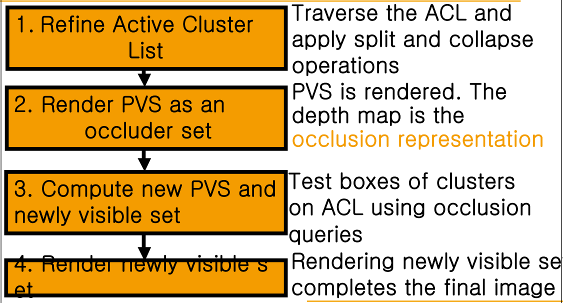
The PM contained within a cluster is refined prior to rendering



Occlusion Culling

- **Exploit temporal coherence**
  - ◆ **Use visible clusters from the previous frame as potentially visible set (PVS) in the current frame**
- **Cluster bounding boxes tested using occlusion queries**

Rendering Algorithm



1. Refine Active Cluster List — Traverse the ACL and apply split and collapse operations

2. Render PVS as an occluder set — PVS is rendered. The depth map is the occlusion representation

3. Compute new PVS and newly visible set — Test boxes of clusters on ACL using occlusion queries

4. Render newly visible s et — Rendering newly visible se completes the final image

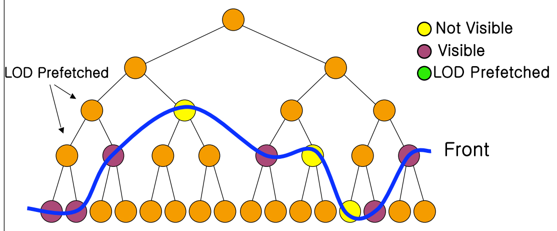
Out-of-core Rendering

- **Entire hierarchy stored in memory**
  - ◆ **Takes 5MB in St. Matthew model**
- **PMs stored on disk**

Out-of-core Rendering

- **Fetch PMs as necessary**
  - ◆ **Buffer for LOD Changes**
  - ◆ **Latency for visibility events**
- **PMs replaced using LRU policy**

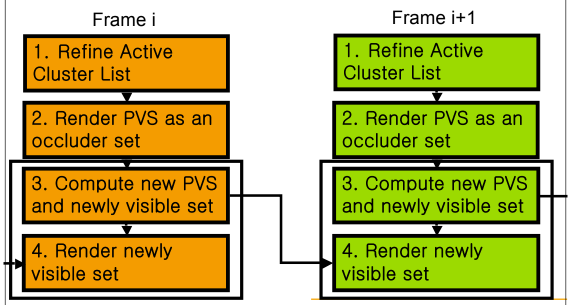
LOD Prefetching



Visibility Prefetching

- **Occlusion events are difficult to predict**
- **We use a frame of latency to reduce stalling**
- **The rendering algorithm is reordered using a pair of offscreen buffers**

Latency


Frame i
1. Refine Active Cluster List
2. Render PVS as an occluder set
3. Compute new PVS and newly visible set
4. Render newly visible set
Frame i+1
1. Refine Active Cluster List
2. Render PVS as an occluder set
3. Compute new PVS and newly visible set
4. Render newly visible set

Outline

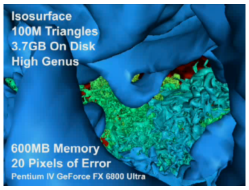
- CHPM representation
- Building a CHPM
- Interactive display
- Implementation and results
- Conclusion and future work

Implementation

- Use GL_ARB_vertex_buffer_object to manage PM data in GPU memory
  - 5% of PMs change refinement level
- Implemented on Windows XP
  - Uses virtual memory through memory mapped files [Lindstrom and Pascucci 02]

Live Demo – Isosurface

- Alienware laptop
  - 3.2GHz Pentium IV PC with 2GB main memory
  - Quadro FX Go5700 with 128MB GPU memory



Live Demo – St. Matthew

- Alienware laptop
  - 3.2GHz Pentium IV PC with 2GB main memory
  - Quadro FX Go5700 with 128MB GPU memory

Saint Matthew Scan
372M Triangles
13GB on Disk
0.29mm Scan Resolution
600MB Main Memory
1 Pixel of Error
Pentium IV GeForce FX 5950 Ultra

Test Configuration

Dell Precision Workstations

- Dual 3.2 GHz Pentium IV PC with 1 GB main memory
- GeForce FX 6800 Ultra with 128MB of video memory

Processing Time and Storage Requirement

Processing 30M triangles per an hour

Model	Triangles (M)	Memory footprint used (MB)	Processing Time (hour)
Power plant	**12.8**	**32**	**0.5**
Isosurface	**100**	**256**	**2.8**
St. Matthew	**372**	**512**	**10.2**

Processing Time and Storage Requirement

- **CHPM requires 88MB per million vertices**
  - ♦**Low compared to 224MB (Hoppe's VDPM) and 108MB (XFastMesh)**
  - ♦**Compression on PMs can further improve the memory overhead and storage overhead**

Runtime Performance

Model	Pixels of error	Frame rate	Memory footprint	Avg # Tris
Power plant	1	28	400MB	542K
Isosurface	20	27	600MB	819K
St. Matthew	1	29	600MB	850K

Image resolution is 512 × 512

Timing Breakdown

Model	Rendering	Occlusion Culling	Refinement
Power plant	86%	13%	1%
Isosurface	94%	5%	1%
St. Matthew	94%	4%	2%

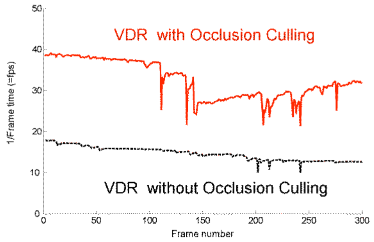
Refinement Comparison

1M Triangle Isosurface

Method	Num Dependency Checks	Total Refinement Time
Vertex Hierarchy	4.2M	1,065 msec
CHPM	223	28 msec

38X speedup

Frame Rate - Isosurface



2X speedup

Limitations

- Latency
  - Reduces stalls but no guarantee
  - Excludes latency-sensitive applications
- Requires high temporal coherence

Advantages

- Refinement time
- Performance comparable to static LOD systems
- Combines out-of-core, VDR, and occlusion culling
- General
  - CAD, scanned, isosurface models

Conclusion

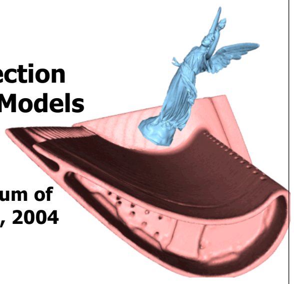
- Quick-VDR: view-dependent rendering of massive models
  - Generates crack-free and drastic high quality simplification by cluster dependencies
  - Provides two levels of refinement using CHPM
  - Applies to a wide variety of massive models

Ongoing and Future Work

- Compression
- Consider dynamic effects
  - Shadows
  - Lighting
- Occlusion event prediction

Application to Collision Detection

- Fast Collision Detection Between Massive Models Using Dynamic Simplification
  - Presented in Symposium of Geometric Processing, 2004



Acknowledgements

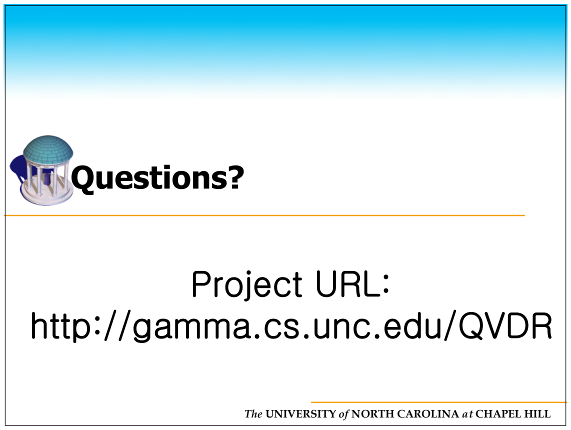
- Anonymous donor for power plant model
- LLNL ASCI VIEWS for isosurface model
- Digital Michelangelo Project at Stanford University for the St. Matthew model

Acknowledgements

- DARPA
- Army Research Office
- Intel Corporation
- Office of Naval Research
- National Science Foundation

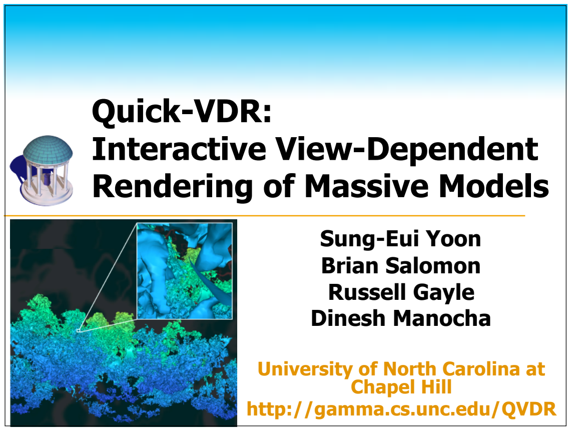
Acknowledgements

- Naga Govindaraju
- Martin Isenburg
- Stefan Gumhold
- Mark Duchaineau
- Peter Lindstrom
- Members of Walkthru group



Questions?

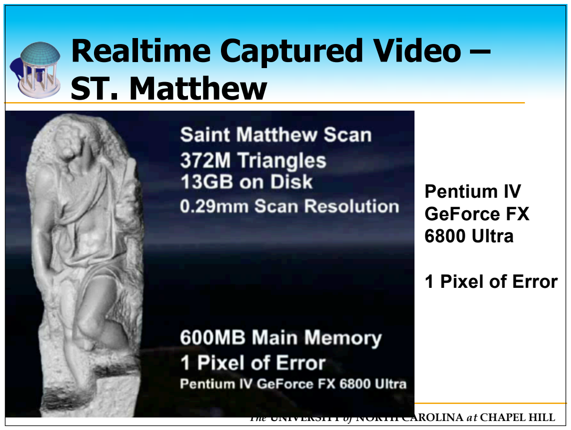
Project URL:
http://gamma.cs.unc.edu/QVDR



Quick-VDR: Interactive View-Dependent Rendering of Massive Models

Sung-Eui Yoon
Brian Salomon
Russell Gayle
Dinesh Manocha

University of North Carolina at Chapel Hill
http://gamma.cs.unc.edu/QVDR



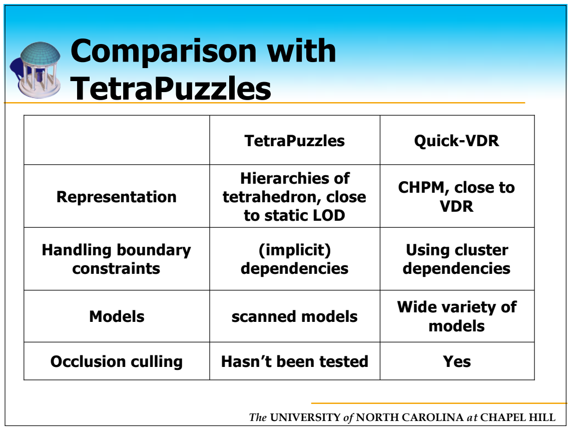
Realtime Captured Video – ST. Matthew

Saint Matthew Scan
372M Triangles
13GB on Disk
0.29mm Scan Resolution

600MB Main Memory
1 Pixel of Error
Pentium IV GeForce FX 6800 Ultra

Pentium IV
GeForce FX
6800 Ultra

1 Pixel of Error

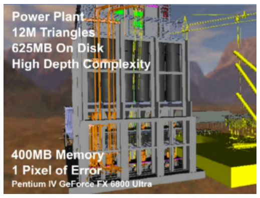


Comparison with TetraPuzzles

	TetraPuzzles	Quick-VDR
Representation	Hierarchies of tetrahedron, close to static LOD	CHPM, close to VDR
Handling boundary constraints	(implicit) dependencies	Using cluster dependencies
Models	scanned models	Wide variety of models
Occlusion culling	Hasn't been tested	Yes

Live Demo – Power plant

- **Alienware laptop**
  - ◆ **3.2GHz Pentium IV PC with 2GB main memory**
  - ◆ **Quadro FX Go5700 with 128MB GPU memory**



Quick-VDR: Interactive View-Dependent Rendering of Massive Models

Sung-Eui Yoon Brian Salomon Russell Gayle Dinesh Manocha
University of North Carolina at Chapel Hill
{sungeui,salomon,rgayle,dm}@cs.unc.edu
http://gamma.cs.unc.edu/QVDR

Abstract

We present a novel approach for interactive view-dependent rendering of massive models. Our algorithm combines view-dependent simplification, occlusion culling, and out-of-core rendering. We represent the model as a clustered hierarchy of progressive meshes (CHPM). We use the cluster hierarchy for coarse-grained selective refinement and progressive meshes for fine-grained local refinement. We present an out-of-core algorithm for computation of a CHPM that includes cluster decomposition, hierarchy generation, and simplification. We make use of novel cluster dependencies in the preprocess to generate crack-free, drastic simplifications at runtime. The clusters are used for occlusion culling and out-of-core rendering. We add a frame of latency to the rendering pipeline to fetch newly visible clusters from the disk and to avoid stalls. The CHPM reduces the refinement cost for view-dependent rendering by more than an order of magnitude as compared to a vertex hierarchy. We have implemented our algorithm on a desktop PC. We can render massive CAD, isosurface, and scanned models, consisting of tens or a few hundreds of millions of triangles at $10-35$ frames per second with little loss in image quality.

Keywords: Interactive display, view-dependent rendering, occlusion culling, external-memory algorithm, levels-of-detail

1 Introduction

Recent advances in acquisition, modeling, and simulation technologies have resulted in large databases of complex geometric models. These gigabyte-sized datasets consist of tens or hundreds of millions of polygons. The enormous size of these datasets poses a number of challenges in terms of interactive display and manipulation on current graphics systems.

View-dependent simplification and rendering have been actively researched for interactive display of large datasets [Hoppe 1997; Luebke and Erikson 1997; Xia et al. 1997]. These algorithms have many appealing properties because they compute different levels-of-detail (LODs) over different regions of the model. The selection of appropriate LODs is based on view-position, local illumination and other properties such as visibility and silhouettes. Most view-dependent algorithms precompute a vertex hierarchy of the model and perform incremental computations between successive frames. This reduces the "popping" artifacts that can occur while switching between different LODs. The algorithms generally maintain a cut, or *active vertex front*, across the hierarchy and use it for mesh refinement. The front is traversed each frame and is updated based on the change in view parameters. In order to preserve the local topology, dependencies are introduced between simplification operations.

Current representations and refinement algorithms for view-dependent rendering do not scale well to large models composed of tens or hundreds of millions of triangles. The refinement cost is a function of the front size and may be prohibitively expensive for massive models. Furthermore, resolving dependencies in the vertex hierarchy can be expensive (e.g. hundreds of milliseconds or more per frame).

In addition to reducing the refinement cost, it is necessary to integrate view-dependent simplification algorithms with occlusion culling and out-of-core rendering. Occlusion culling computes a set of potentially visible primitives during each frame and is needed to handle high depth complexity models. Out-of-core rendering techniques operate with a bounded memory footprint and are required to render massive models on commodity graphics systems with limited memory. Algorithms for occlusion culling and out-of-core techniques also perform computations based on the view parameters. However, no known algorithms integrate conservative occlusion culling and out-of-core rendering with vertex hierarchies.

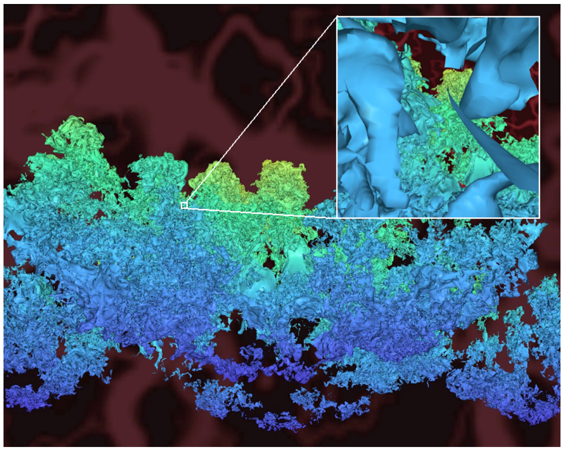


Figure 1: This image shows the application of Quick-VDR to a complex isosurface (100M triangles) generated from a very high resolution 3D simulation of Richtmyer-Meshkov instability and turbulence mixing. The right inset image shows a zoomed view. The isosurface has high depth complexity, holes, and a very high genus. Quick-VDR can render it at $11-21$ frames per second on a PC with NVIDIA GeForce FX5950 card and uses a memory footprint of 600MB.

Main Contributions: We present a new view-dependent rendering algorithm (Quick-VDR) for interactive display of massive models. We use a novel scene representation, a *clustered hierarchy of progressive meshes* (CHPM). The cluster hierarchy is used for coarse-grained view-dependent refinement. The progressive meshes provide fine-grained local refinement to reduce the popping between successive frames without high refinement cost.

Our rendering algorithm uses temporal coherence and occlusion queries for visibility computations at the cluster level. We account for visibility events between successive frames by combining fetching and prefetching techniques for out-of-core rendering. Our rendering algorithm introduces one frame of latency to fetch newly visible clusters from the disk and to avoid stalling the pipeline.

Quick-VDR relies on an out-of-core algorithm to compute a CHPM that performs a hierarchical cluster decomposition and simplification. We introduce the concept of *cluster dependencies* between adjacent clusters to generate crack-free and drastic simplifications of the original model.

We have implemented and tested Quick-VDR on a commodity PC with NVIDIA 5950FX Ultra card. To illustrate the generality of our approach we have highlighted its performance on several models: a complex CAD environment (12M triangles), scanned models (372M triangles), and an isosurface (100M triangles). We can render these models at $10-35$ frames per second using a limited memory footprint of $400-600$MB.

Advantages: Our approach integrates view-dependent simplification, conservative occlusion culling, and out-of-core rendering for high quality interactive display of massive models on current graphics systems. As compared to prior approaches, Quick-VDR offers the following benefits:

1. **Lower refinement cost:** The overhead of view-dependent refinement in the CHPM is one to two orders of magnitude lower

than vertex hierarchies for large models.

2. **Massive models:** We are able to compute drastic simplifications of massive models, using hierarchical simplification with cluster dependencies, necessary for interactive rendering.
3. **Runtime performance:** Quick-VDR renders CHPMs using a bounded memory footprint and exploits the features of current graphics processors to obtain a high frame rate.
4. **Image quality:** We significantly improve the frame rate with little loss in image quality and alleviate popping artifacts between successive frames.
5. **Generality:** Quick-VDR is a general algorithm and applicable to all types of polygonal models, including CAD, scanned, and isosurface.

Organization: The rest of the paper is organized in the following manner. We give a brief overview of related work in Section 2 and describe our scene representation and refinement algorithm in Section 3. Section 4 describes our out-of-core algorithm to generate a CHPM for a large environment. We present the rendering algorithm in Section 5 and highlight its performance in Section 6. We compare our algorithm with other approaches in Section 7 and discuss some of its limitations.

2 Related Work

We give a brief overview of the previous work in view-dependent rendering, out-of-core rendering, occlusion culling, and hybrid approaches to massive model rendering.

2.1 View-Dependent Simplification

View-dependent simplification of complex models has been an active area of research over the last decade. View-dependent rendering originated as an extension of the progressive mesh (PM) [Hoppe 1996]. A PM is a linear sequence of increasingly coarse meshes built from an input mesh by repeatedly applying edge collapse operations. It provides a continuous resolution representation of an input mesh and is useful for efficient storage, rendering, and transmission.

Xia and Varshney [1997] and Hoppe [1997] organized the PM as a vertex hierarchy (or view-dependent progressive mesh (VDPM)) instead of a linear sequence. Luebke and Erikson [1997] developed a similar approach employing octree-based vertex clustering operations and used it for dynamic simplification. El-Sana and Varshney [1999] extended these ideas using a uniform error metric based on cubic interpolants and reduced the cost of runtime tests.

The Multi-Triangulation(MT) is a multiresolution representation that has been used for view-dependent rendering [L. De Floriani 1997]. Pajarola [2001] improved the update rate of runtime mesh selection by exploiting properties of the half-edge mesh representation and applied it to manifold objects. El-Sana and Bachmat [2002] presented a mesh refinement prioritization scheme to improve the runtime performance.

2.2 Out-of-core Computation and Rendering

Many algorithms have been proposed for out-of-core simplification. These include [Lindstrom and Silva 2001; Shaffer and Garland 2001; Cignoni et al. 2003b] for generating static LODs. Hoppe [1998] extended the VDPM framework for terrain rendering by decomposing the terrain data into blocks, generating a block hierarchy and simplifying each block independently. Prince [2000] extended this out-of-core terrain simplification to handle arbitrary polygonal models.

El-Sana and Chang [2000] segment a mesh into sub-meshes such that the boundary faces are preserved while performing edge-collapse operations. DeCoro and Pajarola [2002] present an external data structure for the half-edge hierarchy and an explicit paging system for out-of-core management of view-dependent rendering. Lindstrom [2003] presents an approach for out-of-core simplification and view-dependent visualization. Cignoni et al. [2003a] present an efficient method for out-of-core rendering of terrain data.

2.3 Occlusion Culling

The problem of computing portions of the scene visible from a given viewpoint has been well-studied [Cohen-Or et al. 2001]. Many specialized object-space algorithms have been developed for architectural models or urban environments. For general environments, image-based occlusion representations are widely used and the resulting algorithms use graphics hardware to perform visibility computations [Greene et al. 1993; Zhang et al. 1997; Klosowski and Silva 2001]. In some cases, additional graphics processors have been used for visibility computations [Wonka et al. 2001; Govindaraju et al. 2003]. All of these algorithms load the entire scene graph into main memory.

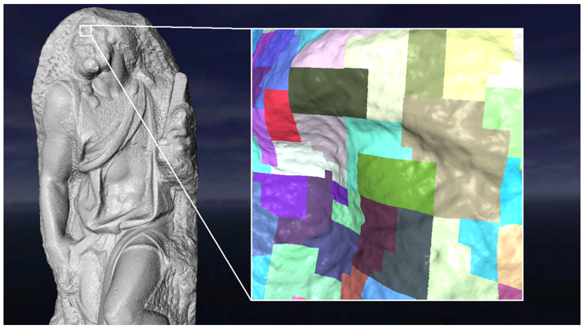


Figure 2: Scan of Michelangelo's St. Matthew. *This $9.6GB$ scanned model consists of $372M$ triangles. The right inset image shows clusters in color from a $64K$ cluster decomposition of the model. Quick-VDR is able to render this model at $13-23$ frames per second using a memory footprint of $600MB$.*

2.4 Hybrid Algorithms for Rendering Acceleration

Many hybrid algorithms have been proposed that combine model simplification with visibility culling or out-of-core data management. The Berkeley Walkthrough system [Funkhouser et al. 1996] combines cells and portals based on visibility computation algorithms with static LODs for architectural models. The MMR system [Aliaga et al. 1999] combines static LODs with occlusion culling and out-of-core computation and is applicable to models that can be partitioned into rectangular cells. The QSplat system [Rusinkiewicz and Levoy 2000] uses a compact bounding volume hierarchy of spheres for view-frustum and backface culling, level-of-detail control and point-based rendering. Erikson et al. [2001] and Samanta et al. [2001] use hierarchies of static LODs (HLODs) and view-frustum culling. Govindaraju et al. [2003] integrate HLODs and conservative occlusion culling for interactive display of large CAD environments. Yoon et al. [2003] presented an in-core algorithm to combine view-dependent simplification with conservative occlusion culling. El-Sana et al. [2001] combined view-dependent rendering with approximate occlusion culling. The *iWalk* system [Correa et al. 2002] partitions the space into cells and performs out-of-core rendering of large architectural and CAD models on commodity hardware using non-conservative occlusion culling.

3 Overview

In this section we introduce some of the terminology and representations used by Quick-VDR. We also give a brief overview of our approach for out-of-core hierarchical simplification and rendering.

3.1 View-Dependent Rendering of Massive Datasets

Most of the prior work on view-dependent simplification and rendering of large datasets uses vertex hierarchies such as VDPM. These approaches augment each edge collapse with *dependency* information related to the local neighborhood at the time of the edge collapse during construction. This information is used to prevent "fold-overs" whereby a face normal is reversed at runtime. However, many issues arise in applying these approaches to massive datasets composed of tens or hundreds of millions of triangles. Traversing and refining an *active vertex front* across a vertex hierarchy composed of tens of millions of polygons can take hundreds of milliseconds per frame. Also, resolving the dependencies can lead to non-localized memory accesses which can be problematic for out-of-core rendering. Moreover, performing occlusion culling and out-of-core rendering using vertex hierarchies can become expensive.

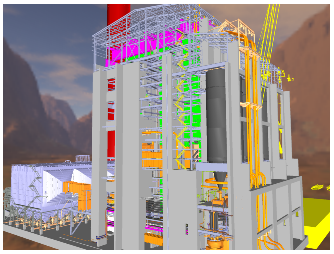


Figure 3: Power Plant. *A rendering of the power plant model using our runtime algorithm. This model consists of over 12M triangles and has high depth complexity. It is rendered at an average of* 33 *FPS using* 400*MB of main memory by our system.*

3.2 Scene Representation

We propose a novel representation, a clustered hierarchy of progressive meshes (CHPM), for view-dependent rendering of massive datasets. The CHPM consists of two parts:

Cluster Hierarchy: We represent the entire dataset as a hierarchy of clusters, which are spatially localized mesh regions. Each cluster consists of a few thousand triangles. The clusters provide the capability to perform coarse-grained view-dependent (or selective) refinement of the model. They are also used for visibility computations and out-of-core rendering.

Progressive Mesh: We precompute a simplification of each cluster and represent a linear sequence of edge collapses as a progressive mesh (PM). The PMs are used for fine-grained local refinement and to compute an error-bounded simplification of each cluster at runtime.

We refine the CHPM at two levels. First we perform a coarse-grained refinement at the cluster level. Next we refine the PMs of the selected clusters. The PM refinement provides smooth LOD transitions.

3.2.1 Cluster Hierarchy

Conceptually, a cluster hierarchy is similar to a vertex hierarchy. However, every node of a cluster hierarchy represents a set of vertices and faces rather than a single vertex. At runtime, we maintain an *active cluster list* (ACL), which is similar to an active front in a vertex hierarchy and perform selective refinement on this list via the following operations:

- **Cluster-split:** A cluster in the active cluster list is replaced by its children.
- **Cluster-collapse:** Sibling clusters are replaced by their parent.

These operations are analogous to the vertex split and collapse in a vertex hierarchy but provide a more coarse-grained approach to selective refinement.

3.2.2 Progressive Meshes and Refinement

A PM is a mesh sequence built from an input mesh by a sequence of edge collapse operations. The inverse operation, a vertex split, restores the original vertices and replaces the removed triangles. We use the notation M_A^0 to represent the most simplified or *base mesh* of cluster A. Moreover, M_A^i is computed by applying a vertex split operation to M_A^{i-1} (as shown in Fig. 4). Each PM is stored as a base mesh and a series of vertex split operations. For each cluster in the ACL we select the position in the edge collapse sequence that meets the allowed error for the cluster with the least number of faces. We take advantage of temporal coherence by starting with the position from the previous frame. In practice, refining a PM is a very fast operation and requires no dependency checks.

The PMs allow us to perform smooth LOD transitions at the level of a single cluster. In order to perform globally smooth LOD transitions we require that the changes to the ACL between successive frames are also smooth. If cluster C is the parent of clusters A and B, we set the highest resolution mesh approximation of cluster C's PM to be the union of the base meshes of cluster A and B's PMs. That is, $M_C^k = M_A^0 \bigcup M_B^0$ (see Fig. 4). Therefore, the cluster-collapse and cluster-split operations introduce no popping artifacts.

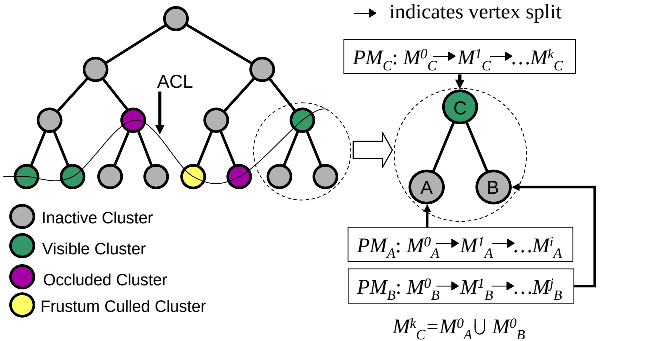

→ indicates vertex split
PM_C: $M^0_C \rightarrow M^1_C \rightarrow \ldots M^k_C$
ACL
C
A
B
Inactive Cluster
Visible Cluster
Occluded Cluster
Frustum Culled Cluster
PM_A: $M^0_A \rightarrow M^1_A \rightarrow \ldots M^i_A$
PM_B: $M^0_B \rightarrow M^1_B \rightarrow \ldots M^j_B$
$M^k_C = M^0_A \cup M^0_B$

Figure 4: CHPM: Clustered Hierarchy of Progressive Meshes. *At runtime the active cluster list (ACL) represents a front in the cluster hierarchy containing the clusters of the current mesh (left). Clusters on the ACL are classified as visible, frustum culled, or occlusion culled. The PMs (right) of visible clusters are refined to meet the screen space error bound by selecting a mesh from the PM mesh sequence. When the ACL changes, smooth LOD transitions occur because the most refined mesh of each PM is equal to the union of the base meshes of its children.*

3.3 Simplification Error Bounds

A key issue is computation of errors associated with the LODs generated at runtime. Each cluster contains progressive mesh that can be refined within a range of object space error values. We refer to this range as the *error-range* of a cluster and is expressed as a pair: (*min-error*, *max-error*). The *max-error* is the error value associated with the base mesh (M^0) and the *min-error* is the error value associated with the highest resolution mesh (e.g. M_A^i, M_B^j, and M_C^k in Fig. 4).

The allowed runtime error is expressed in screen-space as a *pixels-of-error* (POE) value. Using the POE value and the minimum distance between a cluster and the viewpoint, we compute the maximum object-space error that is allowed for the cluster, called the *error-bound*. View-dependent refinement of each cluster based on the *error-bound* will be explained in Sec. 5.1.

3.4 Preprocess

Given a large dataset, we compute a CHPM representation. Our out-of-core algorithm begins by decomposing the input mesh into a set of clusters. To support transparent accesses on a large mesh during simplification, we preserve all inter-cluster connectivity information. The clusters are input to a cluster hierarchy generation algorithm which builds a balanced hierarchy in a top-down manner. We perform out-of-core hierarchical simplification using the cluster hierarchy.

Each cluster should be independently refinable at runtime for efficiency and out-of-core rendering. For this reason, boundary constraints on simplification are introduced during hierarchical simplification. While guaranteeing crack-free simplifications at runtime, these constraints can prevent drastic simplification and may dramatically increase the number of faces rendered at runtime. To alleviate these problems we introduce *cluster dependencies* that allow boundary simplification while maintaining crack-free rendering at runtime. The role of the dependencies in simplification is detailed in Sec. 4.4 and at runtime in Sec. 5.2.

3.5 Rendering Algorithm

Quick-VDR uses the CHPM as a scene representation for out-of-core view-dependent rendering and occlusion culling. Coarse-grained selective refinement is accomplished by applying cluster-split and cluster-collapse operations to the ACL. Cluster dependencies assure that consistent cluster boundaries are rendered and that we are able to compute drastic simplifications. We use temporal coherence to accelerate refinement and to perform occlusion culling.

Quick-VDR uses the operating system's virtual memory manager through a memory mapped file for out-of-core rendering. In order to overcome the problem of accurately predicting the occlusion events, we introduce one frame of latency in the runtime pipeline. This allows us to load newly visible clusters to avoid stalling the rendering pipeline.

4 Building a CHPM

In this section we present an out-of-core algorithm to compute CHPMs for massive models. Our algorithm proceeds in three steps. First, we decompose the input mesh into a set of clusters. The decomposition occurs in several passes to avoid loading the entire input mesh at once. Next, we construct the cluster hierarchy by repeatedly subdividing the mesh in a top-down manner. Finally, we compute progressive meshes for each cluster by performing a bottom-up traversal of the hierarchy.

4.1 Cluster Decomposition

The clusters form the underlying representation for both the preprocessing step as well as out-of-core view-dependent rendering with occlusion culling. We decompose the model into clusters, which are spatially localized portions of the input mesh. The generated clusters should be nearly equally sized in terms of number of triangles for several reasons. This property is desirable for out-of-core mesh processing to minimize the memory requirements. If the cluster size as well as the number of clusters required in memory at one time are bounded, then simplification and hierarchy construction can be performed with a constant memory footprint. Moreover, enforcing spatial locality and uniform size provides higher performance for occlusion culling and selective refinement.

The out-of-core cluster decomposition algorithm proceeds in four passes. The first three passes only consider the vertices of the original model and create the clusters while the fourth assigns the faces to the clusters. We use a variation of the cluster decomposition algorithm for out-of-core compression of large datasets presented in [Isenburg and Gumhold 2003]. However, our goal is to decompose the mesh for out-of-core processing and view-dependent rendering. As a result, we only compute and store the connectivity information used by the simplification algorithm. The four passes of the algorithm are:

First vertex pass: We compute the bounding box of the mesh.

Second vertex pass: We compute balanced-size clusters of vertices (e.g. 2K vertices). Vertices are assigned to cells of a uniform 3D grid which may be subdivided to deal with irregular distribution of geometry. A graph is built with nodes representing the non-empty cells weighted by vertex count. Edges are inserted between each cell and its k nearest neighbors using an approximate nearest neighbor algorithm [Arya and Mount 1993] (e.g. k=6). We use a graph partitioning algorithm [Hendrickson and Leland 1995] to partition the graph and compute balanced-size clusters.

Third vertex pass: Based on the output of the partitioning, we assign vertices to clusters and reindex the vertices. The new index is a cluster/vertex pair that is used to locate the vertex in the decomposition. A mapping is created that maps the original vertex indices to the new pair of indices. This mapping can be quite large so it is stored in a file that can be accessed in blocks with LRU paging to allow the remainder of the preprocess to operate in a constant memory size.

Face pass: In the final pass, we assign each face to a single cluster that contains at least one of its vertices. The mapping file created in the previous pass is used to locate the vertices. The vertices of faces spanning multiple clusters are marked as constrained for simplification. These vertices make up the boundaries between clusters and are referred to as *shared vertices* while the remaining vertices are referred to as *interior vertices*.

The resulting cluster decomposition consists of manageable mesh pieces that can be transparently accessed in an out-of-core manner for hierarchy generation and simplification, while preserving all the original connectivity information. Different clusters computed for the St. Matthew model are shown in Fig. 2.

4.2 Cluster Hierarchy Generation

In this section, we present an algorithm to compute the cluster hierarchy. The clusters computed by the decomposition algorithm described in the previous section are used as the input to hierarchy generation. Our goal is to compute a hierarchy of clusters with the following properties:

Nearly equal cluster size As previously discussed, consistent cluster size is important for memory management, occlusion culling, and selective refinement. Clusters at all levels of the hierarchy must possess this property.

Balanced cluster hierarchy During hierarchical simplification, cluster geometry is repeatedly simplified and merged in a bottom up traversal. The hierarchy must be well balanced so that merged clusters have nearly identical *error-range*s.

Minimize shared vertices The number of shared vertices at the cluster boundary should be minimized for simplification. Otherwise, in order to maintain consistent cluster boundaries, the simplification will be over-constrained and may result in lower fidelity approximations of the original model.

High spatial locality The cluster hierarchy should have high spatial locality for occlusion culling and selective refinement.

We achieve these goals by transforming the problem of computing a cluster hierarchy into a graph partitioning problem and compute the hierarchy in a top down manner. Each cluster is represented as a node in a graph, weighted by the number of vertices. Clusters are connected by an edge in the graph if they share vertices or are within a threshold distance of each other. The edges are weighted by the number of shared vertices and the inverse of the distance between the clusters, with greater priority placed on the number of shared vertices. The cluster hierarchy is then constructed in a top-down manner by recursively partitioning the graph into halves considering the weights, thus producing a binary tree. The weights guide the partitioning algorithm [Karypis and Kumar 1998] to produce clusters with spatial locality while tending towards fewer shared vertices. The top down partitioning creates an almost balanced hierarchy.

4.3 Out-of-Core Hierarchical Simplification

We simplify the mesh by traversing the cluster hierarchy in a bottom-up manner. Each level of the cluster hierarchy is simplified in a single pass so the simplification requires $\lceil log_2(n) + 1\rceil$ passes where n is the number of leaf clusters. During each pass only the cluster being simplified and clusters with which it shares vertices must be resident in memory.

Simplification operations are ordered by a priority queue based upon quadric errors [Garland and Heckbert 1997]. We build the progressive meshes (PMs) for each cluster by applying “half-edge collapses”. The half-edge collapse, in which an edge is contracted to one of the original vertices, is used to avoid creation of new vertices during simplification. Edges adjacent to shared vertices are not collapsed during simplification. The edge collapses and associated error values are stored along with the most refined mesh of a PM. After creating the PM, the *error-range* of the cluster is computed based on the errors of the PM’s original and base mesh.

When proceeding to the next level up the hierarchy, the mesh within each cluster’s PM is initialized by merging the base meshes of the children. Constraints on vertices that are shared by two clusters being merged are removed thereby allowing simplification of the merged boundary. Since the intermediate clusters should be nearly the same size as the leaf level clusters, each cluster is simplified to half its original face count at each level of the hierarchy.

As simplification proceeds, a file is created for the progressive mesh of each cluster. However, handling many small files at runtime is inefficient. The PM files are merged into one file which can be memory mapped to allow the OS to perform memory management of the PMs and optimize disk access patterns during runtime rendering. The file is stored in a breadth first manner in an attempt to match the probable access pattern during runtime refinement.

4.4 Boundary Constraints and Cluster Dependencies

In order to support out-of-core rendering and to allow efficient refinement at runtime, the PMs of each cluster should be independently refinable while still maintaining a crack-free consistent mesh. To achieve this, our algorithm detects the shared vertices and restricts collapsing the edges adjacent to them during hierarchical simplification. As simplification proceeds up the hierarchy, these constraints are removed because the clusters sharing the vertices have been merged.

While these constraints assure crack-free boundaries between clusters at runtime, they can be overly restrictive. After simplifying several levels of the hierarchy most of the vertices in the base mesh of the PM are shared vertices. As illustrated in Fig. 5 this problem arises along boundaries between clusters that are merged at higher levels in the hierarchy. This can degrade the quality of simplification, and impedes drastic simplification. In Fig. 5 notice that the boundaries between clusters A and B and clusters C and D are merged in the next level of the hierarchy (E and F). However, the boundary between B and C is not merged until higher up the hierarchy, but it is already drastically under-simplified compared to the interior. This constraint problem is common to many hierarchical simplification algorithms that decompose a large mesh for view-dependent rendering [Hoppe 1998; Prince 2000] or compute hierarchies of static LODs (HLODs) [Erikson et al. 2001; Govindaraju et al. 2003; Samanta et al. 2001].

We introduce *cluster-level dependencies* to address this constraint problem. The intuition behind dependencies is that precomputed simplification constraints on shared vertices can be replaced by runtime dependencies. During hierarchical simplification, we may collapse an edge adjacent to a shared vertex. The clusters sharing that vertex are marked as dependent upon each other. This boundary simplification occurs on the merged mesh prior to PM generation for the cluster. In Fig. 5 clusters E and F are marked dependent and thereby allow the boundary to be simplified.

At runtime, splitting a cluster forces all its dependent clusters to split so that the boundaries are rendered without cracks. Likewise, a parent cluster cannot be collapsed unless all of its dependent clusters have also been collapsed. In Fig. 5, clusters E and F must be split together and clusters A, B, C, and D must be collapsed together (assuming E and F are dependent). For example, if clusters B and F are rendered during the same frame, their boundary will be rendered inconsistently and may have cracks.

Although cluster dependencies allow boundary simplification, we need to use them carefully. Since splitting a cluster forces its dependent clusters to split, dependencies will cause some clusters to be rendered that are overly conservative in terms of their *error-bound*. Furthermore, the boundaries change in one frame which may cause popping artifacts. This can be exacerbated by "chained" dependencies in which one cluster is dependent upon another cluster which is in turn dependent upon a third cluster, and so on.

To avoid these potential runtime problems, we prioritize clusters for boundary simplification. At each level of hierarchical simplification the clusters are entered into a priority queue. Priorities are assigned as the ratio of average error of shared vertices to the average error of interior vertices. A cluster, A, is removed from the head of the priority queue. For each cluster, B, that shares at least j (e.g. 5) vertices with A we apply boundary simplification between A and B if the following conditions are met:

1. A and B will not be merged within a small number of levels up the cluster hierarchy (e.g., 2).
2. A and B have similar *error-range*.
3. A dependency between A and B will not introduce a chain (unless all the clusters in the chain share vertices).

This is repeated for each cluster in the priority queue. The first condition avoids creating dependencies between clusters that are resolved within only a few additional hierarchy levels. The second condition discourages dependencies between those clusters that are unlikely to be simultaneously present in the ACL at runtime. The third condition prevents long dependency chains and preserves selective refinement at the cluster level. The cluster dependencies ensure that a sufficient number of shared vertices are collapsed at each level of the hierarchy while still generating and rendering crack-free simplifications at runtime.

5 Interactive Out-of-Core Display

In the previous section, we described an algorithm to compute the CHPM. In this section, we present a novel rendering algorithm that uses the CHPM for occlusion culling, view-dependent refinement and out-of-core rendering. The entire representation including the PMs is stored on the disk. We load the coarse-grained cluster hierarchy into main memory and keep a working set of PMs in main memory. The cluster hierarchy without the PMs is typically a few megabytes for our benchmark models (e.g. 5MB for St. Matthew model). We perform coarse-grained refinement at the cluster level and fine-grained refinement at the level of PMs. We introduce a frame of latency in the rendering pipeline in order to fetch the PMs of newly visible clusters from the disk and avoid stalls in the rendering pipeline.

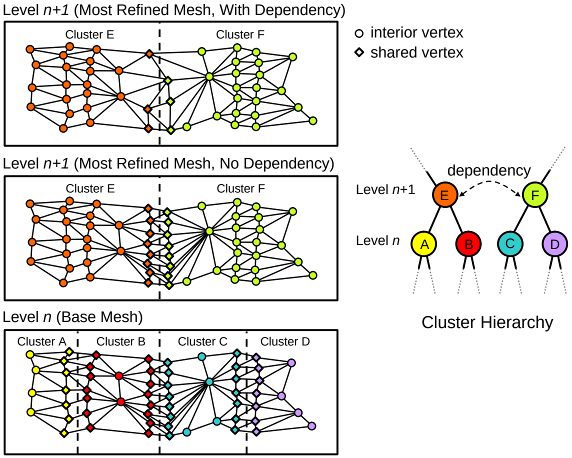


Figure 5: Dependencies. *After simplifying level n of the hierarchy the boundaries AB, BC, and CD are all under-simplified because they are constrained. When initializing the base meshes of E and F prior to simplifying level $n+1$, two of these boundaries, AB and CD, are no longer constrained because they have been merged. The boundary BC was not merged and will remain under-simplified. We can, however, simplify the faces along this boundary if we mark E and F as dependent.*

5.1 View-Dependent Refinement

Our algorithm maintains an active cluster list (ACL), which is a cut in the tree representing the cluster hierarchy. During each frame, we refine the ACL based on the current viewing parameters. Specifically, we traverse the ACL and compute the *error-bound* for each cluster. Each cluster on the active front whose *error-bound* is less than the *min-error* of its *error-range* is split because the PM cannot meet the *error-bound*. Similarly, sibling clusters that have a greater *error-bound* than *max-error* are collapsed. Each PM in the ACL is refined prior to being rendered by choosing the mesh in the PM mesh sequence with the lowest face count that meets the *error-bound*.

5.2 Handling Cluster Dependencies

Our simplification algorithm introduces dependencies between the clusters so that we can simplify their boundaries during the preprocess. We use these dependencies to generate a crack-free simplification at runtime. Cluster-collapses occur to reduce the polygon count in the current refinement. However, prior to collapsing a pair of sibling clusters we must check the parent's dependencies. If the children of any dependent clusters cannot also be collapsed, then the initial cluster collapse cannot occur. These checks occur at the cluster level and are relatively inexpensive.

5.3 Rendering Algorithm

Our rendering algorithm combines occlusion culling, view-dependent refinement, and out-of-core rendering. We first explain the sequence of operations performed during each frame for occlusion culling and view-dependent refinement. After that we present the algorithm for out-of-core rendering by introducing one frame of latency. This extra frame time is used to load the PMs of newly visible clusters from the disk.

5.4 Conservative Occlusion Culling

We exploit temporal coherence in occlusion culling. Each frame our algorithm computes a potentially visible set of clusters (PVS) and a newly visible set (NVS), which is a subset of the PVS. The

PVS for frame i is denoted as PVS_i and the NVS as NVS_i. An occlusion representation (OR_i), represented as a depth buffer, is computed by rendering PVS_{i-1} as an occluder set. Using OR_i we determine PVS_i. The overall rendering algorithm is:

Step 1: Refine ACL. The ACL is refined as described in Sec. 5.1 based on the camera parameters for frame i.

Step 2: Render PVS_{i-1} to compute OR_i: We refine clusters in PVS_{i-1} based on the viewpoint, compute a simplification for each cluster and render them to compute OR_i. OR_i is represented as a depth map that is used for occlusion culling. These clusters are rendered to both the depth and color buffers.

Step 3: Compute NVS_i and PVS_i: The bounding boxes of all the clusters in the ACL are tested for occlusion against OR_i. This test is performed with hardware occlusion queries at the resolution of image precision. PVS_i contains all the clusters with visible bounding boxes, while NVS_i contains the clusters with visible bounding boxes that were not in PVS_{i-1}.

Step 4: Render NVS_i: The PMs of clusters in NVS_i are refined and rendered, generating the final image for frame i.

5.5 Out-of-Core Rendering

Our algorithm works with a fixed memory footprint of main memory and graphics card memory. The entire cluster hierarchy is in main memory and we fetch the PMs of the clusters needed for the current frame as well as prefetch some PMs of clusters for subsequent frames. Additionally, we store the vertices and faces of active clusters in GPU memory. By rendering the primitives directly from GPU memory, AGP bus bandwidth requirement is reduced and we obtain an increased triangle throughput.

Our out-of-core rendering algorithm uses the paging mechanism of the operating system by mapping a file into read-only logical address space [Lindstrom and Pascucci 2002]. To fully take advantage of this mechanism, we store our view-dependent representation in a memory coherent manner, as described in Sec 4.3. We use two separate threads: a fetch thread and a main thread. The fetch thread is used to prepare data for PMs that are likely to be used in the future. This thread provides hints to OS and converts the PM data to the runtime format while a main thread handles refinement, occlusion culling, and rendering.

5.5.1 LOD Prefetching

When we update clusters in the ACL by performing cluster-collapse and cluster-split operations, the children and parent clusters are activated. The PMs of these clusters may not be loaded in the main memory and GPU memory. This can stall the rendering pipeline. To prevent these stalls whenever a cluster is added to the ACL we prefetch its parent and children clusters. Thus, we attempt to keep one level of the hierarchy above and below the current ACL in main memory.

5.5.2 Visibility Fetching

Predicting visibility or occlusion events is difficult, especially in complex models with high depth complexity and small holes. As a result, our algorithm introduces a frame of latency in the rendering pipeline and fetches the PMs of the newly visible cluster in the ACL from the disk.

In our rendering algorithm visibility events are detected in Step 3, and the newly visible clusters are added to NVS_i (Sec. 5.4). These clusters are then rendered in Step 4, which will likely not allow enough time to load these clusters without stalling. Step 2, rendering OR_i, is the most time consuming step of the rendering algorithm. Therefore, we delay the rendering of NVS_i until the end of Step 2 of the next frame and perform rendering of PVS_{i-1} and fetching PMs from the disk in parallel using a fetch thread. Our rendering pipeline is reordered to include a frame of latency thereby increasing the time allowed to load a cluster to avoid stall.

During frame i we perform Steps 1 through 3 of the rendering algorithm with the camera parameters for frame i. However, we perform Step 4 for frame $i-1$ and generate the final image for frame $i-1$. The overall pipeline of the algorithm proceeds as: 1_i, 2_i, 3_i, 4_{i-1}, 1_{i+1}, 2_{i+1}, 3_{i+1}, 4_i, $\ldots$, where n_j refers to Step n of frame j (as shown in Fig. 6). In this reordered pipeline, the PM of a cluster in NVS_i can be loaded anytime from step 3_i until step 2_{i+1} when it must be rendered for OR_{i+1}. However, rendering the already loaded PMs in OR consumes most of the frame time, so there is almost a full frame time to load NVS.

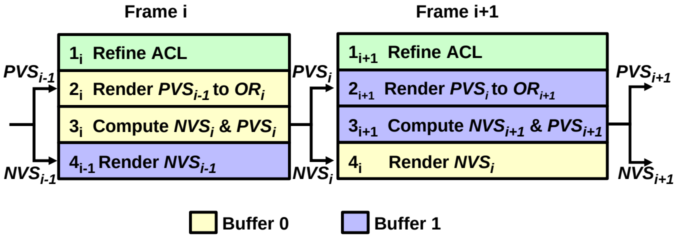

Frame i
Frame i+1
1_i Refine ACL
2_i Render PVS_{i-1} to OR_i
3_i Compute NVS_i & PVS_i
4_{i-1} Render NVS_{i-1}
1_{i+1} Refine ACL
2_{i+1} Render PVS_i to OR_{i+1}
3_{i+1} Compute NVS_{i+1} & PVS_{i+1}
4_i Render NVS_i
PVS_{i-1}
NVS_{i-1}
PVS_i
NVS_i
PVS_{i+1}
NVS_{i+1}
Buffer 0
Buffer 1

Figure 6: Our Rendering Pipeline. *In frame i occlusion culling is performed for frame i but the final image for frame $i-1$ is displayed. This allows extra time for loading the PMs of newly visible clusters. Two off-screen buffers facilitate this interleaving of successive frames. The partial rendering of frame i is stored in one buffer while occlusion culling for frame $i+1$ occurs in the other buffer.*

To implement this pipeline, we use a pair of off-screen buffers. One buffer holds the partial rendering of a frame from Step 2 so that it may be composited with the newly visible clusters in Step 4 the following frame. The odd numbered frames use the first buffer while the even-numbered frames use the second buffer, so that each consecutive pair of frames can render to separate buffers. Fig. 6 illustrates how the buffers are used for two consecutive frames.

6 Implementation and Performance

In this section we describe our implementation and highlight its performance on massive models.

6.1 Implementation

We have implemented our out-of-core simplification and runtime system on a dual 2.4GHz Pentium-IV PC, with 1GB of RAM and a GeForce Ultra FX 5950 GPU with 128MB of video memory. Our system runs on Windows XP and uses the operating system's virtual memory through memory mapped files. Windows XP imposes a 2GB limitation for mapping a file to user-addressable address space. We overcome this limitation by mapping a 32MB portion of the file at a time and remapping when a PM is required from outside this range.

We use the METIS graph partitioning library [Karypis and Kumar 1998] for cluster computation. We use NVIDIA OpenGL extension GL_NV_occlusion_query to perform occlusion queries. We are able to perform an average of approximately 400K occlusion queries per second on the bounding boxes.

We achieve high throughput from graphics cards by storing the mesh data on the GPU, thereby reducing the data transferred to the GPU each frame. We use the GL_ARB_vertex_buffer_object OpenGL extension that performs GPU memory management for both the vertex and the face arrays. However, we generate some new faces during each frame by performing vertex splits or edge collapse operations during local refinement of each PM. In practice, only a small number of PMs require refinement during each frame. As a result, we only transmit the faces of these PMs to the GPU and the other faces are cached in the GPU memory.

6.2 Massive Models

Our algorithm has been applied to three complex models, a coal-fired power plant composed of more than 12 million polygons and 1200 objects (Fig. 3), the St. Matthew model consisting of a single 372 million polygon object (Fig. 2), and an isosurface model consisting of 100 million polygons also originally represented as a single object (Fig. 1). The details of these models are shown in Table 1. We generated paths in each of our test models and used them to test the performance of our algorithm. These paths are shown in the accompanying video.

6.3 Performance

We have applied our out-of-core CHPM generation preprocess to each of the models. Table 1 presents preprocessing time for each

Model	PP	Isosurface	St. Matthew
Triangles (M)	12.2	100	372
Original Size (MB)	485	$2,543$	$9,611$
Num Clusters (K)	5.8	32	65
Memory footprint used (MB)	32	256	512
Size of CHPM (MB)	625	$3,726$	$13,992$
Processing time (min)	64	350	$2,369$

Table 1: Preprocess *Preprocess timings and storage requirements for test models. We are able to compute a CHPM for each environment using our out-of-core algorithm and a memory footprint of* $32 - 512$*MB.*

model on the PC. Hierarchical simplification takes approximately 85% of the preprocess time. The remainder of the time is dominated by the face pass of the cluster decomposition. This pass makes random accesses to the out-of-core vertex index mapping table to locate face vertices in the cluster decomposition. We could use an external sort of the mapping table to improve access patterns as in [Lindstrom and Silva 2001].

We are able to render all these models at interactive rates (10-30 frames per second) on a single PC. Fig. 7 illustrates the performance of the system on a complex path in isosurface model. Table 2 shows the average frame rate, front size, and number of edge collapse and vertex split operations performed for paths in each of our test models. Table 3 shows the average breakdown of the frame time for each model. Rendering costs dominate the frame time.

6.3.1 Out-of-core

Our system relies on the underlying operating systems virtual memory management for paging of PMs and, as discussed in Sec. 5.5.2, uses a frame of latency to hide load times of newly visible clusters. The frame rates of a sample path of the isosurface model are shown in Fig. 7.

6.3.2 Occlusion culling

Occlusion culling is very important for rendering models with high depth complexity such as the power plant and isosurface models. Fig. 7 highlights the benefit of occlusion culling by comparing the frame rate of our system over a path with occlusion culling enabled and disabled. On average the frame rate is $25 - 55\%$ higher when occlusion culling enabled.

7 Comparisons and Limitations

In this section, we analyze the performance of Quick-VDR. We also highlight the benefits over prior approaches and describe some of its limitations.

Refinement Cost of CHPMs vs. Vertex Hierarchies: Most of the earlier algorithms for view-dependent simplification use a vertex hierarchy. These algorithm compute an active vertex front in the hierarchy and handle dependencies at the vertex or edge level.

We compared the refinement cost of CHPM with an implementation of a vertex hierarchy (VDPM) for an isosurface with about 1M triangles (see Table 4). We have observed that CHPM refinement cost is one-two orders of magnitude lower, even without occlusion culling. This lowered cost is due to the following factors:

1. Our clusters consist of thousands of triangles. As a result, the size of ACL is typically more than one-two orders of magnitude smaller than the size of active front in a vertex hierarchy.
2. We perform coarse-grained refinement operations at the cluster level and use a single conservative error bound for an entire cluster. Therefore, refinement of individual PMs is much faster than it would be by performing per-vertex computations across an active vertex front.
3. Handling dependencies at the cluster level is significantly cheaper than those at the vertex level.

Conservative Occlusion Culling: Quick-VDR performs conservative occlusion culling up to image precision. The occlusion computations are performed at the cluster level. The size of ACL is typically a few hundred clusters so performing occlusion culling takes $1 - 10\%$ of the total frame time.

Model	POE	Avg FPS	Avg Front Size	Avg # Ecol/Vsplit	Avg # Tri(K)
Power plant	1	33	1219	410	403
Isosurface	40	16	937	164	765
St. Matthew	1	17	366	802	771

Table 2: Runtime Performance: *We highlight the performance on the three benchmarks. The average frame rate, average front size, and average number of edge collapse and vertex splits are presented for a sample path in each model. All the data is acquired at* 512×512 *resolution. We use a* 400*MB memory footprint for the power plant model and* 600*MB for other models.*

Model	Refining	Occlusion Culling	Rendering
Power plant	3.6%	10.7%	85.7%
Isosurface	1.9%	1.9%	96.2%
St. Matthew	4.2%	1.4%	94.4%

Table 3: Runtime Timing Breakdown. *This table shows the percentage of frame time spent on the three major computations of the runtime algorithm. More than* 85% *of the time is spent in rendering the potential occluders and visible primitives.*

Storage Overhead: Our CHPM implementation requires on average 88MB per million vertices. This is low compared to Hoppe's [1997] VDPM representation (224MB) and XFastMesh (108MB) [DeCoro and Pajarola 2002]. Moreover, CHPM can easily represent models with non-manifold topologies.

Out-of-Core Computation: Our out-of-core preprocess is able to construct a CHPM from large datasets using a constant-sized memory footprint. Moreover, our hierarchical simplification algorithm produces nearly in-core quality progressive meshes and preserves the mesh connectivity.

Our current implementation does not achieve the same performance of [Lindstrom 2003] in terms of triangles simplified per second. Lindstrom [2003] applies external-memory sorts to his out-of-core data structures to improve the access patterns and we can also use them to improve the performance of our system. However, Lindstrom [2003] does not preserve all the faces and vertices in the leaf level of the hierarchy.

Quick-VDR introduces a frame of latency to fetch PMs of the newly visible cluster from the disk. This is needed to take into account the visibility events that can occur between successive frames. Earlier algorithms that combine visibility computations with out-of-core rendering decompose large CAD environments into rectangular cells and do not introduce additional latency [Aliaga et al. 1999; Correa et al. 2002]. However, it may not be easy to decompose large isosurfaces for visibility-based prefetching. Moreover, the MMR system [Aliaga et al. 1999] uses image-based impostors and can introduce additional popping artifacts. The iWalk system [Correa et al. 2002] performs approximate occlusion culling.

Application to Massive Models: Our algorithm has been applied to complex models composed of a few hundred million polygons. In contrast, view-dependent algorithms were applied to scanned models with $8 - 10$ million triangles [DeCoro and Pajarola 2002], 2M triangle isosurface and the power plant model [Yoon et al. 2003] or were combined with approximate occlusion culling [El-Sana and Bachmat 2002]. Lindstrom's algorithm [2003] does not perform occlusion culling and has been applied it to a 47M triangle isosurface. It is difficult to perform a direct comparisons with these approaches as they used an older generation of the hardware and it may not have the same set of features (e.g. occlusion queries). Yoon et al. [2003] also used clusters for occlusion culling. However, their underlying representation is a vertex hierarchy and their algorithm does not scale to massive models. The main reasons for the high frame-rate performance of Quick-VDR on massive models are:

- Low refinement cost during each frame.
- High GPU throughput obtained by rendering PMs directly from GPU memory
- Significant occlusion culling based on the cluster hierarchy.
- Out-of-core computations at the cluster level.

Limitations The main limitation of our approach is one frame of

Method	Vertex Hierarchy	CHPM
Num Dependency Checks	4.2M	223
Refinement Time(ms)	1,221	32

Table 4: Refinement Performance. *A comparison of refinement cost between a CHPM and vertex hierarchy in a* 1*M triangle isosurface. This table measures the time to fully refine the mesh from the base mesh.*

latency in the rendering pipeline. Our visibility fetching scheme ameliorates the stalling problem but does not eliminate the problem. The set of dynamic LODs or simplifications computed by the CHPM could be smaller than the ones computed using a full vertex hierarchy. This is because of our decomposition of the model into a cluster and representation of each cluster as a linear sequence of edge collapses. Moreover, cluster dependencies that force us to perform additional cluster-split operations might cause popping artifacts. We use one error bound for the entire cluster. As a result our simplification error bounds can be more conservative as compared to the ones used in vertex hierarchies and we may render more triangles per frame to guarantee the screen-space POE bound. Our occlusion culling algorithm assumes high temporal coherence between successive frames. Its effectiveness varies as a function of coherence between successive frames.

8 Conclusion and Future Work

We have presented Quick-VDR, a novel algorithm for view-dependent rendering of massive models. Quick-VDR represents the scene using a CHPM and provides us with the ability to perform coarse-grained as well as fine-grained refinement. It significantly reduces the refinement cost as compared to earlier approaches based on vertex hierarchies. The cluster hierarchy enables occlusion culling and out-of-core rendering. Quick-VDR relies on an out-of-core algorithm to compute a CHPM and combines view-dependent simplification, occlusion culling and out-of-core rendering. Quick-VDR has been applied to massive models with a few hundred million triangles and can render them at interactive rates using commodity graphics systems.

Many avenues for future work lie ahead. In addition to overcoming the limitations of the current approach, we would like to use geomorphing [Hoppe 1996] to smooth the popping artifacts. Moreover, the two level refinement could be extended to consider view-dependent effects such as specular highlights, silhouettes and shadows. We would also like to explore methods for predicting occlusion events so that we can further improve our out-of-core computation and eliminate the frame of latency in the rendering pipeline. Finally, we would like to use the CHPM representation for multiresolution compression and collision detection between massive models.

References

ALIAGA, D., COHEN, J., WILSON, A., ZHANG, H., ERIKSON, C., HOFF, K., HUDSON, T., STUERZLINGER, W., BAKER, E., BASTOS, R., WHITTON, M., BROOKS, F., AND MANOCHA, D. 1999. MMR: An integrated massive model rendering system using geometric and image-based acceleration. In *Proc. of ACM Symposium on Interactive 3D Graphics*, 199–206.

ARYA, S., AND MOUNT, D. M. 1993. Approximate nearest neighbor queries in fixed dimensions. In *Proc. 4th ACM-SIAM Sympos. Discrete Algorithms*, 271–280.

CIGNONI, P., GANOVELLI, F., GOBBETTI, E., MARTON, F., PONCHIO, F., AND SCOPIGNO, R. 2003. Planet-sized batched dynamic adaptive meshes (p-bdam). In *IEEE Visualization*.

CIGNONI, P., MONTANI, C., ROCCHINI, C., AND SCOPIGNO, R. 2003. External memory management and simplification of huge meshes. In *IEEE Transaction on Visualization and Computer Graphics*.

COHEN-OR, D., CHRYSANTHOU, Y., AND SILVA, C. 2001. A survey of visibility for walkthrough applications. *SIGGRAPH Course Notes # 30*.

CORREA, W., KLOSOWSKI, J., AND SILVA, C. 2002. iwalk: Interactive out-of-core rendering of large models. In *Technical Report TR-653-02, Princeton University*.

DECORO, C., AND PAJAROLA, R. 2002. Xfastmesh: View-dependent meshing from external memory. In *IEEE Visualization*.

EL-SANA, J., AND BACHMAT, E. 2002. Optimized view-dependent rendering for large polygonal dataset. *IEEE Visualization*, 77–84.

EL-SANA, J., AND CHIANG, Y.-J. 2000. External memory view-dependent simplification. *Computer Graphics Forum 19*, 3, 139–150.

EL-SANA, J., AND VARSHNEY, A. 1999. Generalized view-dependent simplification. *Computer Graphics Forum*, C83–C94.

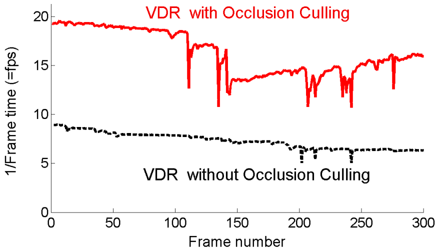


Figure 7: Frame Rate in Isosurface Model. *Frame rates are shown for a sample path using our system. For comparison we show our system without occlusion culling.*

EL-SANA, J., SOKOLOVSKY, N., AND SILVA, C. 2001. Integrating occlusion culling with view-dependent rendering. *Proc. of IEEE Visualization*.

ERIKSON, C., MANOCHA, D., AND BAXTER, B. 2001. Hlods for fast display of large static and dynmaic environments. *Proc. of ACM Symposium on Interactive 3D Graphics*.

FUNKHOUSER, T., KHORRAMABADI, D., SEQUIN, C., AND TELLER, S. 1996. The ucb system for interactive visualization of large architectural models. *Presence 5*, 1, 13–44.

GARLAND, M., AND HECKBERT, P. 1997. Surface simplification using quadric error bounds. *Proc. of ACM SIGGRAPH*, 209–216.

GOVINDARAJU, N., SUD, A., YOON, S., AND MANOCHA, D. 2003. Interactive visibility culling in complex environments with occlusion-switches. *Proc. of ACM Symposium on Interactive 3D Graphics*, 103–112.

GREENE, N., KASS, M., AND MILLER, G. 1993. Hierarchical z-buffer visibility. In *Proc. of ACM SIGGRAPH*, 231–238.

HENDRICKSON, B., AND LELAND, R. 1995. A multilevel algorithm for partitioning graphs. In *Super Computing*.

HOPPE, H. 1996. Progressive meshes. In *Proc. of ACM SIGGRAPH*, 99–108.

HOPPE, H. 1997. View dependent refinement of progressive meshes. In *ACM SIGGRAPH Conference Proceedings*, 189–198.

HOPPE, H. 1998. Smooth view-dependent level-of-detail control and its application to terrain rendering. In *IEEE Visualization Conference Proceedings*, 35–42.

ISENBURG, M., AND GUMHOLD, S. 2003. Out-of-core compression for gigantic polygon meshes. In *ACM Trans. on Graphics (Proc. of ACM SIGGRAPH)*, vol. 22.

KARYPIS, G., AND KUMAR, V. 1998. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*.

KLOSOWSKI, J., AND SILVA, C. 2001. Efficient conservative visiblity culling using the prioritized-layered projection algorithm. *IEEE Trans. on Visualization and Computer Graphics 7*, 4, 365–379.

L. DE FLORIANI, P. MAGILLO, E. P. 1997. Building and traversing a surface at variable resolution. In *IEEE Visualization*.

LINDSTROM, P., AND PASCUCCI, V. 2002. Terrain simplification simplified: A general framework for view-dependent out-of-core visualization. In *IEEE Transaction on Visualization and Computer Graphics*, 239–254.

LINDSTROM, P., AND SILVA, C. 2001. A memory insensitive technique for large model simplification. In *Proc. of IEEE Visualization*, 121–126.

LINDSTROM, P. 2003. Out-of-core construction and visualization of multiresolution surfaces. In *ACM Symposium on Interactive 3D Graphics*.

LUEBKE, D., AND ERIKSON, C. 1997. View-dependent simplification of arbitrary polygon environments. In *Proc. of ACM SIGGRAPH*.

PAJAROLA, R. 2001. Fastmesh: Efficient view-dependent mesh. In *Proc. of Pacific Graphics*, 22–30.

PRINCE, C. 2000. *Progressive Meshes for large Models of Arbitrary Topology*. Master's thesis, University of Washington.

RUSINKIEWICZ, S., AND LEVOY, M. 2000. Qsplat: A multiresolution point rendering system for large meshes. *Proc. of ACM SIGGRAPH*.

SAMANTA, R., FUNKHOUSER, T., AND LI, K. 2001. Parallel rendering with k-way replication. *IEEE Symposium on Parallel and Large-Data Visualization and Graphics*.

SHAFFER, E., AND GARLAND, M. 2001. Effient adaptive simplification of massive meshes. In *Proc. of IEEE Visualization*.

WONKA, P., WIMMER, M., AND SILLION, F. 2001. Instant visibility. In *Proc. of Eurographics*.

XIA, J., EL-SANA, J., AND VARSHNEY, A. 1997. Adaptive real-time level-of-detail-based rendering for polygonal models. *IEEE Transactions on Visualization and Computer Graphics 3*, 2 (June), 171–183.

YOON, S., SALOMON, B., AND MANOCHA, D. 2003. Interactive view-dependent rendering with conservative occlusion culling in complex environments. *Proc. of IEEE Visualization*.

ZHANG, H., MANOCHA, D., HUDSON, T., AND HOFF, K. 1997. Visibility culling using hierarchical occlusion maps. *Proc. of ACM SIGGRAPH*.

Cache-Oblivious Mesh Layouts

Sung-Eui Yoon,[1] Peter Lindstrom[2]
Valerio Pascucci,[2] Dinesh Manocha[1]
1: University of North Carolina - Chapel Hill
2: Lawrence Livermore National Laboratory

http://gamma.cs.unc.edu/COL

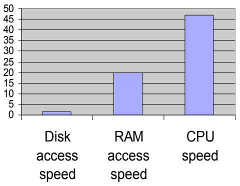
Goal

- Compute cache-coherent layouts of polygonal meshes
  - For geometric processing and visualization
  - Handle any kinds of polygonal models (e.g., irregular geometry)

Motivation

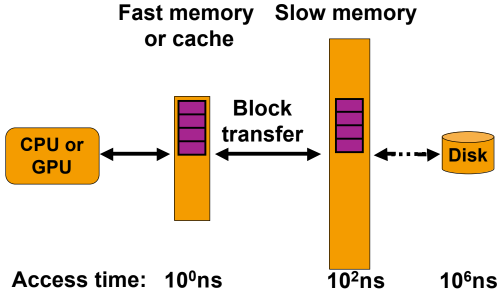
- High growth rate of computational power of CPUs and GPUs

Growth rate during 1993 – 2004



Courtesy: http://www.hcibook.com/e3/online/moores-law/

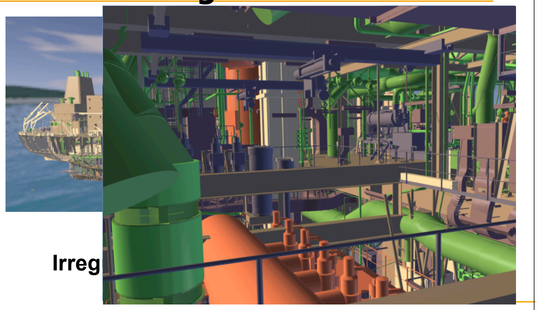
Memory Hierarchies and Caches



Cache-Coherent Layouts

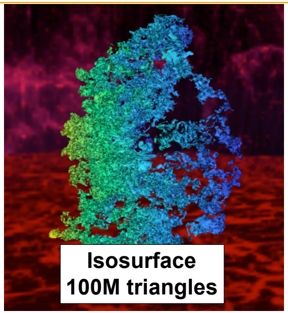
- Cache-Aware
  - Optimized for particular cache parameters (e.g., block size)
- Cache-Oblivious
  - Minimizes data access time without any knowledge of cache parameters
  - Directly applicable to various hardware and memory hierarchies

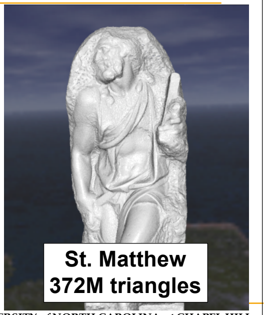
CAD Model – Double Eagle Tanker Model



Irreg

Isosurface and Scanned Models


Isosurface
100M triangles


St. Matthew
372M triangles

Main Contribution

- Algorithm to compute cache-oblivious layouts of polygonal meshes

Cache-oblivious metric

Multilevel optimization framework

Applicable to hierarchical representations

Live Demo – View-Dependent Rendering (VDR)

- Based on multiresolution hierarchy
  - ♦Dynamically computes simplification
  - ♦Cache-oblivious layout is used to minimize GPU vertex cache misses

Double Eagle Tanker
82 Million triangles

GeForce Go
6800 Ultra

Related Work

- Cache-coherent algorithms
- Mesh layouts

Cache-Coherent Algorithms

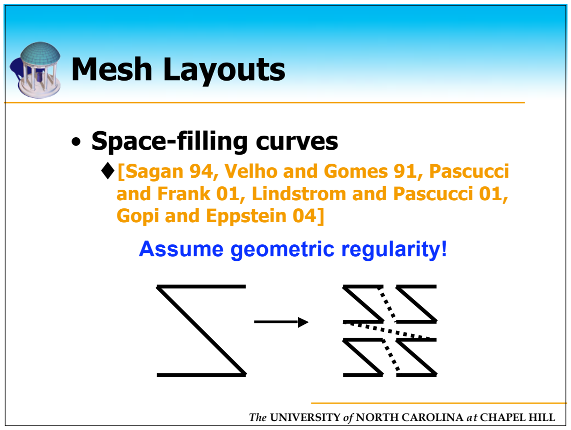
- Cache-aware [Coleman and McKinley 95, Vitter 01, Sen et al. 02]
- Cache-oblivious [Frigo et al. 99, Arge et al. 04]

Focus on specific problems such as sorting and linear algebra computations

Mesh Layouts

- Rendering sequences
  - ♦Triangle strips
  - ♦[Deering 95, Hoppe 99, Bogomjakov and Gotsman 02]
- Processing sequences
  - ♦[Isenburg and Gumhold 03, Isenburg and Lindstrom 04]

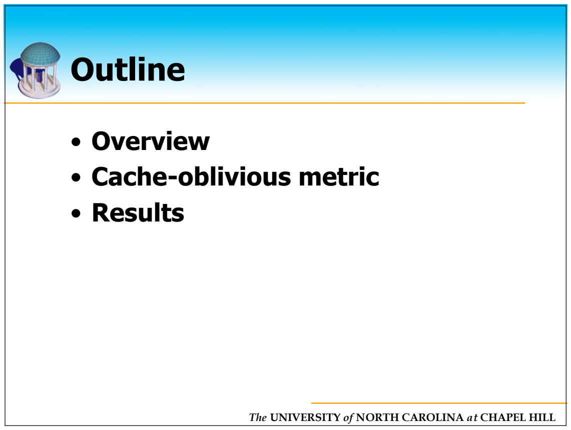
Assume that access pattern globally follows the layout order!



Mesh Layouts

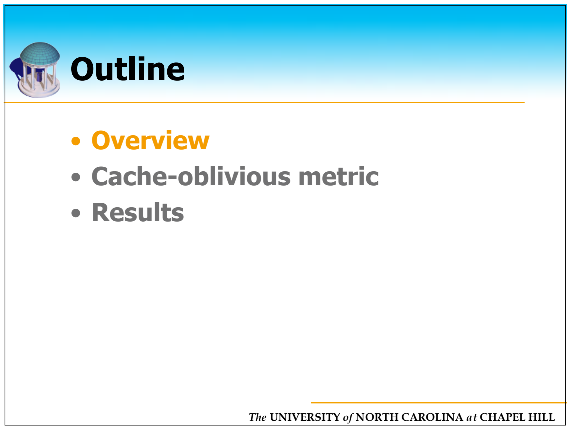
- Space-filling curves
  - ◆[Sagan 94, Velho and Gomes 91, Pascucci and Frank 01, Lindstrom and Pascucci 01, Gopi and Eppstein 04]

Assume geometric regularity!



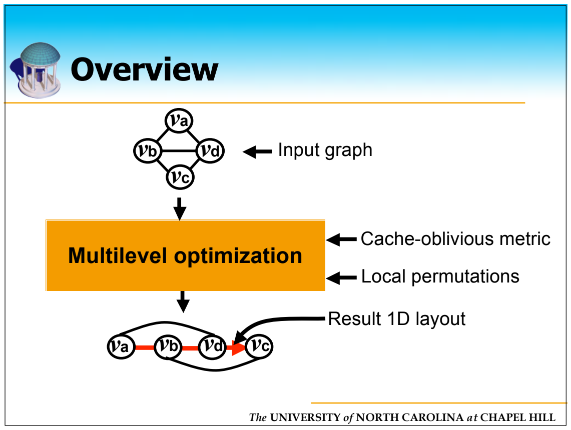
Outline

- Overview
- Cache-oblivious metric
- Results



Outline

- Overview
- Cache-oblivious metric
- Results



Overview

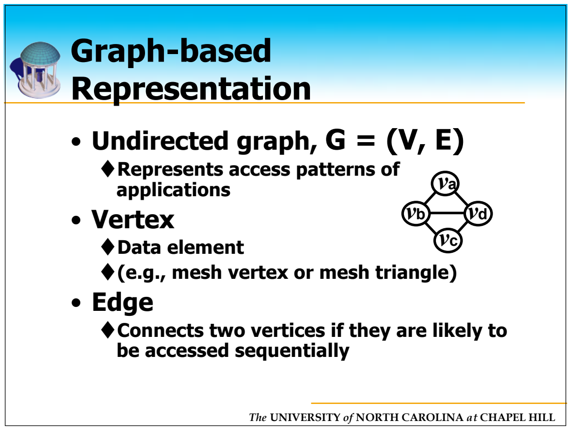
Input graph

Multilevel optimization

Cache-oblivious metric

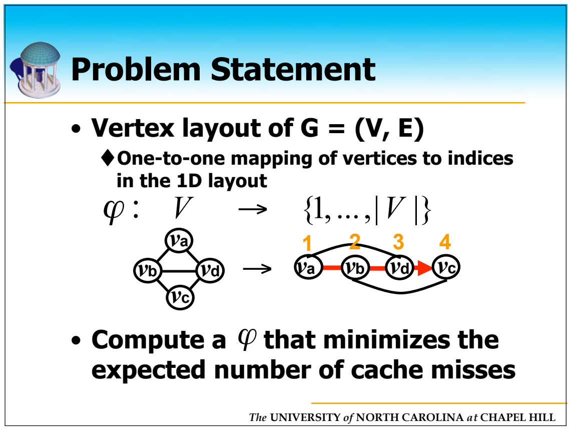
Local permutations

Result 1D layout



Graph-based Representation

- Undirected graph, G = (V, E)
  - ◆Represents access patterns of applications
- Vertex
  - ◆Data element
  - ◆(e.g., mesh vertex or mesh triangle)
- Edge
  - ◆Connects two vertices if they are likely to be accessed sequentially

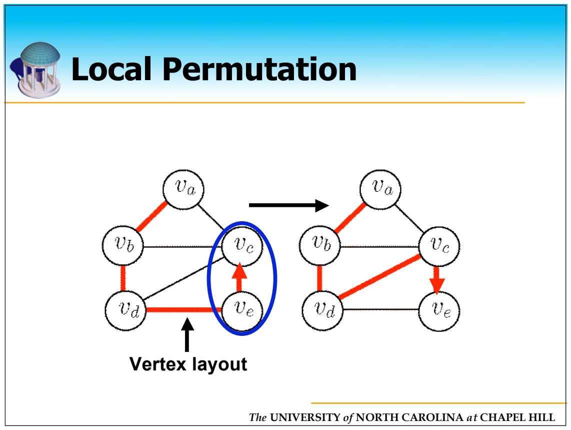


Problem Statement

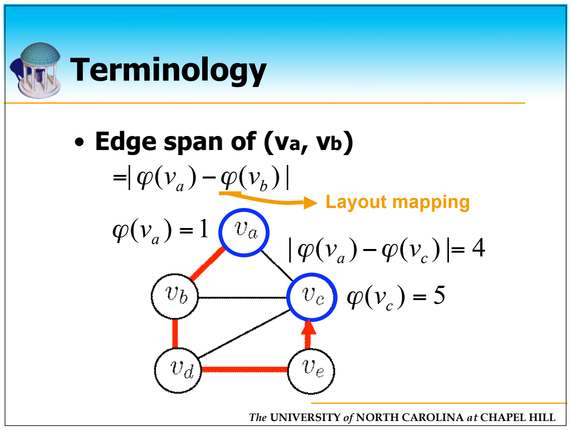
- Vertex layout of G = (V, E)
  - ◆One-to-one mapping of vertices to indices in the 1D layout

$$\varphi : \quad V \quad \rightarrow \quad \{1, \ldots, |V|\}$$

- Compute a φ that minimizes the expected number of cache misses



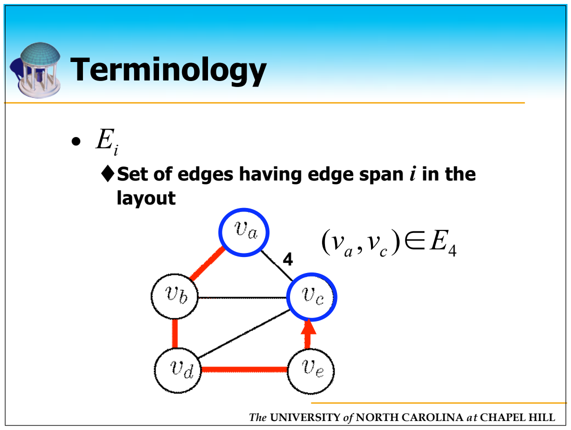
Local Permutation



Terminology

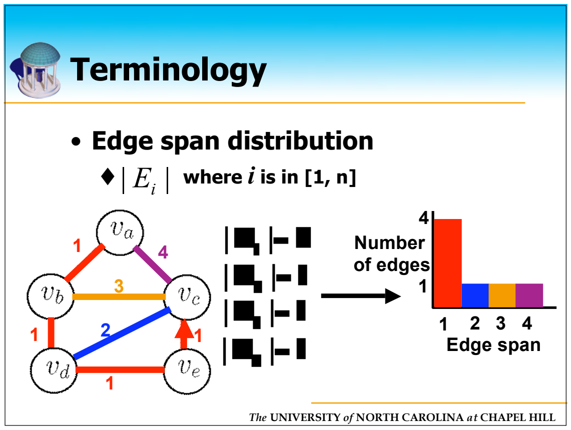
- Edge span of (v_a, v_b)

$=|\varphi(v_a)-\varphi(v_b)|$



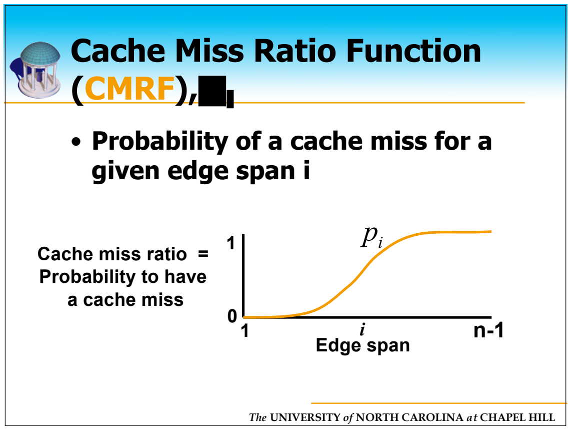
Terminology

- E_i
  - Set of edges having edge span *i* in the layout



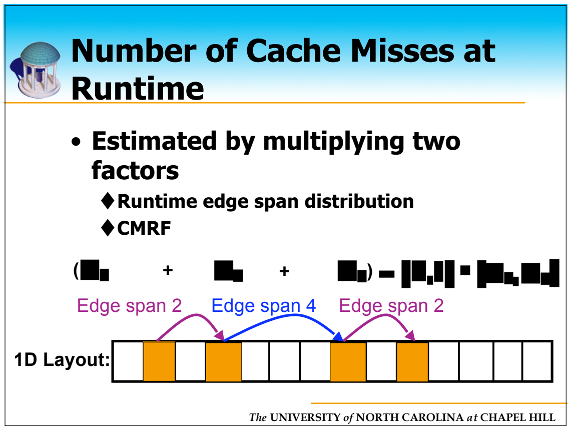
Terminology

- Edge span distribution
  - $|E_i|$ where *i* is in [1, n]



Cache Miss Ratio Function (CMRF), [illegible]

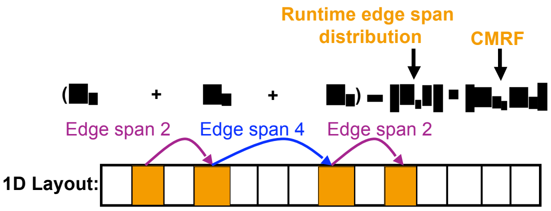
- Probability of a cache miss for a given edge span i



Number of Cache Misses at Runtime

- Estimated by multiplying two factors
  - Runtime edge span distribution
  - CMRF

Number of Cache Misses at Runtime



Expected Number of Cache Misses

Edge span distribution of the layout

The number of vertices

$$\sum_{i=1}^{n-1}|E_i|\,p_i$$

- ♦Approximate runtime edge span distribution with one of the layout

Outline

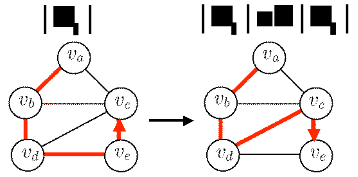
- Overview
- Cache-oblivious metric
- Results

Cache-Oblivious Metric

- Decides if a local permutation reduces number of cache misses
  - ♦Probabilistic formulation
  - ♦Reduces to geometric volume computation

Does a Local Permutation Decrease Cache Misses?

$$\sum_{i=1}^{n-1}|E_i|\,p_i \overset{?}{>} \sum_{i=1}^{n-1}(|E_i|+\Delta|E_i|)\,p_i$$



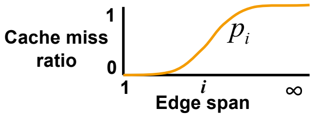
Does a Local Permutation Decrease Cache Misses?

$$\sum_{i=1}^{n-1}\cancel{|E_i|\,p_i} > \sum_{i=1}^{n-1}(\cancel{|E_i|}+\Delta|E_i|)\,p_i$$

$$\rightarrow \sum_{i=1}^{n-1}\Delta|E_i|\,p_i < 0$$

Monotonocity of CMRF, p_i

- Assume CMRF is a monotonically increasing function of edge span



Exact Cache-Oblivious Metric

$$\rightarrow \sum_{i=1}^{n-1} \Delta |E_i| \, p_i < 0$$

Monotonicity of CMRF

where

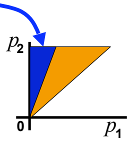
$$\boxed{0 \le p_1 \le p_2 \le \ldots \le p_{n-2} \le p_{n-1} \le 1}$$

All the possible cache configurations

Geometric Formulation

Half hyperspace

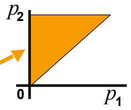
$$\sum_{i=1}^{n-1} \Delta |E_i| \, p_i < 0$$



where

$$0 \le p_1 \le p_2 \le \ldots \le p_{n-2} \le p_{n-1} \le 1$$

Closed hyperspace■ ■▪–▮



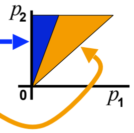
Geometric Volume Computation

- Assume each CMRF to be equally likely

$$\sum_{i=1}^{n-1} \Delta |E_i| \, p_i < 0$$

where

$$0 \le p_1 \le p_2 \le \ldots \le p_{n-2} \le p_{n-1} \le 1$$



- Half hyperspace (blue area)
  - ♦Space of CMRFs that reduce cache misses

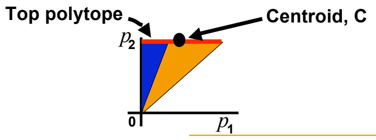
Geometric Volume Computation

Time complexity

- ♦Exact: ■■▪▪▮ [Lasserre and Zeron 01]
- ♦Approximate: ■■▪▮ [Kannan et al. 97]

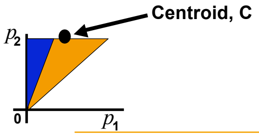
Fast and Approximate Volume Comparison

- Define a top polytope in closed hyperspace
- Compute the centroid, C, of the top polytope



Fast and Approximate Volume Comparison

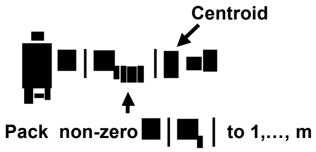
- Use the centroid for approximate volume comparison
  - The volume containing the centroid is likely to be larger



Bound of Approximation

- 0.1% ~ 0.3% compared to the exact metric

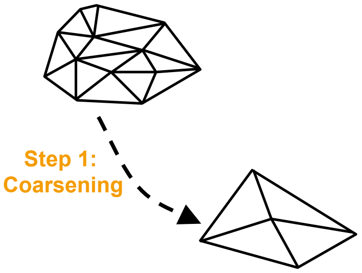
Final Approximate Metric



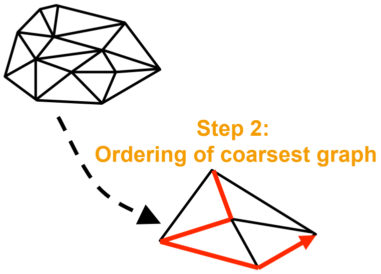
Layout Optimization

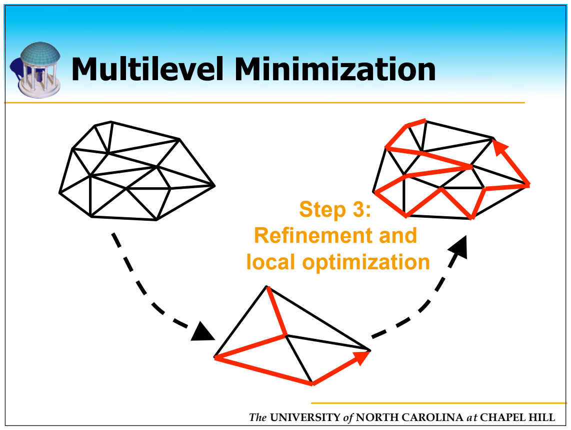
- Find an optimal layout that minimizes our metric
  - Combinatorial optimization problem

Multilevel Minimization


Step 1:
Coarsening

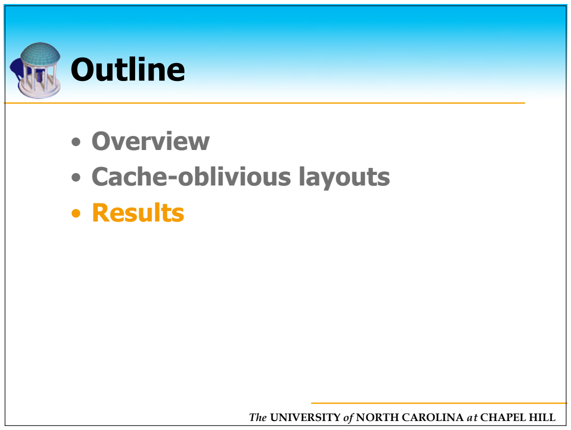
Multilevel Minimization


Step 2:
Ordering of coarsest graph



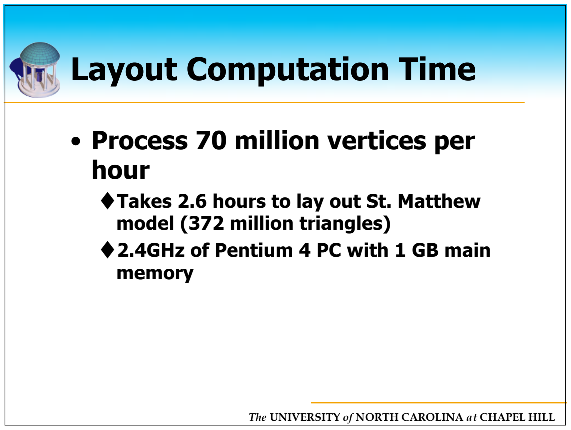
Multilevel Minimization

Step 3:
Refinement and
local optimization



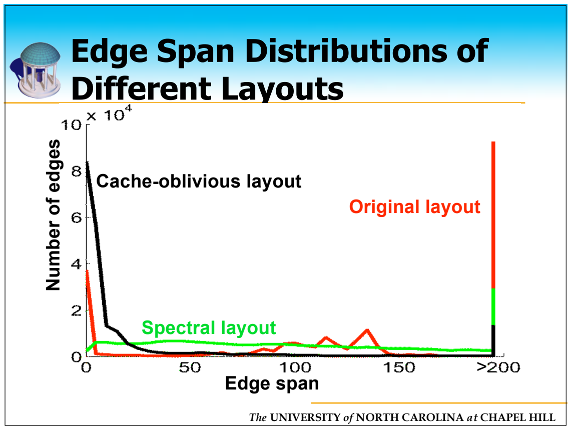
Outline

- Overview
- Cache-oblivious layouts
- **Results**

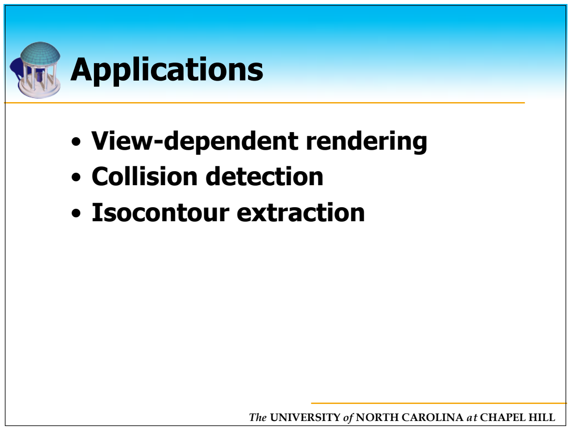


Layout Computation Time

- **Process 70 million vertices per hour**
  - ♦ **Takes 2.6 hours to lay out St. Matthew model (372 million triangles)**
  - ♦ **2.4GHz of Pentium 4 PC with 1 GB main memory**

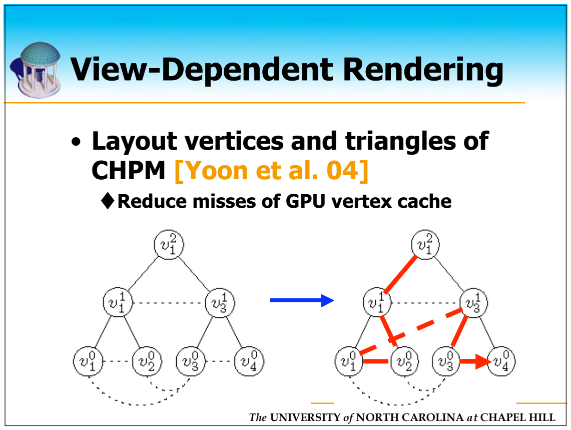


Edge Span Distributions of Different Layouts



Applications

- **View-dependent rendering**
- **Collision detection**
- **Isocontour extraction**



View-Dependent Rendering

- **Layout vertices and triangles of CHPM [Yoon et al. 04]**
  - ♦ **Reduce misses of GPU vertex cache**

View-Dependent Rendering

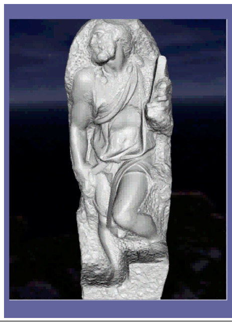
Peak performance: 145 M tri / s on GeForce 6800 Ultra

Models	# of Tri.	Our layout	Simplification layout [Yoon et al. 04]
St. Matthew	372M	106 M/s	23 M/s
Isosurface	100M	90 M/s	20 M/s
Double Eagle Tanker	82M	47 M/s	22 M/s

4.5X

2.1X

Realtime Captured Video – St. Matthew Model

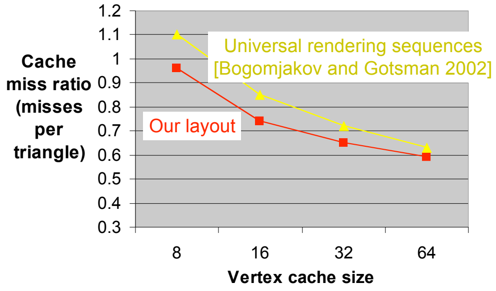


St. Matthew

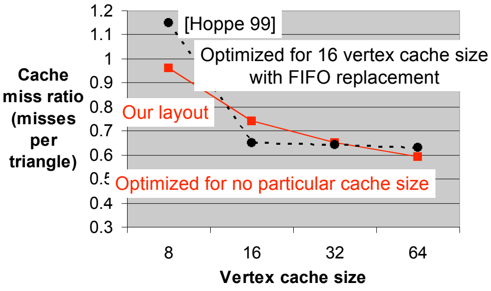
372 Million triangles
9 Gigabyte

GPU: GeForce 6800

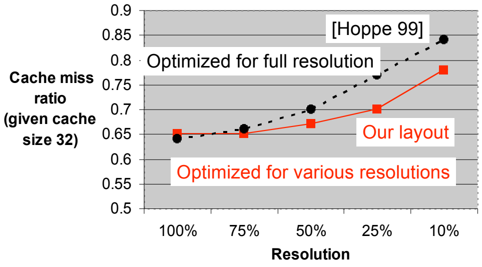
Comparison with Other Rendering Sequences



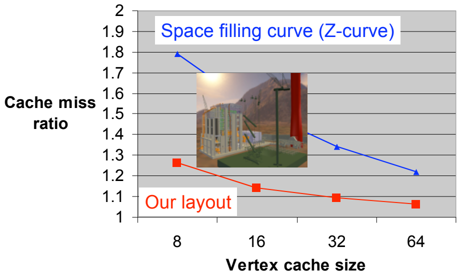
Comparison with Other Rendering Sequences



Performance during View-Dependent Rendering



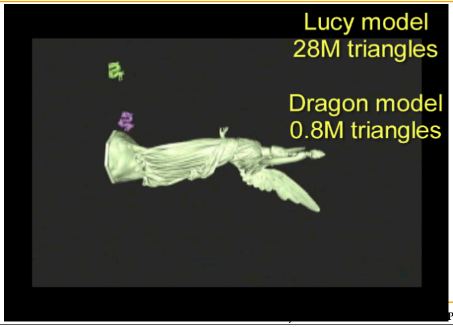
Comparison with Space Filling Curve on Power Plant Model



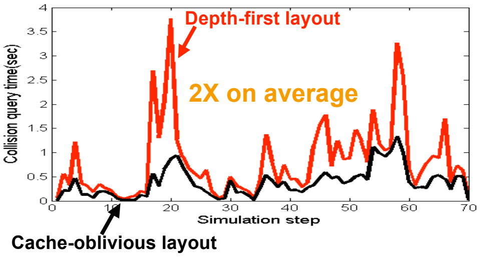
Collision Detection

- Bounding volume hierarchies
  - ♦Widely used to accelerate the performance of collision detection
  - ♦Traversed to find contacting area
  - ♦Uses pre-computed layouts of OBB trees [Gottschalk et al. 96]

Rigid Body Simulation



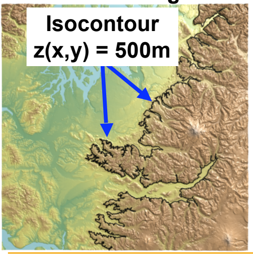
Collision Detection Time



Isocontour Extraction

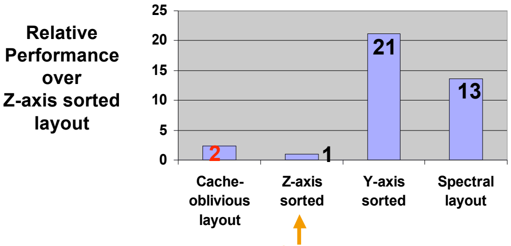
- Contour tree [van Kreveld et al. 97]
- Use mesh as the input graph
- Extract an isocontour that is orthogonal to z-axis

Puget sound, 134 M triangles



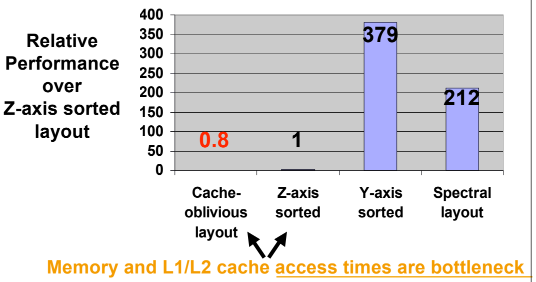
Comparison – First Extraction of Z(x,y) = 500m

Disk access time is bottleneck



Nearly optimized for particular isocontour

Comparison – Second Extraction of Z(x,y) = 500m



Memory and L1/L2 cache access times are bottleneck

Limitations

- Assumptions on CMRF
  - ♦May not work well for all applications
- Does not compute global optimum
  - ♦Greedy solution

Advantages

- General
  - ♦Applicable to all kinds of polygonal models
  - ♦Works well for various applications
- Cache-oblivious
  - ♦Can have benefit from CPU/GPU cache to memory and disk
- No modification of runtime application
  - ♦Only layout computation

OpenCCL: Cache-Coherent Layouts of Graphs and Meshes

- Source codes for computing a cache-coherent layout
- Easy to use

Google "Cache Oblivious Mesh Layout"
or
Http://gamma.cs.unc.edu/COL

```
Graph.AddEdge (1, 2);

int Order [NumVertex];
Graph.ComputeOrdering (Order);
```

Conclusion

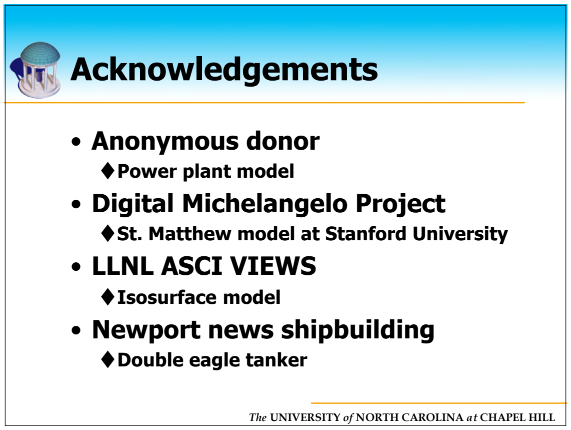
- Novel algorithm for computing cache-oblivious mesh layouts
  - ♦Cast the problem as an optimization
  - ♦Probabilistically compute the expected number of caches misses
  - ♦Achieve significant improvements (2 to 20X) without modifying runtime applications

Ongoing and Future Work

- Apply to other applications
  - ♦Simplification and approximate collision detection [Yoon et al. 04]
  - ♦Shortest path computation, etc.
- Investigate optimality

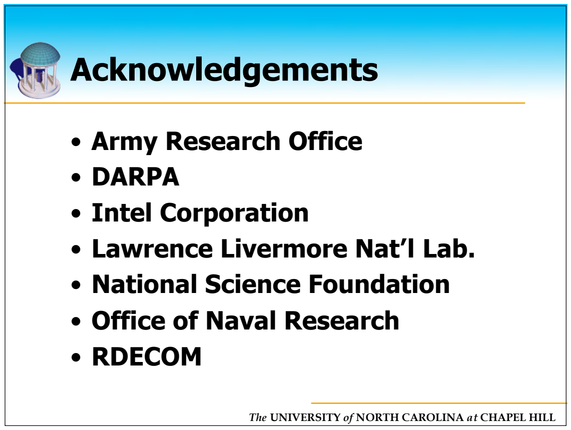
Ongoing and Future Work

- Cache-Oblivious Layouts of Bounding Volume Hierarchies [Yoon and Manocha 05]
  - ♦Tech. Report, University of North Carolina at Chapel Hill



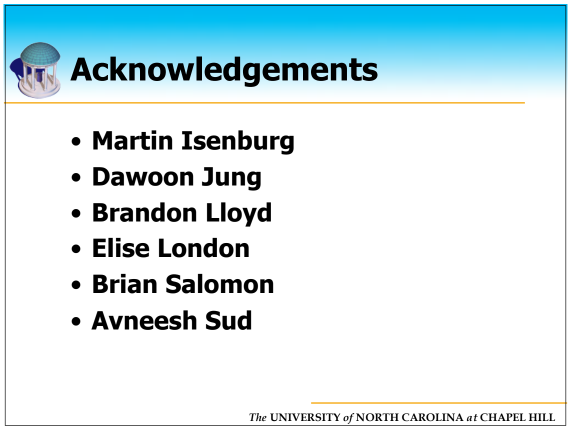
Acknowledgements

- Anonymous donor
  - ♦Power plant model
- Digital Michelangelo Project
  - ♦St. Matthew model at Stanford University
- LLNL ASCI VIEWS
  - ♦Isosurface model
- Newport news shipbuilding
  - ♦Double eagle tanker



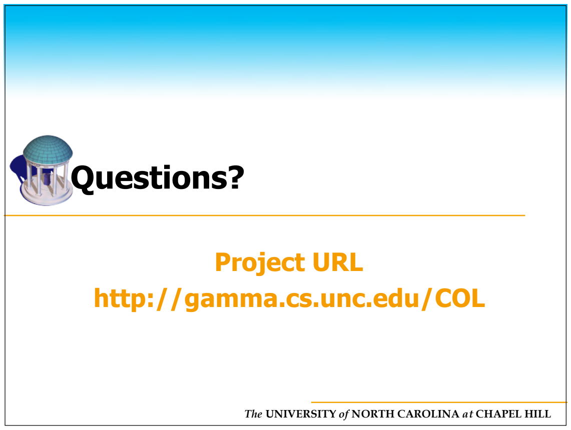
Acknowledgements

- Army Research Office
- DARPA
- Intel Corporation
- Lawrence Livermore Nat'l Lab.
- National Science Foundation
- Office of Naval Research
- RDECOM



Acknowledgements

- Martin Isenburg
- Dawoon Jung
- Brandon Lloyd
- Elise London
- Brian Salomon
- Avneesh Sud



Questions?

Project URL
http://gamma.cs.unc.edu/COL

Subdivided Shadow Maps

Technical Report TR05-024

Brandon Lloyd, Sung-eui Yoon, David Tuft, Dinesh Manocha
University of North Carolina at Chapel Hill

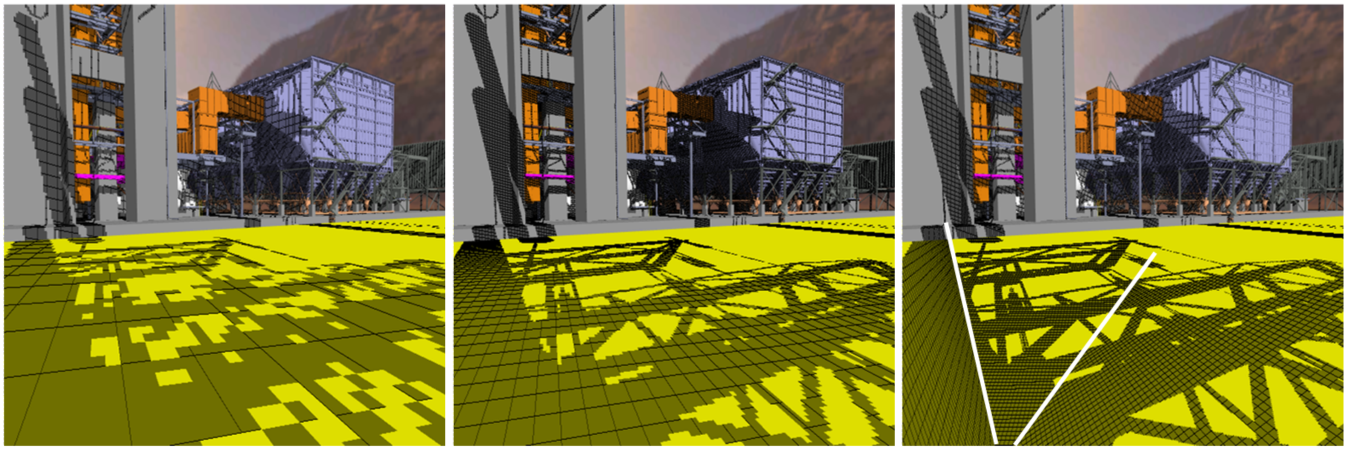


Figure 1: Comparison between trapezoidal shadow maps (TSMs) and subdivided shadow maps. a) TSM with 1K×1K shadow map, b) TSM with 4K×4K shadow map, and c) 1K×1K subdivided shadow map. This configuration with a small angle between the light and view directions is difficult for prior methods. Even with the largest shadow map that can be allocated on current hardware, TSMs are not able to match the quality of subdivided shadow maps for this view.

Abstract

We present a technique for reducing perspective aliasing error in shadow maps. From the viewpoint of the light, the scene is first split into subdivisions defined by the visible faces of the camera frustum. The frustum subdivisions may be further subdivided along their corresponding faces. We apply a separate shadow map warp to each resulting subdivision. This produces significantly less error than applying a single shadow map warp to the whole scene We layout the subdivisions in rectangular regions within a single shadow map, using the maximum error of each subdivision to assign larger regions to subdivisions with higher error. Our method runs well on commodity graphics hardware and is easy to integrate into existing shadow map systems. We are able to achieve interactive performance (8-25 fps) on a power plant model (12M triangles), a double eagle tanker model (82M triangles), and the St. Matthew model (370M triangles) running on a PC with a GeForce 7800 GTX. We observed significantly less aliasing compared to prior shadow map warping algorithms.

1 Introduction

Shadows are an important component of an interactive rendering system. They add realism and important visual cues. In this paper we restrict ourselves to hard shadows generated by directional light sources using shadow maps. For the directional light source, the standard shadow map algorithm proposed by Williams [1978] renders the scene with an orthographic projection from the light's view, and uses the resulting depth map to determine which surfaces lie in shadow. Shadow maps are a particularly attractive algorithm because they are easy to implement, they support a wide variety of geometry representations, and there exists wide support for shadow maps in current graphics hardware. The main drawback of shadow maps is aliasing at the shadow edges. Shadow map aliasing caused by the orientation of the surface onto which the shadow is projected is called *projective aliasing*. Aliasing due to limited shadow map resolution is called *perspective aliasing*. A number of algorithms have been proposed that address perspective aliasing by warping the shadow map so as to allocate more shadow map resolution near the viewer where the aliasing is the worst [Stamminger and Drettakis 2002; Wimmer et al. 2004; Martin and Tan 2004]. The warping is performed by applying a transformation to the scene before it is rendered into the shadow map. The warping reduces aliasing artifacts for most light/camera configurations. However when the angle between the light and view directions is small, all previous warping algorithms revert back to standard shadow maps, resulting in large perspective aliasing error.

Main Results: In this paper we present an algorithm to reduce perspective aliasing error by subdividing the scene and applying a separate warping function to each subdivision. Our algorithm uses two types of subdivisions:

- **Frustum subdivisions.** From the light's view, the algorithm first subdivides the eye view frustum into subdivisions corresponding to the visible frustum faces.
- **Face subdivisions.** The frustum subdivisions may be further subdivided along the length of their corresponding faces.

Frustum subdivisions dramatically reduce the error for configurations with small angles between the light and view

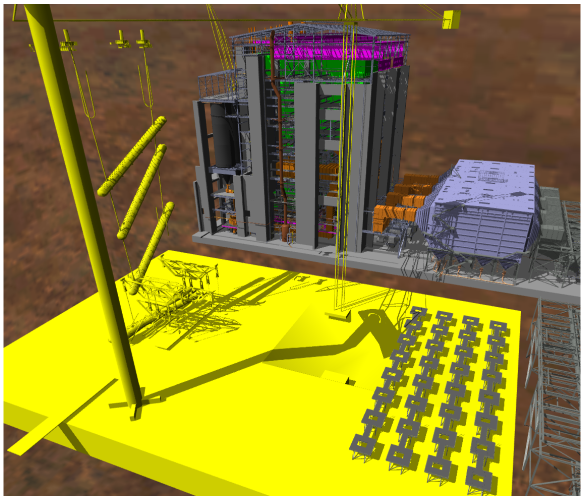


Figure 2: **Power plant model.** *This image shows a power plant model consisting of more than* 12 *million triangles. We are able to render the model with high quality shadows at* 8-20 *frames per second on a PC with GeForce 7800 GTX.*

directions. Face subdivisions reduce error for all configurations. By warping face subdivisions we more closely approximate the optimal warping function that would completely eliminate aliasing.

We formulate the maximum error in each subdivision. Using this information we can allocate to each subdivision a rectangular region in the shadow map. The regions are proportional in size to the maximum error of their corresponding subdivisions. Allocating space in the texture map in this way insures that the maximum error over the entire scene in minimized. Once the subdivisions have been rendered into their separate regions of the shadow map, a fragment program is used to render the final image.

Compared to previous shadow map warping algorithms, subdivided shadow maps provide a more even distribution of error over all light directions and lower error overall, with only a modest increase in rendering cost. We have integrated our algorithm with a rendering system for large models that runs on commodity hardware. With a GeForce 7800 GPU our system achieves interactive performance (8-25 fps) on a power plant model consisting of 12 million triangles, a double eagle tanker model consisting of 82 million triangles, and the St. Matthew model consisting of 372 million triangles. We found that the performance with the subdivided shadow map was only about twice as slow as with a standard shadow map. We were able to reduce the maximum aliasing error by a factor of up to 10 times.

The rest of this paper is organized as follows. In Section 2 we briefly discuss related work in shadow map computation. Section 3 provides the background for quantifying the perspective aliasing error and the parameterization of the warping functions that we use. The subdivision algorithm is described in Section 4. We discuss various implementation details and our results in Section 5. Finally, we conclude with some ideas for future work.

2 Previous Work

Many techniques have been proposed for shadow generation. In this section, we limit ourselves to shadow maps and some hybrid combinations with object-space techniques. Shadow maps were first introduced by Williams [1978]. Segal et al. [1992] later implemented them on standard graphics hardware. Many algorithms have been proposed to address the aliasing problems with shadow maps. Adaptive shadow maps [Fernando et al. 2001] use a hierarchy of small shadow maps to allocate resolution where it is required. Increased programmability of GPUs has facilitated implementations of adaptive shadow maps for hardware rendering [Lefohn et al. 2005], but performance can be slow. Instead of using a regular grid of shadow samples, irregular shadow maps [Aila and Laine 2004; Johnson et al. 2004] use a sample distribution that corresponds exactly to the image samples for the eye, thus avoid the aliasing problem altogether. However, irregular shadow maps are difficult to implement on current hardware. Shadow map warping was introduced with perspective shadow maps (PSMs) [Stamminger and Drettakis 2002]. PSMs use the camera's perspective transform to warp the shadow map. A singularity may arise with PSMs that requires special handling [Kozlov 2004]. Light-space perspective shadow maps (LSPSMs) [Wimmer et al. 2004] are a generalization PSMs that do not have the singularity problem because they use a perspective projection that is oriented perpendicular to the light direction. Trapezoidal shadow maps (TSMs) [Martin and Tan 2004] are very similar to LSPSMs, except that they use a different formulation for the perspective projection. Chong et al. [2004] use a general projective transform to ensure that there is a one-to-one correspondence between pixels in the image and the texels in the shadow map, but only for a small number of planes within the scene. Others have discussed using separate shadow maps for different parts of the scene [Tadamura et al. 1999; Forum 2003; Aldridge 2004].

Pure object-space shadow algorithms do not have aliasing problems. Some hybrid algorithms combine object-space techniques with shadow maps to reduce aliasing. McCool et al. [2000] construct shadow volumes from a shadow map. Sen et al. [2003] create a shadow map that more accurately represents shadow edges. Both of these techniques, while generating better looking shadow edges, may miss small features that cannot be represented in the shadow map. Chan and Durand [2004] use shadow maps to restrict shadow volume rendering to the shadow edges. Govindaraju et al. [2003] use shadow polygons for the most aliased areas and a shadow map everywhere else.

3 Shadow map warping

In this section we review perspective aliasing, we show how aliasing can be controlled by reparameterizing the shadow map, and we quantify the error. We also present our own parameterization of the warping function and derive the error equations. We follow the general outline of the excellent analysis provided by Wimmer et al. [2004], but our explanation is cast in somewhat different terms. Our notation also differs slightly.

3.1 Shadow map warping

We illustrate the concept of shadow map warping using the 2D configuration shown in Figure 3. A directional light source is positioned perpendicular to the view direction of a

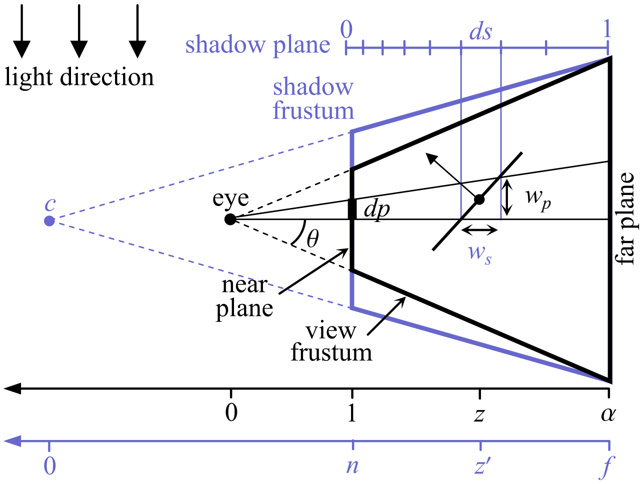


Figure 3: **Perspective aliasing and the shadow map warping parameterization.** *Perspective aliasing occurs when shadow texel beams are wider than pixel beams. The shadow map is warped using a perspective projection placed around the view frustum. This is a canonical view frustum with near plane at 1. The parameter α represents the ratio of the far to near plane distances.*

perspective camera. The shape of the view frustum is parameterized by α, the near to far plane distance ratio. This will permit us later to more easily understand how the aliasing errors change when we change the shape of the frustum.

Perspective aliasing. The notion of perspective aliasing with a shadow map can be understood intuitively in terms of the relative widths of shadow and pixel beams. In Figure 3 parallel shadow beams emanate from the texels of the shadow plane with widths w_s. Pixel beams likewise emanate from the pixels on the view plane. The widths of the pixel beams w_p increase with z. If shadow and pixel beams fall on a surface that is oriented with its normal half-way between the view and light directions, the size of the resulting footprints depends only on the relative widths of the beams, $m = w_s/w_p$. There is no projective aliasing in this case. When the shadow beams are wider than the pixel beams, i.e. $m > 1$, a shadow beam footprint is covered by multiple pixels beams. Thus the footprints of individual shadow texels can be clearly distinguished in the image and perspective aliasing occurs. Ideally we would like $w_s = w_p$ everywhere in the scene in order to avoid aliasing and to make the best use of the shadow map resolution.

Parameterization. As shown in Figure 3, we would like to reparameterize the shadow plane so that the widths of the shadow texel beams more closely match those of the pixel beams they intersect. Since texel spacing is inversely proportional to the derivative of the texture parameterization, the width of a shadow texel beam can be expressed as:

$$w_s = \frac{1}{r_s}\frac{dz}{ds},$$

where r_s is the resolution of the shadow map. The width of a pixel beam depends on the image pixel resolution r_p and is proportional to z.

$$w_p = z\frac{2\tan\theta}{r_p}.$$

To minimize the perspective aliasing error, m, we would like to have the shadow texel width also be proportional to z. In other words, we seek a parameterization of the shadow plane s whose derivative ds is proportional to $1/z$:

$$s = \int_0^1 ds = \int_1^z \frac{1}{z}dz = \ln z.$$

Thus the optimal parameterization of the shadow plane is a logarithmic function. The only non-linear transform that current graphics hardware provides is a perspective projection, which gives texel spacing proportional to $1/z^2$ (see Appendix). Therefore, the best we can do is to minimize the resolution mismatch error according to some optimality criterion.

The parameterization of the shadow plane with a perspective projection can be specified in a manner similar to that used for the camera. With near and far planes that coincide with those of the camera, the parameterization is given by:

$$s = \frac{1}{2} + \frac{f+n}{2(f-n)} + \frac{fn}{z'(f-n)}.$$

The position of the center of projection c is controlled by near plane distance n, which is a free parameter. When $n = 1$, c coincides with the eye and the warping effect is strongest. At $n = \infty$ the perspective projection becomes orthogonal and the texel spacing becomes uniform as in the standard shadow mapping algorithm. The fundamental difference between the recently proposed shadow map warping algorithms (PSM[Stamminger and Drettakis 2002], LSPSM[Wimmer et al. 2004], and TSM[Martin and Tan 2004]) is the choice of n. PSMs place n at 1. LSPSMs choose n such that the maximum error for the whole frustum is minimized. TSMs choose n such that a user specified portion at the front of the frustum is mapped to the 80% line on the shadow plane.

Quantifying error. The choice of the n parameter greatly affects the perspective aliasing error m. To compute m we first replace z' in the parameterization function s with $(z - 1 + n)$ and differentiate:

$$\frac{ds}{dz} = \frac{fn}{(z-1+n)^2(f-n)}. \qquad (1)$$

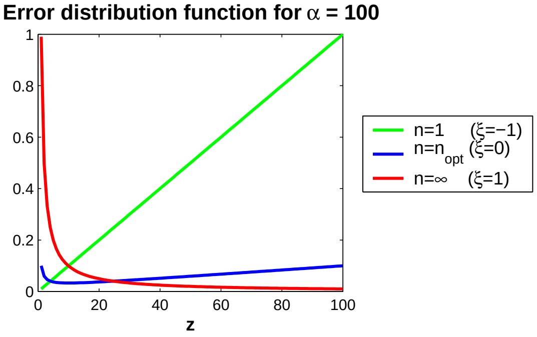


Figure 4: **Error distribution function.** *This graph shows g plotted for various values of n and corresponding ξ.*

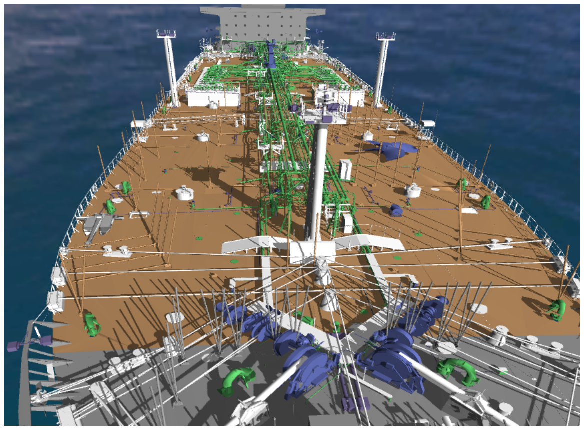


Figure 5: **Double eagle tanker model.** *This image shows a double eagle tanker model consisting of* 82 *million triangles. We are able to render the model at* 5-10 *frame per second with high shadow quality.*

If we substitute $f = n + \alpha - 1$ into Equation 1 we obtain for m:

$$m = \frac{w_s}{w_p} = \frac{r_p}{r_s} \frac{(\alpha - 1)}{2\tan\theta}\, g \qquad (2)$$

$$g = \frac{(z - 1 + n)^2}{zn(n + \alpha - 1)} \qquad (3)$$

The first term captures the effect of the relative resolutions of the image and the shadow map. The second term is characterized by the shape of the camera frustum. The last term, g is the error distribution function. Figure 3.1 shows g plotted over the range of the frustum $z \in [1, \alpha]$ for various values of n. The function g reaches its maximum value of 1 on the far and near planes with $n = 1$ and $n = \infty$ respectively. The least maximum value of g, $g_0 = \min_n \max_z(g) = 1/\sqrt{\alpha}$, occurs at the optimal n parameter for LSPSMs $n_{opt} = 1 + \sqrt{\alpha}$.

3.2 Reparameterizing the error distribution function

We reparameterize the error distribution function g to facilitate the error analysis for subdivision and to provide a more intuitive control over the maximum error. The n parameter is cumbersome because it is tied to scale of the view frustum and it is not directly related to the maximum error. We observe that as n decreases from infinity to 1, the maximum value of g over the whole frustum $\max_z(g)$ moves from 1 on the near plane to g_0 and back up to 1 again on the far plane. Based on this behavior we choose a new parameter ξ that controls the value and location of the maximum error. Over the range $\xi \in [-1, 0]$, we want $g_{\max}$ to linearly interpolate on the near plane from the maximum value of 1 to the minimum g_0 . For $\xi \in [0, 1]$, $g_{\max}$ should move back to 1 on the far plane.

$$\xi = \begin{cases} \frac{g_0 - g(n,1)}{1 - g_0} = \frac{\sqrt{\alpha} - n + 1}{n + \alpha - 1}, & n \geq n_{opt}, \\ \frac{g(n,\alpha) - g_0}{1 - g_0} = \frac{\sqrt{\alpha} - n + 1}{n\sqrt{\alpha}}, & n < n_{opt}. \end{cases}$$

We can then solve for n in terms of ξ:

$$n = \begin{cases} \frac{\sqrt{\alpha} + 1 - \xi(\alpha - 1)}{\xi + 1}, & \xi \leq 0, \\ \frac{\sqrt{\alpha} + 1}{\xi\sqrt{\alpha} + 1}, & \xi > 0. \end{cases} \qquad (4)$$

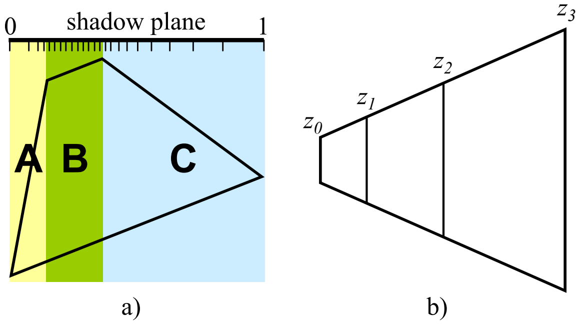


Figure 6: **Subdivisions.** ***a)*** Frustum subdivisions *divide the frustum into faces over which the texel spacing increases monotonically.* ***b)*** Face subdivision *split along a face to more closely approximate the optimal shadow map parameterization.*

Plugging these values for n back into Equations 2 and 3 with $z = \alpha$ for $\xi \leq 0$ and $z = 1$ for $\xi > 0$, we get an expression for the maximum error over the entire frustum in terms of ξ:

$$m_{\max} = \frac{r_p}{r_s} \frac{(\alpha - 1)}{2\tan\theta} \frac{1 + |\xi|(\sqrt{\alpha} - 1)}{\sqrt{\alpha}}. \qquad (5)$$

4 Subdivision Algorithm

Subdividing the scene and applying a separate warping function to each subdivision can drastically reduce perspective aliasing error. In this section we discuss two types of subdivision: frustum subdivision and face subdivision.

Frustum subdivision. Till now we have only considered the configuration with the light direction perpendicular to the view direction. Figure 6(a) shows an example of a more general configuration. Recall that to minimize perspective aliasing, the shadow texel beams should be smaller than any pixel beams that they intersect. When the light is behind the camera, the narrowest pixel beams are encountered on the faces of the view frustum that face the light. To avoid aliasing, the shadow texels must become narrower in region A with increasing s. The shadow texel spacing is constant in region B because the minimum pixel beam width is the size of the pixels on the image plane, which is also constant. In region C the texel spacing begins to increase again. Previous shadow map warping algorithms use a single monotonic warping function for the entire frustum. In at least one of the sections, the texels spacing will grow in the opposite direction of the warping function, leading to high errors. Therefore, other warping algorithms reduce the amount of warping as the angle between the light and view directions becomes smaller. When they are parallel, a standard orthogonal projection is used. Frustum subdivision avoids the light direction problem by subdividing along the frustum faces (edges in 2D). Since a warp is applied to each subdivision independently the error is more tightly bounded.

We allocate shadow texels to each frustum subdivision according to the maximum error in each subdivision. If subdivision i has a maximum error of e_i and there are r_s texels in the shadow map, then the number of shadow texels allocated for each subdivision is:

$$r_{si} = r_s \frac{e_i}{\Sigma_j e_j}. \qquad (6)$$

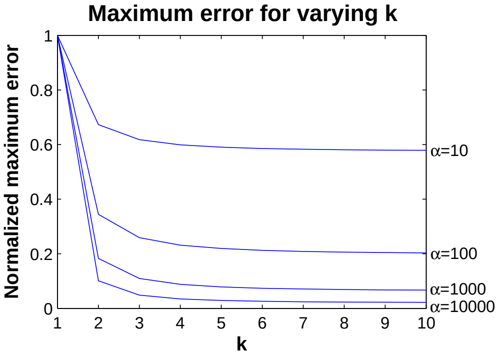


Figure 7: **Maximum error as a function of the number of subdivisions k for varying values of α.** *Each plot is normalized to show the overall shape. 2 or 3 subdivision bring the largest reduction in error.*

This allocation heuristic insures that the maximum error is spread evenly over all the subdivisions. If the same warping parameter ξ is used for all subdivisions, the heuristic also tends to allocate more texels to larger subdivisions. According to Equation 5, the maximum error with fixed ξ depends on α. For larger faces, α will be larger and the error greater.

Face subdivision. Once the frustum has been subdivided by its faces, further subdivisions along the faces create a better approximation of the optimal logarithmic transform, thus reducing error even more. If we subdivide along a face such that each subdivision is self-similar (as shown in Figure 6(b)), the maximum error in each subdivision will be the same. For each subdivision $i = 1, 2, ..., k$ we choose new near and far planes $[z_{i-1}, z_i]$:

$$z_i = \alpha^{i/k}.$$

With k subdivisions the ratio of the far to near plane distances, α_k, becomes $\alpha_k = \alpha_0^{1/k}$. According to texture allocation heuristic shown in Equation 6, each face subdivision i will be allotted $r_{si} = 1/k$ of the original shadow texels. Plugging in these values into Equation (5), the maximum error as a function of k becomes:

$$m_{\max} = k\frac{r_p}{r_s}\;\frac{(\alpha^{1/k}-1)}{2\tan\theta}\;\frac{(1+|\xi|(\sqrt{\alpha^{1/k}}-1))}{\sqrt{\alpha^{1/k}}}. \qquad (7)$$

Figure 7 shows how the maximum error changes with the number of face subdivisions k. Most of the error reductions come with only 2 or 3 subdivisions. This is important because we would like to keep the number of sections to minimum to avoid any extra rendering overhead.

4.1 Shadow warping and subdivision in 3D

So far we have only considered 2D scenes. Most of the machinery presented in the previous sections extends to 3D. As in 2D we first perform frustum subdivision. A frustum in general position, as shown in Figure 8(a), has up to 5 subdivisions defined by the faces, 4 sides and the near (or far) plane. If the edges shared by the sides and the near plane are shifted slightly to include the viewpoint, only 4 subdivision need to be used to cover the whole frustum. This introduces a small overlap between the subdivisions and only slightly more error. After frustum subdivision, we perform face subdivision. Next, each subdivision is transformed to a canonical space via a shear and a scale where the warping functions are computed (see Figure 8(b)). We use the same warping parameter, ξ, for each subdivision. Finally, the subdivisions are laid out in the shadow map (see Figure 8(c)).

Error in the x direction. There are two major differences with subdivided shadow maps in 2D and 3D. The first difference is that there is an added dimension in the shadow map, t, which corresponds to the x direction in Figure 3. The perspective projection applied to z also affects x. The error in x is optimal when $\xi = -1$, in which case the frustum of the perspective projection exactly matches that of the camera (as in PSMs). The error in z, however, is at its maximum with $\xi = -1$. Unfortunately, it is not possible to minimize the error in both the x and z directions at the same time.

We can follow a similar analysis for the x to obtain the maximum x error for a subdivision. We start with the width of a shadow texel beam in the t direction:

$$\begin{aligned} w_t &= \frac{2\alpha\tan\theta}{r_t}\;\frac{z'}{f} \\ &= \frac{2\alpha\tan\theta}{r_t}\;\frac{(z+n-1)}{(n+\alpha-1)}. \end{aligned}$$

The error function in x is then given by:

$$m_x \;=\; \frac{w_t}{w_p} = \frac{r_p}{r_t}\;\alpha\;\frac{(n+z-1)}{z(n+\alpha-1)}. \qquad (8)$$

The maximum x error always occurs on the near plane. Plugging in $z = 1$ and the expressions for n in Equation 4 we obtain for the maximum error in x:

$$m_{x\max} = \frac{r_p}{r_t}\;\alpha\;\begin{cases} \frac{1-\xi(\sqrt{\alpha}-1)}{\sqrt{\alpha}}, & \xi \le 0, \\ \frac{1}{\sqrt{\alpha}(1+\xi(\sqrt{\alpha}-1))}, & \xi > 0. \end{cases} \qquad (9)$$

Subdivision layout. The second major difference has to do with how the sections are laid out in the shadow map. We take advantage of the fact that the errors of each subdivision are correlated when the same warping parameter is used for all subdivisions. The maximum x errors is the same for subdivisions from opposite sides of the frustum (AC and BD) if the faces have been subdivided. For the maximum z error there is a trade off between opposite sides of the frustum. If the error is relatively high for one subdivision, it be low for its opposite subdivision. Based on these observations, we first split the shadow map in the t direction according to the maximum x error of the AC and BD pairs. Then we independently split in the s direction for subdivisions arising from each face pair according to maximum z error, e.g. between A and C.

5 Results

In this section we describe our implementation of subdivided shadow maps and highlight their performance on large models compared to other shadow map methods.

5.1 Implementation

We have implemented our algorithm on 2.4GHz Pentium-IV PC with 1GB RAM and a GeForce 7800 GTX with 256MB of video memory.

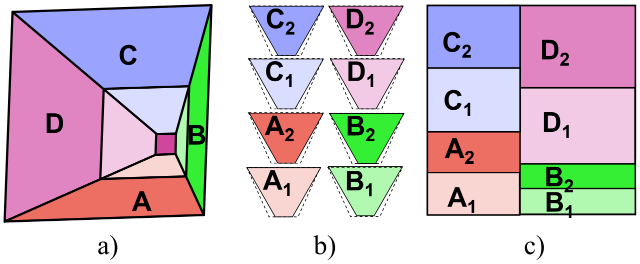


Figure 8: **Basic algorithm.** ***a)*** *Frustum and face subdivisions performed on faces of the eye view frustum as seen from the light.* ***b)*** *Each subdivision transformed to a canonical space and fitted with a warp (dashed frustum).* ***c)*** *Texture space layout according the maximum error in each subdivision.*

Model	Vert. (M)	Tri. (M)	Obj.
Power plant	11	12.2	1200
Double eagle tanker	77	82	3346
St. Matthew	186	372	1

Table 1: **Benchmark models:** *Model complexity of benchmark models is shown.*

Benchmark models. We tested out algorithm with three complex models, a coal-fired power plant composed of more than 12 million polygons and 1200 objects (Fig. 2), a double eagle tanker model consisting of 82 million polygon and 3346 objects (Fig. 5), the St. Matthew model consisting of a single 372 million polygon and single object (Fig. 11). The details of these models are shown in Table 1. We generated paths in each of our test models and used them to test the performance of our algorithm. These paths of the power plant and St. Matthew model are shown in the accompanying video.

Fragment program. We store the shadow maps for each subdivision in the same texture. In the fragment program we must be able to choose which shadow map to use at any pixel. We use the method described by Aldridge [2004]. We define planes parallel to the light through the edges of the visible faces. The normals are oriented facing left in the light's view. The dot product of a vertex with each of the edge planes is stored in separate channels of a texture coordinate and interpolated over the scene. For edges not visible to the light we set the dot product to 0. The following code will set to 1 only the channel corresponding to the face in the which a fragment lies, while setting all other channels to 0.

```
faceSelect = sign(dotProducts);
faceSelect = saturate(faceSelect * saturate(-faceSelect.yzwx));
```

Similarly we define split planes parallel to the light that pass through the edge induced by a face subdivision and track the dot product of a vertex with the split planes.

```
splitSelect = splitDotProducts >= 0;
```

The `faceSelect` and `splitSelect` are then combined to select the set of texture coordinate corresponding to the subdivision in which the fragment lies:

```
select0 = faceSelect * splitSelect;
select1 = faceSelect * (1.0 - splitSelect);
texCoord = select0.r * texCoord0 + select1.r * texCoord4 +
           select0.g * texCoord1 + select1.g * texCoord5 +
           select0.b * texCoord2 + select1.b * texCoord6 +
           select0.a * texCoord3 + select1.a * texCoord7;
```

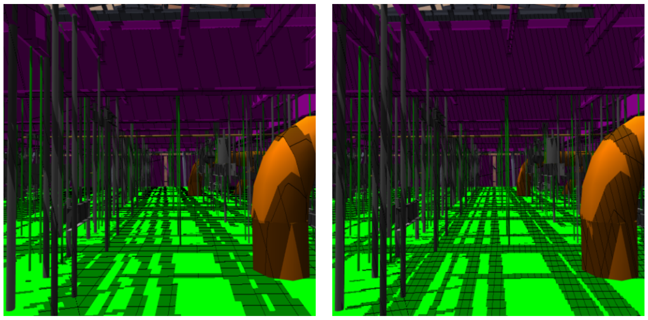


Figure 9: **Subdivided shadow maps.** *a) without and b) with face subdivision. For this scene with $\alpha = 100$, face subdivision leads to nearly* 5 *times improvement in maximum error.*

Model	St. Matthew	Power plant
SSM	1.16	1.67
SSM*	1.25	2.48

Table 2: **Overdraw Factors.** *Overdraw factors of the subdivided shadow maps with and without subdivisions (SSM and SSM*) are shown. Overdraw factors of the power plant model are much higher than those of St. Matthew since size and irregularity of objects of the power plant is much higher.*

Integration. Our method is simple to integrate with code that already uses shadow maps. Using functions we provide, the user first queries in which sections an object falls. In the subdivided shadow map render function we loop over each subdivision, setting up the view parameters and invoking a user supplied render callback. We supply the current subdivision as a parameter to the callback so that the user can render the objects which fall in this subdivisions. In this way our code does not need to know the details of the rest of the rendering system and the system does not need to know about how the shadow maps are handled.

Interactive shadow generation on complex models. To highlight the performance and high quality of our shadow maps on complex and massive models, we integrate our algorithm with out-of-core view-dependent rendering system [Yoon et al. 2004]. The rendering system uses a set of clusters decomposed from an input model as a scene representation of the model and computes progressive meshes to provide smooth level-of-detail transitions for each cluster at runtime. We also take advantage of visibility culling techniques [Govindaraju et al. 2003] to effectively cull away portion of the mesh that are not visible to the eye or the light.

5.2 Analysis and Limitation

For many of our comparisons we chose TSMs over PSMs and LSPSMs because TSMs gave slightly better quality.

The size of the objects in the scene has a great impact on the rendering time for the shadow map. Since each subdivision must be rendered into the shadow map separately,

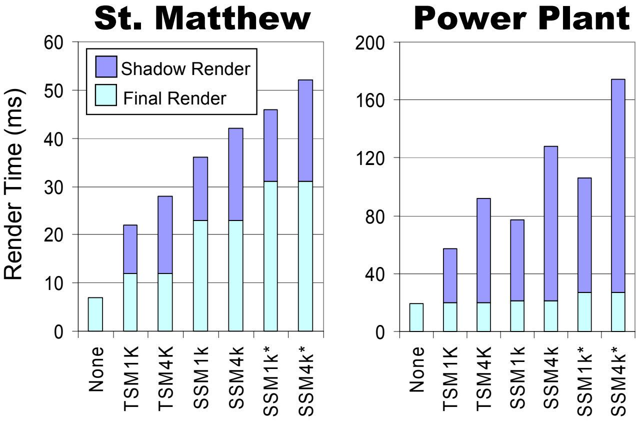


Figure 10: *Timing comparison between trapezoidal shadow maps (TSM) and subdivided shadow maps with and without face subdivisions (SSM* and SSM) at 1K×1K and 4K×4K resolution.*

objects that fall into more than one subdivision will be rendered multiple times. If a scene is decomposed into sub-objects that are sufficiently smaller than the subdivisions, most sub-objects will be rendered only once. Therefore, the rendering cost of transforming scene geometry will not increase much. The subdivided shadow map has the same resolution as a standard shadow map, so the fill-rate consumption will also about the same. But if the objects are large, they will span multiple subdivisions, hurting performance. We measure excessive rendering with an *overdraw factor*:

$$\text{overdraw} = \frac{\text{number of object renderings}}{\text{number of objects}}$$

The graph in Figure 10 shows that when more subdivisions are used, the shadow map rendering time for the power plant increases much more than for the statue. The reason for this is that the power plant has much more irregular geometry resulting in clusters that are larger that those of the statue. This leads to much higher overdraw factors (see Table 2).

The image rendering time for subdivided shadow maps is higher than for TSMs. This is due largely to the use of a more complicated fragment program for subdivided shadow maps. The fragment program that supports a face subdivision is even more complicated, leading to even longer render times. If the overdraw factor is low, the total frame time for subdivided shadow maps should be roughly twice that for a shadow warping algorithm. However, the quality may be much more than twice. Figure 1 shows a comparison of subdivided shadow maps with TSMs. For this view even the largest shadow map we could allocate, 4K×4K, was not sufficient to match the quality that we achieved with a 1K×1K subdivided shadow map.

The error reduction with subdivided shadow maps will be most dramatic when the angle between the light and view directions is small because other shadow warping algorithms do not handle this case well. When the light and view directions are perpendicular in the optimal configuration, frustum subdivision does not improve the quality much over the other shadow warping algorithms. A single set of face subdivisions, however, reduces the error quite a bit. Figure 9 SSMs with and without face subdivision.

Often the improvement in shadow quality from using face subdivision does not appear to be as good as predicted by Equation 7. The equation predicts the maximum over an entire subdivision. If there are no surfaces where the error is at its maximum, the reduction of error will not be visible. The error is reduced in other parts of the scene as well, but not as much.

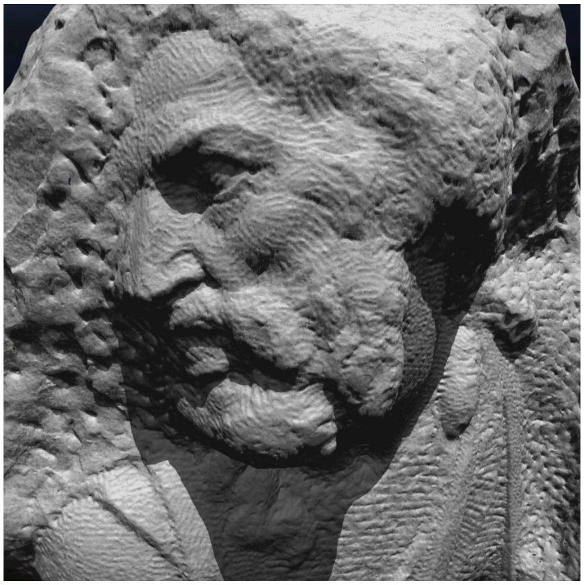


Figure 11: **St. Matthew model with shadows.** *This image shows St. Matthew model consisting of* 372 *million triangles. We are able to render the model with high quality shadows at* 15-25 *frames per second on commodity hardware with a GeForce 7800 GTX GPU.*

One problem with subdivided shadow maps is that the texels of the shadow map can be excessively sheared. The shear comes from the transformation of the subdivisions to the canonical space for warping. The texel shearing can be alleviated somewhat by first rotating the subdivision before shearing. This introduces more error but may yield more pleasing results.

Conclusion and Future Work

We have presented a shadow map algorithm for reducing perspective aliasing by applying separate warp functions to subdivisions of the scene individually. The algorithm can reduce the maximum aliasing by a factor of 10 or more while running only about twice as slow as other shadow map warping algorithms.

We would like to extend our technique to be used with point lights. The analysis is a bit more involved for point lights because now shadow texel beams change width with distance. Cube maps can be seen as a form of subdivision. We conjecture that warping the faces of the cube maps could lead to higher quality shadows.

References

Aila, T., and Laine, S. 2004. Alias-free shadow maps. In *Proceedings of Eurographics Symposium on Rendering 2004*, Eurographics Association, 161–166.

Aldridge, G. 2004. Generalized trapezoidal shadow mapping for infinite directional lighting. *http://legion.gibbering.net/projectx/paper/shadow%20mapping/*.

Chan, E., and Durand, F. 2004. An efficient hybrid shadow rendering algorithm. In *Proceedings of the Eurographics Symposium on Rendering*, Eurographics Association, 185–195.

Chong, H., and Gortler, S. 2004. A lixel for every pixel. In *Proceedings of the Eurographics Symposium on Rendering*, Eurographics Association, 167–172.

Fernando, R., Fernandez, S., Bala, K., and Greenberg, D. 2001. Adaptive shadow maps. In *Proceedings of ACM SIGGRAPH 2001*, 387–390.

Forum. 2003. Shadow mapping in large environments. *http://www.gamedev.net/community/forums/topic.asp?topic_id=179941*.

Govindaraju, N., Lloyd, B., Yoon, S., Sud, A., and Manocha, D. 2003. Interactive shadow generation in complex environments. *Proc. of ACM SIGGRAPH/ACM Trans. on Graphics 22*, 3, 501–510.

Johnson, G., Mark, W., and Burns, C. 2004. The irregular z-buffer and its application to shadow mapping. In *The University of Texas at Austin, Department of Computer Sciences. Technical Report TR-04-09*.

Kozlov, S. 2004. *Perspective Shadow Maps: Care and Feeding.* Addison-Wesley, 214–244.

Lefohn, A., Sengupta, S., Kniss, J. M., Strzodka, R., and Owens, J. D. 2005. Dynamic adaptive shadow maps on graphics hardware. In *ACM SIGGRAPH 2005 Conference Abstracts and Applications*.

Martin, T., and Tan, T.-S. 2004. Anti-aliasing and continuity with trapezoidal shadow maps. In *Proceedings of the Eurographics Symposium on Rendering*, Eurographics Association, 153–160.

McCool, M. 2000. Shadow volume reconstruction from depth maps. *ACM Trans. on Graphics 19*, 1, 1–26.

Segal, M., Korobkin, C., van Widenfelt, R., Foran, J., and Haeberli, P. 1992. Fast shadows and lighting effects using texture mapping. In *Computer Graphics (SIGGRAPH '92 Proceedings)*, vol. 26, 249–252.

Sen, P., Cammarano, M., and Hanrahan, P. 2003. Shadow silhouette maps. *ACM Trans. on Graphics (Proc. of ACM SIGGRAPH) 22*.

Stamminger, M., and Drettakis, G. 2002. Perspective shadow maps. In *Proceedings of ACM SIGGRAPH 2002*, 557–562.

Tadamura, K., Qin, X., Jiao, G., and Nakamae, E. 1999. Rendering optimal solar shadows using plural sunlight depth buffers. In *Computer Graphics International 1999*, 166.

Williams, L. 1978. Casting curved shadows on curved surfaces. In *Computer Graphics (SIGGRAPH '78 Proceedings)*, vol. 12, 270–274.

Wimmer, M., Scherzer, D., and Purgathofer, W. 2004. Light space perspective shadow maps. In *Proceedings of the Eurographics Symposium on Rendering*, Eurographics Association, 143–152.

Yoon, S.-E., Salomon, B., Gayle, R., and Manocha, D. 2004. Quick-VDR: Interactive View-dependent Rendering of Massive Models. *IEEE Visualization*, 131–138.

Appendix

A parameterization s using a general projective transform on z is given by:

$$s = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} z \\ 1 \end{bmatrix} = \begin{bmatrix} az+b \\ cz+d \end{bmatrix}$$

After the perspective divide we have a function:

$$\begin{aligned} s &= \frac{az+b}{cz+d} \\ \frac{ds}{dz} &= \frac{ad-bc}{(cz+d)^2} \end{aligned}$$

Thus the texel spacing in world space, dz/ds, generated by a perspective projection is proportional to z^2.

R-LODs: Fast LOD-based Ray Tracing of Massive Models

Sung-Eui Yoon[1] Christian Lauterbach[2] Dinesh Manocha[2]
[1]Lawrence Livermore National Laboratory [2]University of North Carolina at Chapel Hill
{sungeui,cl,dm}@cs.unc.edu
http://gamma.cs.unc.edu/RAY

Abstract

We present a novel LOD (level-of-detail) algorithm to accelerate ray tracing of massive models. Our approach computes drastic simplifications of the model and the LODs are well integrated with the kd-tree data structure. We introduce a simple and efficient LOD metric to bound the error for primary and secondary rays. The LOD representation has small runtime overhead and our algorithm can be combined with ray coherence techniques and cache-coherent layouts to improve the performance. In practice, the use of LODs can alleviate aliasing artifacts and improve memory coherence. We implement our algorithm on both 32*bit and* 64*bit machines and able to achieve up to* 2–20 *times improvement in frame rate of rendering models consisting of tens or hundreds of millions of triangles with little loss in image quality.*

1 Introduction

In recent years, there has been a renewed interest in real-time ray tracing for interactive applications. This is due to many factors: firstly, processor speed has continued to rise at exponential rates as predicted by Moore's Law and is approaching the raw computational power needed for interactive ray tracing. Secondly, ray tracing algorithms can be parallelized on shared memory and distributed memory systems. Therefore, the current hardware trend towards desktop systems with multi-core CPUs and programmable GPUs can be used to accelerate ray tracing. Finally, recent algorithmic improvements that exploit ray coherence can achieve a significant improvement in rendering time [22, 31].

Our goal is to perform interactive ray tracing of massive models consisting of tens or hundreds of millions of triangles on current desktop systems. Such gigabyte-sized models are the result of advances in model acquisition, computer-aided design (CAD), and simulation technologies and their complexity makes interactive visualization and walk-throughs a challenging task. In the context of rendering massive models, ray tracing has an important property: its asymptotic performance is logarithmic in the number of primitives for a given resolution. This is due to the use of hierarchical data structures such as bounding volume hierarchies or kd-trees. The asymptotic complexity makes ray tracing an attractive choice, especially for rendering of massive models.

The logarithmic growth, however, continues only as long as the system has sufficient main memory to contain the entire model and hierarchical data structures. As models grow much larger, the size of the hierarchical structure also increases linearly and the underlying ray tracer performs its computations in an out-of-core manner, slowing down drastically. A major trend in computing hardware has been the increasing gap between processor speed and memory speed. Moreover, disk I/O accesses are in general more than three orders of magnitude slower than main memory accesses. Because of these gaps, hardware advances are not expected to provide an efficient solution to the problem of ray tracing massive models.

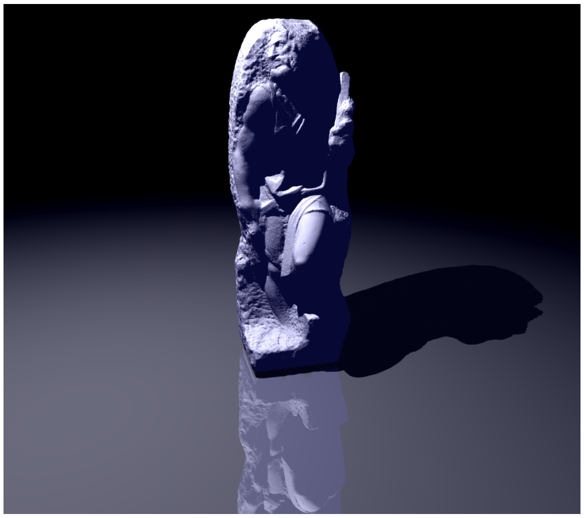


Figure 1. St. Matthew Model: *We use our LOD-based algorithm to accelerate ray tracing of the St. Matthew model with shadows and reflections. We render the* $128M$ *triangle model at* 512×512 *resolution with* 2×2 *anti-aliasing and pixels-of-error (PoE)* $= 4$*. We are able to achieve* $2-3$ *frames per second on two dual-core Xeon processors with 4GB of memory. We observe a* $2-20$ *times increase in the frame rate due to R-LODs with very little loss in image quality.*

Main Contributions: We present a new algorithm to accelerate ray tracing of massive models using geometric levels-of-detail (LODs). Our approach computes simple and drastic simplifications, called *R-LODs*, of the polygonal model. The R-LODs have a compact representation and are tightly integrated with the kd-tree. We present a simple and efficient LOD error metric to bound the error for primary and secondary rays. Additionally, we use techniques based on ray coherence and cache-oblivious layouts to improve the performance of our LOD based ray tracing algorithm. R-LODs also alleviate the temporal aliasing that can arise during rendering of highly tessellated models.

We have implemented and tested our system on two machines running Windows XP 32-bit and 64-bit with two dual-core Xeon CPUs and have evaluated its performance on different kinds of models with $20-128M$ triangles. The performance gain of our LOD based ray tracer is proportional to the reduction in the working set size and the number of intersection tests. The frame rate improvement varies from 2 times on models with small working set size to almost $20-50$ times on models with very large working set size.

Our ray tracing algorithm offers the following *benefits*:

1. **Simplicity:** R-LODs are very easy to implement and

their representation has small runtime overhead. Our algorithm maintains the simplicity, coherence, and performance of the kd-tree data structure.

2. **Interactivity:** The LOD based ray tracer provides a framework for interactive ray tracing due to the fact that we can trade off image quality for improved frame rate.
3. **Front size:** R-LODs reduce the size of the front traversed in the kd-tree. This results in fewer ray intersection tests and decreases the size of the working set.
4. **Coherence:** R-LODs make memory accesses more coherent and reduce the number of L1/L2 cache misses and page faults. Furthermore, they can also improve the performance of ray coherence techniques.
5. **Generality:** Our algorithm is applicable to a wide variety of polygonal models, including scanned and CAD models.

Organization: The rest of the paper is organized in the following manner: section 2 gives a brief summary of prior work in interactive rendering. We give an overview of our approach in Section 3 and present the R-LOD representation and computation algorithm in Section 4. Section 5 shows acceleration techniques based on cache-coherent layouts and ray coherence. We describe the implementation of our ray tracer and analyze its performance on different models in Section 6. Finally, section 7 compares our algorithm with other approaches.

2 Related work

In this section, we give a short overview of interactive ray tracing and the use of LODs for interactive rendering.

2.1 Interactive Ray Tracing

Ray tracing was introduced by Appel [3] and Whitted [36] and is a very well studied field. In this section, we just briefly survey some recent techniques used to accelerate ray tracing, but a detailed description is available in [27]. At a broad level, we classify prior approaches into four categories:

Exploiting ray coherence: The underlying idea here is not to trace each ray by itself, but to utilize the fact that neighboring rays exhibit spatial coherence. Earlier attempts to exploit this concept were beam tracing [11], pencil tracing [26] and cone tracing [2]. More recently, Wald *et al.* [31] group rays into bundles and use them to accelerate traversal and intersection with primitives for all rays simultaneously by taking advantage of SIMD instructions. Reshetov *et al.* [22] propose an algorithm to integrate beam tracing with the kd-tree spatial structure and were able to further exploit ray coherence.

Hardware acceleration: Another trend has been to use hardware support to accelerate ray tracing. Purcell *et al.* [21] show that ray tracing could be adapted to the streaming model of current programmable GPUs, which are mainly designed for rasterization. Schmittler *et al.* [25] and Woop *et al.* [38] present prototypes for a complete and programmable ray tracing hardware architecture to run at interactive rates.

Parallel computing: Ray tracing is easily parallelizable due to the fact that all rays can be traced independently. Parker *et al.* [19], DeMarle *at al.* [7], and Dietrich *et al.* [8] describe an interactive ray tracer for rendering large scientific or CAD datasets running on shared memory or distributed architectures. Wald *et al.* [34] built a ray tracer to run on clusters of commodity hardware machines and were able to achieve interactive frame rates for large architectural and CAD models. Both of these systems are mainly intended for models that could be kept in the main memory of a shared memory system or of PCs used in the cluster.

Large datasets: Many algorithms have been presented to improve the performance of ray tracing on large datasets [7, 10, 20, 32]. Our approach is complimentary to these methods and can be combined with them to further improve the performance.

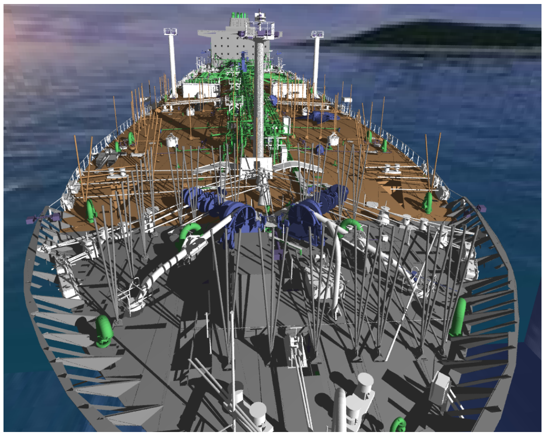


Figure 2. **Double Eagle Tanker**: *The deck of the Double Eagle tanker with shadows is shown using ray tracing. We are able to achieve* 1*-3fps at* 512 *by* 512 *image resolution with* $2x2$ *super-sampling and PoE* $= 4$ *on a dual Xeon workstation. In this model, the working set of the ray tracer is low and we are able to achieve up to* 2 *times improvement in the frame rate.*

2.2 Interactive Rendering using LODs and Out-of-Core Techniques

LODs have been widely used to accelerate rasterization of large polygonal datasets [16]. At a broad level, prior algorithms can be classified into static LODs, view-dependent simplification, image-based representations and hybrid combinations of geometric and image-based representations. Out-of-core algorithms are an active area in computer graphics and visualization with the goal to efficiently handle large datasets [4]. LOD algorithms can be combined with out-of-core techniques to rasterize large polygonal datasets composed of tens or hundreds of millions of polygons at interactive rates on commodity PCs [23, 6, 41, 9].

LOD-based based algorithms can also be applied to accelerate ray tracing. Christensen *et al.* [5] introduce a LOD approach for an offline ray tracer based on ray differentials [12]. Wand and Straßer [35] propose an algorithm for multiresolution ray tracing of point-sampled geometry based on ray-differentials. Another approach is to integrate the LODs into the hierarchical structure [37]. Recently, Stoll *et al.* [28] proposed a novel architecture for dynamic multiresolution ray tracing. They proposed a watertight multiresolution method by interpolating between discrete LODs for each ray. Their discrete LODs are computed from choosing proper tessellation levels for subdivision meshes. Also, efficient algorithms based on depth images can be used to accelerate ray tracing [15, 1].

3 Overview

In this section, we discuss many issues that govern the performance of ray tracing and give an overview of our approach.

3.1 Ray tracing of massive models

In this paper, we restrict ourselves to triangulated models, though our approach can also be extended to other primitives such as point clouds. All efficient ray tracers employ hierarchical data structures to avoid testing each ray with every primitive. We use the kd-tree, which is a special case of the general BSP tree and has recently become a popular choice due to its simplicity and performance [10, 27]. Each node of the kd-tree represents one subdivision of the parent's space and contains information about the axis-aligned plane used for the split as well as pointers to its child nodes. We use the optimized representation proposed by Wald *et al.* [30] and extend

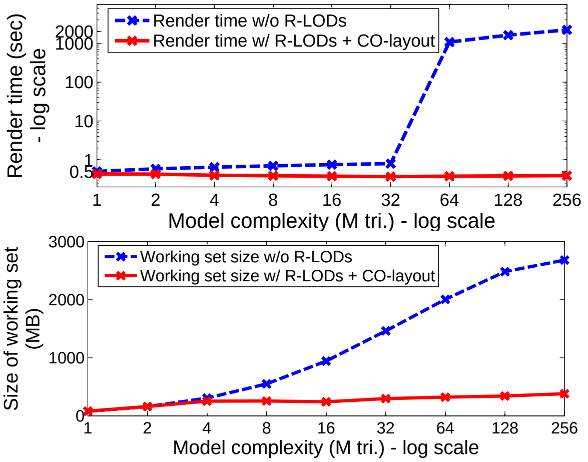


Figure 3. **Performance of Ray Tracing**: *We precompute simplified versions of the St. Matthew model and ray trace each simplification separately from the same viewpoint. We measure the frame time and working set size of the ray-tracer, with and without R-LODs by using different simplified models on a* 64*-bit machine with* 2*GB RAM. Notice the big jump in frame time for ray tracing without R-LODs, as the working set of ray tracing with massive models exceeds the available RAM. On the other hand, our R-LOD based ray tracing combined with cache-oblivious layouts (CO-layout) achieves near-constant performance in terms of frame rate and the working size.*

it to efficiently handle LODs.

Out-of-core ray tracing: Ray tracers taking advantage of hierarchical data structures should exhibit a logarithmic growth rate as a function of the model complexity [31]. We measured the performance of a coherent ray tracer during rendering different simplification levels of the St. Matthew model, as shown in Fig. 3. Our experiment indicates that ray tracing performance increases as a logarithmic function of model complexity as long as the kd-tree and primitives of a model can fit in the main memory. However, once the model size and the working set size of the kd-tree exceeds the available main memory of the machine, the disk I/O significantly affects the performance of the ray tracer.

Ray coherence: Recent approaches that exploit spatial and ray coherence decrease the number of memory accesses and therefore also the number of disk accesses for large models [31, 22]. These algorithms perform traversals and intersections for multiple spatially-coherent rays in a group at the same time. In general, rays in a group exhibit higher coherence at the higher levels of the kd-tree (that usually are in main memory) because each ray in the group is likely to follow same path in the tree as other rays. However, accesses to the nodes deeper in the tree are incoherent and, thus, result in disk cache misses, especially when dealing with massive models, since bounding box of those nodes become smaller compared to width of the ray group. Therefore, in order to accelerate out-of-core ray tracing, we need to reduce the number of accesses made to the nodes deeper in the tree.

3.2 Our Approach

We mainly address the problem of ray tracing massive models. If models have high depth complexity, current traversal algorithms based on kd-trees can efficiently handle such kinds of models. In this case, the working set size is proportional to the number of visible primitives from the primary and secondary rays. Therefore, we primarily deal with the problem of fast ray tracing when the number of visible primitives is high.

We assume that each ray or ray bundle is represented by a pyramidal beam or frustum. As described in the multi-level ray tracing of Reshetov *et al.* [22], during traversal the frustum is checked for intersection with the bounding box of the current kd-tree node by using an inverse frustum culling approach. This results in two interesting cases:

1. Models with large primitives: If the bounding box of the node is larger than the frustum, it is likely that the node intersects with the whole beam, i.e. we can exploit spatial coherence. Typically, architectural models or CAD models result in such cases whenever the model is coarsely tessellated, has large planar primitives or is viewed at close range.

2. Highly tessellated models: In this case, the bounding box is much smaller than the frustum. This implies that the beam needs to be split into smaller sub-beams. However, if the beam represents just one ray, then further subdivision is not possible, even though the sub-tree represented by the node has a high number of descendants and, thus, there is high local geometric complexity. Therefore, ray coherence approaches like multi-level ray tracing and ray packet tracing fall back to normal ray tracing and may not offer much benefit. For example, consider ray tracing a St.Matthew model consisting of 128M triangles at a resolution of 1024^2 primary rays. Assume that every ray hits the model and half of the model's triangles are visible to the eye. In this case, fewer than 1% of the actual triangles are hit by one of the rays. Moreover, each of these triangles is sampled as a representative of several triangles in the subtree. This has two consequences: first, the memory accesses may be *incoherent* because each triangle may lie in a different part of memory. Secondly, *temporal aliasing* can occur between frames since it is likely that a different triangle will be chosen in successive frames.

Our novel LOD-based ray tracing algorithm handles this second case by choosing our precomputed R-LOD representation when traversal determines that a LOD metric is satisfied. This means that traversal can stop before reaching the deep levels of the tree, reducing the number of incoherent accesses and the size of the working set, while maintaining ray coherence so that related techniques still work well. As a result, we obtain significant improvements in rendering speed.

4 LOD Based Ray Tracing

In this section we present the R-LODs that are used to accelerate ray tracing. We first describe our R-LOD representation and the modified traversal algorithm. Then we present our LOD error metric and the R-LOD construction algorithm.

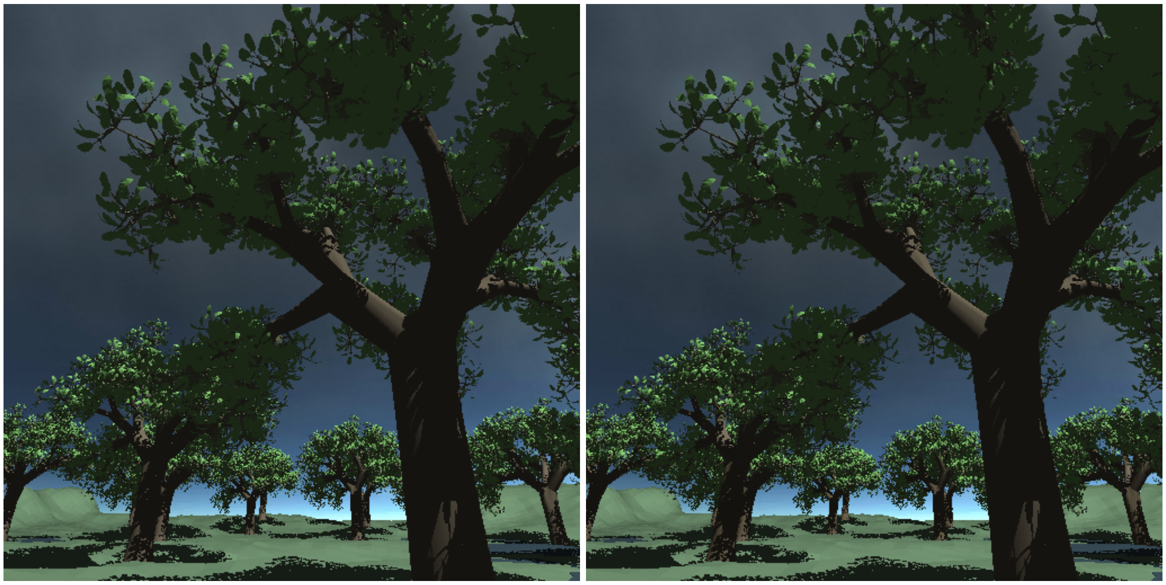
4.1 R-LOD Representation for Ray Tracing

Our goal for interactive ray tracing is to design a LOD representation that retains the benefits of kd-tree based acceleration algorithms, i.e. simplicity, efficiency and low runtime overhead.

A R-LOD consists of a plane with material attributes (e.g. color), which is a drastic simplification of the descendant triangles contained in an inner node of the kd-tree, as shown in Fig. 5. Each R-LOD is also associated with a surface deviation error which is used to quantify the projected screen-space error at runtime.

Let us assume that the original tree has height h, where $h \approx \log_2(n)$, and n is the number of triangles in the original model. The R-LOD associated with a kd-tree node at height k is a simplification into a plane of the 2^k descendant triangles. Our choice to use such a representation is motivated by the following goals:

Simple and efficient LOD representation: Current ray object intersection algorithms based on the kd-tree are highly optimized for interactive ray tracing. We use simple representations for LODs to minimize storage and traversal overhead. Each R-LOD adds 4 bytes to an inner node of the kd-tree. We also use a simple and fast LOD selection algorithm to reduce the traversal overhead.



(a) No LOD

(b) PoE $= 4$

Figure 4. Forest Model: *We render the forest model consisting of* 32 *million triangles with shadow rays using PoE* $= 0$ *and PoE* $= 4$*. The image resolutions are* 512×512 *without anti-aliasing to highlight image quality differences. We are able to render the model given the viewpoint at the* 1.6 *frames per second and achieve* 5 *times improvement by using R-LODs.*

Drastic model simplification: The computational workload of ray tracing is a logarithmic function of the model complexity. If the model size is reduced by a factor of m, the tree traversal overhead reduces by only $\log(m)$. As a result, m has to be a significant number, say 2^3or 2^4.

High quality rendering: Ray tracing is primarily used to generate high-quality rendering. Since the R-LODs are a drastic simplification of the original model, their use can result in visual artifacts. In order to control the errors caused by R-LODs, we associate a deviation error metric and compute a screen-space projection in terms of *pixels-of-error* (PoE) deviation. Also, we assume that our drastically simplified LOD representations are mainly used given small PoE values (e.g., 1–4 pixels at image resolution 1024×1024) for high-quality rendering.

4.2 Runtime Traversal with R-LODs

Our new traversal algorithm is a modification of the efficient traversal algorithm described in Wald's thesis [30] and [29]. We recursively traverse the kd-tree from the root node or the entry-point that is computed using multi-level ray tracing. When we reach an intermediate node associated with a R-LOD, we check whether we can use the R-LOD based on our LOD error metric. If the current R-LOD satisfies the LOD error metric, we perform an intersection test between a R-LOD and the ray. If there is an intersection, we stop the traversal and return the intersection data of the R-LOD to compute shading and shoot secondary rays, if necessary. If there is no intersection, the algorithm does not traverse the child nodes of the intermediate node associated with the R-LOD. Each R-LOD is bounded by a kd-node and therefore, the extent of the plane of the R-LOD is implicitly bounded by the kd-node during tree traversal. The implicit extent of the plane results in a compact R-LOD representation.

4.3 LOD Error Metric Evaluation

We use a projection-based algorithm integrated with surface deviation error to select appropriate LODs for ray tracing.

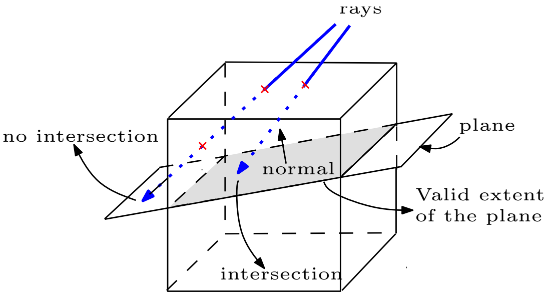


Figure 5. LOD Representation: *A R-LOD consists of a plane with material attributes. It serves as a drastic simplification of triangle primitives contained in the bounding box of the subtree of a kd-tree node. Its extent is implicitly given by its containing kd-node. The plane representation makes the intersection between a ray and a R-LOD very efficiently and results in a compact representation.*

Conservative projection algorithm: We use a projection method to efficiently compare the screen-space area of the R-LOD after the perspective projection with the PoE in the screen-space. Conceptually, we position a projection plane at the intersection between the ray and the kd-tree node. The plane is set to be orthogonal to the ray, as shown in Fig. 6. We enclose the R-LOD (and its corresponding simplified geometry) in a sphere. The area of the R-LOD projected onto the projection plane is conservatively measured by computing πR^2, where R is the radius of the sphere. Let R_p be the radius of a sphere inscribed in a rectangular shape pixel of the image screen. In this case, R_p is simply half of the width of the pixel. Then, the projected area of a pixel in the projection plane satisfies the following relationship:

$$\frac{d_{near}}{R_p} = \frac{d_{min}}{\hat{R}_p} \Rightarrow \hat{R}_p = d_{min}\frac{R_p}{d_{near}} = d_{min}C, \qquad (1)$$

where $\hat{R_p}$ is the projected radius of R_p, d_{near} is the distance from the viewer to the image plane, and d_{min} is the distance from the viewer to the intersection point between the ray and the kd-node. Since $\frac{R_p}{d_{near}}$ $(= C)$ is a constant, the projected radius, $\hat{R_p}$, is a simple linear function of the distance, d_{min}, along the ray from the eye to the intersecting node. We select an R-LOD if $\hat{R_p}$, is bigger than the radius, R, associated with the R-LOD. Our LOD metric is very efficient as it requires only one multiplication and d_{min} is already known during the tree traversal.

Surface deviation: The error metric described above conservatively measures the projected screen-space area of the R-LOD. We augment the metric to take into account the surface deviation of a R-LOD. For this we first measure the surface deviation between the plane of the R-LOD and all the contained triangles. We combine the surface deviation and the projected screen-space area of the R-LOD in the following geometric formulation. We compute the volume of the surface deviation along the plane and add this volume to the volume of the sphere enclosing the R-LOD. We then treat the summed volume as a volume of an imaginary sphere and use its radius as the error bound of the R-LOD. In this geometric formulation, these two seeming different error bounds can be treated in a uniform manner.

Error quantization: The exact representation of the plane and associated materials takes 32 bytes. Instead of directly associating this information with each node of the kd-tree, we quantize the error bounds associated with the R-LODs and store the quantized error bound as well as an R-LOD index in a 4 byte structure as the part of the kd-node in order to reduce the working set size during traversals. Therefore, only if the error bound of an R-LOD is satisfied within the PoE bound, we load the exact R-LOD representation by using the R-LOD index. When considering a path from a leaf node of the kd-tree to the root node, the error bounds associated with the nodes increase as a geometric series. Therefore, we use a geometric distribution equation to quantize the error values associated with the R-LODs. We found that 5 bits are enough (i.e. 10%–20% quantization error) to conservatively quantize the error bound of the R-LODs in our benchmarks; therefore, each R-LOD index is stored in 27 bits, which are enough to indicate all the R-LODs in our tested models.

Secondary rays: Our LOD metric based on conservative projection also extends to secondary rays. These include *reflection* (in which a ray reflects at an intersection point with a reflective triangle) and *shadow* rays. This is mainly because these secondary rays can be expressed as a linear transformation [11]. In the case of reflection, the radius, R_p, of the sphere inscribed in the pixels of the image space increases linearly based on the sum of the distance from the viewer to an intersecting reflective triangle, and to another intersecting object along the primary or reflective secondary rays. Similarly, our metric also works well for shadow rays and again we use a linear transformation. One issue with using LODs for shadow rays is that they can result in self-shadowing artifacts when different versions of the R-LODs are selected by the primary ray and the shadow ray. We overcome this problem by ignoring the intersections between the shadow ray and the primitives that are within the LOD error bounds associated with the R-LOD selected by the primary ray.

Our projection-based method does not work with *refraction*, since refraction is not a linear transformation [11]. In this case, we use a more general, but expensive method based on ray differentials [12], to decide whether an R-LOD satisfies the PoE bound after refraction.

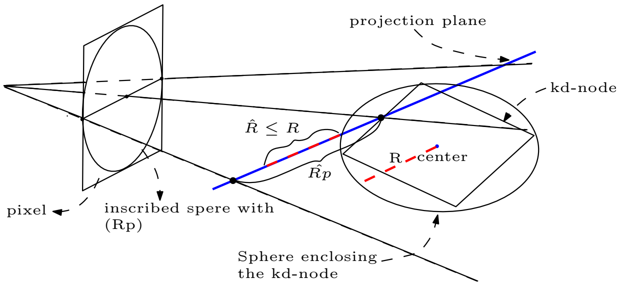


Figure 6. **Projection-based LOD Metric**: *We place a projection plane at the intersection point between a ray and the kd-node. The plane is orthogonal to the ray. Based on this projection plane, we conservatively check whether the R-LOD satisfies the error metric.*

4.4 R-LOD Construction

Our goal is to compute a plane that approximates the triangles that are contained in the subtree of an intermediate kd-node and also their material properties. If a triangle contained in the subtree is not fully contained in the bounding box of the node, we clip the triangle against the box and do not consider the clipped portion of the triangle. We use principal component analysis (PCA [13]), to compute the plane. PCA computes the eigenvectors that provide a statistical description of input points. We perform PCA computation based on the vertices of the triangles, but also take into account the size of the triangles by associating the area of the triangle as a weight for each vertex. The plane is computed based on the eigenvector associated with the largest eigenvalue and this eigenvector represents the normal to the plane[1]. We compute material properties that are mean values of the contained triangles and associate them with the R-LOD. The surface deviation of the plane against the geometric primitives is computed based on the smallest eigenvalue, which corresponds to a variance of geometry along the normal of the plane.

Hierarchical R-LOD computation: We can compute the R-LODs associated with each node of the tree in a bottom-up manner. However, a naive algorithm would compute the R-LOD for each node independently and this can result in a $O(n \log n)$ algorithm.

Instead, we present a R-LOD computation algorithm that has linear time complexity and is well suited for out-of-core computation. Each element, σ_{ij}, of (i, j)th component of a covariance matrix for PCA is defined as the following:

$$\sigma_{ij} = \sum_{k=1}^{n} (V_i^k - \mu_i)(V_j^k - \mu_j), \tag{2}$$

where V_i^k is the ith component (e.g. x, y, and z) of kth vertex data, μ_i is the mean of V_i^k, and n is the number of vertices. This equation can be reformulated as:

$$\sigma_{i,j} = \sum_{k=1}^{n} V_i^k V_j^k - \frac{2}{n} \sum_{k=1}^{n} V_i^k \sum_{k=1}^{n} V_j^k + \frac{1}{n^2} \sum_{k=1}^{n} V_i^k \sum_{k=1}^{n} V_j^k, \tag{3}$$

It follows that if we can compute and store the sums of V_i^k, V_j^k, $V_i^k V_j^k$, and n, we can compute the covariance with these sums and n for any intermediate node. In order to compute the covariance matrix of a parent node, we simply add these

[1] The direction of the normal is chosen to be closer to the average normal of triangles.

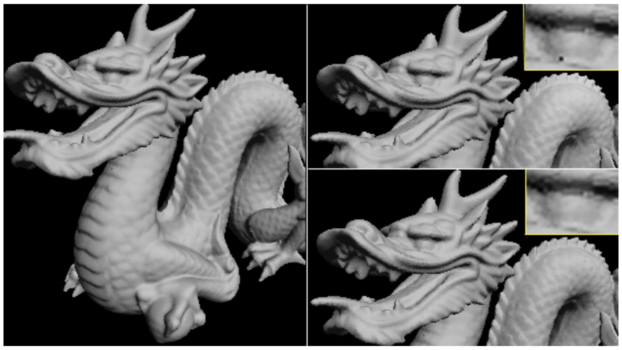


Figure 7. C^0 **Discontinuity**: *The left image shows the Stanford dragon model as rendered by our approach with PoE* $= 0$, *i.e. using original triangles. The top right image was acquired by setting PoE* $= 5$ *at* 512×512 *image resolution with no expansion of R-LODs. As can be seen in the area of the dragon's eye, there is a hole caused by* C^0 *discontinuity of our LOD representation. By allowing a small amount of expansion of R-LODs, we can avoid having holes in the final image as shown in the bottom right image. Close-ups of the eye are shown in boxes with yellow borders.*

variables as a weighted sum of the number of vertices contained in each child node. This property is particularly useful to compute the R-LODs of inner nodes in the kd-tree in an out-of-core manner. Our algorithm has linear time complexity and its memory overhead is a function of the height of the tree. In practice, the memory overhead in our benchmarks is less than 1MB.

4.5 C^0 Discontinuity between R-LODs

Our LOD computation algorithm computes a drastic simplification. Therefore, if the underlying triangles have high curvature, the PCA-based approximation can have high surface deviation. In this case, it is possible that our algorithm does not maintain C^0 continuity between R-LODs, which can result in some holes in the resulting image (see Fig. 7). This kind of problem has been well-studied in the LOD and point-based rendering literature. Particularly, many techniques in the LOD literature have been proposed to patch these holes using precomputed data structures or runtime algorithms [6, 41]. However, those approaches can increase the storage and runtime overhead of ray tracing algorithms. In our implementation, we do not use any patching techniques.

Instead, we ameliorate this problem through our R-LOD selection algorithm. A very low PoE bound should be used to limit the error introduced by the R-LODs. The low PoE bound also minimizes temporal popping that can arise when we switch between the R-LODs of parent and children nodes during successive frames. Moreover, we assign higher weight to surface deviation computation as part of the error metric computation; therefore, higher resolutions are used in the region with high curvature.

Expansion of R-LODs: In addition to these two heuristics, we also expand the extents of R-LODs to remove holes caused by C^0 discontinuity between R-LODs. Please note that as the surface deviation increases, it is likely that gaps become larger. Therefore, we increase the extent of a R-LOD as a function of the surface deviation associated with the R-LOD. This expansion is efficiently considered during the plane and ray intersection as an additional numerical tolerance. In practice, we found that combining these heuristics work well to remove holes caused by C^0 discontinuity without introducing any noticeable visual artifact given low PoE error bounds (see Fig. 7).

5 Utilizing Coherences

In this section, we describe approaches to improve the performance of our ray tracing algorithm using ray coherence and cache-coherent layouts.

5.1 Ray Coherence

We define ray coherence as the coherence of rays in tree traversal and intersection, i.e. rays may take a similar path in the tree and may hit the same triangles. For primary rays, our ray tracer starts out by assuming there is ray coherence and shoots a beam using the algorithm presented in [22]. We compute a common entry point in the tree for all rays in the beam, at which the beam is split into either sub-beams or ray packets depending on its size. For the latter case, we use the coherent ray tracing algorithm [31] which works on a 2×2 packet of rays in parallel using current processors' SIMD instructions. During all traversal, we check whether we need to use R-LODs that have appropriate resolution based on our LOD metric. If so, we stop traversal of that subtree and intersect with the simplified representation. If the given model is highly tessellated, beam tracing and the use of SIMD instructions may not work well and can even lead to a decrease in performance (as explained in Section 3.2). However, we found that the use of R-LODs alleviates this problem, as we generally do not traverse as deep into the tree and therefore execute less overhead intersections. Secondary rays can be also handled in a similar manner.

5.2 Cache Coherence

It is highly desirable to maintain cache coherence during runtime tree traversals to help to achieve good performance. We apply the cache-oblivious mesh layout algorithm [39] to compute cache-coherent layouts of the kd-nodes. We interpret the kd-tree as a graph that can represent the expected runtime access patterns. The quality of the layout depends on the structure of the input graph. In order to predict the runtime behavior of tree traversal by the graph, we use a simple method to compute the probability that a node will be accessed given that its parent node has been accessed before, based on their geometric relationship [40]. The ray tracing algorithm traverses the child nodes from the parent node when there is an intersection between a ray and the bounding box of the parent node. Therefore, we estimate that the probability that the child node is accessed increases as its surface area compared to its parent node increases. This property is already well exploited by the kd-tree construction algorithms by using the surface areas of the bounding boxes of the kd-nodes [17]. The layouts can increase the performance of the ray tracer by $10 - 60\%$ on massive models. This is in addition to the speedups obtained by R-LODs.

6 Implementation and Results

In this section, we describe our implementation and highlight the performance of our ray tracer on different benchmarks.

6.1 Implementation

We have implemented our R-LOD construction algorithm and ray tracer on both 32-bit and 64-bit machines that have two dual-core Xeon processors running 32-bit and 64-bit Windows XP, respectively. For runtime ray tracing, we use memory mapped files to efficiently access large files of geometry and kd-tree. However, in the 32-bit OS, we can only map up to 3GB total memory. To deal with larger data, we have implemented explicit out-of-core memory access management. This is not necessary in the 64-bit OS where we just use implicit OS memory mapping functionality.

In order to construct the kd-tree for a model that does not fit into main memory, we first subdivide the model into voxels in an out-of-core manner and then build the kd-tree for each of these voxels individually in core [41]. This step can also be performed in parallel on different voxels for speeding up the

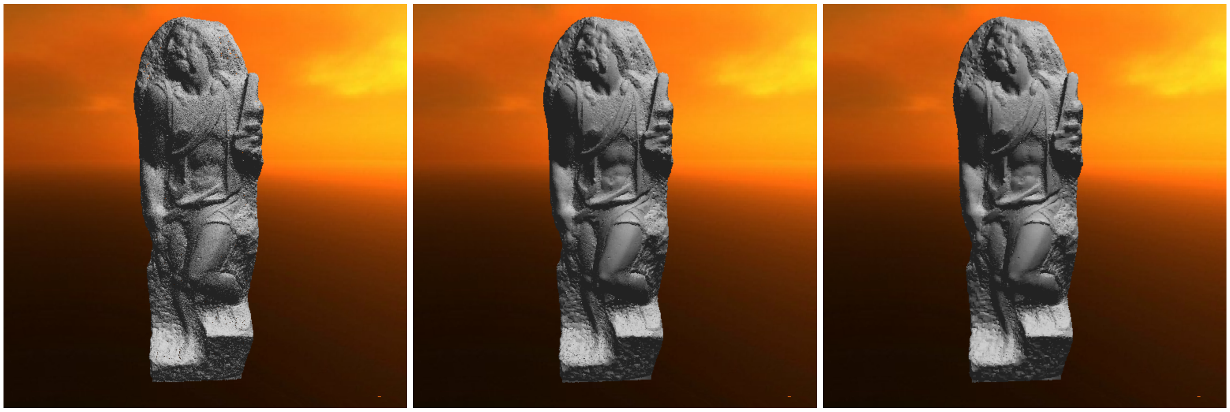

(a) No LOD (b) PoE = 2.5 (c) PoE = 5

Figure 8. *Images of the St. Matthew model with different PoE values are shown at 512×512 image resolutions. We do not use anti-aliasing to highlight image quality difference. Please note that the original St. Matthew model has many holes. The use of R-LODs can alleviate aliasing artifacts and improve the performance of massive models.*

Model	Vert.	Tri.	Node	Size	R-LOD
	(M)	(M)	(M)	(GB)	Comp. (min)
Forest	19	32	105	4.1	10
Double eagle	77.7	81.7	173	9.1	32
St. Matthew	128	256	378	26	124

Table 1. Benchmark models
Model complexity, the number of kd-nodes, the total size of kd-tree, geometry, and R-LODs, and the construction time of R-LODs are shown.

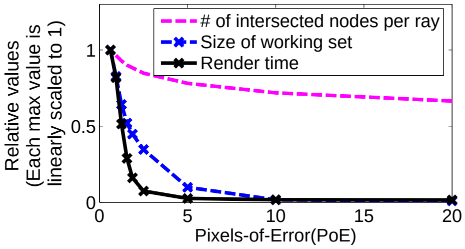


Figure 9. Performance variation as a function of PoE: *We show the relative benefit of R-LODs on different aspects of overall performance of ray tracing St. Matthew model. We measured the rendering time, average number of processed node per ray, and size of working set on a 32-bit Xeon machine with 2GB RAM. All these values are shown in a scale-invariant manner by linearly scaling their maximum values to 1. The performance of our LOD-based ray tracer drastically decreases as we linearly increase the PoE. Moreover, the graph indicates that there is high correlation between the performance of the ray tracer and the size of working set. Image shots generated by tested PoE values can be seen in Fig. 8.*

construction. Afterwards, the kd-tree for each voxel is merged into the global tree, which is used for ray tracing.

Since we have found that the quality of the kd-tree is the most important factor for fast ray tracing, we build the kd-tree using the surface-area heuristic [17, 10] and some further improvements as presented by [22]. Especially important is to introduce extra splits for empty areas in order to bound the geometry more tightly for our R-LOD representation.

6.2 Results

Benchmarks: We have applied our LOD-based ray tracing algorithm to different benchmarks as shown in Table 1. We computed different paths through these models and measured the performance of the ray tracer with and without LODs using a small PoE metric. We use a resolution of 512×512 pixels for interactive rendering. We also use 2×2 super-sampling per pixel; therefore, we effectively shoot $1K \times 1K$ rays from the eye for each frame. We are able to render most of these models at $5 - 12$ frames a second with primary rays and $1 - 8$ frames a second when we include reflections and shadow rays. These results are shown in the video.

Preprocessing: We only compute R-LODs for a subset of the nodes in the kd-tree to avoid excessive memory overhead. Our current implementation selects every third node on the path from the root node to the leaf node. Our unoptimized R-LOD construction implementation can process 2–3 million triangles per minute; most of the processing time is spent on reading data from the disk. The size of R-LODs associated with each node takes less than 10% of the total storage. However, if we consider the additional 4 bytes for R-LOD index and quantized error bound in the kd-nodes, total storage overhead of our R-LOD nodes is roughly 33% compared to the optimized kd-tree representation[30].

Performance variation as a function of PoE: We vary the PoE metric for the St. Matthew model (256M triangles) and measure its benefit on the rendering time, average number of processed nodes per ray, and size of working set per frame. The working set is measured at a granularity of 4KB. In order to show the relative benefit, we linearly scale each value into $[0, 1]$ by scaling the maximum value of each item to 1. The min and max values of each item are as follows: rendering time (ms)(160, 11914), size of the working set(MB) (2, 1565), and average number of processed nodes per ray (13.6, 22.42). As can be seen in Fig. 9, the performance of the ray tracer increases drastically as we linearly increase the PoE values.

Runtime performance: The benefit of LODs varies with the reduction in the working set size. For a highly tessellated St. Matthew model with 128M triangles, we achieve more than one order of magnitude reduction in the size of the working set and almost two orders of magnitude improvement in the frame rate. This model has low depth complexity and more than half the primitives are visible from the eye. We show the frame rates obtained during rendering of the St. Matthew model with and without R-LODs and cache-oblivious layouts in Fig. 10. Moreover, we are able to achieve 2.6 frames per second while rendering the model with shadow and reflection with little loss of image quality (see. Fig. 1). For the forest model shown in Fig. 4, we are able to achieve more than five times improvement by using R-LODs.

In the case of the Double Eagle tanker, we get 10%–200% improvement. This model has high depth complexity and is not highly tessellated. As a result, the performance improvement due to LODs is limited. An image of the tanker with shadows is shown in Fig. 2.

7 Analysis and Comparison

In this section, we analyze the performance of our ray tracing algorithm and also compare its performance to prior approaches. We also discuss some limitations of our approach.

7.1 Analysis

We first examine different aspects of our R-LOD representation.

R-LOD overhead: Our algorithm introduces 4 bytes of additional storage for each kd-node. We measure the additional computational overhead of evaluating our LOD metric during traversal by comparing the runtime performance on the Stanford scanned dragon model (870K triangles) of the standard ray tracer using 8 byte sized kd-nodes and of our ray tracer, which uses 12 byte-sized nodes with stored R-LODs. In order to measure the overhead of R-LODs, we set our PoE metric to 0 during LOD tree traversal; consequently, the image quality is the same in both cases. We found that the R-LOD overhead for storage and traversal reduces the performance by 2%–5%, as compared to ray tracing as described in [30].

Performance gains: The use of R-LODs reduces both computational workload and memory requirements. A major benefit of R-LODs is the reduction of the working set size and cache miss ratios of the runtime algorithm. This size decreases almost as a exponential function of the PoE as shown in Fig. 9. As a result, we get fewer L1/L2 cache misses and page faults and our new ray tracing algorithm is more cache coherent.

7.2 Comparison to other approaches

Our algorithm integrates R-LODs with the kd-tree representation for ray tracing. The idea of using an integrated hierarchical representation for traversal, visibility and simplification has been used by other algorithms for interactive rendering. These include the QSplat system [23], which uses a hierarchy of spheres and a screen space PoE metric to stop the tree traversal at a node. However, QSplat is mainly designed for point datasets or dense meshes arising from scanned models. Moreover, our LOD computation and error metric evaluation algorithms are different from QSplat as we take into account primary and secondary rays. The Quick-VDR system [41] uses a two-level multiresolution hierarchy called CHPM for view-dependent simplification and visibility culling of large polygonal models. However, the CHPM representation has a high memory overhead and does not lend itself well to ray tracing.

Several other ray tracing algorithms based on LODs have been proposed. The algorithm that is closest to our approach is the out-of-core ray tracer described in [32]. While we use R-LODs to perform fewer node and triangle intersections, Wald *et al.* use a simplified version only when the data is not in main memory in order to hide the latency incurred by loading data from the disk. This approach works well when the working set is smaller than main memory. Our LOD based algorithm is complimentary to their work and uses a different representation to reduce the size of the working set and perform fewer ray intersections.

Pharr *et al.* [20] describe an algorithm to optimize memory coherence in ray tracing. In their approach, the rays are reordered so that they access the scene data in a coherent manner. Their prime application was accelerating ray tracing for offline rendering. Our LOD based approach is quite complimentary to their algorithm. The LOD-based renderer described by Christensen *et al.*[5] differs from ours in two respects. Firstly, it uses subdivision meshes. Therefore, it is primarily useful for computing appropriate tessellation levels from the coarsest resolution. On the other hand, we compute the R-LODs from the original mesh. Secondly, Christensen *et al.* use ray differentials, which is expensive for real-time ray tracing. In contrast, our LOD metric is very efficient and optimized for interactive rendering.

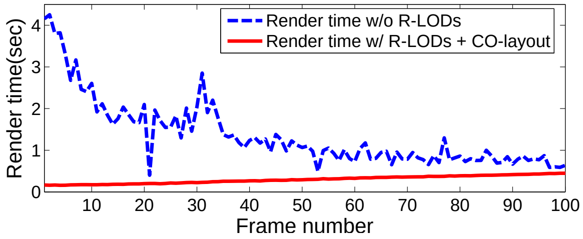


Figure 10. **Frame time with and without R-LODs**: *The graphs shows frame times while rendering the $128M$ St. Matthew model with/without R-LODs and cache-oblivious (CO) layout. We measure frame time when we approach the model starting from the viewpoint shown in Fig. 8. The path is also shown in the video.*

7.3 Limitations

Our approach has certain limitations. First of all, any LOD-based acceleration technique can result in visual artifacts. We minimize these artifacts by using a low PoE bound and combining the projected screen-space error and surface deviation error of an R-LOD. If we use a high PoE bound, the R-LODs may result in holes on the simplified representation. This visual artifact can be removed by employing implicit surfaces[33, 14] as a LOD and thereby sacrificing some of the efficiency of our LOD representation. Moreover, our current R-LOD representation is a drastic simplification of the underlying geometric primitives and their material properties. As a result, the R-LOD representation may not provide high quality simplification for surfaces that have highly varying BRDF. One possibility is to use a more complex reflectance representation [18] in such cases. Also, our LOD metric does not give guarantees on the errors in the path traced by the secondary rays and the illumination computed at each pixel. However, we indirectly reduce the differences by reducing errors associated with the R-LODs. Finally, our efficient projection-based LOD error metric can currently handle planar reflections and shadow rays, but not refraction nor non-planar reflection.

8 Conclusion and Future Work

We have presented a novel LOD-based ray tracing algorithm to improve the performance of ray tracing massive models. We use the R-LOD representation as a drastic simplification of geometric primitives contained in the subtree of a kd-node and select the LODs based on our projection-based LOD error metric. We have described a hierarchical R-LOD construction algorithm that has linear time complexity and is well suited for out-of-core computation. The use of R-LODs results in fewer intersection tests and can significantly improve the memory coherence of the ray tracing algorithm. We have observed more than an order of magnitude speedup on massive models, and most of these gains are due to improved memory coherence and fewer cache misses.

There are many avenues for future work. In addition to addressing current limitations of our approaches, we would like to extend our current R-LOD representation to support smooth implicit surfaces to improve the rendering quality, and still have a compact representation. Moreover, we would like to extend our approach to handle other kinds of input model types such as point clouds [24] and higher order primitives. It might be useful to integrate approximate visibility criteria within our efficient LOD metric to further improve the performance ray tracing on massive models with high depth complexity. Also, we would like to consider visibility issues during construction of R-LODs in order to have better visual quality. Furthermore, we are interested in evaluating our ray tracer on other complex datasets and measuring the performance benefit. LODs could also be potentially useful in the context of designing future hardware for interactive ray tracing.

Acknowledgments

We would like to thank Matt Pharr, Ingo Wald, Eric Haines, and David Kasik for providing helpful feedback on an earlier draft of this paper. The St. Matthew statue are courtesy of the Digital Michelangelo Project at Stanford University. the Double Eagle tanker is courtesy of Newport News Shipbuilding. Simplified St. Matthew models are computed by a streaming simplification tool provided by Peter Lindstrom. This work was supported in part by ARO Contracts DAAD19-02-1-0390 and W911NF-04-1-0088, NSF awards 0400134 and 0118743, ONR Contract N00014-01-1-0496, DARPA/RDECOM Contract N61339-04-C-0043 and Intel. Some of the work was performed under the auspices of the U.S. Department of Energy by the University of California, Lawrence Livermore National Laboratory.

References

[1] M. Agrawala, R. Ramamoorthi, and A. Moll. Efficient image-based methods for rendering soft shadows. In *ACM SIGGRAPH*, pages 375–384, 2000.

[2] J. Amanatides. Ray tracing with cones. In *Computer Graphics (SIGGRAPH '84 Proceedings)*, volume 18, pages 129–135, July 1984.

[3] A. Appel. Some techniques for shading machine renderings of solids. In *AFIPS 1968 Spring Joint Computer Conf.*, volume 32, pages 37–45, 1968.

[4] Y.-J. Chiang, J. El-Sana, P. Lindstrom, R. Pajarola, and C. T. Silva. Out-of-core algorithms for scientific visualization and computer graphics. *IEEE Visualization 2003 Course Notes*, 2003.

[5] P. H. Christensen, D. M. Laur, J. Fong, W. L. Wooten, and D. Batali. Ray differentials and multiresolution geometry caching for distribution ray tracing in complex scenes. *Computer Graphics Forum*, 22(3):543–552, Sept. 2003.

[6] P. Cignoni, F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio, and R. Scopigno. Adaptive tetrapuzzles: efficient out-of-core construction and visualization of gigantic multiresolution polygonal models. *ACM Trans. Graph.*, 23(3):796–803, 2004.

[7] D. E. DeMarle, C. P. Gribble, and S. G. Parker. Memory-savvy distributed interactive ray tracing. In *EGPGV*, pages 93–100, 2004.

[8] A. Dietrich, I. Wald, and P. Slusallek. Large-Scale CAD Model Visualization on a Scalable Shared-Memory Architecture. In *Proceedings of 10th International Fall Workshop - Vision, Modeling, and Visualization (VMV) 2005*, pages 303–310, 2005.

[9] E. Gobbetti and F. Marton. Far voxels: A multiresolution framework for interactive rendering of huge complex 3D models on commodity graphics platforms. *ACM Trans. Graph.*, 24(3):878–885, 2005.

[10] V. Havran. *Heuristic Ray Shooting Algorithms*. PhD thesis, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, November 2000.

[11] P. S. Heckbert and P. Hanrahan. Beam tracing polygonal objects. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 119–127, New York, NY, USA, 1984. ACM Press.

[12] H. Igehy. Tracing ray differentials. In *ACM SIGGRAPH*, pages 179–186, 1999.

[13] I. Jolliffe. Principle component analysis. In *Springer-Veriag*, 1986.

[14] D. Levin. Mesh-independent surface interpolation. In *Geometric Modeling for Scientific Visualization*, pages 37–49, 2003.

[15] D. Lischinski and A. Rappoport. Image-based rendering for non-diffuse synthetic scenes. In *Eurographics Rendering Workshop 98*, pages 301–314, 1998.

[16] D. Luebke, M. Reddy, J. Cohen, A. Varshney, B. Watson, and R. Huebner. *Level of Detail for 3D Graphics*. Morgan-Kaufmann, 2002.

[17] J. D. MacDonald and K. S. Booth. Heuristics for ray tracing using space subdivision. *Visual Computer*, 1990.

[18] F. Neyret. Modeling, animating, and rendering complex scenes using volumetric textures. *IEEE Transactions on Visualization and Computer Graphics*, 1998.

[19] S. Parker, W. Martin, P. Sloan, P. Shirley, B. Smits, and C. Hansen. Interactive ray tracing. *Symposium on Interactive 3D Graphics*, 1999.

[20] M. Pharr, C. Kolb, R. Gershbein, and P. Hanrahan. Rendering complex scenes with memory-coherent ray tracing. In *Proc. of ACM SIGGRAPH*, pages 101–108, 1997.

[21] T. Purcell, I. Buck, W. Mark, and P. Hanrahan. Ray tracing on programmable graphics hardware. *ACM Trans. on Graphics (Proc. of SIGGRAPH'02)*, 21(3):703–712, 2002.

[22] A. Reshetov, A. Soupikov, and J. Hurley. Multi-level ray tracing algorithm. *ACM Trans. Graph.*, 24(3):1176–1185, 2005.

[23] S. Rusinkiewicz and M. Levoy. Qsplat: A multiresolution point rendering system for large meshes. *Proc. of ACM SIGGRAPH*, pages 343–352, 2000.

[24] G. Schaufler and H. W. Jensen. Ray tracing point sampled geometry. In *Rendering Techniques*, pages 319–328, 2000.

[25] J. Schmittler, S. Woop, D. Wagner, W. J. Paul, and P. Slusallek. Realtime ray tracing of dynamic scenes on an FPGA chip. In *HWWS '04: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 95–106, New York, NY, USA, 2004. ACM Press.

[26] M. Shinya, T. Takahashi, and S. Naito. Principles and applications of pencil tracing. In *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 45–54, July 1987.

[27] P. Shirley, P. Slusallek, B. Mark, G. Stoll, and I. Wald. Introduction to real-time ray tracing. *SIGGRAPH Course Notes*, 2005.

[28] G. Stoll, W. R. Mark, P. Djeu, R. Wang, and I. Elhassan. Razor: An Architecture for Dynamic Multiresolution Ray Tracing. Technical Report TR-06-21, Dept of Computer Sciences, University of Texas, 2006.

[29] K. Sung and P. Shirley. Ray tracing with the BSP tree. *Graphics Gems III*, pages 271–274, 1992.

[30] I. Wald. *Realtime Ray Tracing and Interactive Global Illumination*. PhD thesis, Computer Graphics Group, Saarland University, 2004.

[31] I. Wald, C. Benthin, M. Wagner, and P. Slusallek. Interactive rendering with coherent ray tracing. In A. Chalmers and T.-M. Rhyne, editors, *Computer Graphics Forum (Proceedings of EUROGRAPHICS 2001)*, volume 20, pages 153–164. Blackwell Publishers, Oxford, 2001.

[32] I. Wald, A. Dietrich, and P. Slusallek. An Interactive Out-of-Core Rendering Framework for Visualizing Massively Complex Models. In *Proceedings of the Eurographics Symposium on Rendering*, 2004.

[33] I. Wald and H.-P. Seidel. Interactive Ray Tracing of Point Based Models. In *Proceedings of 2005 Symposium on Point Based Graphics*, 2005.

[34] I. Wald, P. Slusallek, and C. Benthin. Interactive distributed ray tracing of highly complex models. In S.J.Gortler and K.Myszkowski, editors, *Rendering Techniques 2001 (Proceedings of the 12th EUROGRAPHICS Workshop on Rendering*, pages 277–288. Springer, 2001.

[35] M. Wand and W. Straßer. Multi-resolution point-sampled raytracing. In *Graphics Interface*, 2003.

[36] T. Whitted. An improved illumination model for shaded display. *Commun. ACM*, 23(6):343–349, 1980.

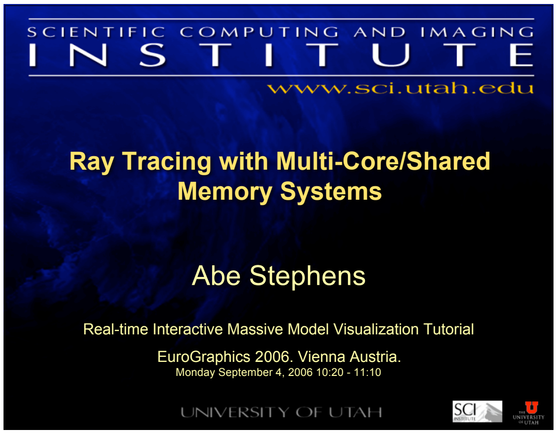
[37] C. Wiley, I. A.T. Campbell, S. Szygenda, D. Fussell, and F. Hudson. Multiresolution BSP trees applied to terrain, transparency, and general objects. In W. Davis, M. Mantei, and V. Klassen, editors, *Graphics Interface*, pages 88–96, May 1997.

[38] S. Woop, J. Schmittler, and P. Slusallek. RPU: a programmable ray processing unit for realtime ray tracing. *ACM Trans. Graph.*, 24(3):434–444, 2005.

[39] S.-E. Yoon, P. Lindstrom, V. Pascucci, and D. Manocha. Cache-Oblivious Mesh Layouts. *Proc. of ACM SIGGRAPH*, 2005.

[40] S.-E. Yoon and D. Manocha. Cache-Efficient Layouts of Bounding Volume Hierarchies. In *Computer Graphics Forum (Eurographics)*, 2006. Conditionally accepted.

[41] S.-E. Yoon, B. Salomon, R. Gayle, and D. Manocha. Quick-VDR: Interactive View-dependent Rendering of Massive Models. *IEEE Visualization*, pages 131–138, 2004.



SCIENTIFIC COMPUTING AND IMAGING INSTITUTE

www.sci.utah.edu

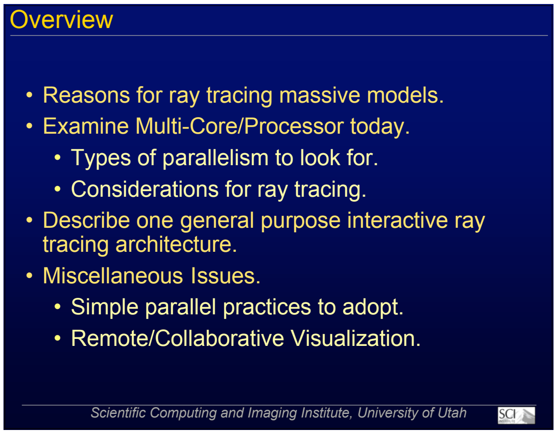
Ray Tracing with Multi-Core/Shared Memory Systems

Abe Stephens

Real-time Interactive Massive Model Visualization Tutorial

EuroGraphics 2006. Vienna Austria.
Monday September 4, 2006 10:20 - 11:10

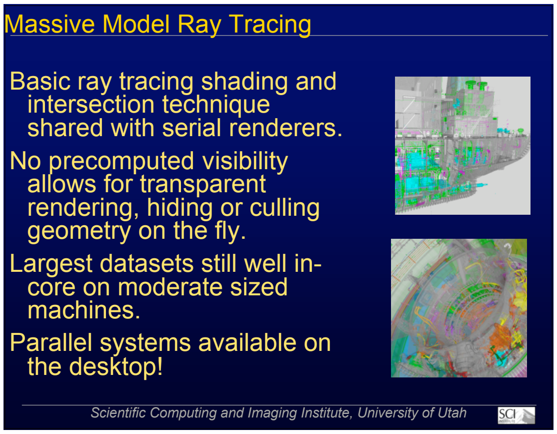
UNIVERSITY OF UTAH



Overview

- Reasons for ray tracing massive models.
- Examine Multi-Core/Processor today.
  - Types of parallelism to look for.
  - Considerations for ray tracing.
- Describe one general purpose interactive ray tracing architecture.
- Miscellaneous Issues.
  - Simple parallel practices to adopt.
  - Remote/Collaborative Visualization.

Scientific Computing and Imaging Institute, University of Utah



Massive Model Ray Tracing

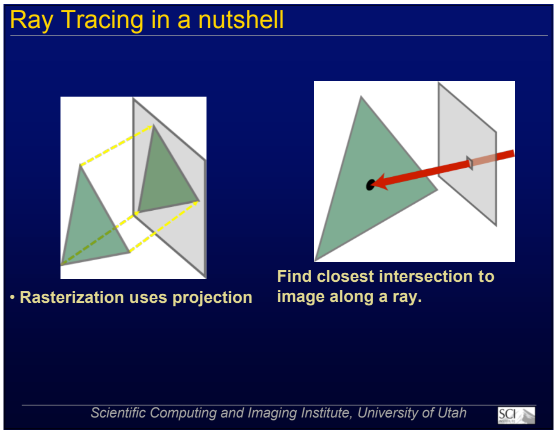
Basic ray tracing shading and intersection technique shared with serial renderers.

No precomputed visibility allows for transparent rendering, hiding or culling geometry on the fly.

Largest datasets still well in-core on moderate sized machines.

Parallel systems available on the desktop!

Scientific Computing and Imaging Institute, University of Utah

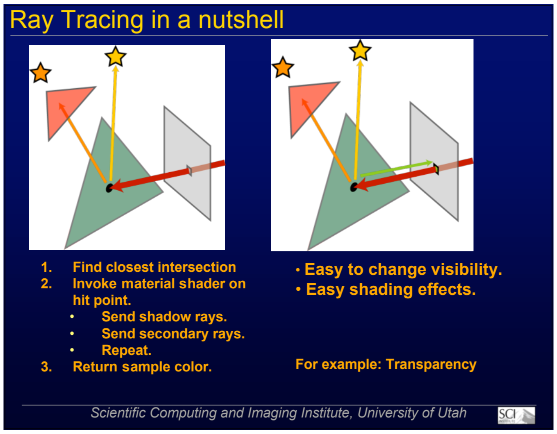


Ray Tracing in a nutshell

- Rasterization uses projection

Find closest intersection to image along a ray.

Scientific Computing and Imaging Institute, University of Utah



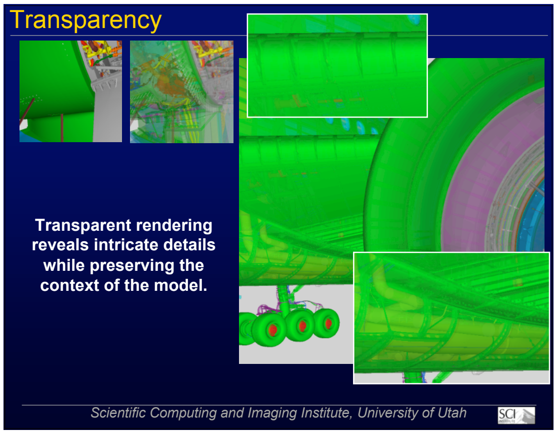
Ray Tracing in a nutshell

1. Find closest intersection
2. Invoke material shader on hit point.
   - Send shadow rays.
   - Send secondary rays.
   - Repeat.
3. Return sample color.

- Easy to change visibility.
- Easy shading effects.

For example: Transparency

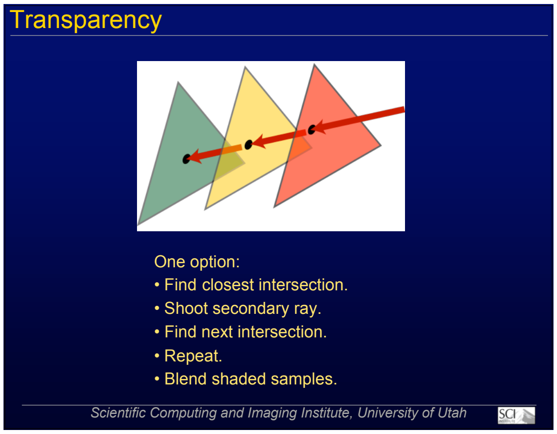
Scientific Computing and Imaging Institute, University of Utah



Transparency

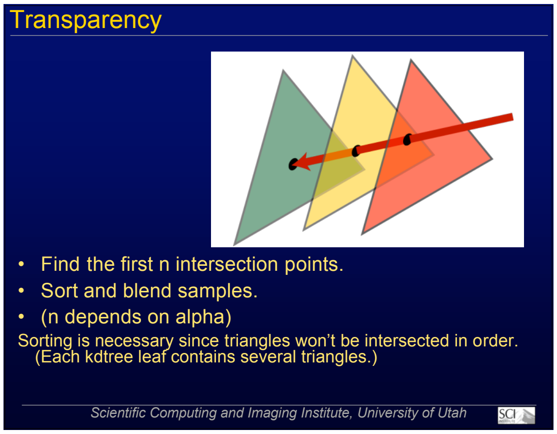
Transparent rendering reveals intricate details while preserving the context of the model.

Scientific Computing and Imaging Institute, University of Utah



Transparency

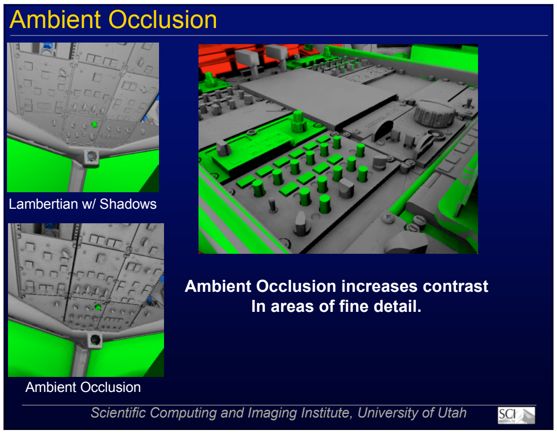
One option:
- Find closest intersection.
- Shoot secondary ray.
- Find next intersection.
- Repeat.
- Blend shaded samples.



Transparency

- Find the first n intersection points.
- Sort and blend samples.
- (n depends on alpha)

Sorting is necessary since triangles won't be intersected in order. (Each kdtree leaf contains several triangles.)

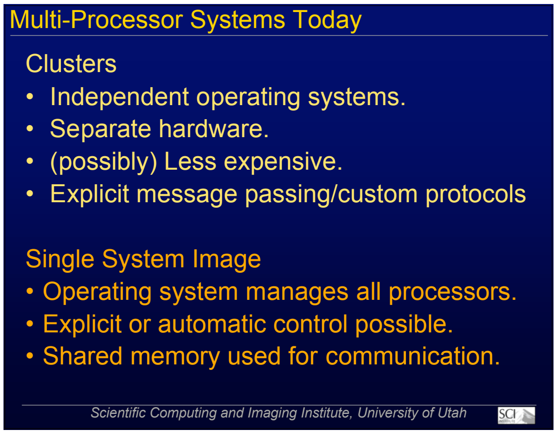


Ambient Occlusion

Lambertian w/ Shadows

Ambient Occlusion

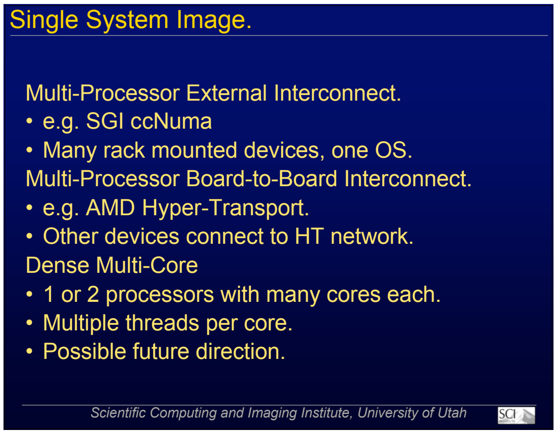
Ambient Occlusion increases contrast In areas of fine detail.



Multi-Processor Systems Today

Clusters
- Independent operating systems.
- Separate hardware.
- (possibly) Less expensive.
- Explicit message passing/custom protocols

Single System Image
- Operating system manages all processors.
- Explicit or automatic control possible.
- Shared memory used for communication.

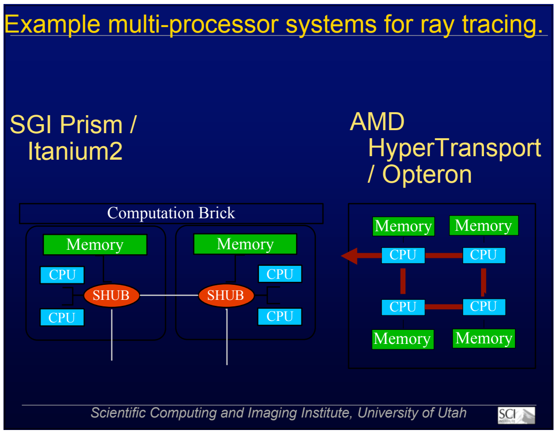


Single System Image.

Multi-Processor External Interconnect.
- e.g. SGI ccNuma
- Many rack mounted devices, one OS.

Multi-Processor Board-to-Board Interconnect.
- e.g. AMD Hyper-Transport.
- Other devices connect to HT network.

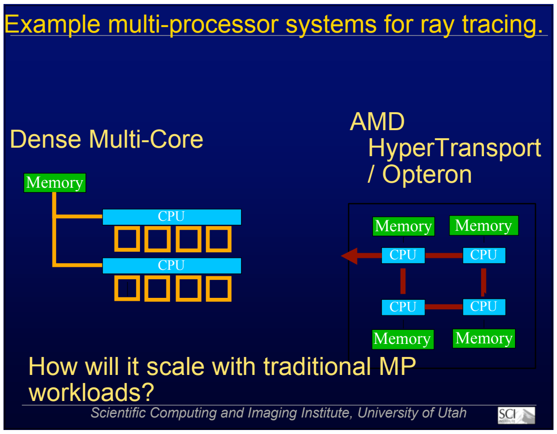
Dense Multi-Core
- 1 or 2 processors with many cores each.
- Multiple threads per core.
- Possible future direction.



Example multi-processor systems for ray tracing.

SGI Prism / Itanium2

AMD HyperTransport / Opteron



Example multi-processor systems for ray tracing.

Dense Multi-Core

Memory

CPU

CPU

AMD
HyperTransport
/ Opteron

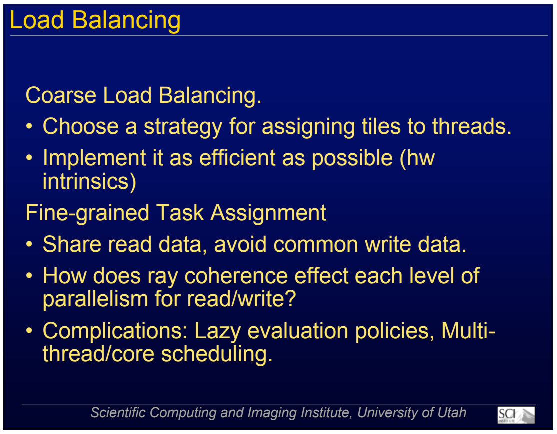
Memory Memory

CPU CPU

CPU CPU

Memory Memory

How will it scale with traditional MP workloads?



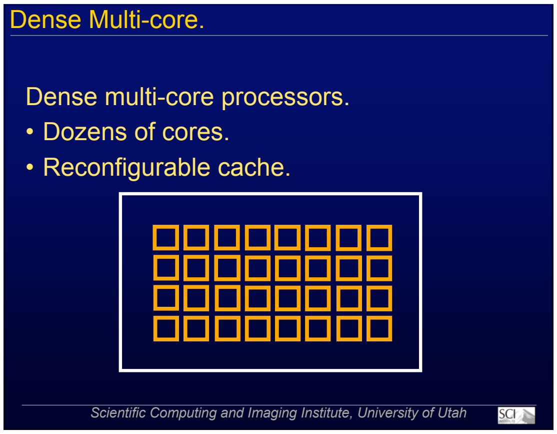
Load Balancing

Coarse Load Balancing.

- Choose a strategy for assigning tiles to threads.
- Implement it as efficient as possible (hw intrinsics)

Fine-grained Task Assignment

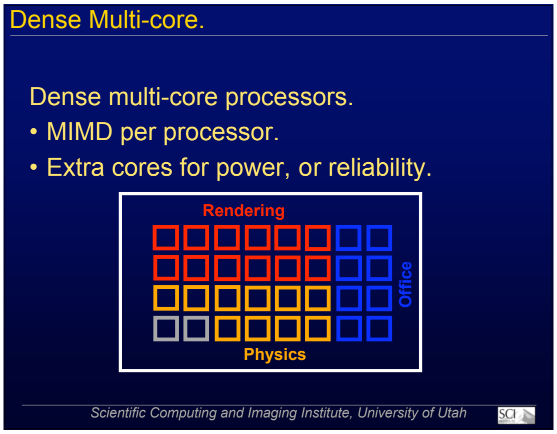
- Share read data, avoid common write data.
- How does ray coherence effect each level of parallelism for read/write?
- Complications: Lazy evaluation policies, Multi-thread/core scheduling.



Dense Multi-core.

Dense multi-core processors.

- Dozens of cores.
- Reconfigurable cache.



Dense Multi-core.

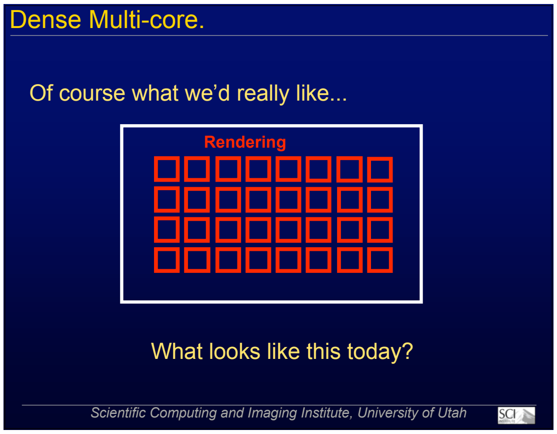
Dense multi-core processors.

- MIMD per processor.
- Extra cores for power, or reliability.

Rendering

Office

Physics

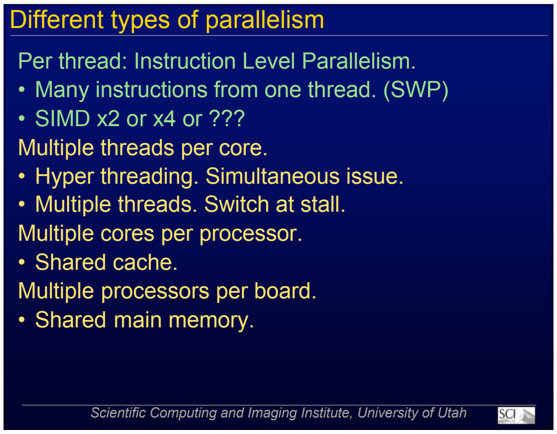


Dense Multi-core.

Of course what we'd really like…

Rendering

What looks like this today?



Different types of parallelism

Per thread: Instruction Level Parallelism.

- Many instructions from one thread. (SWP)
- SIMD x2 or x4 or ???

Multiple threads per core.

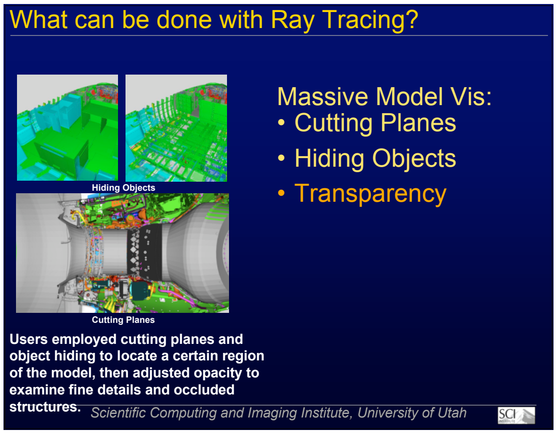
- Hyper threading. Simultaneous issue.
- Multiple threads. Switch at stall.

Multiple cores per processor.

- Shared cache.

Multiple processors per board.

- Shared main memory.



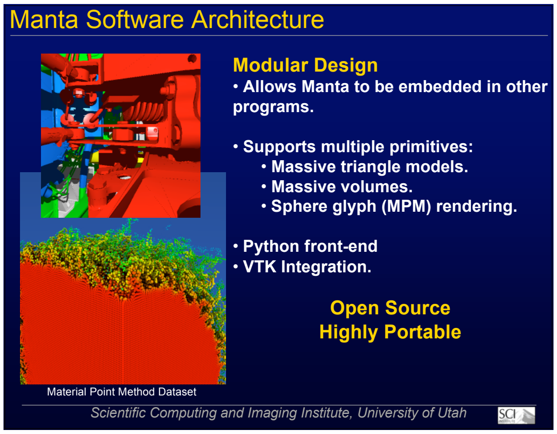
What can be done with Ray Tracing?

Hiding Objects

Cutting Planes

Users employed cutting planes and object hiding to locate a certain region of the model, then adjusted opacity to examine fine details and occluded structures.

Massive Model Vis:
- Cutting Planes
- Hiding Objects
- Transparency



Manta Software Architecture

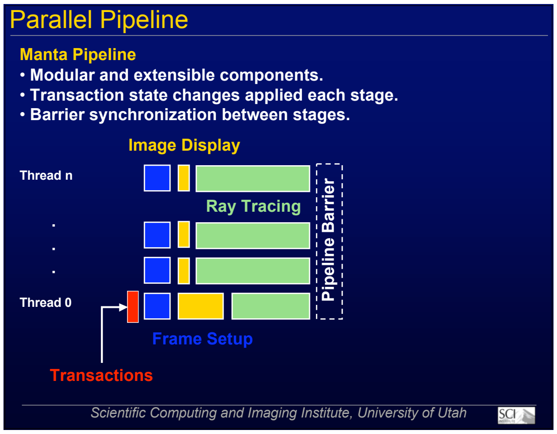
Material Point Method Dataset

Modular Design
- Allows Manta to be embedded in other programs.

- Supports multiple primitives:
  - Massive triangle models.
  - Massive volumes.
  - Sphere glyph (MPM) rendering.

- Python front-end
- VTK Integration.

Open Source
Highly Portable



Parallel Pipeline

Manta Pipeline
- Modular and extensible components.
- Transaction state changes applied each stage.
- Barrier synchronization between stages.

Image Display

Thread n

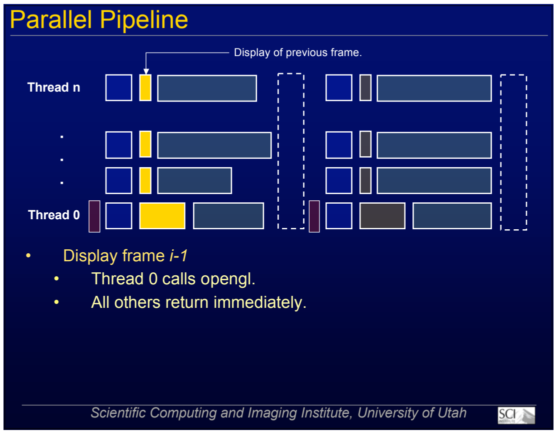
Ray Tracing

Pipeline Barrier

Thread 0

Frame Setup

Transactions



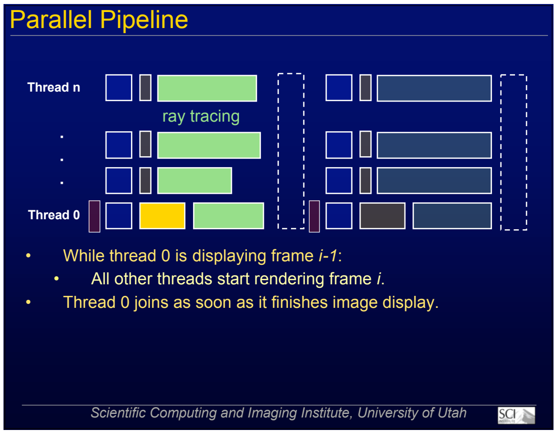
Parallel Pipeline

Display of previous frame.

Thread n

Thread 0

- Display frame *i-1*
  - Thread 0 calls opengl.
  - All others return immediately.



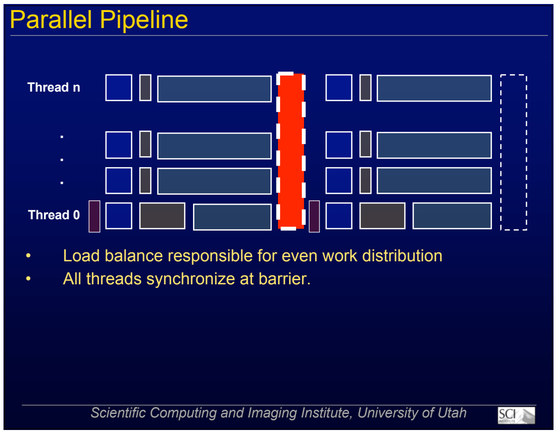
Parallel Pipeline

Thread n

ray tracing

Thread 0

- While thread 0 is displaying frame *i-1*:
  - All other threads start rendering frame *i*.
- Thread 0 joins as soon as it finishes image display.

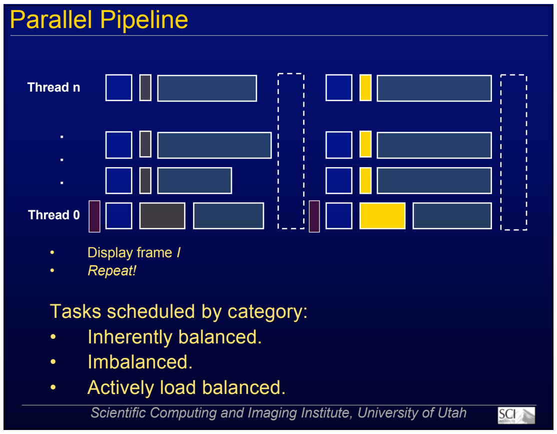


Parallel Pipeline

Thread n

Thread 0

- Load balance responsible for even work distribution
- All threads synchronize at barrier.

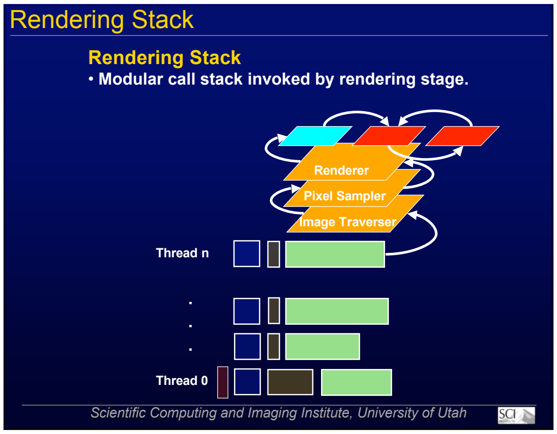


Parallel Pipeline

- Display frame *I*
- *Repeat!*

Tasks scheduled by category:

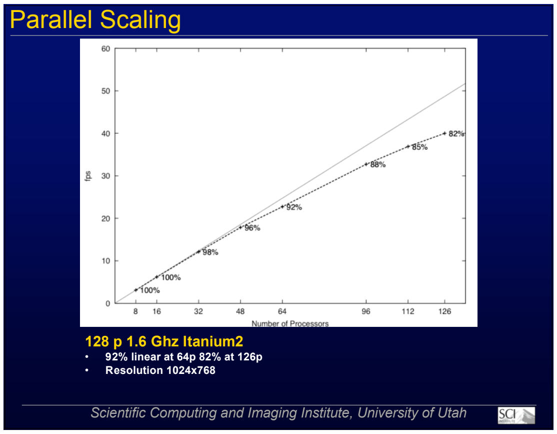
- Inherently balanced.
- Imbalanced.
- Actively load balanced.



Rendering Stack

Rendering Stack

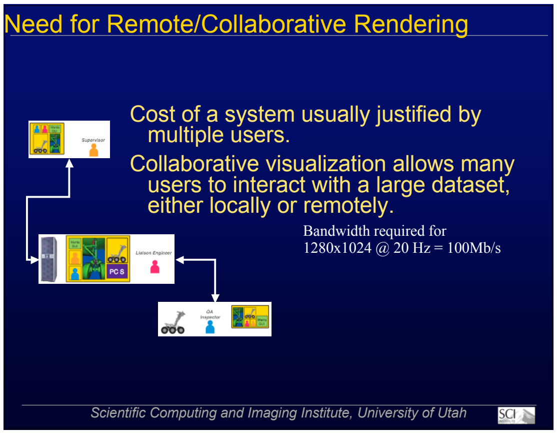
- **Modular call stack invoked by rendering stage.**



Parallel Scaling

128 p 1.6 Ghz Itanium2

- **92% linear at 64p 82% at 126p**
- **Resolution 1024x768**

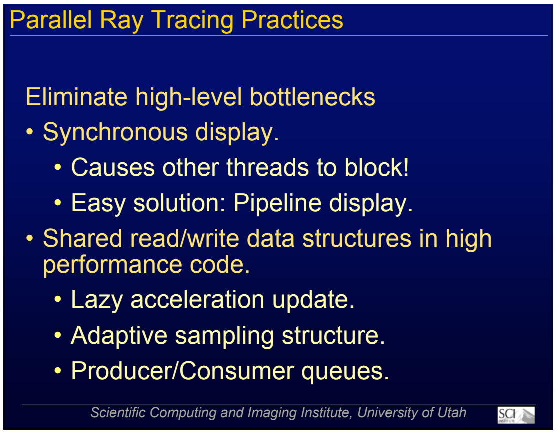


Need for Remote/Collaborative Rendering

Cost of a system usually justified by multiple users.

Collaborative visualization allows many users to interact with a large dataset, either locally or remotely.

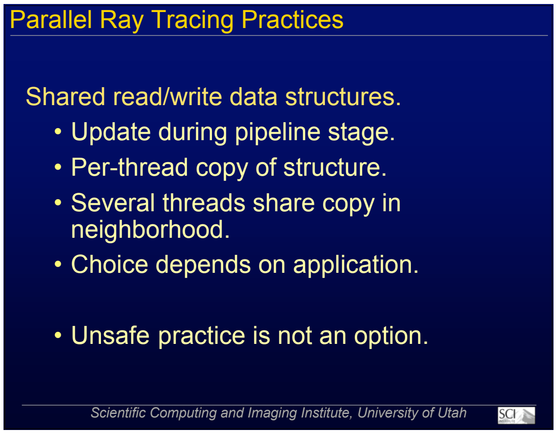
Bandwidth required for
1280x1024 @ 20 Hz = 100Mb/s



Parallel Ray Tracing Practices

Eliminate high-level bottlenecks

- Synchronous display.
  - Causes other threads to block!
  - Easy solution: Pipeline display.
- Shared read/write data structures in high performance code.
  - Lazy acceleration update.
  - Adaptive sampling structure.
  - Producer/Consumer queues.



Parallel Ray Tracing Practices

Shared read/write data structures.

- Update during pipeline stage.
- Per-thread copy of structure.
- Several threads share copy in neighborhood.
- Choice depends on application.

- Unsafe practice is not an option.

Eurographics 2006 Course Notes

Visibility-guided Rendering to Accelerate 3D Graphics Hardware Performance

Beat Bruderlin, Mathias Heyer, Sebastian Pfützner
Technical University of Ilmenau and 3DInteractive GmbH
www.3dinteractive.de

Hardware accelerators for 3D graphics (GPUs) have become ubiquitous in PCs and laptops. They are very powerful for real-time visualization of scenes up to a few million polygons at very high (almost photo-realistic) quality. GPUs have been successfully applied in computer games and engineering visualization. However, a straightforward use of GPUs, as is the current practice, can no longer deal with the ever larger datasets of complete and fully-detailed engineering models such as airplanes, industrial plants, cars, etc. This is a severe limitation in light of the data explosion, which we currently encounter in industry. Already some large models are 100 to 1000 times larger than what can be handled in real time by GPUs.

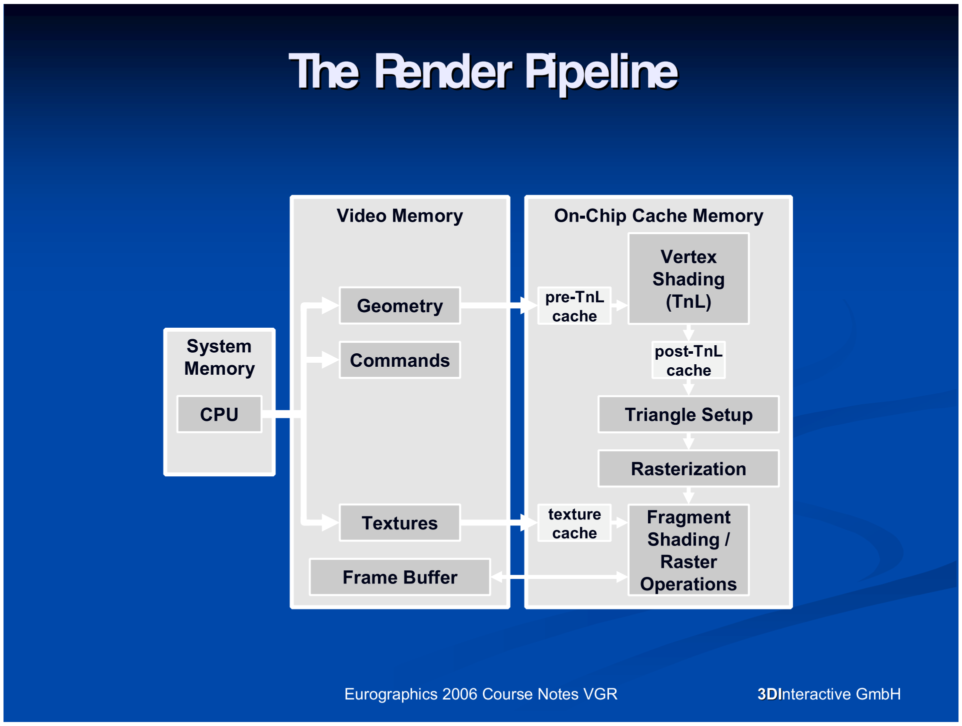
Visibility-guided Rendering (VGR) is a novel approach for real-time rendering of very large 3d datasets, overcoming the limitations of graphics hardware. The key advantage of VGR is to efficiently determine visibility of the vast majority of polygons before they are sent to the GPU. Different culling techniques are presented, which are used in combination. Spatial data structures, as well as hardware features of the GPU can be exploited to implement the culling techniques optimally. The visibility-guided rendering approach relies heavily on efficient memory management, concurrency between CPU and GPU, as well as optimal use of low-level OS functionality to handle very large data sets. The presentation concludes with a live demo of the VGR software.

The Classic Visualization Approaches: Raster Conversion and Ray Tracing

Before explaining the details of Visibility-guided Rendering we identify the main differences between two seemingly opposite classical rendering approaches at a high abstraction level:

- Sampling-based rendering (CPU-based Ray Tracing) and
- Rasterization-based rendering (i.e. Hardware-accelerated OpenGL)

In the following we try to explain some of the current limitations of GPUs and develop ideas for overcoming these limitations. In a simplified view of the render pipeline of a hardware-based rasterization approach all polygons of a scene are copied from the system memory into the video memory of the GPU, where they are transformed into the camera coordinate system and then converted into a pixel raster using a polygon filling algorithm. Only at the end visibility is determined on a per-pixel base (fragment operation), using the well-known z-buffer approach [SA03].



The Render Pipeline

System Memory

CPU

Video Memory

Geometry

Commands

Textures

Frame Buffer

On-Chip Cache Memory

pre-TnL cache

Vertex Shading (TnL)

post-TnL cache

Triangle Setup

Rasterization

texture cache

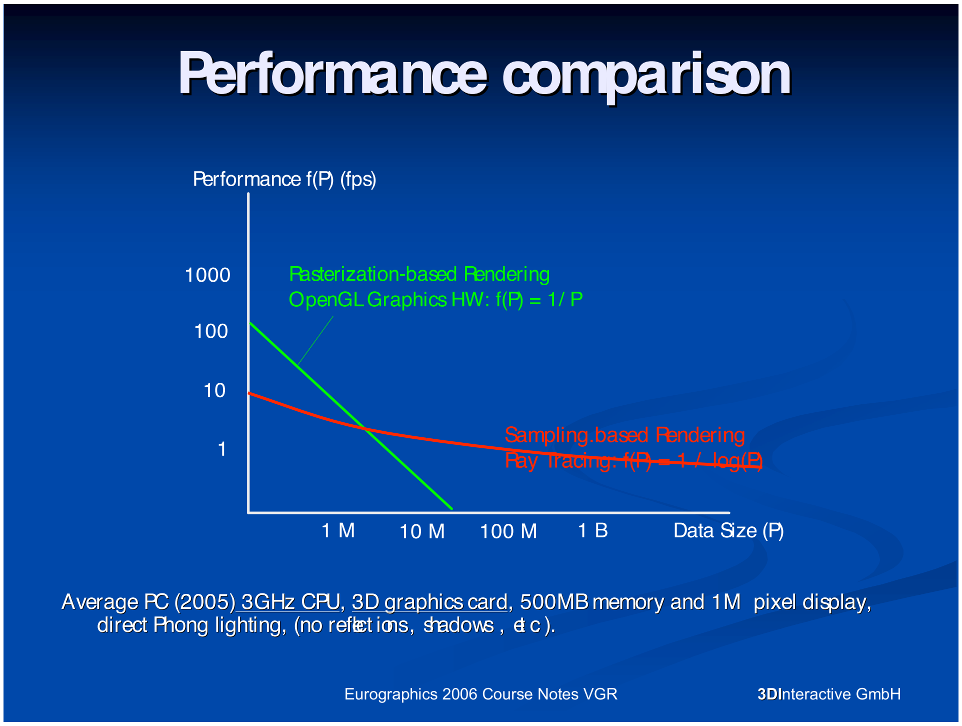
Fragment Shading / Raster Operations

Eurographics 2006 Course Notes VGR

3DInteractive GmbH

Understanding the process, it becomes clear that the amount of work for the render pipeline has to be roughly proportional to the number of polygons in the scene. The asymptotic behavior of this approach expressed in the so-called O-notation is O (n) (pronounced O of n) where n is the number of polygons. Therefore, the performance of the brute-force render pipeline approach (measured in frames per second) is inversely proportional to the number of polygons.

A sampling-based approach (such as ray tracing) works the other way round: A ray is cast from the camera origin through each pixel, and the intersection of the ray with the front-most polygon is determined. For each ray, a spatial search needs to be carried out among the polygons of the scene. Fortunately, efficient data structures and algorithms exist for this task which determine the visible polygon roughly in O(log n)-time per pixel for compact scenes or in O (cube_root(n) * log n)) time for “polygon soups”. The visibility of individual polygons (i.e. how many pixels of a polygon are visible) is determined as a side effect of this. To obtain the render performance for any scene we just need to divide the per-pixel performance by the number of pixels (roughly 1 million for a computer monitor). The theoretical (asymptotic) behavior expressed by the O-notation ignores constant factors. Therefore the over all asymptotic render performance will be the same as the per-pixel performance.



Performance comparison

Average PC (2005) 3GHz CPU, 3D graphics card, 500MB memory and 1M pixel display, direct Phong lighting, (no reflections, shadows, etc).

Eurographics 2006 Course Notes VGR 3DInteractive GmbH

Looking at the actual frame rate (see illustration), which means to also consider the constant hardware-specific factor, we observe that hardware accelerated Open GL performs better for small scenes with up to several hundred thousand to a few million polygons, each potentially generating many visible pixels on average. However, scenes with over 10 million polygons can no longer be rendered in real time, even with the fastest graphics cards. Beyond this limit, ray tracing clearly outperforms the rasterization approach. However, ray tracing on a single CPU of a PC or laptop is still not fast enough for real-time visualization.

This simplified view certainly ignores some of the shortcuts and optimization steps that can be
taken by each approach, as well as possible parallelization. On the other hand, neither of the theoretical analyses accounts for the additional difficulties occurring with very large scenes. - Just to mention the most obvious: Large scenes require over a hundred times more memory than the video ram of the graphics card provides, and over 10 times more than the physical main memory of most PCs. We need to take memory management and OS issues into account. Also the processors of the GPU have to be synchronized with the CPU (see also [Spi03], [WDS01).

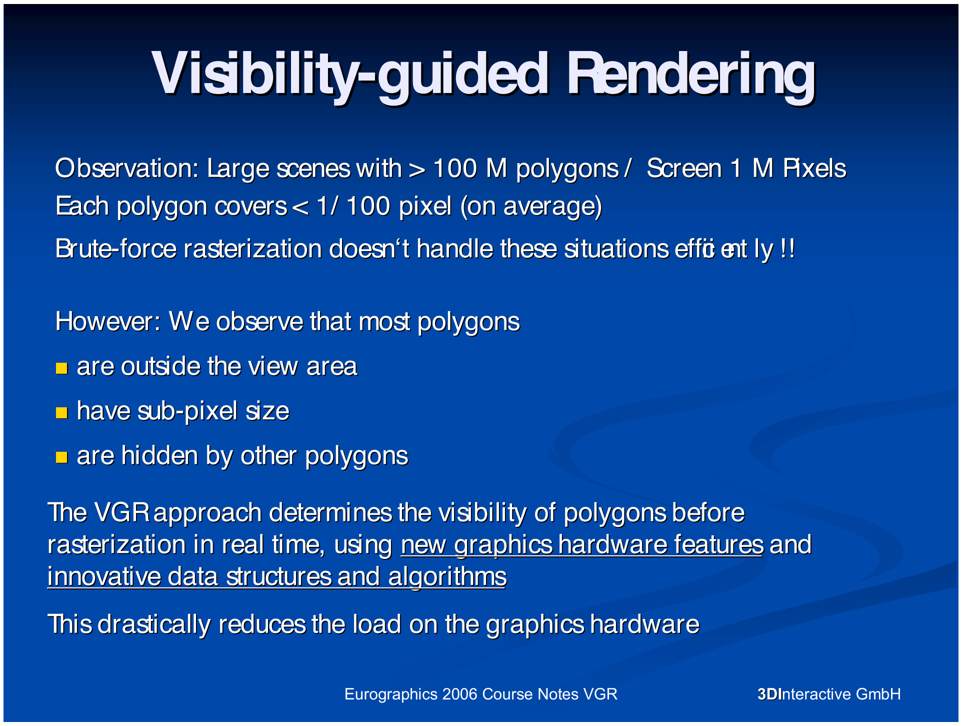
Visibility-guided Rendering / VGR

Prerequisites to VGR

After addressing the main pros and cons of the two classic approaches (rasterization and ray-tracing), we want to further analyze the rendering tasks and find a way out of the dilemma.

A key observation that led to visibility-guided rendering is that very large scenes contain often 100 million or more polygons. The number of pixels on the screen is limited usually to about one million or less. In other words, there are about 100 times more polygons in the scene than pixels on the screen. This, in turn, means, that the average footprint of a polygon (the number of pixels it leaves on the final image) is a small fraction of a pixel (less that 1/100 pixel). Rasterization does not deal with such situations very efficiently, as the render pipeline is optimized for real time rendering of 1000 times larger footprints (covering 100 pixels or more).

We may ask ourselves, what happens to individual the polygons of large scenes? Well, some are really very small and indeed only cover only a fraction of a pixel. Some polygons are hidden by other polygons, or they are outside of the view frustum.



Visibility-guided Rendering

Observation: Large scenes with > 100 M polygons / Screen 1 M Pixels

Each polygon covers < 1/ 100 pixel (on average)

Brute-force rasterization doesn't handle these situations efficiently !!

However: We observe that most polygons

- are outside the view area
- have sub-pixel size
- are hidden by other polygons

The VGR approach determines the visibility of polygons before rasterization in real time, using new graphics hardware features and innovative data structures and algorithms

This drastically reduces the load on the graphics hardware

Eurographics 2006 Course Notes VGR 3DInteractive GmbH

The main idea of Visibility-guided Rendering (VGR) [HeBr04],[HPB05] is to pre-determine the visibility of large parts of polygons before sending them to the GPU, and this way reduce the unnecessary overwork of the render pipeline.

Several known culling techniques ([AM00],[ISNM02],[BWPP04],[GH00],[HeBr04],[HPB05],
[HSLM02],[GSYM02],[ST99],[Sta03],[Zha98]) help the overall approach have been investigated in the literature. Occlusion culling can be used to determine groups of polygons that are hidden by visible polygons in the foreground. View frustum culling determines groups of polygons that are outside of the view frustum. Detail culling determines parts that are too small to contribute to the final image. Various level-of-detail approaches have also been proposed to replace high-polygonal scenes by low-poly data, that are accurate enough to give the same visual appearance at higher (real-time) frame rates (> 10 fps). Alternatively point rendering and stochastic approaches have been proposed [EMI00], [GKM93], [RuLe00],
[WFPHS01])

Individually these methods will significantly reduce the polygon count work in some situations, but not in others. Only a combination of them has the potential to work in a general situation. Many of the methods work well in theory. In practice, however, they generate additional overhead which cancels the benefit. In addition, many practical problems with data sizes exceeding the memory have to be considered as well.

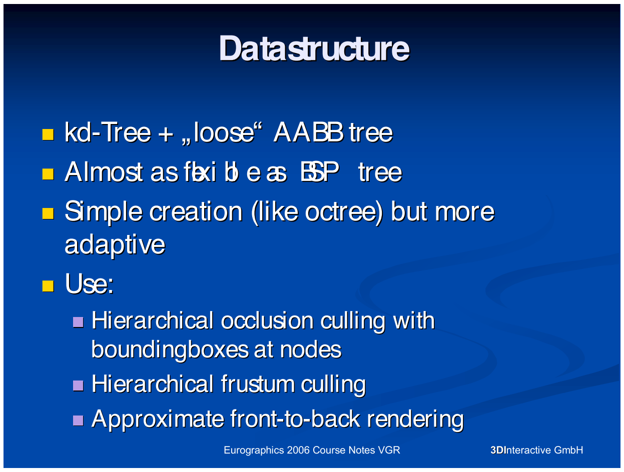
The new approach of visibility-guided rendering (VGR, [HeBr04],[HPB05]) uses several of these culling techniques in combination, and addresses the overhead issue in a novel way. We will not discuss all of the methods in detail, as some of them are relatively straight forward to implement. For instance, backface culling is supported by Open GL and needs no further explanation. It only makes sense for closed objects, for which it may reduce the load on the GPU by somewhat less than a factor of two on average. Mesh-based level of detail approaches (e.g. progressive meshes) have been ruled out because they only work with extensive preprocessing, which we tried to avoid (see design decisions) since precomputing also negatively impacts dynamic scenes. Detail culling works well for low density scenes. For high density scenes, culling small polygons which are part of a larger, connected surface would create visible holes. Fortunately these situations can be handled well by point rendering. Point clouds can be used as replacements for polygons that are smaller than one pixel on the screen. At larger distance, we can reduce the point sets in a relatively straight-forward way, maintaining the visual appearance. [RuLe00] [WFPHS01]

The culling techniques referred to above (detail culling, backface culling, LOD or point-based rendering) can be applied locally and do not depend on other parts in the scene. The most complicated culling technique is occlusion culling, since the visibility (or inversely the occlusion) of polygons is based on the interaction between objects. More concretely, an object is hidden, if it is hidden by at least one visible part of the scene. The “hidden” relation is idempotent. This means, if an object is hidden by hidden parts, it is doubly hidden, which for the viewer is just as hidden as once hidden ☺ To avoid superfluous occlusion tests we need some sort of spatial sorting, to test objects for occlusion front to back.

Below, we introduce a very useful spatial tree data structure and explain how it is used for occlusion culling. The same kd-tree is also used for view frustum culling and for hierarchical LOD.

The kd-tree: An efficient spatial data structure

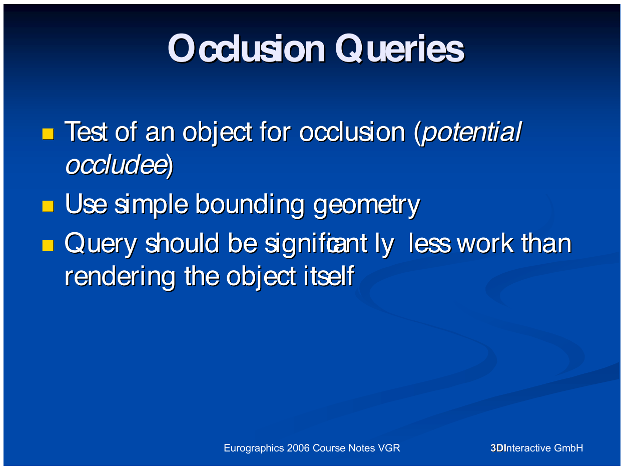
A kd-tree is a binary tree data structure, which at each level subdivides space in a potentially different spatial direction. At each intermediate node of the tree we store the range of data in the corresponding coordinate direction. From these ranges so called bounding volumes (e.g. bounding boxes, here symbolized by orange rectangles) can be derived which include all polygons of the tree underneath. At the leaf nodes pointers to OpenGL vertex buffers and index buffers reference polygons associated with that node [nVi03].



Datastructure

- kd-Tree + „loose“ AABB tree
- Almost as flexible as BSP tree
- Simple creation (like octree) but more adaptive
- Use:
  - Hierarchical occlusion culling with boundingboxes at nodes
  - Hierarchical frustum culling
  - Approximate front-to-back rendering

Eurographics 2006 Course Notes VGR 3DInteractive GmbH



Occlusion Queries

- Test of an object for occlusion (*potential occludee*)
- Use simple bounding geometry
- Query should be significantly less work than rendering the object itself

Eurographics 2006 Course Notes VGR 3DInteractive GmbH

Kd-trees support efficient spatial search queries, such as finding the closest (front-most) triangles to the camera, which have the highest probability of being visible. Also bounding box tests can be used to determine the visibility/ invisibility of groups of triangles hierarchically, and in view order, to effectively exclude a majority of polygons from being sent to the graphics card at all.
In addition, the same hierarchy can be used to represent levels of details. The initial scene consisting of triangles represents the highest level of detail, which is stored in the leaf nodes. At intermediate nodes we represent partial scenes by “point clouds.” which are a combination of the point sets of their two child trees. This way the kd-tree can efficiently derive all visibility information from spatial location and level of detail in real time during the rendering process.

Design Criteria

For the design of the overall system, several additional requirements on the implementation guided the design decision:

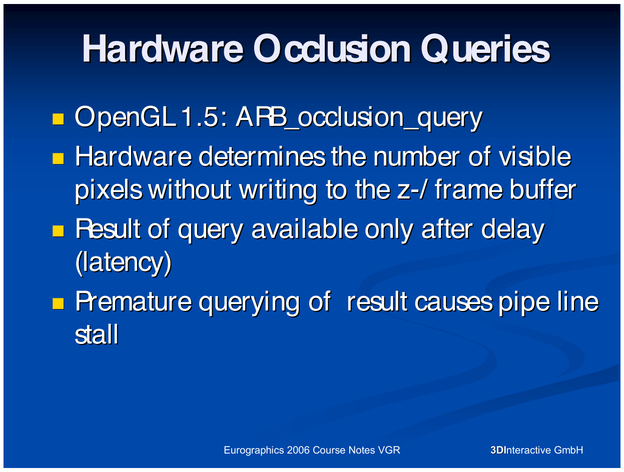
- We want to invest little or no pre-processing overhead, if possible. Most everything should be done in real-time during the rendering process itself.
- Rendering should also support real-time interaction and dynamic scenes.
- We would like to use standard hardware technology, i.e. standard PCs or laptops and standard hardware graphics hardware (64 – 256 MB VRAM)
- Also standard (32 bit) operating systems, and memory requirements (500 MB to 2 GB) should suffice.
- The approach should be scalable and extensible. It should:
    - Potentially take advantage of over 4GB, 64-bit addressable memory.
    - Potentially take advantage of programmable shader technology of state-of-the art graphics cards.
    - Potentially take advantage of multi CPU, multi GPU render servers to improve performance.
    - Potentially handle non-local illumination (specular or diffuse object interreflection, shadows) or global illumination in a hybrid approach using graphics hardware in combination with ray tracing.

Visibility-guided Rendering applies several culling methods in combination with innovative data structures and algorithms and advanced hardware features of graphics cards to dramatically improve the render performance of large scenes. It has been optimized to efficiently address all the requirements listed above.

In the following we present some details about the occlusion culling approach used in visibility-guided rendering. VGR relies heavily on efficient memory management, concurrency between CPU and GPU, as well as optimal use of low-level OS functionality to handle very large models. Ultimately we can show that spatial algorithms and data structures, in combination with the performance of modern graphics cards can achieve similar asymptotic behavior as ray tracing with the additional advantage of real time behavior, due to massive parallel compute power of modern graphics cards.

Occlusion culling:

Occlusion queries can be used to determine potentially visible groups of polygons before sending them to the render pipeline. We can test simple bounding geometry (e.g. bounding box) for visibility. A key prerequisite of the Visibility-guided Rendering approach is an efficient spatial data structure (kd-tree introduced above) to support the visibility determination of large scenes.

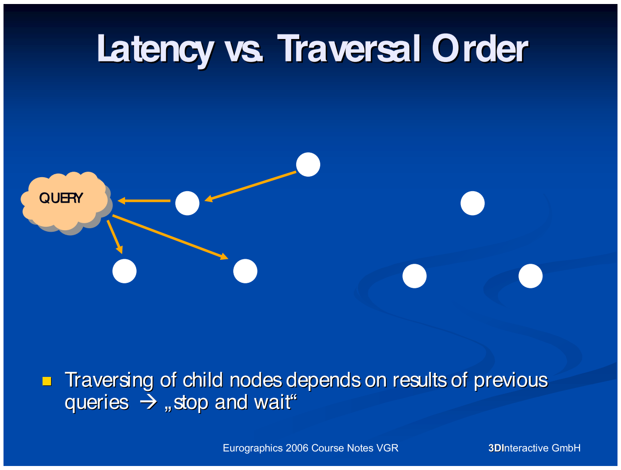


Hardware Occlusion Queries

- OpenGL 1.5: ARB_occlusion_query
- Hardware determines the number of visible pixels without writing to the z-/ frame buffer
- Result of query available only after delay (latency)
- Premature querying of result causes pipe line stall

Eurographics 2006 Course Notes VGR

3DInteractive GmbH



Latency vs. Traversal Order

- Traversing of child nodes depends on results of previous queries → „stop and wait“

Eurographics 2006 Course Notes VGR

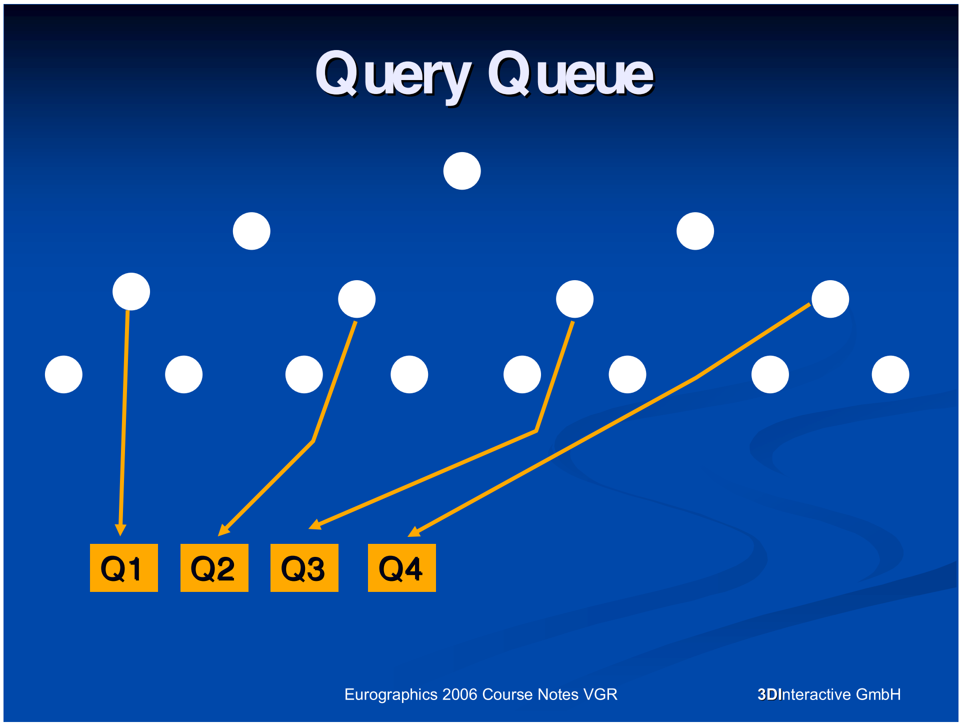
3DInteractive GmbH

The graphics hardware (GPU) can be employed, not only to render the polygons, but also to determine the visibility of bounding boxes, using the “ARB-occlusion-query” functionality [SA03]. This hardware feature determines how many pixels of any sequence of OpenGL render tasks would be visible, without actually setting the pixels in the frame buffer.

The visibility of polygon groups is carried out in the order of the view direction (z-direction) of the camera. Once the front most polygon groups have been drawn, we test the visibility of the potentially hidden bounding boxes of polygon groups further away. The visibility of bounding boxes is also determined hierarchically (top down). The children of an invisible node can be declared invisible without checking their bounding boxes explicitly. None of their polygons need to be rendered. Polygons associated with leaf nodes with visible bounding boxes are rendered entirely. The visibility of polygons further away may change from visible to hidden, after more polygons have been rendered, but never the other way round. Therefore it makes sense to intersperse rendering and occlusion tests for each frame.

A potential problem of this approach is that the GPU determines the visibility of objects asynchronously from the CPU which has to control the render tasks on the GPU and transfer the data from main memory to VRAM. In essence the GPU and the CPU have to collaborate closely, as is demonstrated in this illustration

As is often the case with parallel processing, latency occurs because of one processor having to wait for the other. A straight forward use of the visibility determination with hardware occlusion check and spatial queries results in a stop-and-go behavior and pipeline stalls of the graphics card.

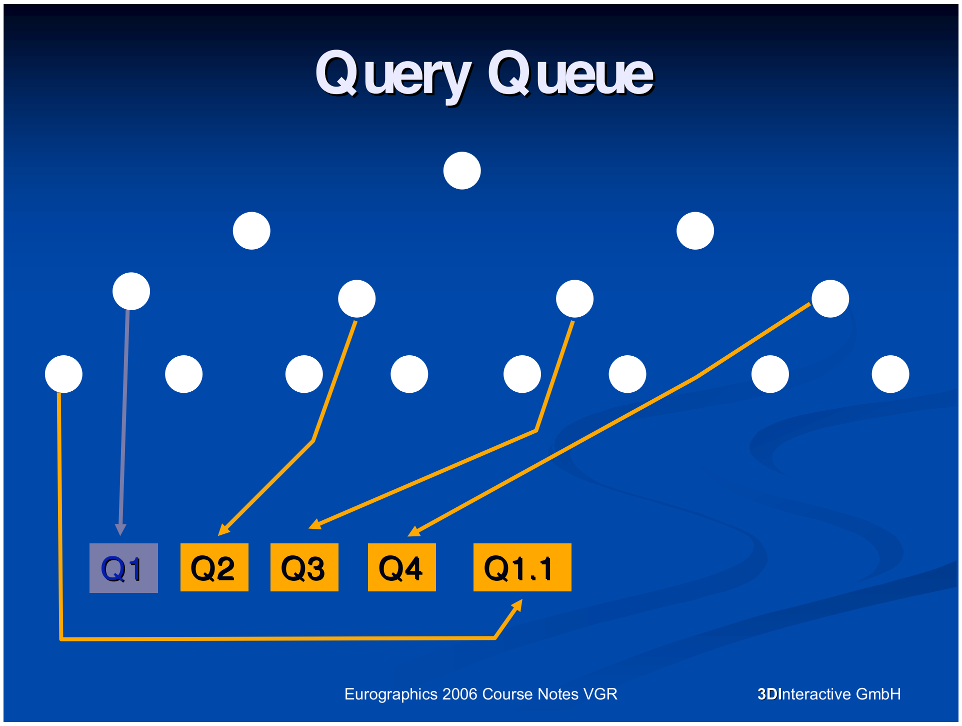


Query Queue

Eurographics 2006 Course Notes VGR

3DInteractive GmbH

To avoid the latency problem we devise a pipeline data structure (priority queue) to communicate between the kd-tree data structure controlled by the CPU and the render tasks an occlusion query handled by the GPU. The tree is traversed top down, front to back in depth-first order, in combination with a breadth-first traversal. The bounding boxes of the intermediate nodes result in occlusion queries that are pushed onto the queue and handled by the GPU in order of distance to the viewer (z-direction).

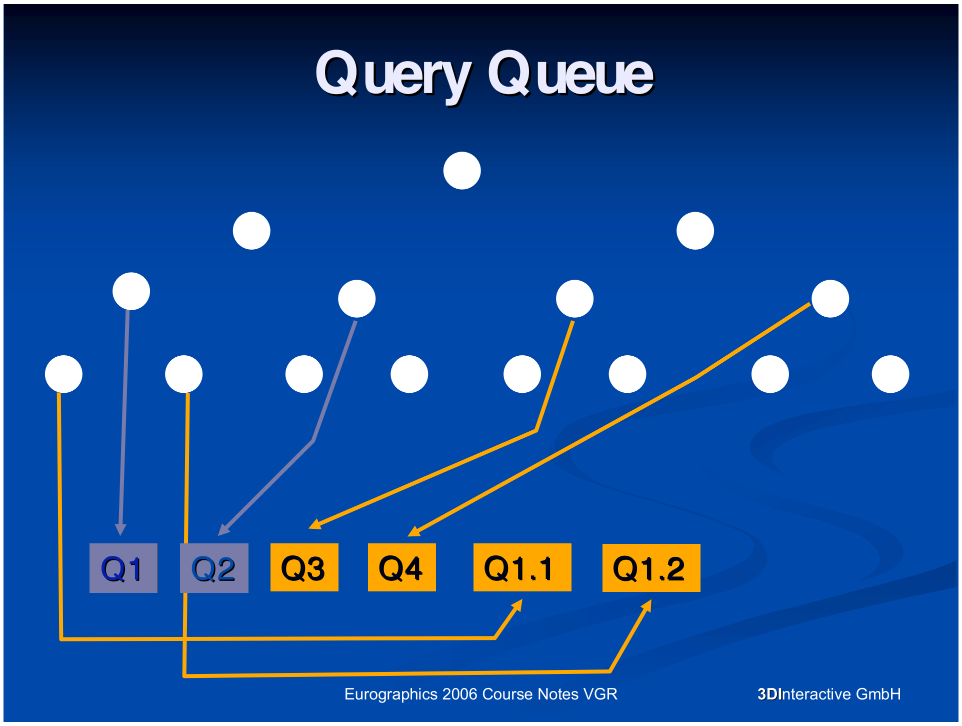


Query Queue

Eurographics 2006 Course Notes VGR

3DInteractive GmbH

After a certain delay the first visibility results are available to the CPU. Depending on the visibility of the parent nodes the algorithm generates new queries for the child nodes of visible parent nodes. If leaf nodes are visible, the corresponding polygons will be drawn. The asynchronous execution of producing queries by the CPU and consuming them by the GPU avoids the previously described pipeline stalls. The length of the query queue can be adapted such that there ideally is no wait for query results; there is always something to do for the GPU.



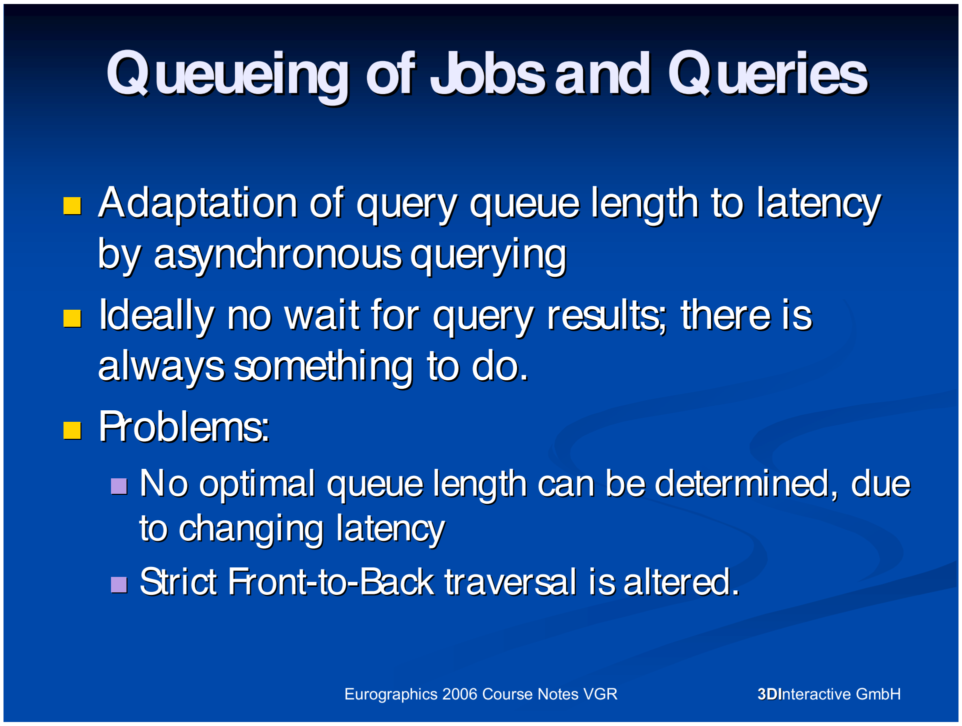
Query Queue

Eurographics 2006 Course Notes VGR

3DInteractive GmbH

There are some potential problems, however:

- There is no a-priory optimal queue length due to changing latency and task complexity. In our approach the length of the queue is adopted dynamically in accordance with actual latency during rendering.
- The strict front-to-back tree traversal (depth-first traversal) is altered due to the delay of the queue. This leads to less than optimal render actions with more overdraw. Possibly the visibility of some leaves will be determined before polygons in front of them have been rendered (false positives). In practice this overdraw problem is negligible, however, compared to the problem of pipeline stalls that would otherwise occur.



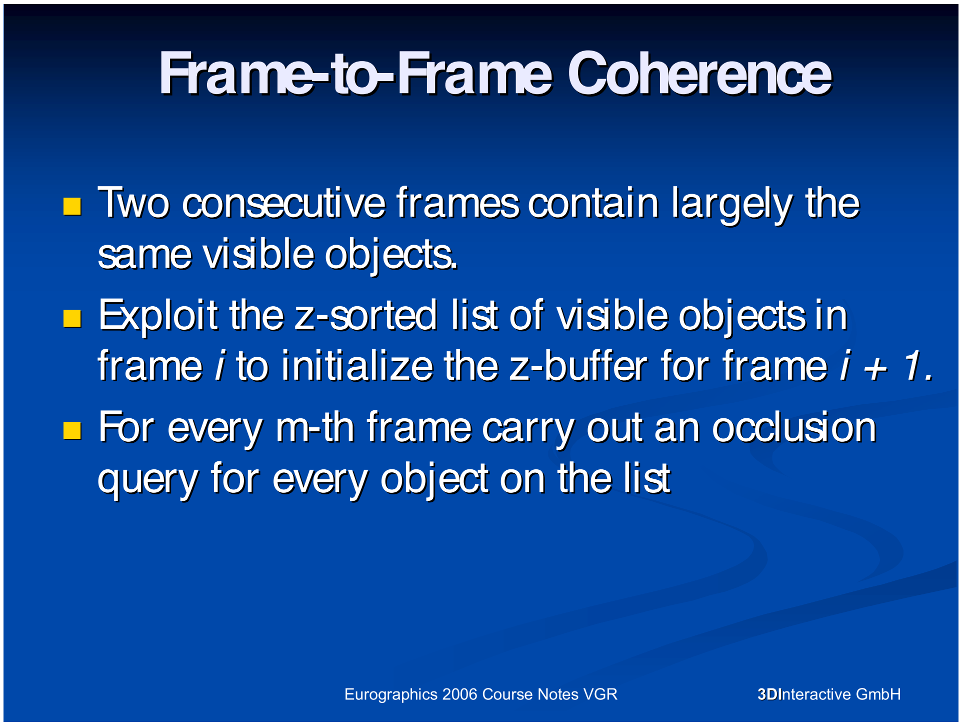
Queueing of Jobs and Queries

- Adaptation of query queue length to latency by asynchronous querying
- Ideally no wait for query results; there is always something to do.
- Problems:
  - No optimal queue length can be determined, due to changing latency
  - Strict Front-to-Back traversal is altered.

Eurographics 2006 Course Notes VGR 3DInteractive GmbH

There are many more issues due to synchronization between GPU and CPU, but also additional properties of the space time continuum we can take advantage of. One such issue is temporal coherence (or the close similarity in visibility between consecutive frames), which will be discussed next.

Exploiting Temporal Coherence

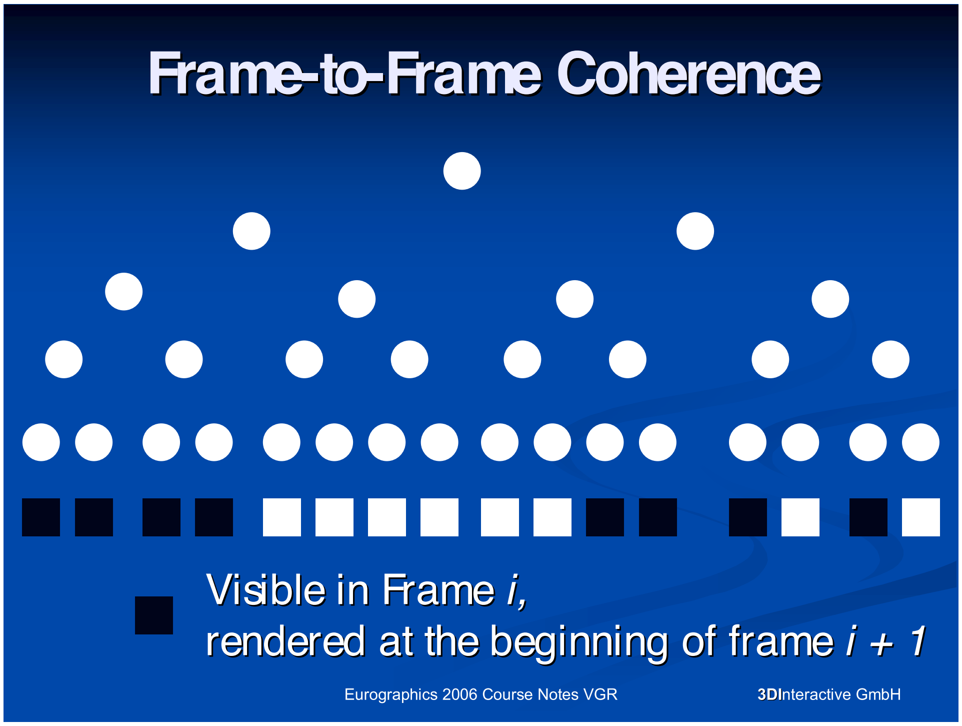


Frame-to-Frame Coherence

- Two consecutive frames contain largely the same visible objects.
- Exploit the z-sorted list of visible objects in frame *i* to initialize the z-buffer for frame *i + 1*.
- For every m-th frame carry out an occlusion query for every object on the list

Eurographics 2006 Course Notes VGR 3DInteractive GmbH

It is very likely that in two consecutive frames largely the same triangles are hidden or visible, respectively. Only very few new ones become visible and very few become hidden from one frame to the next. This temporal coherence is exploited by first rendering the previously visible nodes in each frame, without an occlusion check, to avoid overhead. These nodes are rendered front to back (z-sorted) to minimize overdraw. Then we perform an occlusion test for potentially newly visible nodes. At last a visibility test for a fraction of the initially rendered nodes is carried out, to verify if they are really still visible. This guarantees that potentially visible nodes are rendered immediately, however invisible nodes are not immediately discarded, but will be after m frames.



Frame-to-Frame Coherence

■ Visible in Frame *i*,
rendered at the beginning of frame *i + 1*

Eurographics 2006 Course Notes VGR

3DInteractive GmbH

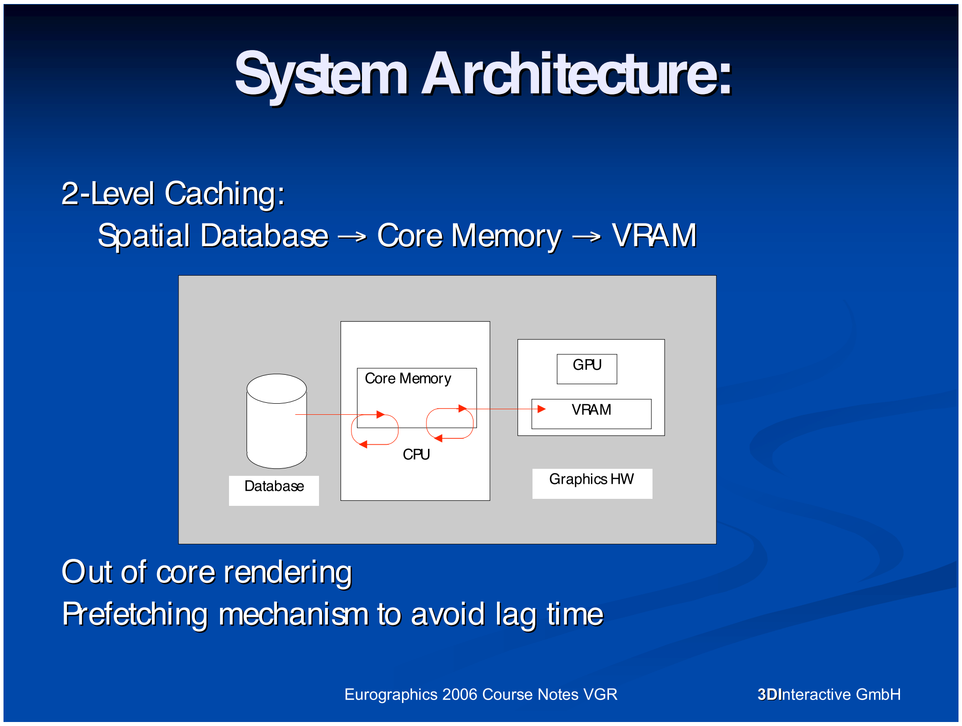
To further increase efficiency due to temporal coherence, the invisible nodes (white) and the visible nodes (black) have been marked in the previous frame. The visibility of a parent of one visible and one invisible subtree is considered undecided (the bounding box is visible) . These undecided nodes are collected in a list, which we call visibility band. At the beginning of each frame, after rendering the previously visible leaves, we start the visibility check at the visibility band, in the front-to-back, top-down order described above (pushing the queries onto the queue, rendering the visible leaves, falling off the end). As mentioned above, we only descend down the invisible subtrees in each frame and increase the counter for the roots of the visible subtrees. Only if the counter exceeds a predetermined value m, a visibility check is performed for that node as well.

Out-of-Core RenderingMemory Issues:

VGR employs different memory management approaches for pre-processing and rendering. Rendering poses higher real-time requirements for out-of-core memory management compared to the requirements of pre-processing. More complex strategies are therefore required.

Some of the issues are:

- The database size may exceed memory size by orders of magnitude
  - Eg. The Boeing 777, with 350.000.000 triangles, compact representation with simple vertex colors, no normals: 8 GB, with normals: 12 GB, other Models: over 30 GB.
  - Available system memory, typically 1 – 1.5 GB (about 1/10 of the scene)
  - Graphics memory (VRAM) 100 – 200 MB (i.e. 1/100 of the largest models)
  - Often only about 1/1000 of the polygons are visible and need to be rendered (e.g. 1 million triangles out of 1 billion in the scene)
- Data management is necessary at several levels (VRAM, System Memory, Disk)



System Architecture:

2-Level Caching:
Spatial Database → Core Memory → VRAM

Out of core rendering
Prefetching mechanism to avoid lag time

Eurographics 2006 Course Notes VGR 3DInteractive GmbH

Fast spatial queries can be enabled by memory-coherent data organization. The latency during data access on disk can be reduced by prefetching. Latencies at read time are not acceptable, therefore asynchronous loading of geometry is required. This results in approximate (preview quality) rendering during load times (see below). The goal is the fastest-possible image development during loading of geometry from the database by means of adequate loading, replacing and prefetching strategies.

Prefetching:

The data of a scene resides in a database on the hard disk. It is represented by a kd-tree, as previously described. The data is loaded to systems memory by an asynchronous process in a priority order. The leaves marked visible have the highest priority. Next come the hidden parts of the scene that are within the view frustum. A low priority is attributed to scene parts that are outside of the view frustum, followed by leaf nodes below the so called LOD band, which have the lowest priority. – The LOD band is, similar to the visibility band (see above), a list of sub-tree nodes that contains details that are smaller on the screen a given threshold (e.g. 1 pixel). This way we can prioritize parts of the scene that have the biggest visual impact and load small details later. Whenever there is time, additional scene parts are loaded to preempt future camera motion.

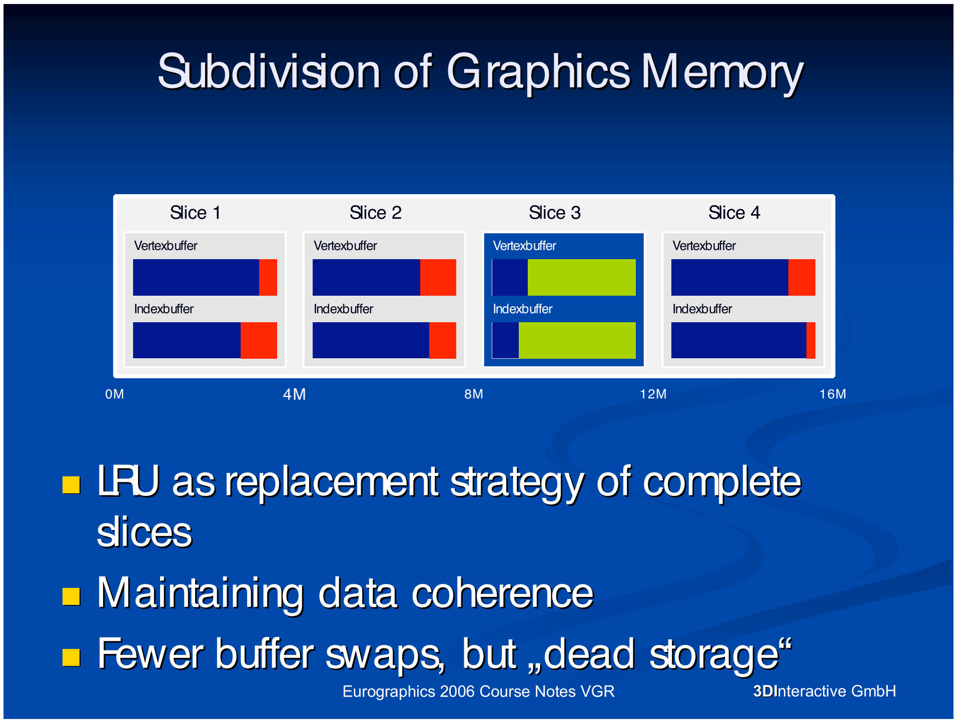
- In each frame, there is potentially a different distribution of priorities. However, due to the temporal coherence from frame to frame, only a few changes need to be applied simultaneously.
- The prefetching-and-replacement strategy tries to keep as many leaf nodes as possible in main memory, starting at the leaves with the highest priority.
- Leaves with lower priority that have stayed in memory for the longest time will be removed from the systems memory first, such that only the data that is important for the actual view is kept in memory.
- Leaf nodes marked as “visible” will never be removed from memory, to avoid flickering.

System Memory Management

During preprocessing geometry was subdivided into chunks of approximately equal size. For rendering, a chunk is either completely loaded into system memory or not at all (atomicity). Since chunks all have about equal size, there will be hardly any fragmentation, and no special system memory caching structure was employed for rendering.

VRAM Management

In addition to systems memory, also the graphics memory (VRAM) needs to be managed. VRAM is usually much smaller than main memory. Therefore more often geometry needs to be replaced, which could lead to more fragmentation.



Subdivision of Graphics Memory

- LRU as replacement strategy of complete slices
- Maintaining data coherence
- Fewer buffer swaps, but „dead storage“

Eurographics 2006 Course Notes VGR **3DI**nteractive GmbH

Management of data in VRAM is done with OpenGL Vertex- and Index buffers. If a leaf node has its own buffer, it needs to be swapped after each frame, which is an expensive operation. (app. 1000 – 5000 times per frame).

Subdivision of VRAM in a few large slices is theoretically better, however it causes an internal fragmentation by continuous replacement. Fragmentation is avoided by always filling a slice from front to back, in render priority (only visible nodes are in VRAM).

If there are no more free slices, a decision is made using the LRU (least recently used) strategy, as to which slice can be completely emptied and refilled. Several nodes are replaced at once in VRAM, and new nodes are entered again in the order in which they are rendered. This maintains the coherence of data and finally, this accelerates rendering.

> Advantage: Fewer buffer swaps
>
> Disadvantage: unused, “dead” memory at the end of each slice (app. 5% – 10% of VRAM)

Data is loaded from main memory into VRAM. This causes a slow down during the first frames after loading, but is hardly noticeable later on.

The Role of Preprocessing

The task of pre-processing is to hierarchically and spatially subdivide the scene into sets of roughly 1000 to 8000 triangles, which are stored as the leaves of a kd-tree. This data structure enables the efficient data access for VGR frustum culling, detail culling and occlusion culling.

For VGR pre-processing mostly consists of a simple spatial sorting. No complex data reduction schemes or progressive mesh generation is necessary.

The top-down generation of a kd-tree for very large scenes requires an out-of-core strategy. Parts of geometry data potentially are used several times during the tree generation in different places of the algorithm. Therefore, in principle, data needs to be loaded multiple times. This causes problems with swapping data in and out of systems memory and fragmentation, which are addressed in the next sections. Properly addressing or avoiding these problems leads to a faster pre-processing. Basically the pre-processing time grows asymptotically with O (n log n).

Examples of actual pre-processing times:

- Power plant (12,7 Mio. triangles) in 1.5 min
- The Boeing 777 (350 Mio. triangles) in 70 min.

Generally, the pre-processing time is comparable to data conversion and copying time of other systems.

Swapping

To handle large data sets also during preprocessing an out-of-core approach hat to be taken. This means that data is only read into the system memory temporarily. Two swapping mechanisms have been tested and compared for pre-processing (memory mapped files & explicit swapping of geometry chunks). For 64 bit architectures, theoretically one could have relied entirely on the operating system to handle the swapping. However this was considered as too inefficient)

Memory-mapped files make use of optimized swapping strategy of the operating system:

- Principle: The virtual address space of a process is subdivided into pages (usually 4 kB in size) which can reference pages in main memory, external devices, or on the hard disk.
- Linear view on all data simultaneously. Access via 64-bit indices. Requesting of the necessary sizes in megabytes.
- Storage may be requested and modified by means of defined windows on the data that are mapped to systems memory (memory-mapped files)
- Memory can be read or written transparently. Altered pages will automatically be written to the hard disk.
- By only using one storage window at a time, the swapping of data is

completely managed by the operating system, which reduces fragmentation of the address space.

Problem of memory-mapped files:

- Swapping strategies of the operating systems are kept general and are not optimized for the specific application case.
- Swapping is done only when the system memory fills up and memory requests can no longer be met.
- For instance under MS-Windows only single pages are written out and immediately afterward other pages are read back in. This leads to inefficient I/O behavior.
- The linear view does not fit well with the hierarchical organization of the geometry. This requires continuous copying to maintain coherence.

A better strategy: Explicit memory management that reflects the hierarchical subdivision of data, and also controls read and write actions.

- For every node of the tree that is being generated (i.e. in every recursion step) the complete subtree data needs to be read, handled and written back to disk (which amounts to the complete data set once per level of recursion)
- If the necessary data is smaller than the available systems memory, the complete leaf data can be handled in core.
- Otherwise, the subdivision has to be carried out in stages, where only a subset what fits into memory can be handled simultaneously.

Advantage over the memory-mapped file approach:

- Subdivision of geometry data into chunks that can be handled completely in core.
- Acceleration of pre-processing by up to a factor of 10.
- Swapping of data can be done highly efficiently due to linear access of hard drive (average transfer rates of 50MB/s (vs. vs. only 1 - 5 MB/s with memory mapped files and linear view on data)
- Enables asynchronous writing of data during generation. This further accelerates the process.

Fragmentation

Fragmentation of memory may lead to denial of requests for free memory of certain size despite the availability of free memory, because the gaps are not consecutive. We distinguish between external (system memory/disk) fragmentation and internal fragmentation (inside cache memory).

In our implementation all geometry data of individual objects and fragments generated during subdivision are managed by several caches of fixed size in system memory. This way geometry data no longer fragments the memory (only the management overhead data itself does) Geometry data may fragment the caches themselves, though. However, this can be mitigated by explicit cache de-fragmentation.

On 64-bit architectures the virtual memory is so large, that fragmentation does not matter, and caching can be omitted. Geometry data is not compacted on the hard disk. Therefore no fragmentation is possible there.

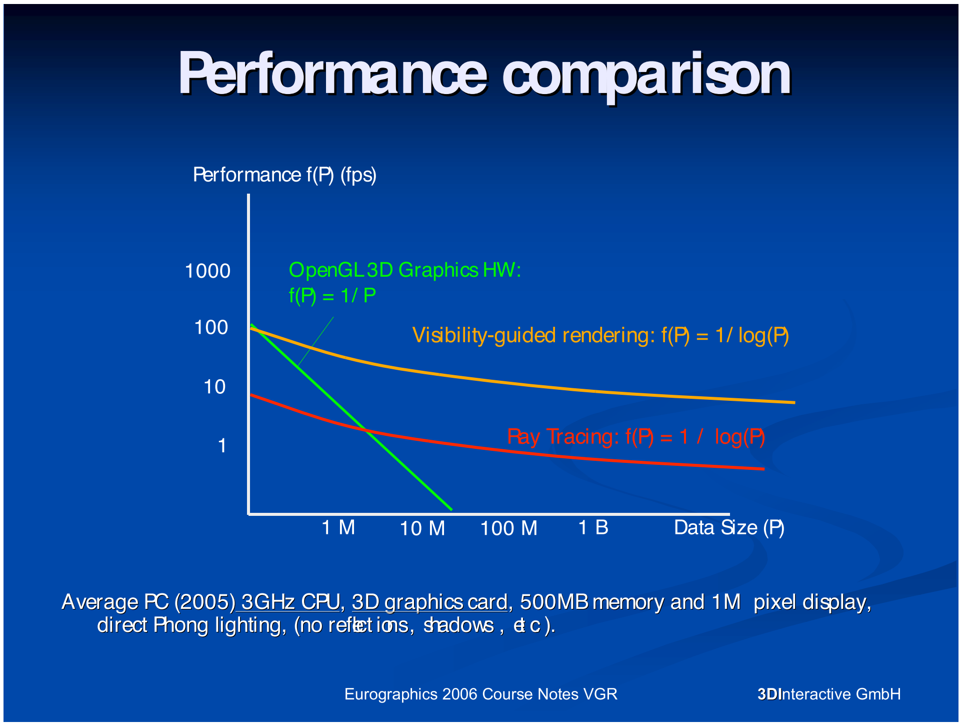
The top-down generation of multi-dimensional hierarchies (kd-trees) promotes coherent data organization on the hard drive (in contrast to rendering, where data is needed in more or less random order) Separate storage of geometry data left and right of the separating plane prevents external fragmentation.

Data are read, handled and deleted in the order in which they were also written. This leads to large gaps that can be utilized well for the final files. As soon as the data of a node completely fits into memory, no more swapping operations are necessary. Therefore no gaps in the swap space smaller than the size of the available system memory can be generated. By preventing fragmentation the available memory can be used more efficiently. This enlarges the useful memory and accelerates the preprocessing.

Conclusion

Visibility guided rendering is well suited for real-time visualization of extremely large engineering CAD models. An example is the CAD model of the Boeing 777, which consists of about 350,000,000 triangles. The interviews3D navigator, which was built on top of the VGR technology allows real-time navigation through the model, as well as interactive picking and editing of any of the approximately 700,000 objects, down to individual screws or washers. Similarly interactive behavior was observed with even larger models that contain over 1.3 billion polygons.

The render performance of VGR behaves asymptotically similar to ray tracing (it is also based on searching in a spatial data structure) but with faster constant factor, thanks to a fast GPU hardware (due its high fill rate and hardware occlusion tests).



Performance comparison

Average PC (2005) 3GHz CPU, 3D graphics card, 500MB memory and 1M pixel display, direct Phong lighting, (no reflections, shadows, etc).

Eurographics 2006 Course Notes VGR

3DInteractive GmbH

The acceleration of VGR over brute-force hardware rendering was achieved by a combination of various culling techniques. Previous limitations of these techniques were overcome by eliminating the overhead as much as possible (usually due to latency and overwork due to exact polygon testing). Efficient memory management and out-of-core rendering are essential to the handling of extremely large models.

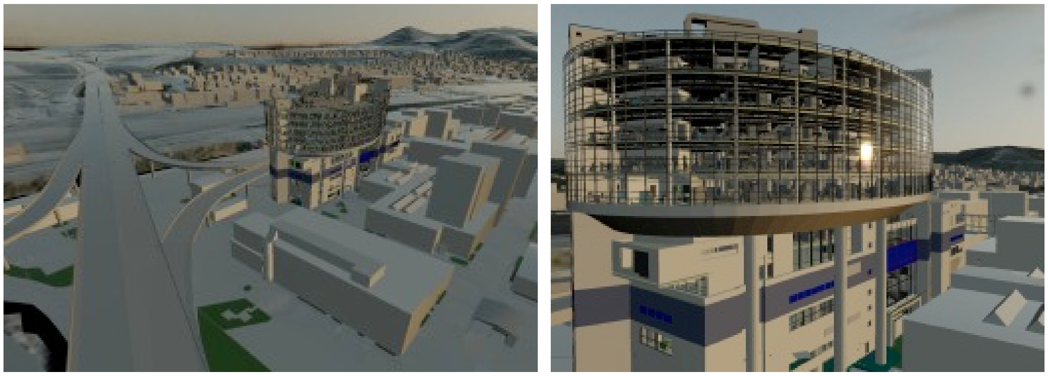
Taking the temporal coherence between consecutive frames into account, we can expect almost constant frame rates, independent of model size. VGR is an output-sensitive approach., i.e. the amount of work depends on how much is ultimately visible on the screen. Since the resolution of a screen is constant (e.g. 1 million pixels) the number of visible polygons cannot actually exceed the number of pixels. -- That’s the theory. In practice we can never achieve100% accuracy, unless we spend a lot of time on preprocessing. It turns out to be faster to deviate from the theoretic ideal, by grouping larger sets of triangles together and make only approximate, yet conservative visibility decisions. Certainly, this generates more overdraw, which has to be handled by the fragment shader of der graphics hardware. However, this is more efficient since the parallel architecture of the graphics hardware is geared towards high polygon and pixel throughput, but does not behave well, if constantly interrupted by many small but costly render queries.

The requirements established in the design decisions have been met:

- Real-time visualization of large models on standard PC and laptop hardware.
- Very large models can be handled by efficient out-of-core rendering and out-of-core pre-processing.
- Preprocessing is very efficient: It relies on simple spatial sorting for which fast approaches exist.
- VGR can be used in combination with programmable pixel and vertex shaders,
- Even hybrid ray tracing (vertex tracing, a combination of hardware rendering and ray tracing, [Prei04] [UlPrBr03]) and other rendering methods (e.g. shadow maps, spherical harmonics for real-time soft shadows) and high dynamic range imaging, etc. can be supported by VGR. This is plausible, since the actual render back-end is based on conventional OpenGL rendering, and all extensions to OpenGL can be applied as well. These issues are not elaborated here any further. Instead, we show some sample screen shots below. For further examples and on-line videos, see: www. 3dinteractive.de

hier entsteht Ihre neue E-Klasse
hier entsteht Ihre neue E-Klasse
DAIMLERCHRYSLER
Start

Real-time rendering of variations of the factory model of Daimlers new E-Class model shown in different lighting situations. Courtesy of DaimlerChrysler Digital Factory Planning.



A digital model of DaimlerChrysler's newly built Van Technology Center seen in its environment in Stuttgart-Untertürkheim (based on satellite data). High-dynamic range imaging environment maps and pixel shaders for the reflective and transparent glass surface have been tested successfully in combination with VGR. Courtesy of DaimlerChrysler Digital Factory Planning.

Literature

Culling techniques/visibility-guided rendering:

[AM00] Ulf Assarsson and Tomas Möller. Optimized view frustum culling algorithms for bounding boxes. Journal of graphics tools,, pages 9-22, 2000.

[ISNM02] W.V. Baxter III, Avneesh Sud, N.K.Govindaraju, and Dinesh Manocha. Giga-Walk: Interactive Walkthrough of Complex Environments. Technical report, University of North Carolina at Chapel Hill, 2002.

[BWPP04] J Bittner, M. Wimmer, H. Piringer, and W. Purgathofer. Coherent hierarchical culling: Hardware occlusion queries made useful. In proceedings of Eurographics Conference, 2004.

[GH00] Gil Gribb and Klaus Hartmann. Fast extraction of the frustum planes from the world-view-projection matrix, June 2000. http://www2.ravensoft.com/users/ggribb/plane%20extraction.pdf.

[HeBr04] M. Heyer, B. Brüderlin, Visibility-guided Rendering for Visualizing Very Large Virtual Reality Scenes, in proceeding 1st GI-Workshop on VR/AR TU-Chemnitz, September 2004 - (see also www.3dinteractive.de)

[HPB05] M. Heyer, S, Pfützner, B. Brüderlin, Visualization Server for Very Large Virtual Reality Scenes, in proceedings, 4th Workshop on Augmented & Virtual Reality in der Produktentstehung, Paderborn, June 2005, Hanser Verlag 2005.

[HSLM02] K.Hillesland, B.Salomon, A.Lastra, and D.Manocha. Fast and Simple Occlusion Culling using Hardware-Based Depth Queries. Technical report, University of North Carolina at Chapel Hill, 2002. further information available at: http://www.cs.unc.edu/salomon/Research/FSOC/.

[GSYM02] Naga K. Govindaraju, Avneesh Sud, Sung-Eui Yoon, and Dinesh Manocha. Interactive Visibility Culling in Complex Environments using Occlusion Switches. Technical report, University of North Carolina at Chapel Hill, 2002.

[ST99] Dieter Schmalstieg and Robert F. Tobler. Fast Projected Area Computation for Three-Dimensional Bounding Boxes . Journal of Graphics Tools, pages 37-43, 1999.

[Sta03] D. Staneker. An Occlusion Culling Toolkit for OpenSG PLUS. Technical report, WSI/GRIS University of Tübingen, Germany, 2003.

[Zha98] Hansong Zhang. Effective Occlusion Culling for the Interactive Display of Arbitrary Models. PhD thesis, University of North Carolina, 1998.

Alternative level of detail and point rendering techniques:

[EMI00] Carl Erikson, Dinesh Manocha, and William V. Baxter III. Hlods for faster display of large static and dynamic environments. Technical report, University of North Carolina at Chapel Hill, 2000.

[GKM93] Ned Greene, Michael Kass, and Gavin Miller. Hierarchical Z-Buffer Visibility. Computer Graphics, SIGGRAPH ´93 Proceedings:pp. 231-238, August 1993.

[RuLe00] Szymon Rusinkiewicz, Marc Levoy. QSplat: A Multiresolution Point Rendering System for Large Meshes, SIGGRAPH 2000

[WFPHS01] M. Wand, M. Fischer, I Peter, F.M. auf der Heide and W. Strasser. The randomized z-buffer algorithm: Interactive rendering of highly complex scenes. In SIGGRAPH 2001 Proceedings, 2001.

Open GL

[nVi03] nVidia. Using Vertex Buffer Objects. WWW, 2003. http://developer.nvidia.com/object/using VBOs.html.

[SA03] Mark Segal and Kurt Akeley. The OpenGL Graphics System: A Specification (Version 1.5), October 2003.

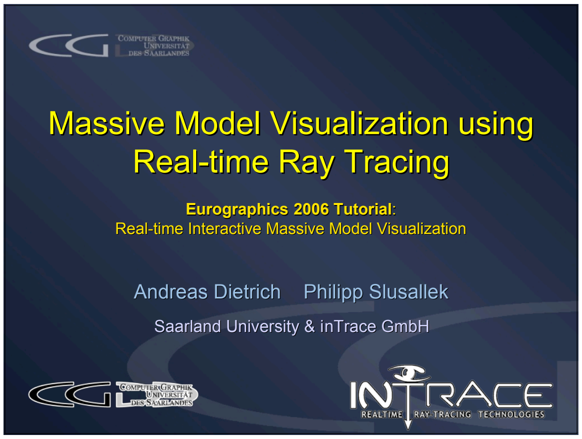
[Spi03] John Spitzer. OpenGL Performance Tuning. Game Developers Conference, 2003.

Real-time ray tracing, vertex tracing & global illumination:

[Prei04] Th Preidel, Global Illumination for Vertex Tracing (in German), Diplomarbeit TU Ilmenau 14. Juni 2004

[UlPrBr03] Th. Ullmann, Th. Preidel, B. Bruderlin, "Efficient Sampling of Textured Scenes in Vertex Tracing," (short presentation) in proceedings of Eurographics Conference, 2003.

[WDS01] I. Wald., A. Dietrich, and P. Slusallek. An interactive out-of-core rendering framework for visualizing massively complex models. In Proc. Eurographics Symposium on Rendering, 2004.

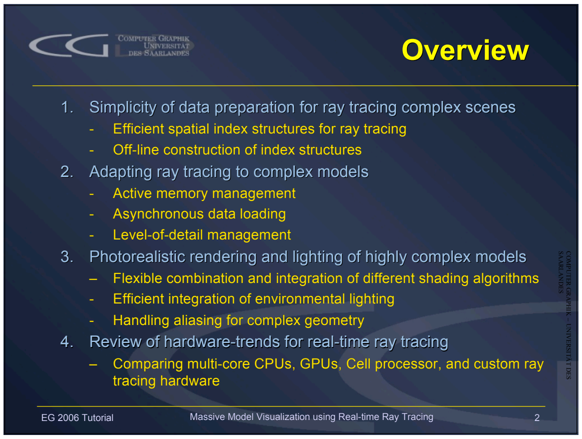


Massive Model Visualization using Real-time Ray Tracing

Eurographics 2006 Tutorial:
Real-time Interactive Massive Model Visualization

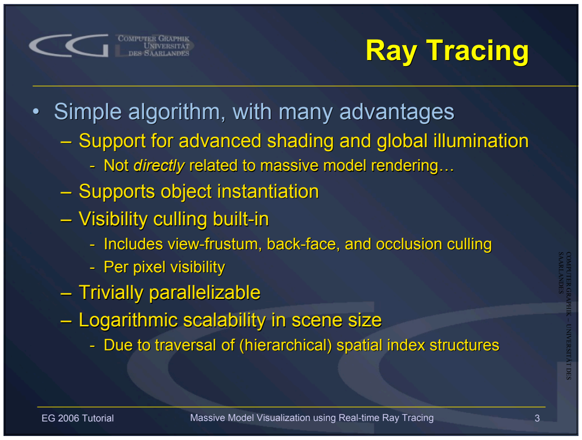
Andreas Dietrich Philipp Slusallek

Saarland University & inTrace GmbH



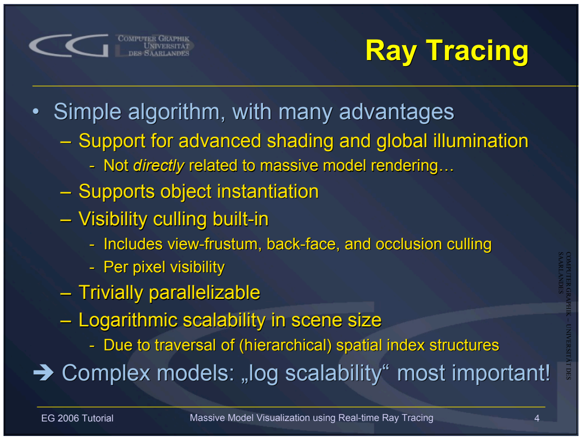
Overview

1. Simplicity of data preparation for ray tracing complex scenes
   - Efficient spatial index structures for ray tracing
   - Off-line construction of index structures
2. Adapting ray tracing to complex models
   - Active memory management
   - Asynchronous data loading
   - Level-of-detail management
3. Photorealistic rendering and lighting of highly complex models
   - Flexible combination and integration of different shading algorithms
   - Efficient integration of environmental lighting
   - Handling aliasing for complex geometry
4. Review of hardware-trends for real-time ray tracing
   - Comparing multi-core CPUs, GPUs, Cell processor, and custom ray tracing hardware



Ray Tracing

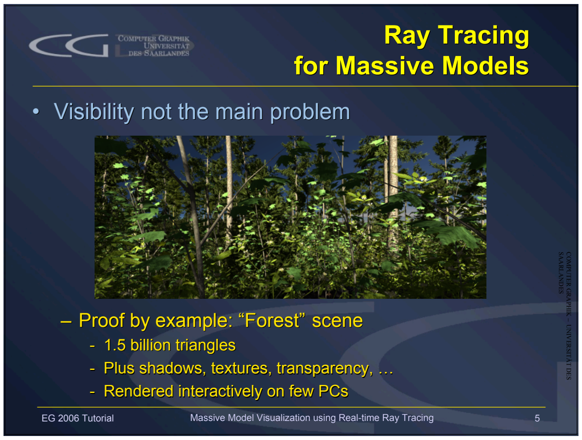
- Simple algorithm, with many advantages
  - Support for advanced shading and global illumination
    - Not *directly* related to massive model rendering…
  - Supports object instantiation
  - Visibility culling built-in
    - Includes view-frustum, back-face, and occlusion culling
    - Per pixel visibility
  - Trivially parallelizable
  - Logarithmic scalability in scene size
    - Due to traversal of (hierarchical) spatial index structures



Ray Tracing

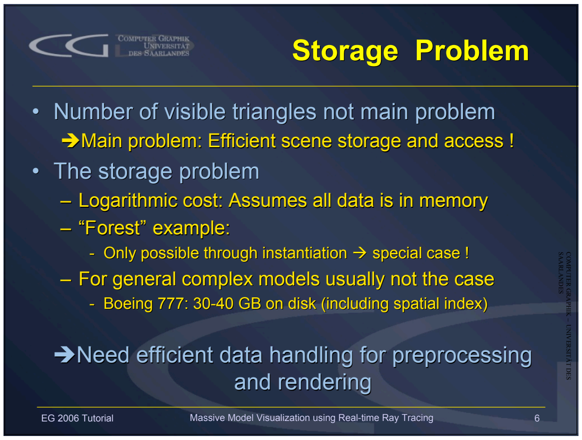
- Simple algorithm, with many advantages
  - Support for advanced shading and global illumination
    - Not *directly* related to massive model rendering…
  - Supports object instantiation
  - Visibility culling built-in
    - Includes view-frustum, back-face, and occlusion culling
    - Per pixel visibility
  - Trivially parallelizable
  - Logarithmic scalability in scene size
    - Due to traversal of (hierarchical) spatial index structures

➔ Complex models: „log scalability“ most important!



Ray Tracing for Massive Models

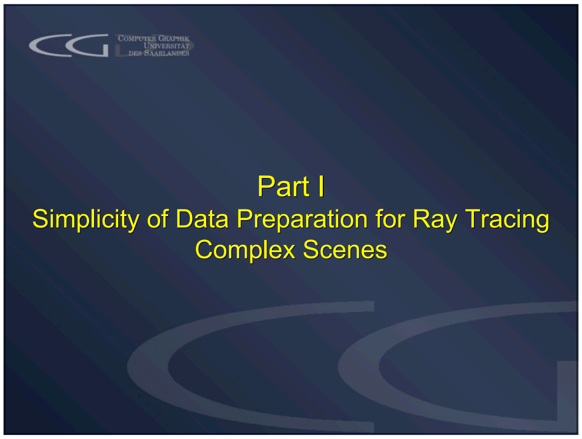
- Visibility not the main problem
  - Proof by example: “Forest” scene
    - 1.5 billion triangles
    - Plus shadows, textures, transparency, …
    - Rendered interactively on few PCs



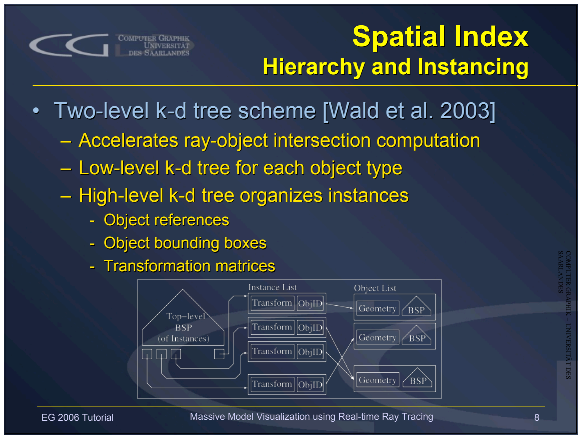
Storage Problem

- Number of visible triangles not main problem
  ➔Main problem: Efficient scene storage and access !
- The storage problem
  - Logarithmic cost: Assumes all data is in memory
  - “Forest” example:
    - Only possible through instantiation → special case !
  - For general complex models usually not the case
    - Boeing 777: 30-40 GB on disk (including spatial index)

➔Need efficient data handling for preprocessing and rendering

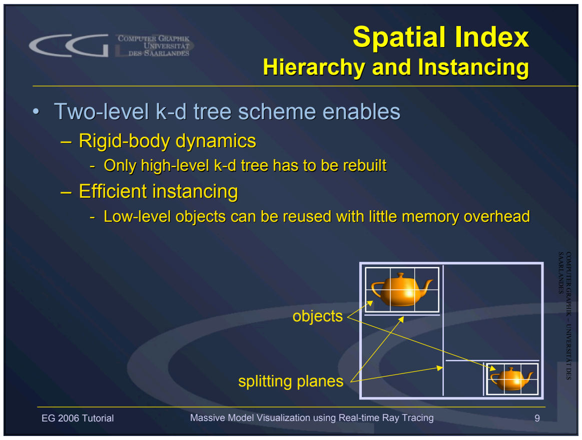


Part I
Simplicity of Data Preparation for Ray Tracing Complex Scenes



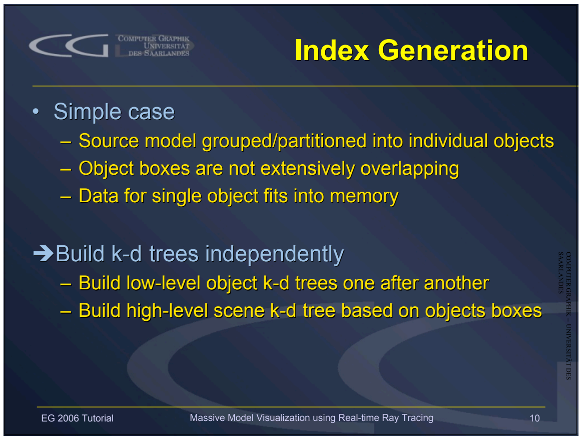
Spatial Index
Hierarchy and Instancing

- Two-level k-d tree scheme [Wald et al. 2003]
  - Accelerates ray-object intersection computation
  - Low-level k-d tree for each object type
  - High-level k-d tree organizes instances
    - Object references
    - Object bounding boxes
    - Transformation matrices



Spatial Index
Hierarchy and Instancing

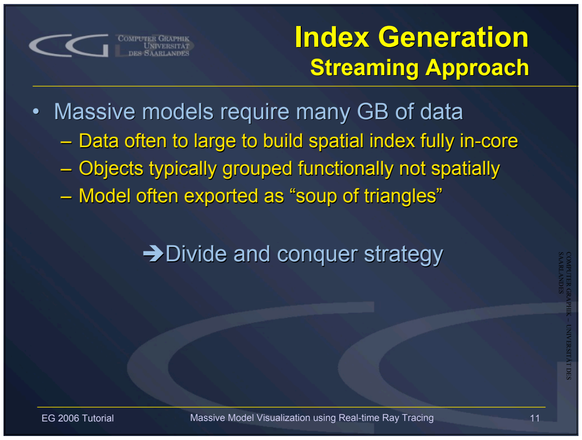
- Two-level k-d tree scheme enables
  - Rigid-body dynamics
    - Only high-level k-d tree has to be rebuilt
  - Efficient instancing
    - Low-level objects can be reused with little memory overhead



Index Generation

- Simple case
  - Source model grouped/partitioned into individual objects
  - Object boxes are not extensively overlapping
  - Data for single object fits into memory

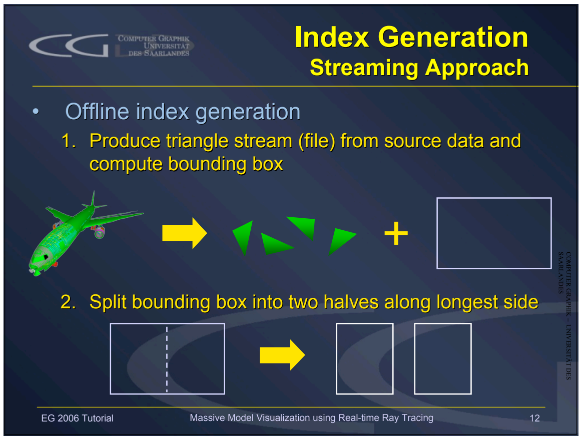
➔Build k-d trees independently
  - Build low-level object k-d trees one after another
  - Build high-level scene k-d tree based on objects boxes



Index Generation
Streaming Approach

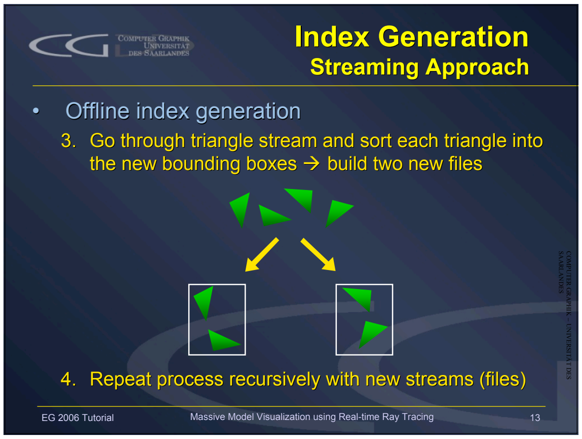
- Massive models require many GB of data
  - Data often to large to build spatial index fully in-core
  - Objects typically grouped functionally not spatially
  - Model often exported as “soup of triangles”

➔Divide and conquer strategy



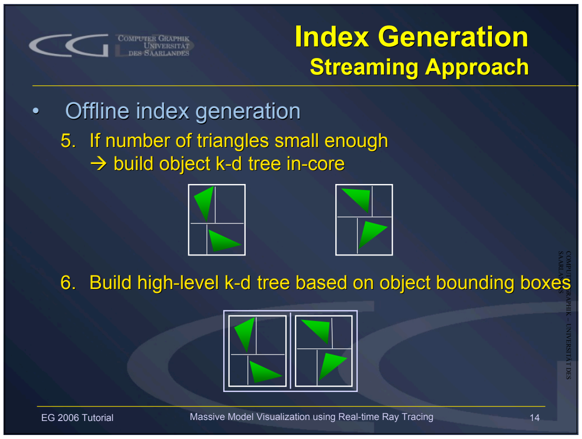
Index Generation
Streaming Approach

- Offline index generation
  1. Produce triangle stream (file) from source data and compute bounding box
  2. Split bounding box into two halves along longest side



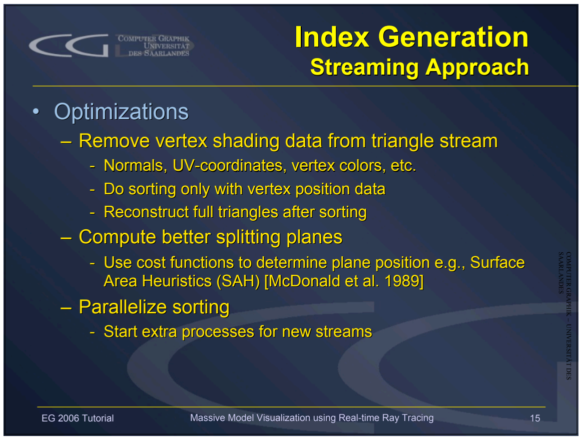
Index Generation
Streaming Approach

- Offline index generation
  3. Go through triangle stream and sort each triangle into the new bounding boxes → build two new files
  4. Repeat process recursively with new streams (files)



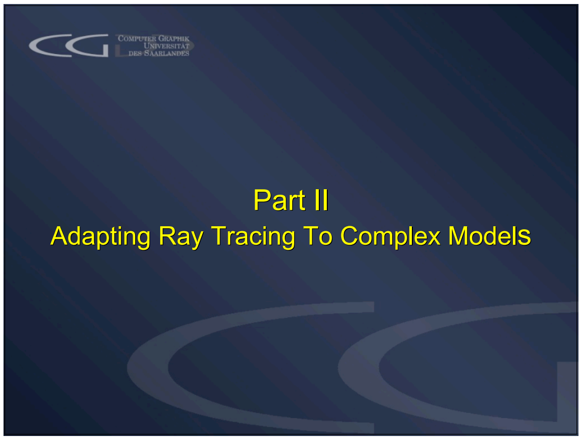
Index Generation
Streaming Approach

- Offline index generation
  5. If number of triangles small enough
     → build object k-d tree in-core
  6. Build high-level k-d tree based on object bounding boxes

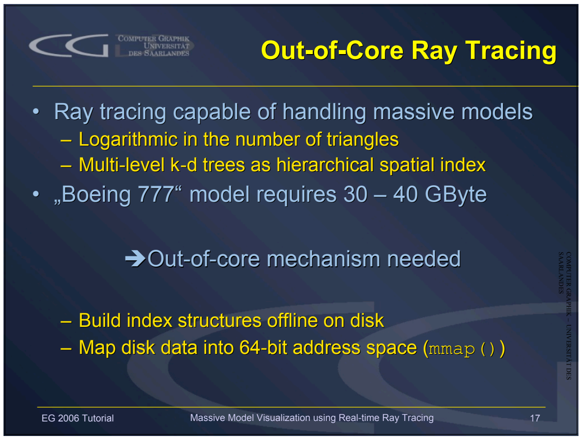


Index Generation
Streaming Approach

- Optimizations
  - Remove vertex shading data from triangle stream
    - Normals, UV-coordinates, vertex colors, etc.
    - Do sorting only with vertex position data
    - Reconstruct full triangles after sorting
  - Compute better splitting planes
    - Use cost functions to determine plane position e.g., Surface Area Heuristics (SAH) [McDonald et al. 1989]
  - Parallelize sorting
    - Start extra processes for new streams



Part II
Adapting Ray Tracing To Complex Models

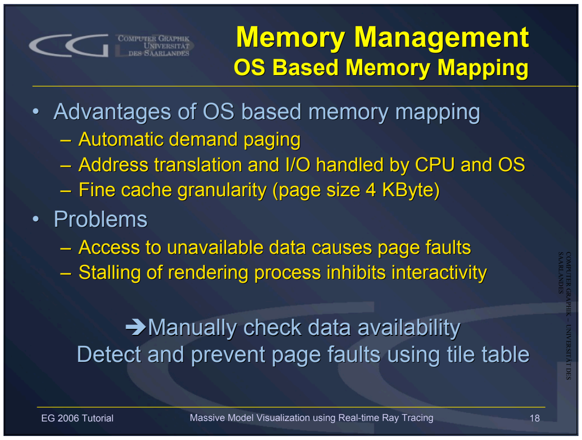


Out-of-Core Ray Tracing

- Ray tracing capable of handling massive models
  - Logarithmic in the number of triangles
  - Multi-level k-d trees as hierarchical spatial index
- „Boeing 777“ model requires 30 – 40 GByte

→Out-of-core mechanism needed

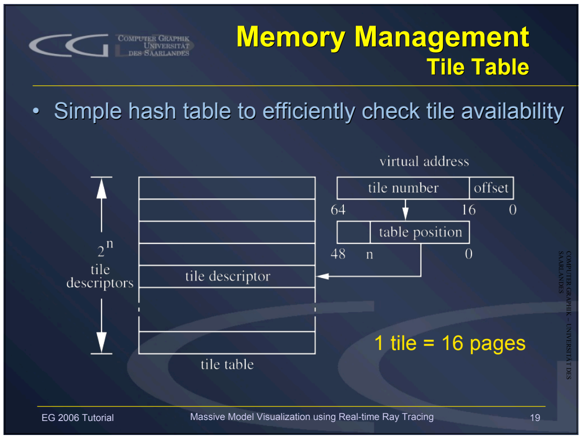
  - Build index structures offline on disk
  - Map disk data into 64-bit address space (`mmap()`)



Memory Management
OS Based Memory Mapping

- Advantages of OS based memory mapping
  - Automatic demand paging
  - Address translation and I/O handled by CPU and OS
  - Fine cache granularity (page size 4 KByte)
- Problems
  - Access to unavailable data causes page faults
  - Stalling of rendering process inhibits interactivity

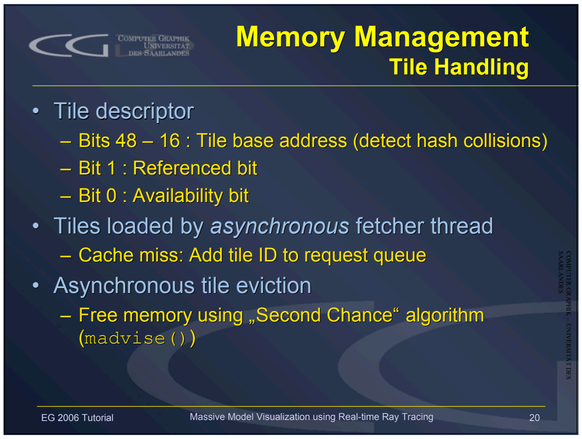
→Manually check data availability
Detect and prevent page faults using tile table



Memory Management
Tile Table

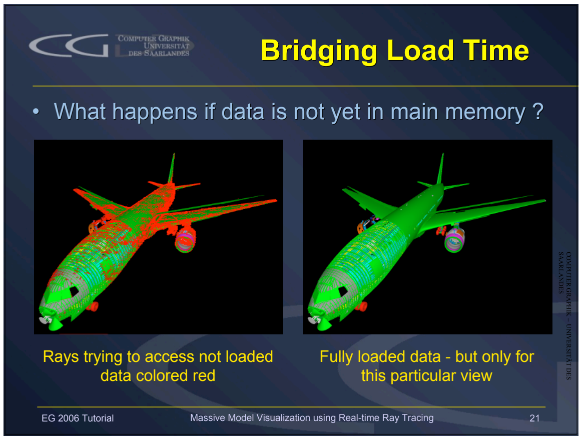
- Simple hash table to efficiently check tile availability

1 tile = 16 pages



Memory Management
Tile Handling

- Tile descriptor
  - Bits 48 – 16 : Tile base address (detect hash collisions)
  - Bit 1 : Referenced bit
  - Bit 0 : Availability bit
- Tiles loaded by *asynchronous* fetcher thread
  - Cache miss: Add tile ID to request queue
- Asynchronous tile eviction
  - Free memory using „Second Chance“ algorithm (`madvise()`)

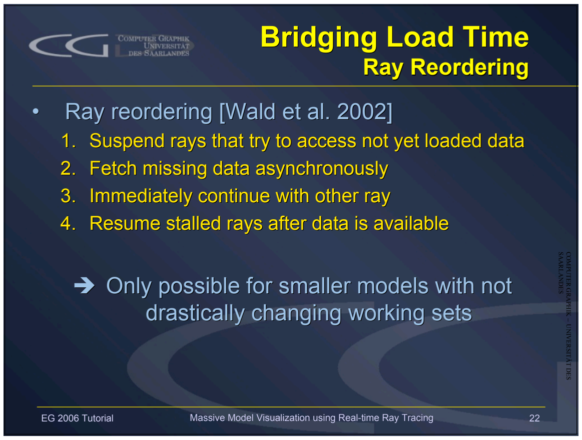


Bridging Load Time

- What happens if data is not yet in main memory ?

Rays trying to access not loaded data colored red

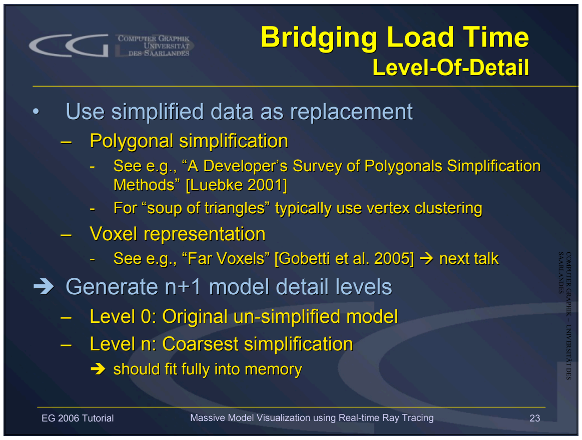
Fully loaded data - but only for this particular view



Bridging Load Time
Ray Reordering

- Ray reordering [Wald et al. 2002]
  1. Suspend rays that try to access not yet loaded data
  2. Fetch missing data asynchronously
  3. Immediately continue with other ray
  4. Resume stalled rays after data is available

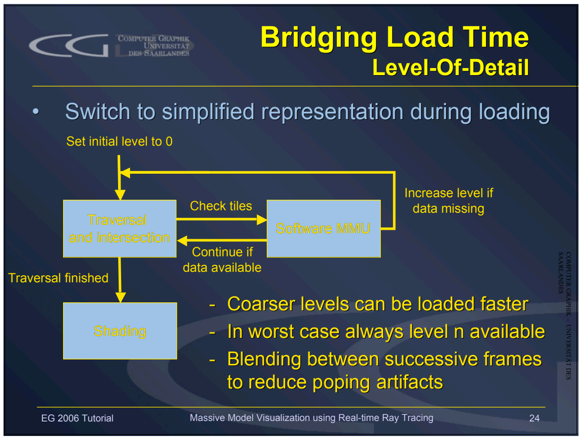
➔ Only possible for smaller models with not drastically changing working sets



Bridging Load Time
Level-Of-Detail

- Use simplified data as replacement
  - Polygonal simplification
    - See e.g., “A Developer’s Survey of Polygonals Simplification Methods” [Luebke 2001]
    - For “soup of triangles” typically use vertex clustering
  - Voxel representation
    - See e.g., “Far Voxels” [Gobetti et al. 2005] → next talk

➔ Generate n+1 model detail levels
  - Level 0: Original un-simplified model
  - Level n: Coarsest simplification
    ➔ should fit fully into memory

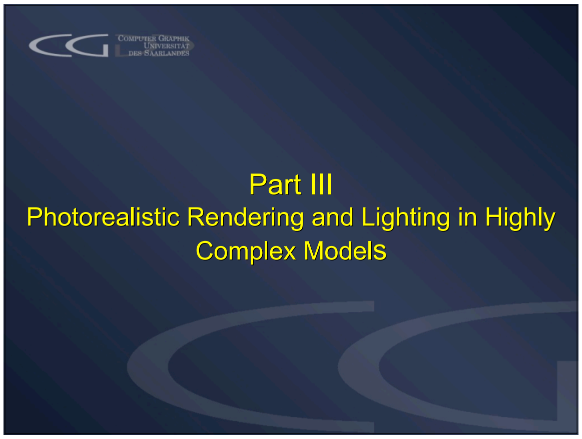


Bridging Load Time
Level-Of-Detail

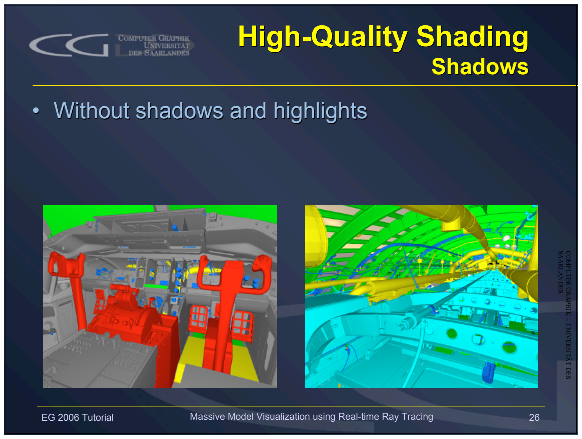
- Switch to simplified representation during loading

Set initial level to 0
Traversal and intersection
Check tiles
Software MMU
Increase level if data missing
Continue if data available
Traversal finished
Shading

- Coarser levels can be loaded faster
- In worst case always level n available
- Blending between successive frames to reduce poping artifacts

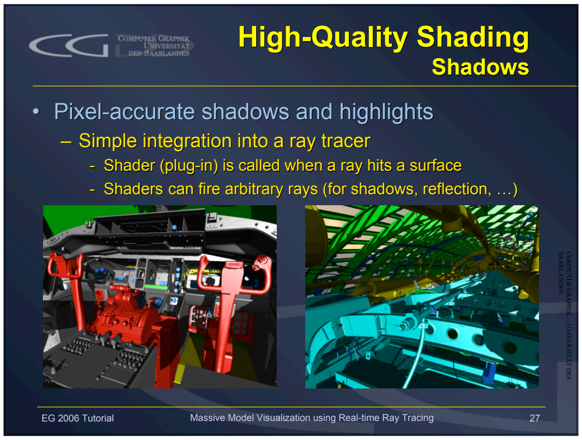


Part III
Photorealistic Rendering and Lighting in Highly Complex Models



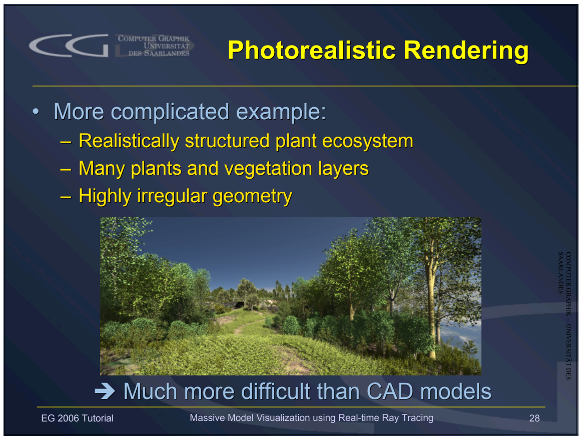
High-Quality Shading
Shadows

- Without shadows and highlights



High-Quality Shading
Shadows

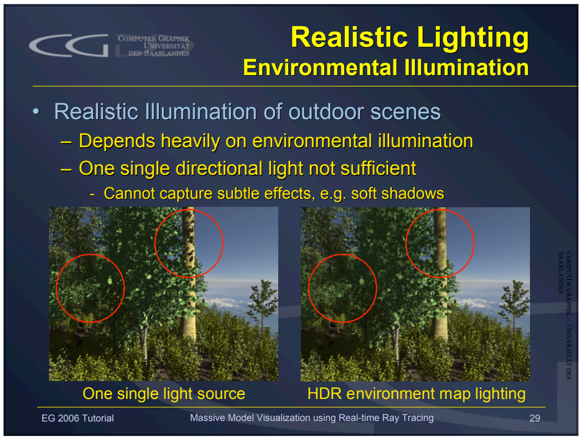
- Pixel-accurate shadows and highlights
  - Simple integration into a ray tracer
    - Shader (plug-in) is called when a ray hits a surface
    - Shaders can fire arbitrary rays (for shadows, reflection, …)



Photorealistic Rendering

- More complicated example:
  - Realistically structured plant ecosystem
  - Many plants and vegetation layers
  - Highly irregular geometry

➔ Much more difficult than CAD models

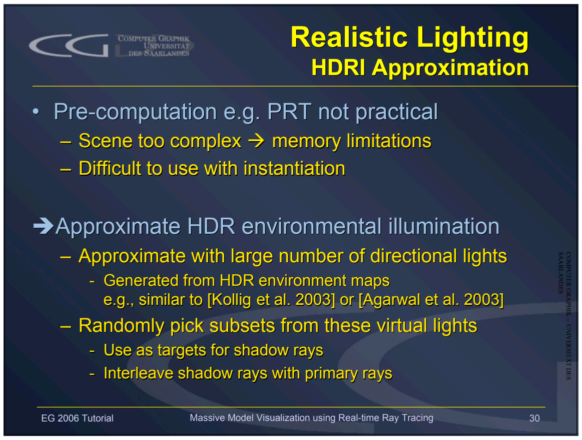


Realistic Lighting
Environmental Illumination

- Realistic Illumination of outdoor scenes
  - Depends heavily on environmental illumination
  - One single directional light not sufficient
    - Cannot capture subtle effects, e.g. soft shadows

One single light source

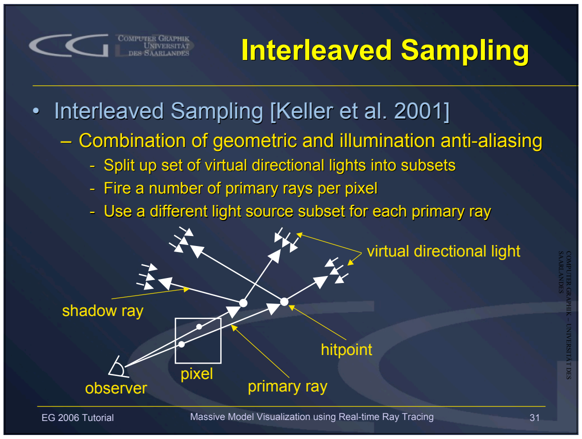
HDR environment map lighting



Realistic Lighting
HDRI Approximation

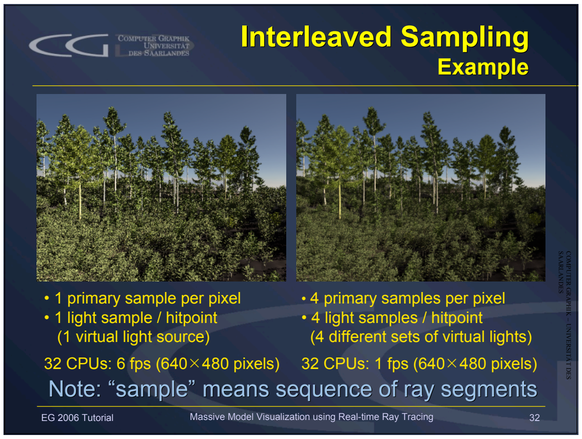
- Pre-computation e.g. PRT not practical
  - Scene too complex → memory limitations
  - Difficult to use with instantiation

➔Approximate HDR environmental illumination
  - Approximate with large number of directional lights
    - Generated from HDR environment maps e.g., similar to [Kollig et al. 2003] or [Agarwal et al. 2003]
  - Randomly pick subsets from these virtual lights
    - Use as targets for shadow rays
    - Interleave shadow rays with primary rays



Interleaved Sampling

- Interleaved Sampling [Keller et al. 2001]
  - Combination of geometric and illumination anti-aliasing
    - Split up set of virtual directional lights into subsets
    - Fire a number of primary rays per pixel
    - Use a different light source subset for each primary ray



Interleaved Sampling Example

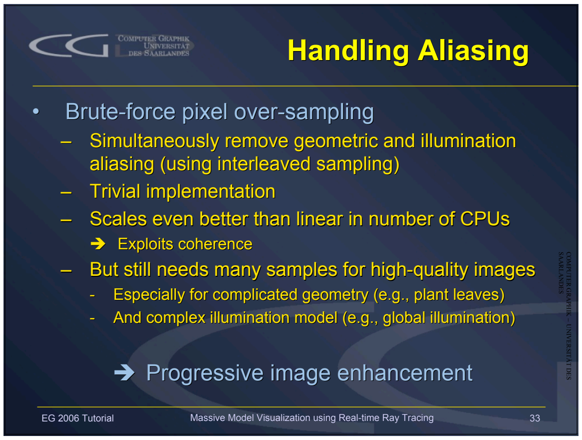
- 1 primary sample per pixel
- 1 light sample / hitpoint (1 virtual light source)

32 CPUs: 6 fps (640×480 pixels)

- 4 primary samples per pixel
- 4 light samples / hitpoint (4 different sets of virtual lights)

32 CPUs: 1 fps (640×480 pixels)

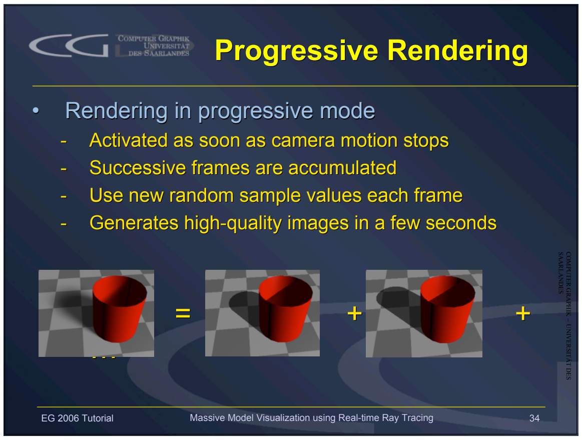
Note: “sample” means sequence of ray segments



Handling Aliasing

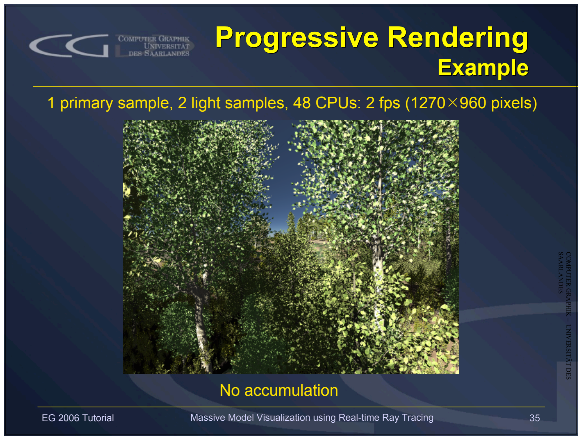
- Brute-force pixel over-sampling
  - Simultaneously remove geometric and illumination aliasing (using interleaved sampling)
  - Trivial implementation
  - Scales even better than linear in number of CPUs
    - ➔ Exploits coherence
  - But still needs many samples for high-quality images
    - Especially for complicated geometry (e.g., plant leaves)
    - And complex illumination model (e.g., global illumination)

➔ Progressive image enhancement



Progressive Rendering

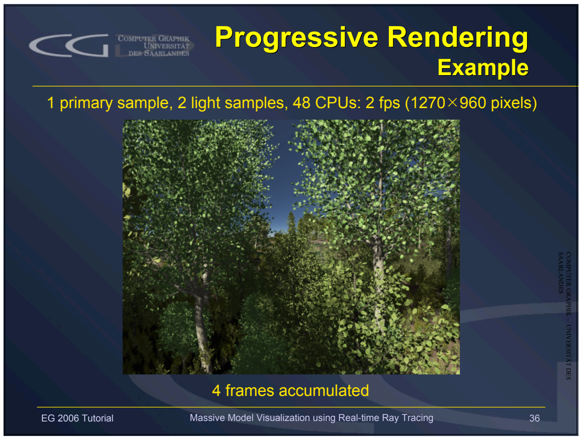
- Rendering in progressive mode
  - Activated as soon as camera motion stops
  - Successive frames are accumulated
  - Use new random sample values each frame
  - Generates high-quality images in a few seconds



Progressive Rendering Example

1 primary sample, 2 light samples, 48 CPUs: 2 fps (1270×960 pixels)

No accumulation



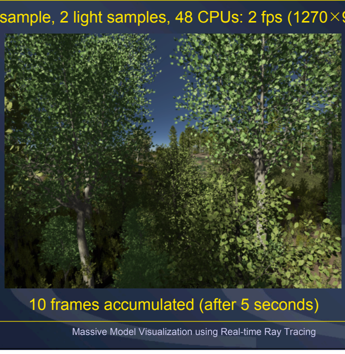
Progressive Rendering Example

1 primary sample, 2 light samples, 48 CPUs: 2 fps (1270×960 pixels)

4 frames accumulated

Progressive Rendering
Example

1 primary sample, 2 light samples, 48 CPUs: 2 fps (1270×960 pixels)



10 frames accumulated (after 5 seconds)

Sample Caching

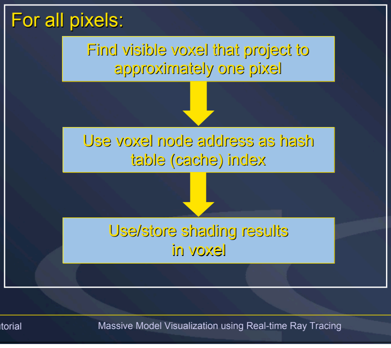
- Sample Cache approach
  - Cache image results from previous frames in acceleration structures (multi-level k-d tree)
  - No preprocessing required
  - Independent from actual scene size

!!! Work in progress !!!

2-3 times speed up compared to standard over-sampling

But very simple implementation

Method Overview



For all pixels:

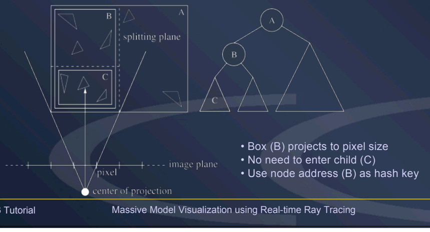
Find visible voxel that project to approximately one pixel

Use voxel node address as hash table (cache) index

Use/store shading results in voxel

Method Overview
Traversal

- Traverse until k-d tree box projects to one pixel
  - Trace ray differentials [Igehy 1999]
  - Check pixel footprint against current k-d tree box
  - Stop if box projects to approximately one pixel


A
B
C
splitting plane
A
B
C
pixel
image plane
center of projection
• Box (B) projects to pixel size
• No need to enter child (C)
• Use node address (B) as hash key

Method Overview
Hash Table

- Use box node index as hash table key
  - Multi-dimensional hash table
  - Node addresses from all hierarchy levels form key
    - E.g. Two-level k-d tree: two node addresses

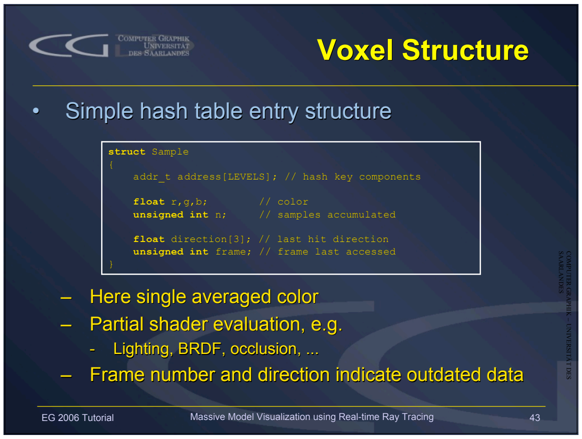
$$index = (p \cdot \sum_i a_i) \bmod s$$

  - Voxel size varies significantly
    - Voxels can be further subdivided (e.g., 2 x 2 x 2 regular grid)
    - Subcell index as additional hash key component
    - ➔ No need to alter existing acceleration structures

Method Overview
Rendering

- Retrieve and update hashed voxel data
  - Basic rendering loop structure

```
for ( each pixel )
    shoot centered test ray;
    if ( suitable voxel found )
        access hash table;
        if ( data not accurate )
            trace and shade new rays;
            update voxel;
        use voxel data;
    else
        standard ray tracing;
```



Voxel Structure

- Simple hash table entry structure

```
struct Sample
{
    addr_t address[LEVELS]; // hash key components

    float r,g,b;          // color
    unsigned int n;       // samples accumulated

    float direction[3]; // last hit direction
    unsigned int frame; // frame last accessed
}
```

- – Here single averaged color
- – Partial shader evaluation, e.g.
  - - Lighting, BRDF, occlusion, ...
- – Frame number and direction indicate outdated data

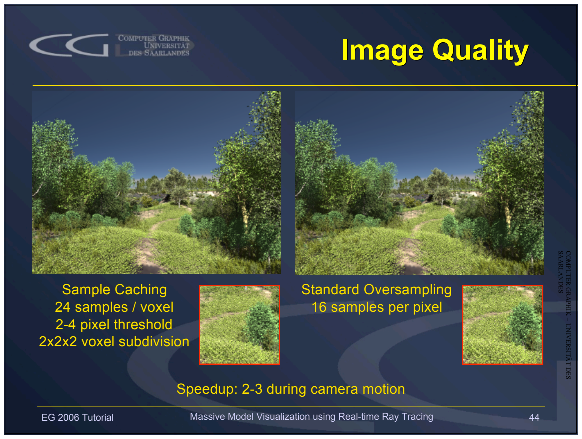
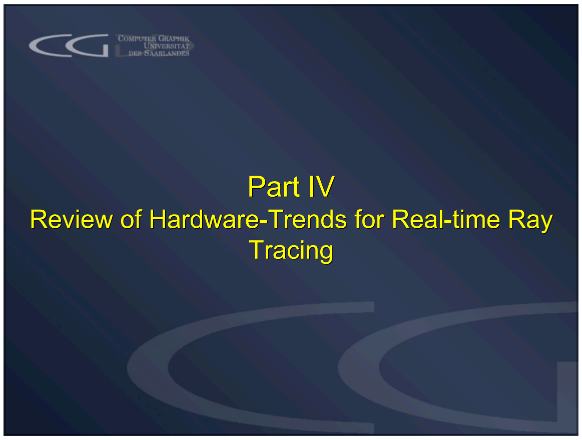


Image Quality

Sample Caching
24 samples / voxel
2-4 pixel threshold
2x2x2 voxel subdivision

Standard Oversampling
16 samples per pixel

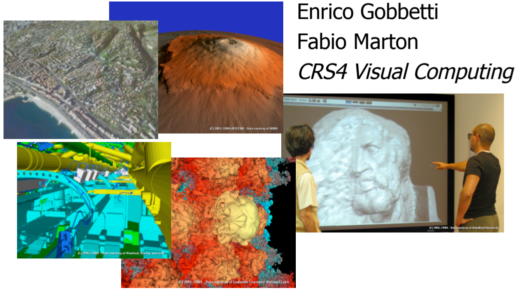
Speedup: 2-3 during camera motion



Part IV
Review of Hardware-Trends for Real-time Ray Tracing

Multi-Resolution Techniques for Exploring Extremely Large and Complex Surfaces

Enrico Gobbetti
Fabio Marton
CRS4 Visual Computing

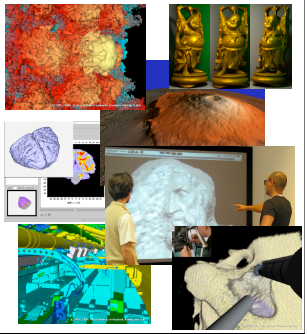


(CRS4 in one slide)

- Interdisciplinary research center focused on computational sciences
  - Operational since 1992
- RTD staff of ~80 people
- Turnover of ~7M Euro, of which ~50% from external funding
  - EU/National research project
  - Industrial contracts

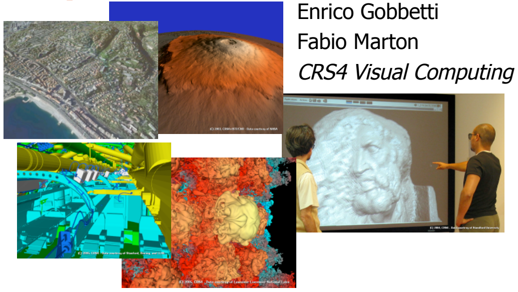
(CRS4 Visual Computing Group)

- Staff
  - 6+ people
- RTD
  - Geometry processing / rendering
  - Scientific visualization
  - Haptics
  - VR & Simulation
- Service
  - Sci Viz + Post production



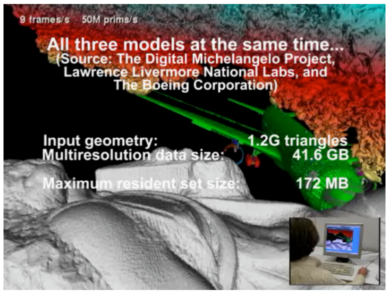
Multi-Resolution Techniques for Exploring Extremely Large and Complex Surfaces

Enrico Gobbetti
Fabio Marton
CRS4 Visual Computing



Goal and Motivation

Accurate interactive inspection of very large models (unlimited size!) on PC platforms…



9 frames/s 50M prims/s

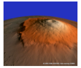
All three models at the same time…
(Source: The Digital Michelangelo Project, Lawrence Livermore National Labs, and The Boeing Corporation)

Input geometry: 1.2G triangles
Multiresolution data size: 41.6 GB
Maximum resident set size: 172 MB

Xeon 2.4GHz / 1GB RAM / 70GB SCSI 320 Disk / NVIDIA 6800GTS

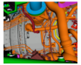
Application domains / data sources

Local Terrain Models
2.5D – Flat – Dense regular sampling

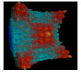


Planetary terrain models
2.5D – Spherical – Dense regular sampling

Laser scanned models
3D – Moderately simple topology – low depth complexity - dense

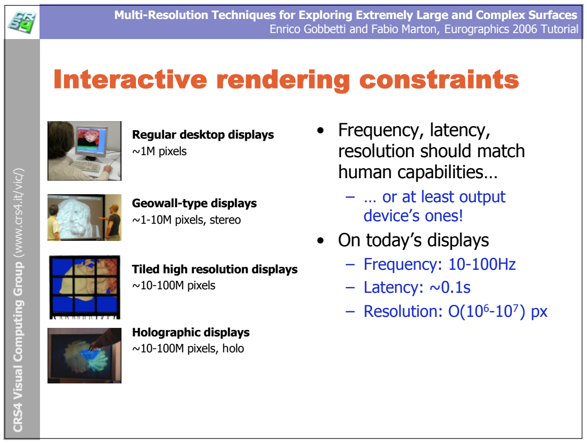


CAD models
3D – complex topology – high depth complexity – structured - 'ugly' mesh



Natural objects / Simulation results
3D – complex topology + high depth complexity + unstructured/high frequency details

- Many important application domains
- Models exceed
  - O(10^8-10^9) samples
  - O(10^9) bytes
- Varying
  - Dimensionality
  - Topology
  - Sampling distribution



Interactive rendering constraints

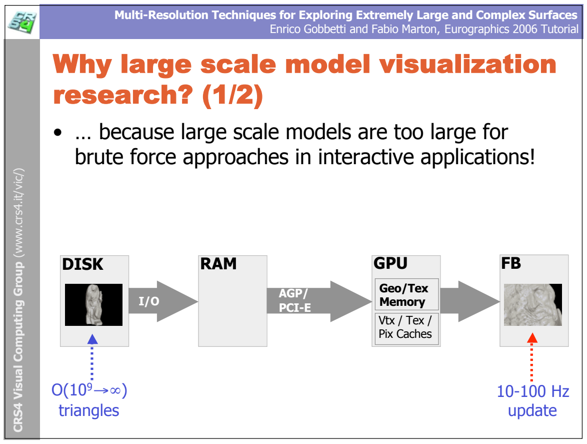
Regular desktop displays
~1M pixels

Geowall-type displays
~1-10M pixels, stereo

Tiled high resolution displays
~10-100M pixels

Holographic displays
~10-100M pixels, holo

- Frequency, latency, resolution should match human capabilities…
  - … or at least output device's ones!
- On today's displays
  - Frequency: 10-100Hz
  - Latency: ~0.1s
  - Resolution: $O(10^6\text{-}10^7)$ px



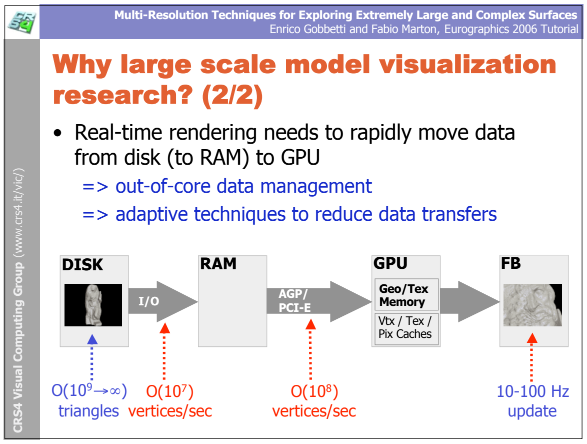
Why large scale model visualization research? (1/2)

- … because large scale models are too large for brute force approaches in interactive applications!

DISK
I/O
RAM
AGP/ PCI-E
GPU
Geo/Tex Memory
Vtx / Tex / Pix Caches
FB

$O(10^9 \rightarrow \infty)$ triangles

10-100 Hz update



Why large scale model visualization research? (2/2)

- Real-time rendering needs to rapidly move data from disk (to RAM) to GPU

=> out-of-core data management

=> adaptive techniques to reduce data transfers

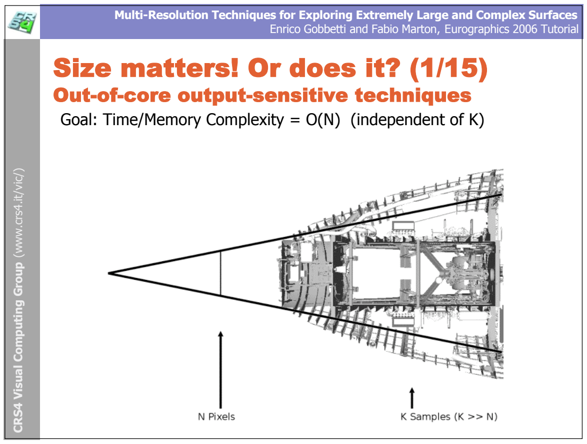
DISK
I/O
RAM
AGP/ PCI-E
GPU
Geo/Tex Memory
Vtx / Tex / Pix Caches
FB

$O(10^9 \rightarrow \infty)$ triangles

$O(10^7)$ vertices/sec

$O(10^8)$ vertices/sec

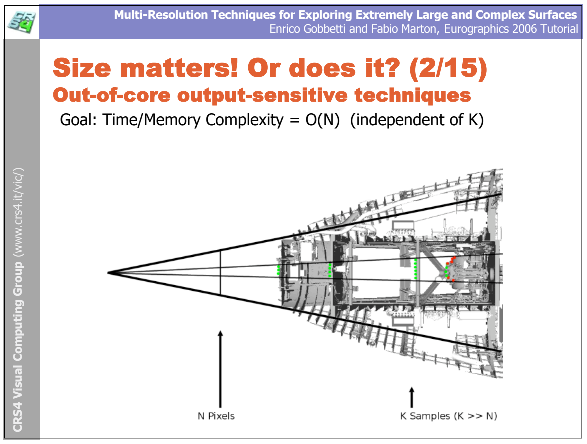
10-100 Hz update



Size matters! Or does it? (1/15)

Out-of-core output-sensitive techniques

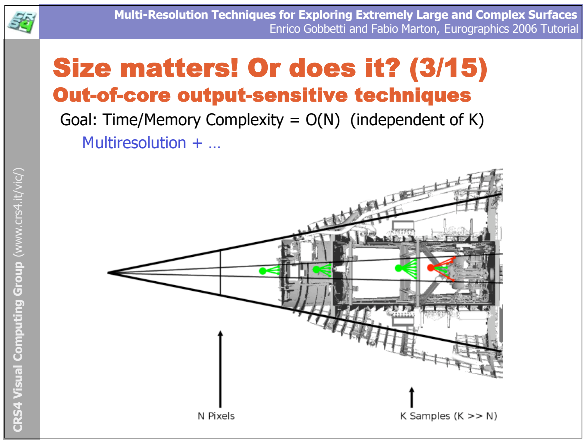
Goal: Time/Memory Complexity = O(N) (independent of K)



Size matters! Or does it? (2/15)

Out-of-core output-sensitive techniques

Goal: Time/Memory Complexity = O(N) (independent of K)

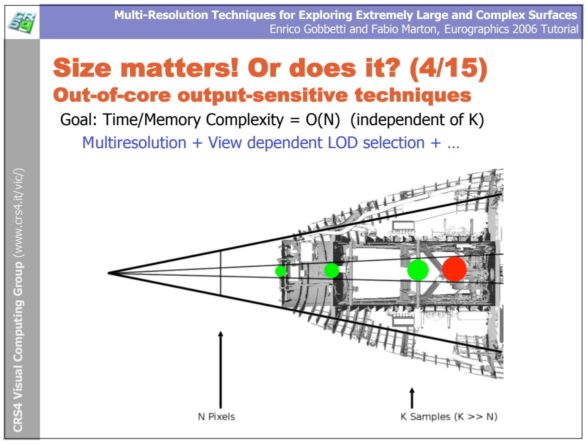


Size matters! Or does it? (3/15)

Out-of-core output-sensitive techniques

Goal: Time/Memory Complexity = O(N) (independent of K)

Multiresolution + …



Multi-Resolution Techniques for Exploring Extremely Large and Complex Surfaces
Enrico Gobbetti and Fabio Marton, Eurographics 2006 Tutorial

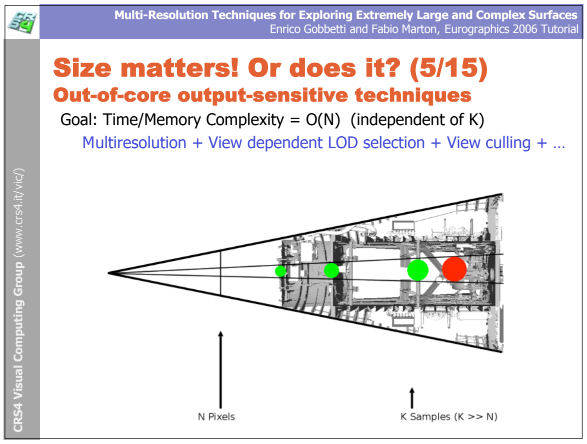
CRS4 Visual Computing Group (www.crs4.it/vic/)

Size matters! Or does it? (4/15)

Out-of-core output-sensitive techniques

Goal: Time/Memory Complexity = O(N) (independent of K)

Multiresolution + View dependent LOD selection + …



Multi-Resolution Techniques for Exploring Extremely Large and Complex Surfaces
Enrico Gobbetti and Fabio Marton, Eurographics 2006 Tutorial

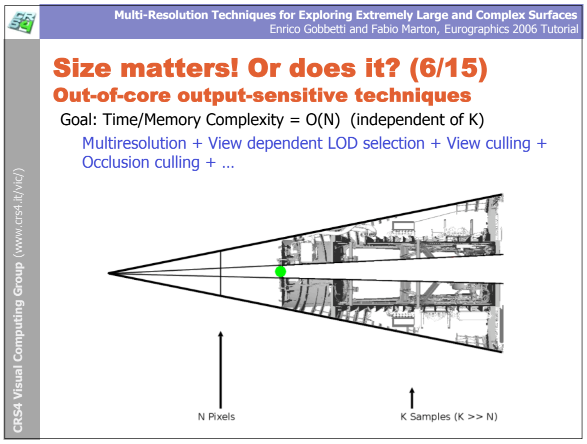
CRS4 Visual Computing Group (www.crs4.it/vic/)

Size matters! Or does it? (5/15)

Out-of-core output-sensitive techniques

Goal: Time/Memory Complexity = O(N) (independent of K)

Multiresolution + View dependent LOD selection + View culling + …



Multi-Resolution Techniques for Exploring Extremely Large and Complex Surfaces
Enrico Gobbetti and Fabio Marton, Eurographics 2006 Tutorial

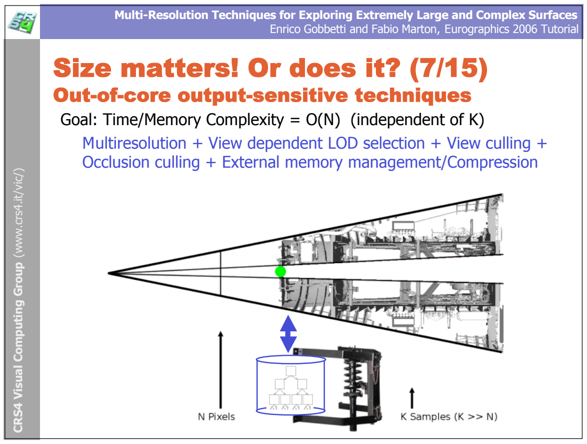
CRS4 Visual Computing Group (www.crs4.it/vic/)

Size matters! Or does it? (6/15)

Out-of-core output-sensitive techniques

Goal: Time/Memory Complexity = O(N) (independent of K)

Multiresolution + View dependent LOD selection + View culling + Occlusion culling + …



Multi-Resolution Techniques for Exploring Extremely Large and Complex Surfaces
Enrico Gobbetti and Fabio Marton, Eurographics 2006 Tutorial

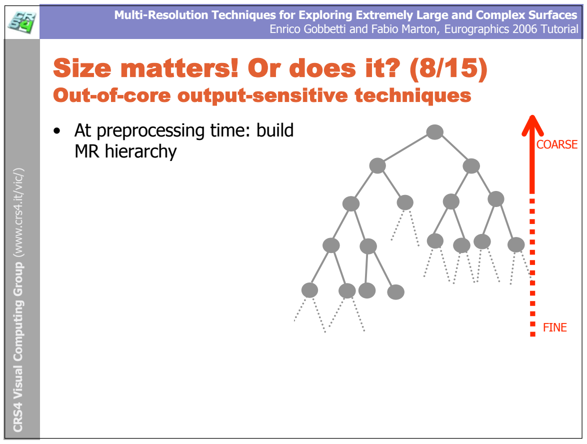
CRS4 Visual Computing Group (www.crs4.it/vic/)

Size matters! Or does it? (7/15)

Out-of-core output-sensitive techniques

Goal: Time/Memory Complexity = O(N) (independent of K)

Multiresolution + View dependent LOD selection + View culling + Occlusion culling + External memory management/Compression



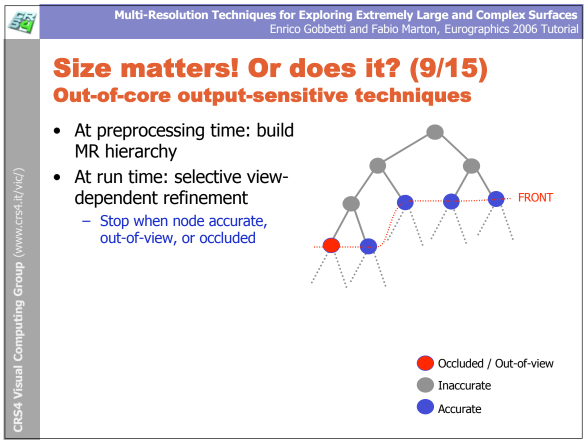
Multi-Resolution Techniques for Exploring Extremely Large and Complex Surfaces
Enrico Gobbetti and Fabio Marton, Eurographics 2006 Tutorial

CRS4 Visual Computing Group (www.crs4.it/vic/)

Size matters! Or does it? (8/15)

Out-of-core output-sensitive techniques

- At preprocessing time: build MR hierarchy



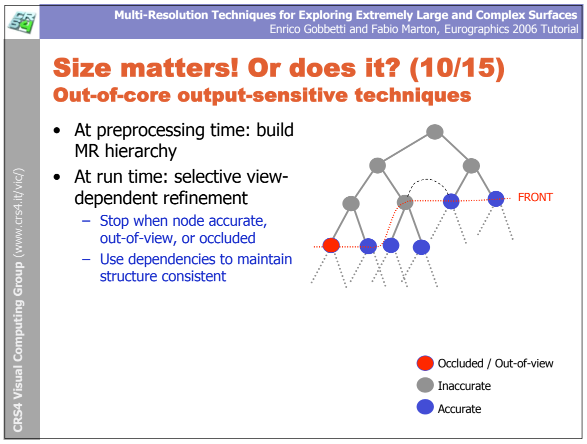
Multi-Resolution Techniques for Exploring Extremely Large and Complex Surfaces
Enrico Gobbetti and Fabio Marton, Eurographics 2006 Tutorial

CRS4 Visual Computing Group (www.crs4.it/vic/)

Size matters! Or does it? (9/15)

Out-of-core output-sensitive techniques

- At preprocessing time: build MR hierarchy
- At run time: selective view-dependent refinement
  - Stop when node accurate, out-of-view, or occluded



Multi-Resolution Techniques for Exploring Extremely Large and Complex Surfaces
Enrico Gobbetti and Fabio Marton, Eurographics 2006 Tutorial

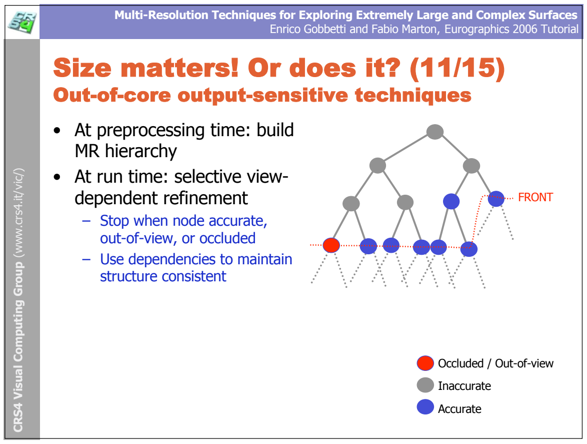
CRS4 Visual Computing Group (www.crs4.it/vic/)

Size matters! Or does it? (10/15)

Out-of-core output-sensitive techniques

- At preprocessing time: build MR hierarchy
- At run time: selective view-dependent refinement
  - Stop when node accurate, out-of-view, or occluded
  - Use dependencies to maintain structure consistent

🔴 Occluded / Out-of-view
● Inaccurate
🔵 Accurate



Multi-Resolution Techniques for Exploring Extremely Large and Complex Surfaces
Enrico Gobbetti and Fabio Marton, Eurographics 2006 Tutorial

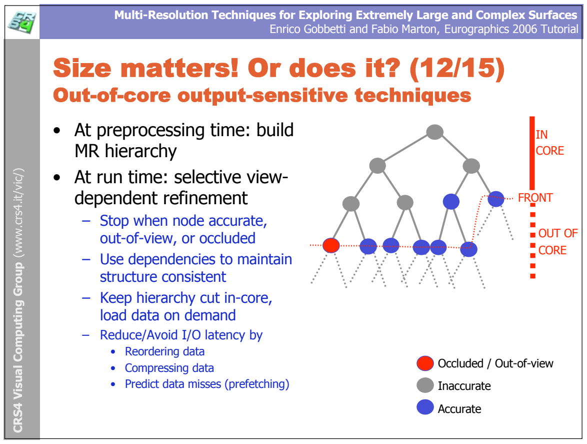
CRS4 Visual Computing Group (www.crs4.it/vic/)

Size matters! Or does it? (11/15)

Out-of-core output-sensitive techniques

- At preprocessing time: build MR hierarchy
- At run time: selective view-dependent refinement
  - Stop when node accurate, out-of-view, or occluded
  - Use dependencies to maintain structure consistent

🔴 Occluded / Out-of-view
● Inaccurate
🔵 Accurate



Multi-Resolution Techniques for Exploring Extremely Large and Complex Surfaces
Enrico Gobbetti and Fabio Marton, Eurographics 2006 Tutorial

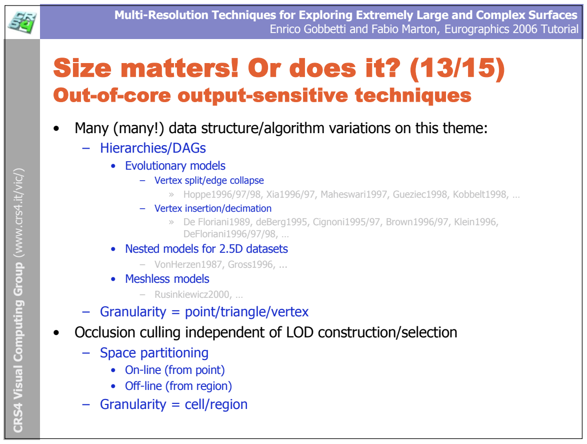
CRS4 Visual Computing Group (www.crs4.it/vic/)

Size matters! Or does it? (12/15)

Out-of-core output-sensitive techniques

- At preprocessing time: build MR hierarchy
- At run time: selective view-dependent refinement
  - Stop when node accurate, out-of-view, or occluded
  - Use dependencies to maintain structure consistent
  - Keep hierarchy cut in-core, load data on demand
  - Reduce/Avoid I/O latency by
    - Reordering data
    - Compressing data
    - Predict data misses (prefetching)

🔴 Occluded / Out-of-view
● Inaccurate
🔵 Accurate



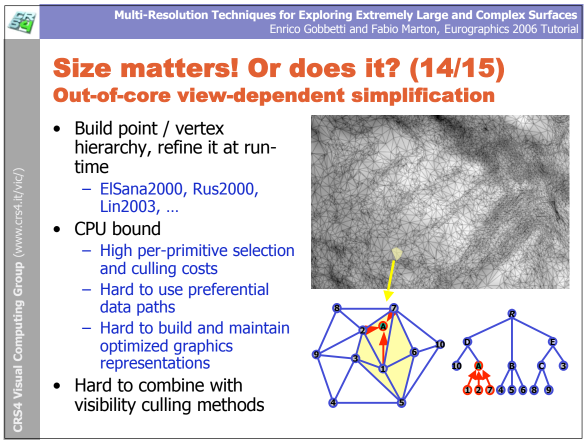
Multi-Resolution Techniques for Exploring Extremely Large and Complex Surfaces
Enrico Gobbetti and Fabio Marton, Eurographics 2006 Tutorial

CRS4 Visual Computing Group (www.crs4.it/vic/)

Size matters! Or does it? (13/15)

Out-of-core output-sensitive techniques

- Many (many!) data structure/algorithm variations on this theme:
  - Hierarchies/DAGs
    - Evolutionary models
      - Vertex split/edge collapse
        - » Hoppe1996/97/98, Xia1996/97, Maheswari1997, Gueziec1998, Kobbelt1998, …
      - Vertex insertion/decimation
        - » De Floriani1989, deBerg1995, Cignoni1995/97, Brown1996/97, Klein1996, DeFloriani1996/97/98, …
    - Nested models for 2.5D datasets
      - VonHerzen1987, Gross1996, …
    - Meshless models
      - Rusinkiewicz2000, …
  - Granularity = point/triangle/vertex
- Occlusion culling independent of LOD construction/selection
  - Space partitioning
    - On-line (from point)
    - Off-line (from region)
  - Granularity = cell/region



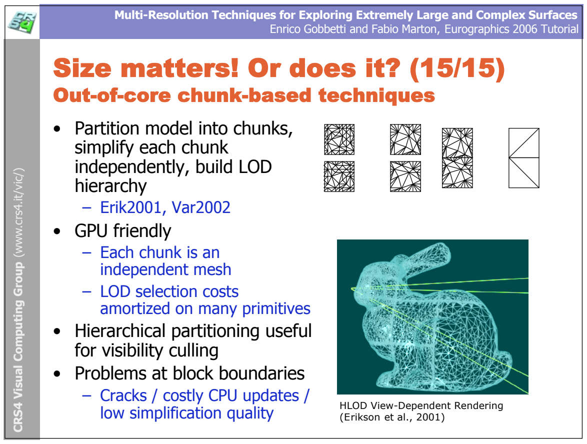
Multi-Resolution Techniques for Exploring Extremely Large and Complex Surfaces
Enrico Gobbetti and Fabio Marton, Eurographics 2006 Tutorial

CRS4 Visual Computing Group (www.crs4.it/vic/)

Size matters! Or does it? (14/15)

Out-of-core view-dependent simplification

- Build point / vertex hierarchy, refine it at run-time
  - ElSana2000, Rus2000, Lin2003, …
- CPU bound
  - High per-primitive selection and culling costs
  - Hard to use preferential data paths
  - Hard to build and maintain optimized graphics representations
- Hard to combine with visibility culling methods



Multi-Resolution Techniques for Exploring Extremely Large and Complex Surfaces
Enrico Gobbetti and Fabio Marton, Eurographics 2006 Tutorial

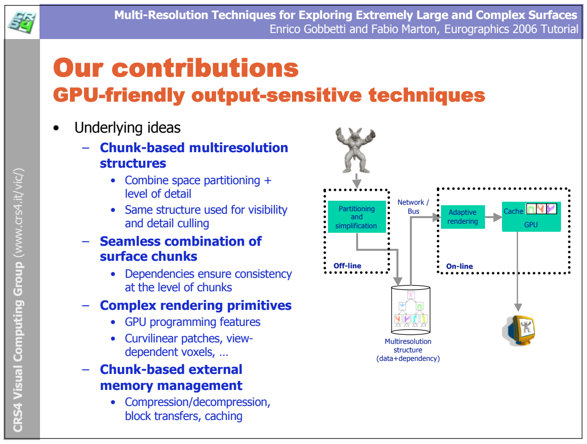
CRS4 Visual Computing Group (www.crs4.it/vic/)

Size matters! Or does it? (15/15)

Out-of-core chunk-based techniques

- Partition model into chunks, simplify each chunk independently, build LOD hierarchy
  - Erik2001, Var2002
- GPU friendly
  - Each chunk is an independent mesh
  - LOD selection costs amortized on many primitives
- Hierarchical partitioning useful for visibility culling
- Problems at block boundaries
  - Cracks / costly CPU updates / low simplification quality

HLOD View-Dependent Rendering
(Erikson et al., 2001)



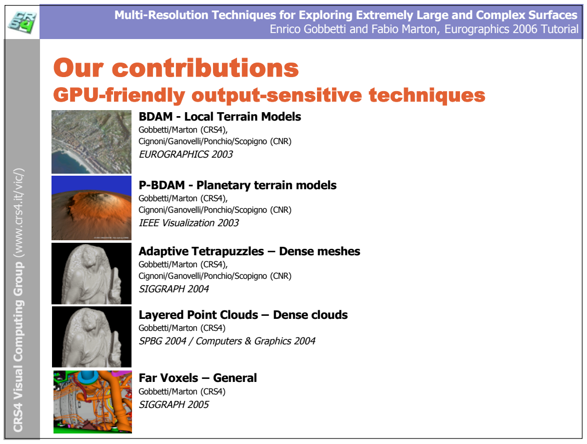
CRS4 Visual Computing Group (www.crs4.it/vic/)

Our contributions

GPU-friendly output-sensitive techniques

- Underlying ideas
  - Chunk-based multiresolution structures
    - Combine space partitioning + level of detail
    - Same structure used for visibility and detail culling
  - Seamless combination of surface chunks
    - Dependencies ensure consistency at the level of chunks
  - Complex rendering primitives
    - GPU programming features
    - Curvilinear patches, view-dependent voxels, …
  - Chunk-based external memory management
    - Compression/decompression, block transfers, caching

Partitioning and simplification
Off-line
Network / Bus
Adaptive rendering
Cache
GPU
On-line
Multiresolution structure (data+dependency)



CRS4 Visual Computing Group (www.crs4.it/vic/)

Our contributions

GPU-friendly output-sensitive techniques

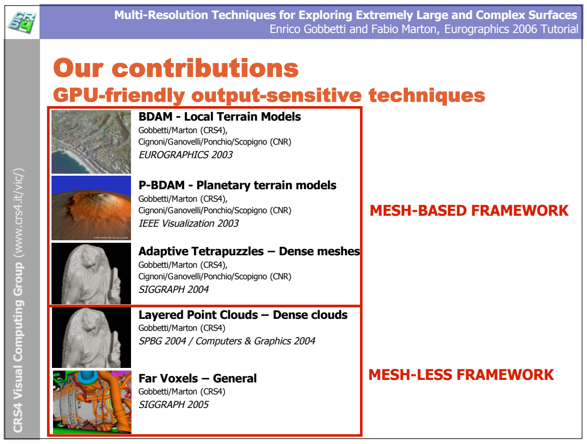
BDAM - Local Terrain Models
Gobbetti/Marton (CRS4),
Cignoni/Ganovelli/Ponchio/Scopigno (CNR)
EUROGRAPHICS 2003

P-BDAM - Planetary terrain models
Gobbetti/Marton (CRS4),
Cignoni/Ganovelli/Ponchio/Scopigno (CNR)
IEEE Visualization 2003

Adaptive Tetrapuzzles – Dense meshes
Gobbetti/Marton (CRS4),
Cignoni/Ganovelli/Ponchio/Scopigno (CNR)
SIGGRAPH 2004

Layered Point Clouds – Dense clouds
Gobbetti/Marton (CRS4)
SPBG 2004 / Computers & Graphics 2004

Far Voxels – General
Gobbetti/Marton (CRS4)
SIGGRAPH 2005



CRS4 Visual Computing Group (www.crs4.it/vic/)

Our contributions

GPU-friendly output-sensitive techniques

BDAM - Local Terrain Models
Gobbetti/Marton (CRS4),
Cignoni/Ganovelli/Ponchio/Scopigno (CNR)
EUROGRAPHICS 2003

P-BDAM - Planetary terrain models
Gobbetti/Marton (CRS4),
Cignoni/Ganovelli/Ponchio/Scopigno (CNR)
IEEE Visualization 2003

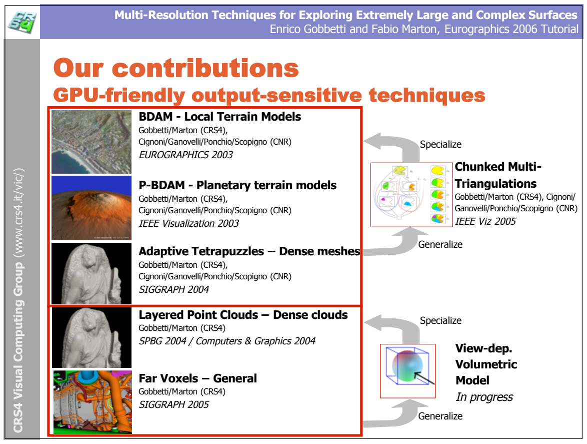
Adaptive Tetrapuzzles – Dense meshes
Gobbetti/Marton (CRS4),
Cignoni/Ganovelli/Ponchio/Scopigno (CNR)
SIGGRAPH 2004

MESH-BASED FRAMEWORK

Layered Point Clouds – Dense clouds
Gobbetti/Marton (CRS4)
SPBG 2004 / Computers & Graphics 2004

Far Voxels – General
Gobbetti/Marton (CRS4)
SIGGRAPH 2005

MESH-LESS FRAMEWORK



CRS4 Visual Computing Group (www.crs4.it/vic/)

Our contributions

GPU-friendly output-sensitive techniques

BDAM - Local Terrain Models
Gobbetti/Marton (CRS4),
Cignoni/Ganovelli/Ponchio/Scopigno (CNR)
EUROGRAPHICS 2003

P-BDAM - Planetary terrain models
Gobbetti/Marton (CRS4),
Cignoni/Ganovelli/Ponchio/Scopigno (CNR)
IEEE Visualization 2003

Adaptive Tetrapuzzles – Dense meshes
Gobbetti/Marton (CRS4),
Cignoni/Ganovelli/Ponchio/Scopigno (CNR)
SIGGRAPH 2004

Specialize

Chunked Multi-Triangulations
Gobbetti/Marton (CRS4), Cignoni/Ganovelli/Ponchio/Scopigno (CNR)
IEEE Viz 2005

Generalize

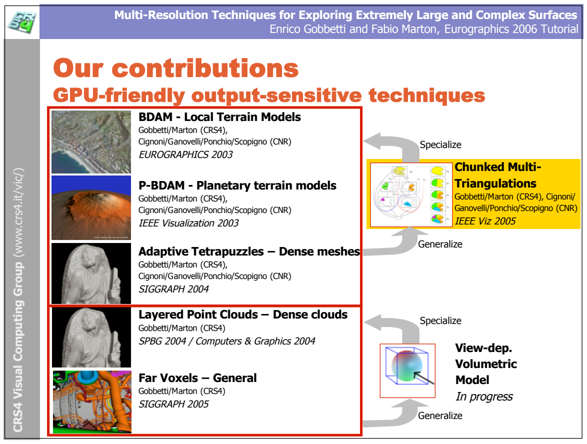
Layered Point Clouds – Dense clouds
Gobbetti/Marton (CRS4)
SPBG 2004 / Computers & Graphics 2004

Far Voxels – General
Gobbetti/Marton (CRS4)
SIGGRAPH 2005

Specialize

View-dep. Volumetric Model
In progress

Generalize



CRS4 Visual Computing Group (www.crs4.it/vic/)

Our contributions

GPU-friendly output-sensitive techniques

BDAM - Local Terrain Models
Gobbetti/Marton (CRS4),
Cignoni/Ganovelli/Ponchio/Scopigno (CNR)
EUROGRAPHICS 2003

P-BDAM - Planetary terrain models
Gobbetti/Marton (CRS4),
Cignoni/Ganovelli/Ponchio/Scopigno (CNR)
IEEE Visualization 2003

Adaptive Tetrapuzzles – Dense meshes
Gobbetti/Marton (CRS4),
Cignoni/Ganovelli/Ponchio/Scopigno (CNR)
SIGGRAPH 2004

Specialize

Chunked Multi-Triangulations
Gobbetti/Marton (CRS4), Cignoni/Ganovelli/Ponchio/Scopigno (CNR)
IEEE Viz 2005

Generalize

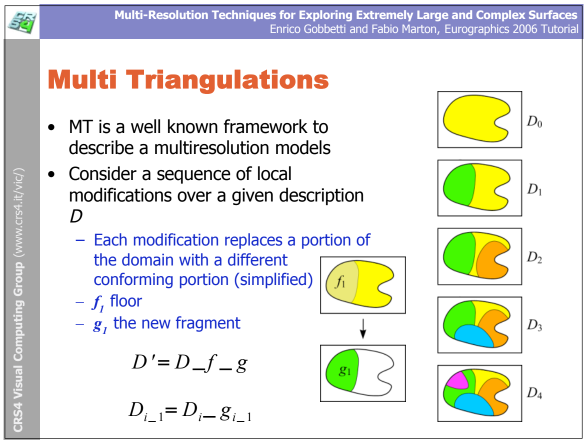
Layered Point Clouds – Dense clouds
Gobbetti/Marton (CRS4)
SPBG 2004 / Computers & Graphics 2004

Far Voxels – General
Gobbetti/Marton (CRS4)
SIGGRAPH 2005

Specialize

View-dep. Volumetric Model
In progress

Generalize



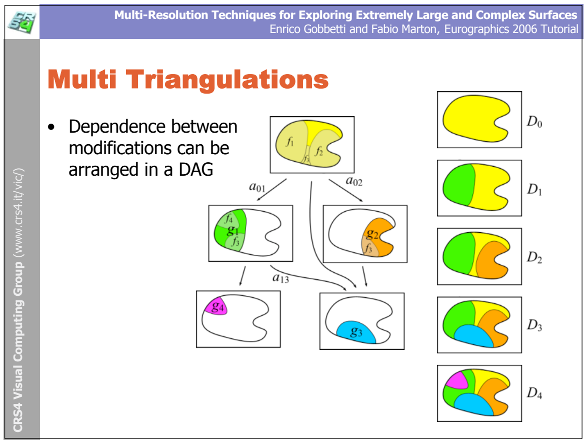
CRS4 Visual Computing Group (www.crs4.it/vic/)

Multi Triangulations

- MT is a well known framework to describe a multiresolution models
- Consider a sequence of local modifications over a given description *D*
  - Each modification replaces a portion of the domain with a different conforming portion (simplified)
  - f_1 floor
  - g_1 the new fragment

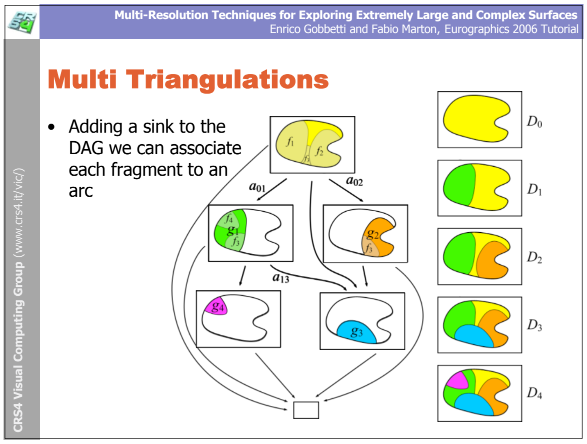
$$D' = D \_ f \_ g$$

$$D_{i\_1} = D_{i} \_ g_{i\_1}$$



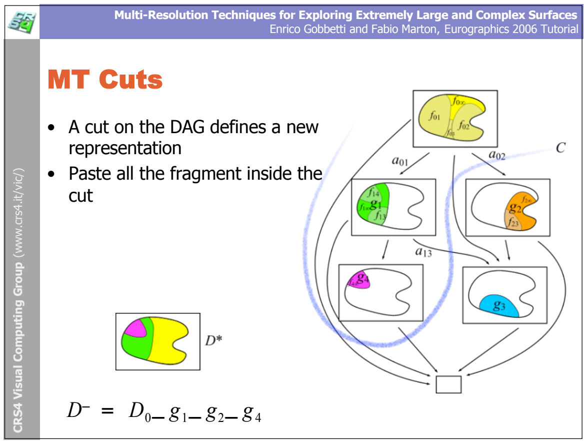
Multi Triangulations

- Dependence between modifications can be arranged in a DAG



Multi Triangulations

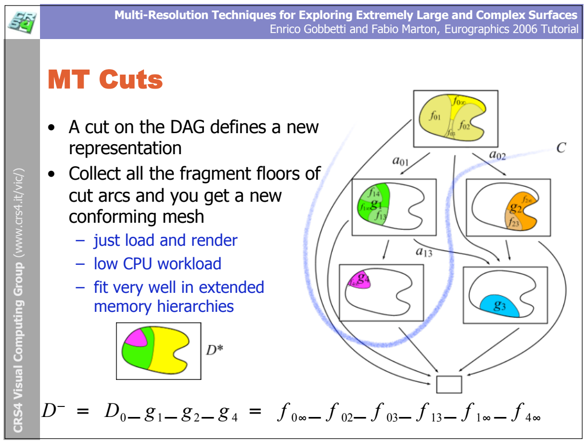
- Adding a sink to the DAG we can associate each fragment to an arc



MT Cuts

- A cut on the DAG defines a new representation
- Paste all the fragment inside the cut

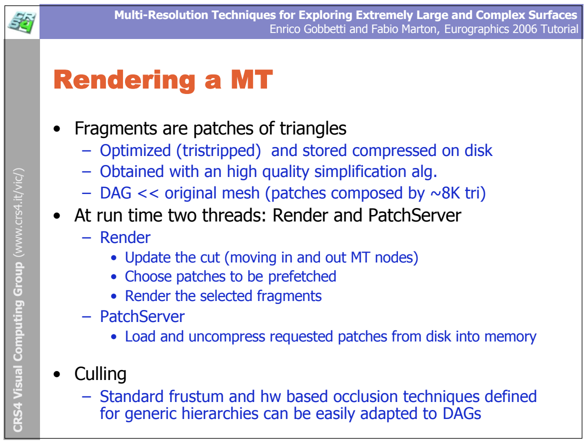
$$D^- = D_0 - g_1 - g_2 - g_4$$



MT Cuts

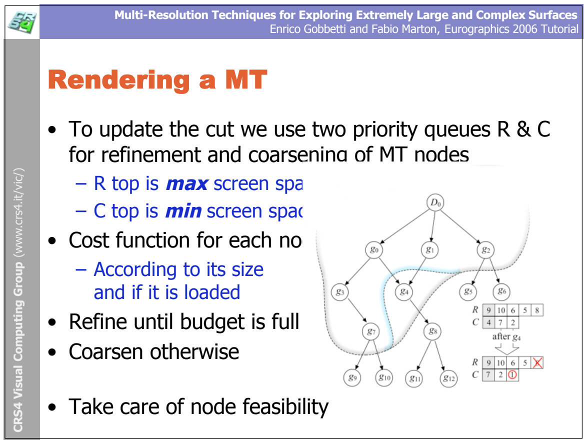
- A cut on the DAG defines a new representation
- Collect all the fragment floors of cut arcs and you get a new conforming mesh
  - just load and render
  - low CPU workload
  - fit very well in extended memory hierarchies

$$D^- = D_0 - g_1 - g_2 - g_4 = f_{0\infty} - f_{02} - f_{03} - f_{13} - f_{1\infty} - f_{4\infty}$$



Rendering a MT

- Fragments are patches of triangles
  - Optimized (tristripped) and stored compressed on disk
  - Obtained with an high quality simplification alg.
  - DAG << original mesh (patches composed by ~8K tri)
- At run time two threads: Render and PatchServer
  - Render
    - Update the cut (moving in and out MT nodes)
    - Choose patches to be prefetched
    - Render the selected fragments
  - PatchServer
    - Load and uncompress requested patches from disk into memory
- Culling
  - Standard frustum and hw based occlusion techniques defined for generic hierarchies can be easily adapted to DAGs

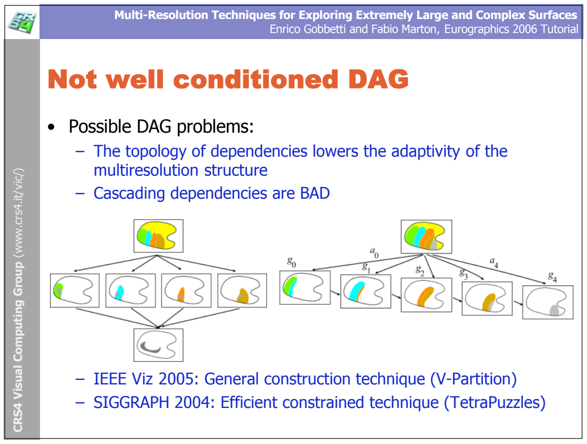


Rendering a MT

- To update the cut we use two priority queues R & C for refinement and coarsening of MT nodes
  - R top is ***max*** screen spa
  - C top is ***min*** screen spac
- Cost function for each no
  - According to its size and if it is loaded
- Refine until budget is full
- Coarsen otherwise
- Take care of node feasibility

R	9	10	6	5	8
C	4	7	2		

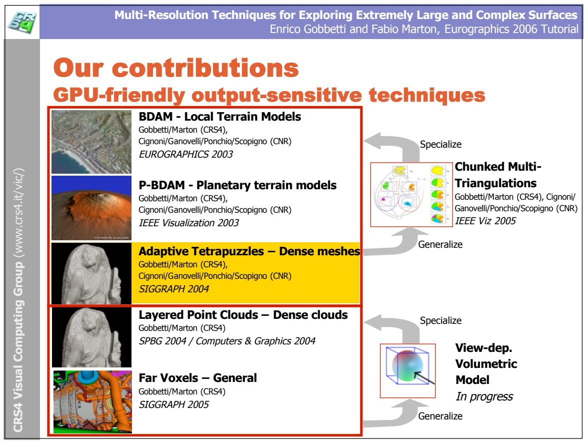
R	9	10	6	5	8
C	7	2	1		



Not well conditioned DAG

- Possible DAG problems:
  - The topology of dependencies lowers the adaptivity of the multiresolution structure
  - Cascading dependencies are BAD

  - IEEE Viz 2005: General construction technique (V-Partition)
  - SIGGRAPH 2004: Efficient constrained technique (TetraPuzzles)



Our contributions

GPU-friendly output-sensitive techniques

BDAM - Local Terrain Models
Gobbetti/Marton (CRS4),
Cignoni/Ganovelli/Ponchio/Scopigno (CNR)
EUROGRAPHICS 2003

P-BDAM - Planetary terrain models
Gobbetti/Marton (CRS4),
Cignoni/Ganovelli/Ponchio/Scopigno (CNR)
IEEE Visualization 2003

Adaptive Tetrapuzzles – Dense meshes
Gobbetti/Marton (CRS4),
Cignoni/Ganovelli/Ponchio/Scopigno (CNR)
SIGGRAPH 2004

Layered Point Clouds – Dense clouds
Gobbetti/Marton (CRS4)
SPBG 2004 / Computers & Graphics 2004

Far Voxels – General
Gobbetti/Marton (CRS4)
SIGGRAPH 2005

Specialize

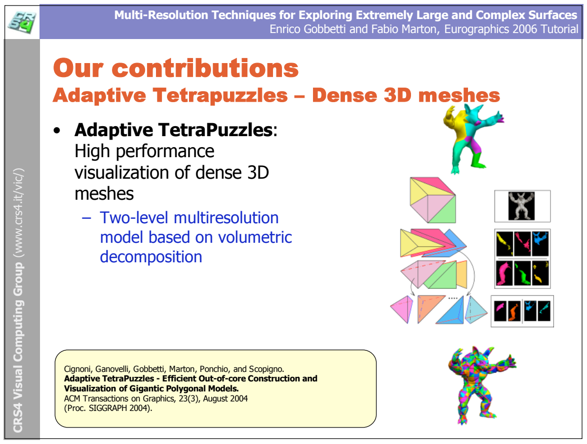
Chunked Multi-Triangulations
Gobbetti/Marton (CRS4), Cignoni/Ganovelli/Ponchio/Scopigno (CNR)
IEEE Viz 2005

Generalize

Specialize

View-dep. Volumetric Model
In progress

Generalize

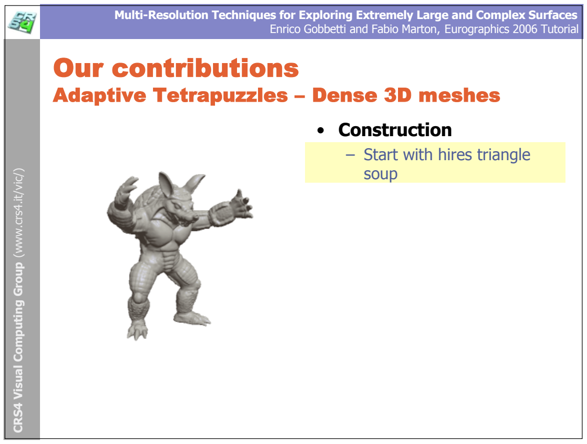


Our contributions

Adaptive Tetrapuzzles – Dense 3D meshes

- **Adaptive TetraPuzzles**: High performance visualization of dense 3D meshes
  - Two-level multiresolution model based on volumetric decomposition

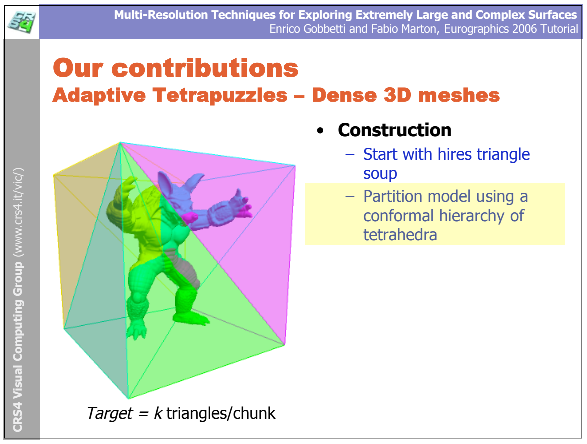
Cignoni, Ganovelli, Gobbetti, Marton, Ponchio, and Scopigno.
Adaptive TetraPuzzles - Efficient Out-of-core Construction and Visualization of Gigantic Polygonal Models.
ACM Transactions on Graphics, 23(3), August 2004
(Proc. SIGGRAPH 2004).



Our contributions

Adaptive Tetrapuzzles – Dense 3D meshes

- **Construction**
  - Start with hires triangle soup

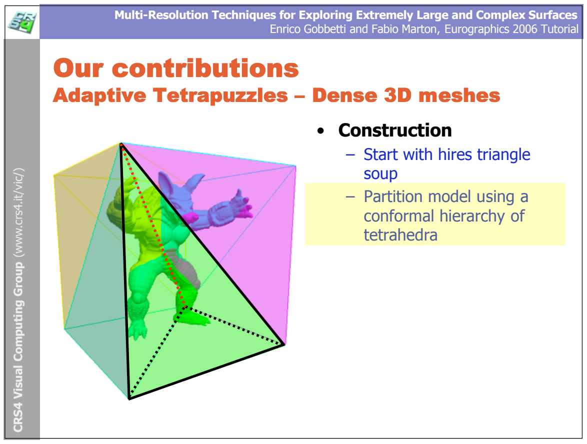


Our contributions

Adaptive Tetrapuzzles – Dense 3D meshes

- **Construction**
  - Start with hires triangle soup
  - Partition model using a conformal hierarchy of tetrahedra

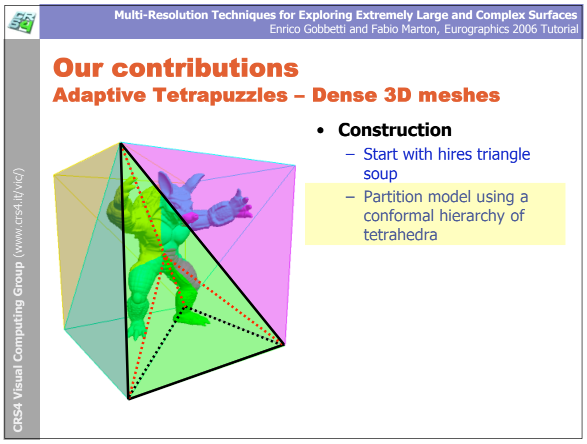
Target = *k* triangles/chunk



Our contributions

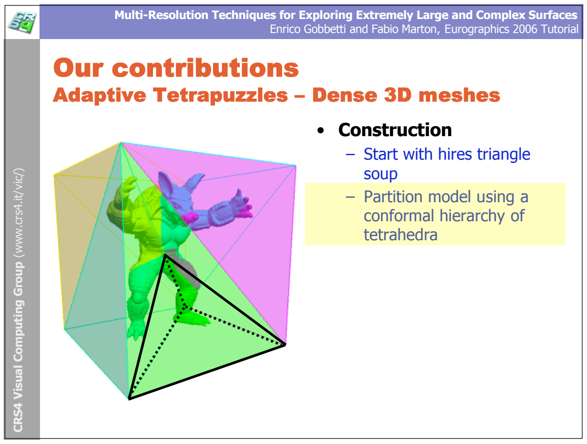
Adaptive Tetrapuzzles – Dense 3D meshes

- **Construction**
  - Start with hires triangle soup
  - Partition model using a conformal hierarchy of tetrahedra



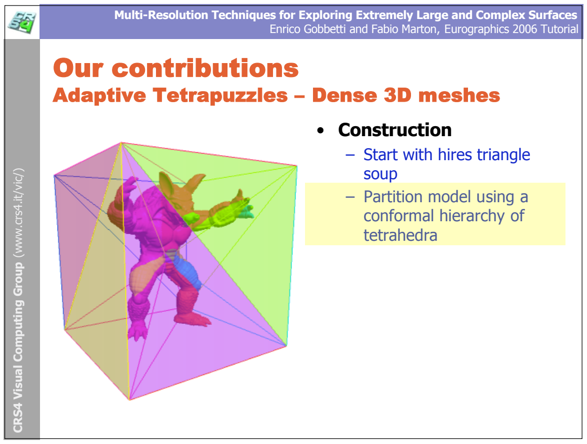
Our contributions
Adaptive Tetrapuzzles – Dense 3D meshes

- **Construction**
  - Start with hires triangle soup
  - Partition model using a conformal hierarchy of tetrahedra



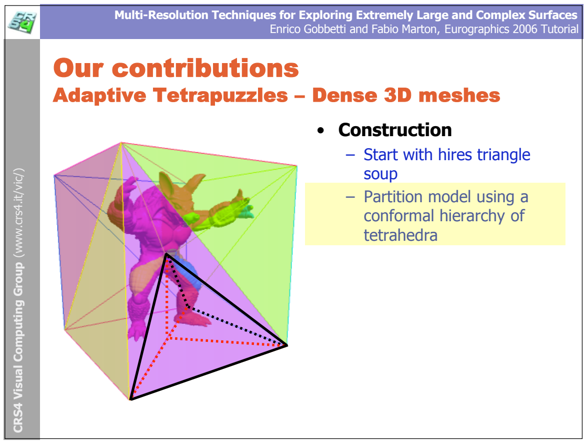
Our contributions
Adaptive Tetrapuzzles – Dense 3D meshes

- **Construction**
  - Start with hires triangle soup
  - Partition model using a conformal hierarchy of tetrahedra



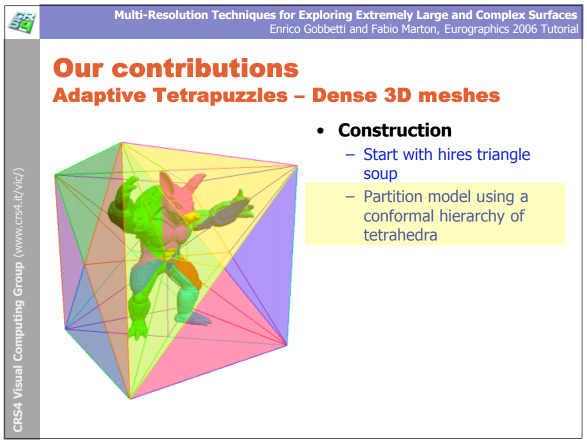
Our contributions
Adaptive Tetrapuzzles – Dense 3D meshes

- **Construction**
  - Start with hires triangle soup
  - Partition model using a conformal hierarchy of tetrahedra



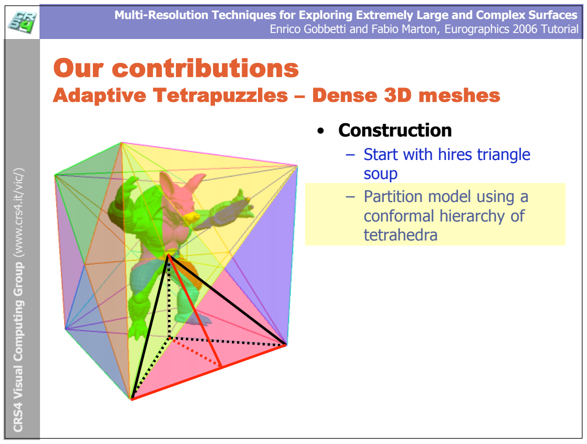
Our contributions
Adaptive Tetrapuzzles – Dense 3D meshes

- **Construction**
  - Start with hires triangle soup
  - Partition model using a conformal hierarchy of tetrahedra



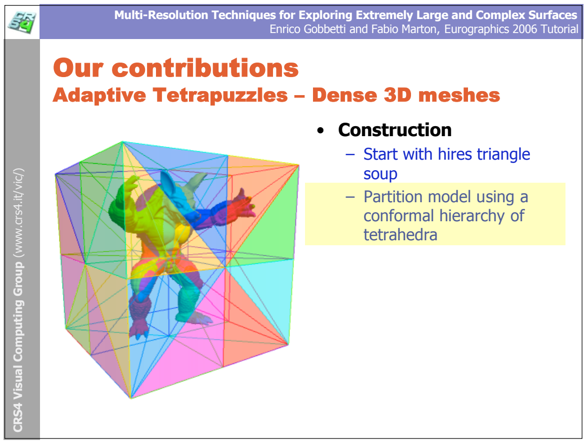
Our contributions
Adaptive Tetrapuzzles – Dense 3D meshes

- **Construction**
  - Start with hires triangle soup
  - Partition model using a conformal hierarchy of tetrahedra



Our contributions
Adaptive Tetrapuzzles – Dense 3D meshes

- **Construction**
  - Start with hires triangle soup
  - Partition model using a conformal hierarchy of tetrahedra

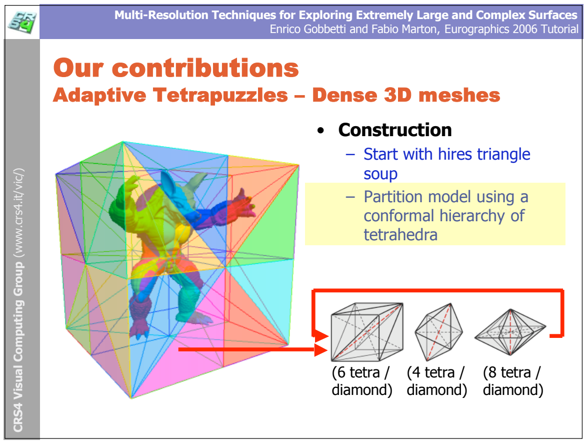


Multi-Resolution Techniques for Exploring Extremely Large and Complex Surfaces
Enrico Gobbetti and Fabio Marton, Eurographics 2006 Tutorial

CRS4 Visual Computing Group (www.crs4.it/vic/)

Our contributions
Adaptive Tetrapuzzles – Dense 3D meshes

- Construction
  - Start with hires triangle soup
  - Partition model using a conformal hierarchy of tetrahedra

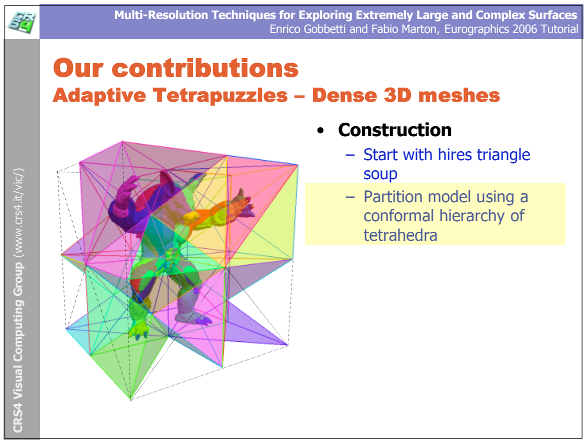


Multi-Resolution Techniques for Exploring Extremely Large and Complex Surfaces
Enrico Gobbetti and Fabio Marton, Eurographics 2006 Tutorial

CRS4 Visual Computing Group (www.crs4.it/vic/)

Our contributions
Adaptive Tetrapuzzles – Dense 3D meshes

- Construction
  - Start with hires triangle soup
  - Partition model using a conformal hierarchy of tetrahedra

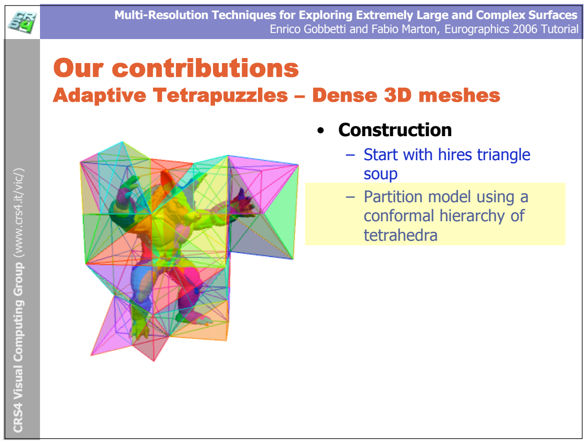


Multi-Resolution Techniques for Exploring Extremely Large and Complex Surfaces
Enrico Gobbetti and Fabio Marton, Eurographics 2006 Tutorial

CRS4 Visual Computing Group (www.crs4.it/vic/)

Our contributions
Adaptive Tetrapuzzles – Dense 3D meshes

- Construction
  - Start with hires triangle soup
  - Partition model using a conformal hierarchy of tetrahedra

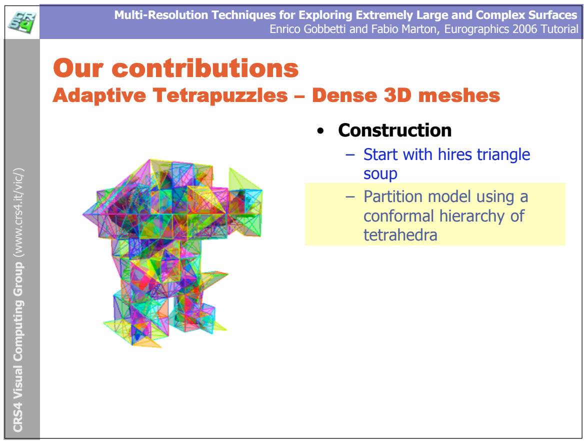


Multi-Resolution Techniques for Exploring Extremely Large and Complex Surfaces
Enrico Gobbetti and Fabio Marton, Eurographics 2006 Tutorial

CRS4 Visual Computing Group (www.crs4.it/vic/)

Our contributions
Adaptive Tetrapuzzles – Dense 3D meshes

- Construction
  - Start with hires triangle soup
  - Partition model using a conformal hierarchy of tetrahedra

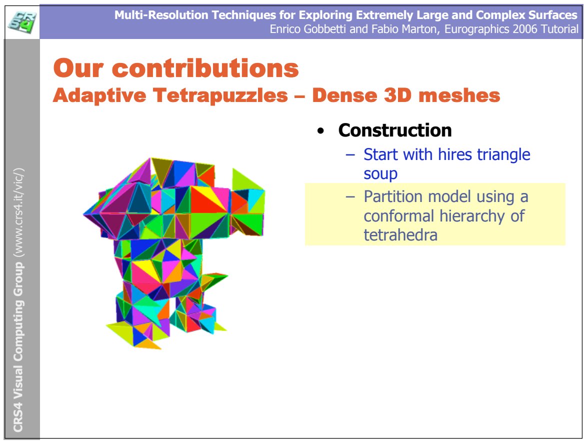


Multi-Resolution Techniques for Exploring Extremely Large and Complex Surfaces
Enrico Gobbetti and Fabio Marton, Eurographics 2006 Tutorial

CRS4 Visual Computing Group (www.crs4.it/vic/)

Our contributions
Adaptive Tetrapuzzles – Dense 3D meshes

- Construction
  - Start with hires triangle soup
  - Partition model using a conformal hierarchy of tetrahedra



Multi-Resolution Techniques for Exploring Extremely Large and Complex Surfaces
Enrico Gobbetti and Fabio Marton, Eurographics 2006 Tutorial

CRS4 Visual Computing Group (www.crs4.it/vic/)

Our contributions
Adaptive Tetrapuzzles – Dense 3D meshes

- Construction
  - Start with hires triangle soup
  - Partition model using a conformal hierarchy of tetrahedra

Our contributions
Adaptive Tetrapuzzles – Dense 3D meshes

k triangles/chunk

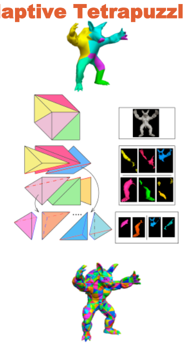
- **Construction**
  - Start with hires triangle soup
  - Partition model using a conformal hierarchy of tetrahedra

Our contributions
Adaptive Tetrapuzzles – Dense 3D meshes

k triangles/chunk

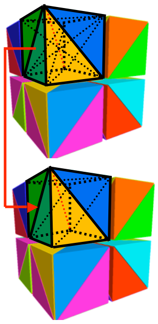
- **Construction**
  - Start with hires triangle soup
  - Partition model using a conformal hierarchy of tetrahedra

Our contributions
Adaptive Tetrapuzzles – Dense 3D meshes



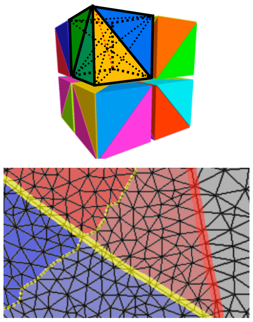
- **Construction**
  - Start with hires triangle soup
  - Partition model using a conformal hierarchy of tetrahedra
  - Construct non-leaf cells by bottom-up recombination and simplification of lower level cells

Our contributions
Adaptive Tetrapuzzles – Dense 3D meshes



- **Construction**
  - Start with hires triangle soup
  - Partition model using a conformal hierarchy of tetrahedra
  - Construct non-leaf cells by bottom-up recombination and simplification of lower level cells

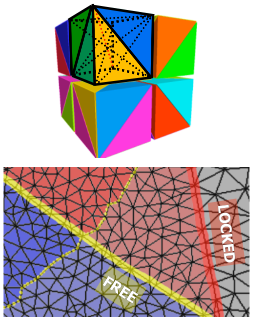
Our contributions
Adaptive Tetrapuzzles – Dense 3D meshes



- **Construction**
  - Start with hires triangle soup
  - Partition model using a conformal hierarchy of tetrahedra
  - Construct non-leaf cells by bottom-up recombination and simplification of lower level cells

🟥 Diamond external boundary
⋯ Diamond internal boundary
— Child tetrahedra boundary

Our contributions
Adaptive Tetrapuzzles – Dense 3D meshes

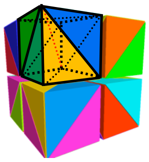


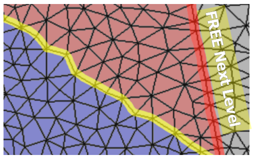
- **Construction**
  - Start with hires triangle soup
  - Partition model using a conformal hierarchy of tetrahedra
  - Construct non-leaf cells by bottom-up recombination and simplification of lower level cells

🟥 Diamond external boundary
⋯ Diamond internal boundary
— Child tetrahedra boundary

Our contributions

Adaptive Tetrapuzzles – Dense 3D meshes



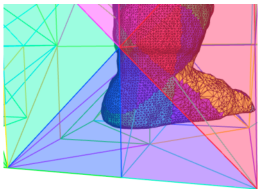


- **Construction**
  - Start with hires triangle soup
  - Partition model using a conformal hierarchy of tetrahedra
  - Construct non-leaf cells by bottom-up recombination and simplification of lower level cells

🟥 Diamond external boundary
🟨 Diamond internal boundary

Our contributions

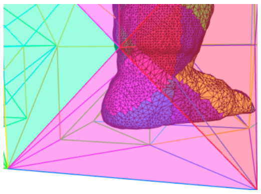
Adaptive Tetrapuzzles – Dense 3D meshes



- **Construction**
  - Start with hires triangle soup
  - Partition model using a conformal hierarchy of tetrahedra
  - Construct non-leaf cells by bottom-up recombination and simplification of lower level cells

Our contributions

Adaptive Tetrapuzzles – Dense 3D meshes

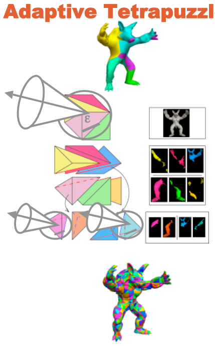


- **Construction**
  - Start with hires triangle soup
  - Partition model using a conformal hierarchy of tetrahedra
  - Construct non-leaf cells by bottom-up recombination and simplification of lower level cells

NO CRACKS / NO GLOBALLY LOCKED BOUNDARY!

Our contributions

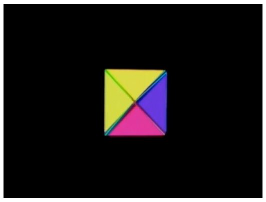
Adaptive Tetrapuzzles – Dense 3D meshes



- **Construction**
  - Start with hires triangle soup
  - Partition model using a conformal hierarchy of tetrahedra
  - Construct non-leaf cells by bottom-up recombination and simplification of lower level cells
- **Rendering**
  - Refine conformal hierarchy, render selected precomputed cells

Our contributions

Adaptive Tetrapuzzles – Dense 3D meshes



View dependent mesh refinement

- **Construction**
  - Start with hires triangle soup
  - Partition model using a conformal hierarchy of tetrahedra
  - Construct non-leaf cells by bottom-up recombination and simplification of lower level cells
- **Rendering**
  - Refine conformal hierarchy, render selected precomputed cells

Our contributions

Adaptive Tetrapuzzles – Dense 3D meshes

Independent diamond processing

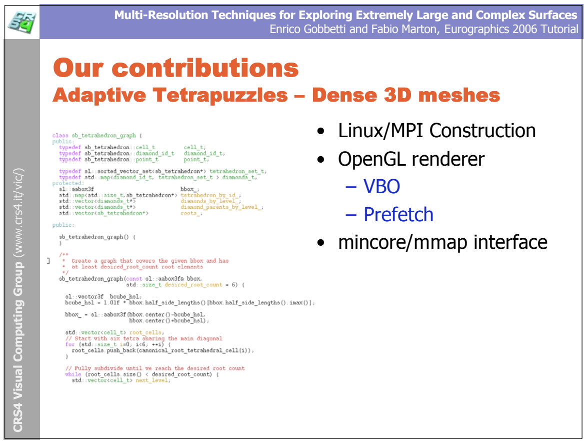
For each mesh chunk:
Simplify + stripify +
compress + eval bounds/error

Out-of-core + parallel

- **Construction**
  - Start with hires triangle soup
  - Partition model using a conformal hierarchy of tetrahedra
  - Construct non-leaf cells by bottom-up recombination and simplification of lower level cells

Out-of-core cull+refine traversal / GPU cached optimized meshes

- **Rendering**
  - Refine conformal hierarchy, render selected precomputed cells

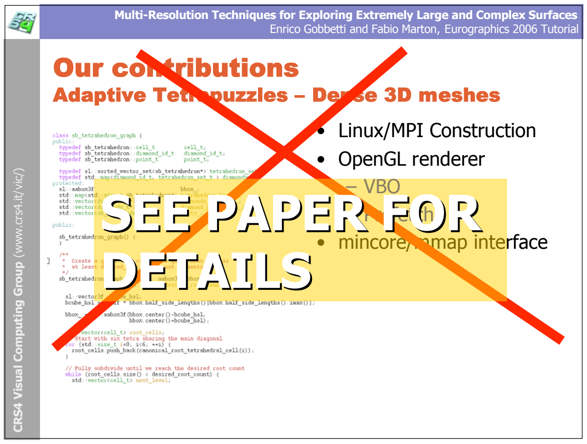


Our contributions

Adaptive Tetrapuzzles – Dense 3D meshes

```
class sb_tetrahedron_graph {
public:
  typedef sb_tetrahedron::cell_t         cell_t;
  typedef sb_tetrahedron::diamond_id_t   diamond_id_t;
  typedef sb_tetrahedron::point_t        point_t;

  typedef sl::sorted_vector_set<sb_tetrahedron*> tetrahedron_set_t;
  typedef std::map<diamond_id_t, tetrahedron_set_t > diamonds_t;
protected:
  sl::aabox3f                             bbox_;
  std::map<std::size_t,sb_tetrahedron*> tetrahedron_by_id_;
  std::vector<diamonds_t*>                diamonds_by_level_;
  std::vector<diamonds_t*>                diamond_parents_by_level_;
  std::vector<sb_tetrahedron*>            roots_;

public:

  sb_tetrahedron_graph() {
  }

  /**
]  *  Create a graph that covers the given bbox and has
   *  at least desired_root_count root elements
   */
  sb_tetrahedron_graph(const sl::aabox3f& bbox,
                       std::size_t desired_root_count = 6) {

    sl::vector3f  bcube_hsl;
    bcube_hsl = 1.01f * bbox.half_side_lengths()[bbox.half_side_lengths().imax()];

    bbox_ = sl::aabox3f(bbox.center()-bcube_hsl,
                        bbox.center()+bcube_hsl);

    std::vector<cell_t> root_cells;
    // Start with six tetra sharing the main diagonal
    for (std::size_t i=0; i<6; ++i) {
      root_cells.push_back(canonical_root_tetrahedral_cell(i));
    }

    // Fully subdivide until we reach the desired root count
    while (root_cells.size() < desired_root_count) {
      std::vector<cell_t> next_level;
```

- Linux/MPI Construction
- OpenGL renderer
  - VBO
  - Prefetch
- mincore/mmap interface

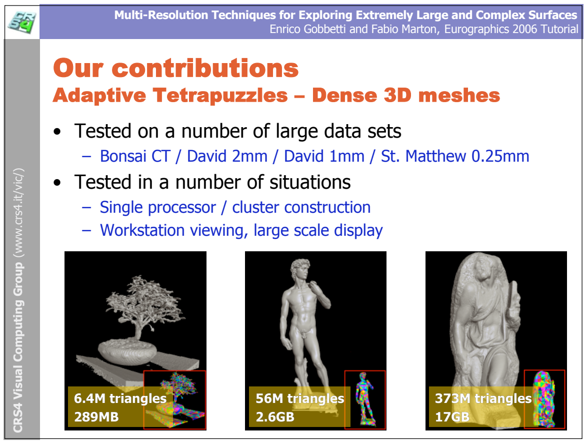


Our contributions

Adaptive Tetrapuzzles – Dense 3D meshes

- Linux/MPI Construction
- OpenGL renderer
  - VBO
  - Prefetch
- mincore/mmap interface

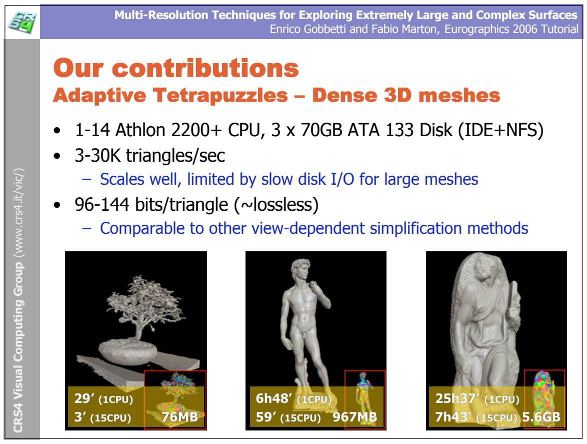
SEE PAPER FOR DETAILS



Our contributions

Adaptive Tetrapuzzles – Dense 3D meshes

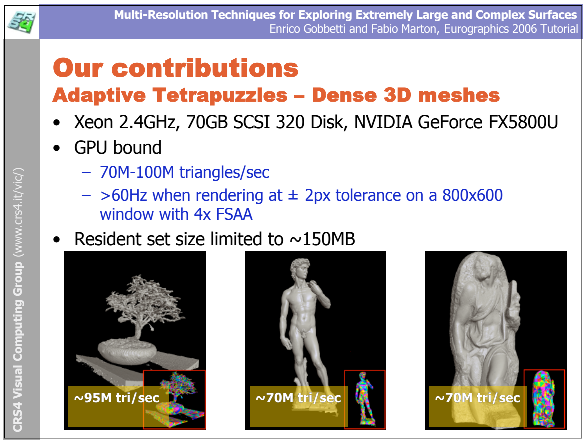
- Tested on a number of large data sets
  - Bonsai CT / David 2mm / David 1mm / St. Matthew 0.25mm
- Tested in a number of situations
  - Single processor / cluster construction
  - Workstation viewing, large scale display



Our contributions

Adaptive Tetrapuzzles – Dense 3D meshes

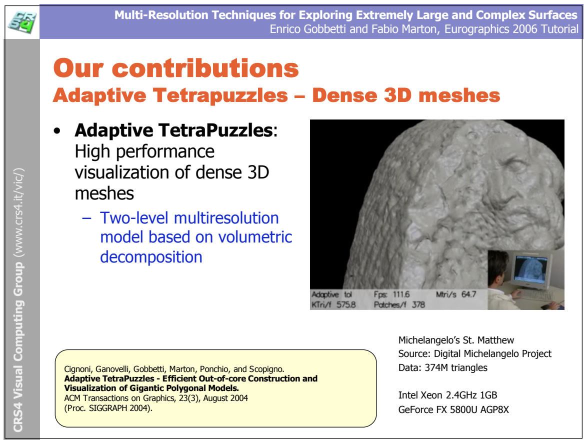
- 1-14 Athlon 2200+ CPU, 3 x 70GB ATA 133 Disk (IDE+NFS)
- 3-30K triangles/sec
  - Scales well, limited by slow disk I/O for large meshes
- 96-144 bits/triangle (~lossless)
  - Comparable to other view-dependent simplification methods



Our contributions

Adaptive Tetrapuzzles – Dense 3D meshes

- Xeon 2.4GHz, 70GB SCSI 320 Disk, NVIDIA GeForce FX5800U
- GPU bound
  - 70M-100M triangles/sec
  - >60Hz when rendering at ± 2px tolerance on a 800x600 window with 4x FSAA
- Resident set size limited to ~150MB



Our contributions

Adaptive Tetrapuzzles – Dense 3D meshes

- **Adaptive TetraPuzzles**: High performance visualization of dense 3D meshes
  - Two-level multiresolution model based on volumetric decomposition

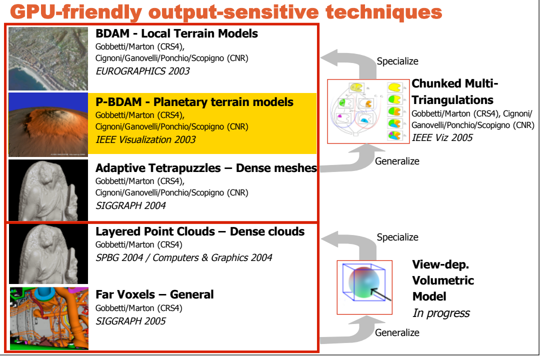
Michelangelo's St. Matthew
Source: Digital Michelangelo Project
Data: 374M triangles

Intel Xeon 2.4GHz 1GB
GeForce FX 5800U AGP8X

Cignoni, Ganovelli, Gobbetti, Marton, Ponchio, and Scopigno.
Adaptive TetraPuzzles - Efficient Out-of-core Construction and Visualization of Gigantic Polygonal Models.
ACM Transactions on Graphics, 23(3), August 2004
(Proc. SIGGRAPH 2004).

Our contributions

GPU-friendly output-sensitive techniques



BDAM - Local Terrain Models
Gobbetti/Marton (CRS4),
Cignoni/Ganovelli/Ponchio/Scopigno (CNR)
EUROGRAPHICS 2003

P-BDAM - Planetary terrain models
Gobbetti/Marton (CRS4),
Cignoni/Ganovelli/Ponchio/Scopigno (CNR)
IEEE Visualization 2003

Adaptive Tetrapuzzles – Dense meshes
Gobbetti/Marton (CRS4),
Cignoni/Ganovelli/Ponchio/Scopigno (CNR)
SIGGRAPH 2004

Layered Point Clouds – Dense clouds
Gobbetti/Marton (CRS4)
SPBG 2004 / Computers & Graphics 2004

Far Voxels – General
Gobbetti/Marton (CRS4)
SIGGRAPH 2005

Specialize

Chunked Multi-Triangulations
Gobbetti/Marton (CRS4), Cignoni/Ganovelli/Ponchio/Scopigno (CNR)
IEEE Viz 2005

Generalize

Specialize

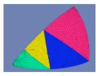
View-dep. Volumetric Model
In progress

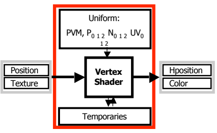
Generalize

Our contributions

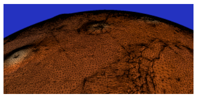
P-BDAM – Planetary terrain models

- **P-BDAM**: High performance planetary terrain visualization technique
  - Handles planet curvature
  - The only accelerated technique with sub-metric global accuracy on entire Earth
  - Parallel construction method





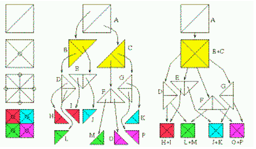
Cignoni, Ganovelli, Gobbetti, Marton, Ponchio, and Scopigno.
Planet-Sized Batched Dynamic Adaptive Meshes (P-BDAM).
In Proceedings IEEE Visualization. Pages 147-155., October 2003.

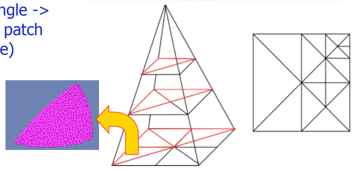


Our contributions

P-BDAM – Planetary terrain models

- Geometry multiresolution data structure
  - Pair of triangle bintrees
    - Each triangle is recursively bisected by splitting it along its longest edge
  - Base domain triangle -> curved triangular patch (displaced triangle)

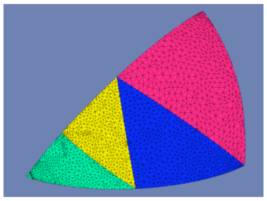


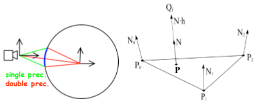


Our contributions

P-BDAM – Planetary terrain models

- Displaced triangle
  - General triangle mesh hi-quality adaptively simplified + stripified during preprocessing
  - Takes into account planet curvature / size
    - 3 double precision corner coordinate
    - Mesh vertex positions computed on GPU from parametric coords
  - Preserves connectivity among adjacent levels using matched triangulations
- Global continuity, compression, submetric accuracy on Earth

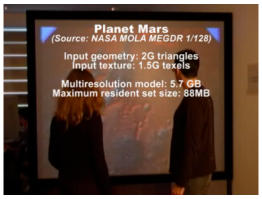




Our contributions

P-BDAM – Planetary terrain models

- **P-BDAM**: High performance planetary terrain visualization technique
  - Handles Earth curvature
  - The only accelerated technique with sub-metric global accuracy on entire Earth
  - Parallel construction method


Planet Mars
(Source: NASA MOLA MEGDR 1/128)
Input geometry: 2G triangles
Input texture: 1.5G texels
Multiresolution model: 5.7 GB
Maximum resident set size: 88MB

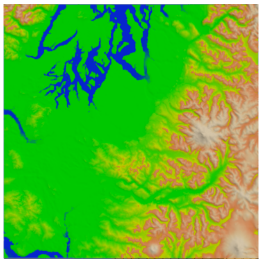
Cignoni, Ganovelli, Gobbetti, Marton, Ponchio, and Scopigno.
Planet-Sized Batched Dynamic Adaptive Meshes (P-BDAM).
In Proceedings IEEE Visualization. Pages 147-155., October 2003.

Planet Mars
Source: NASA MOLA MEGDR
Elevation: 44Kx22K / Color: 44Kx22K

Rendering: 2x1024x768 @ 1px accuracy
Intel Xeon 2.4GHz 1GB
GeForce 6800GT AGP8X

Compression: Regular Grid

- Regular grid instead of irregular triangulation
- Pros:
  - Fastest preprocessing
  - More compact / Less data to send to GPU
  - Easier to apply signal processing algorithms
- Cons:
  - Needs more data to approximate irregular features.

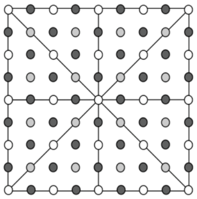


Different levels of resolution of regular grid data of a planar terrain representation

Regular Grid Data Encoding

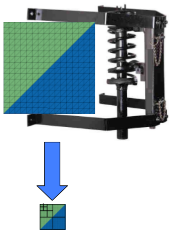
- Compression idea
  - Encode (quantized) new points delta with respect to prediction from previous level
- Hierarchy is a non standard wavelet decomposition of the original dataset!

=> BDAM + Wavelets!



Detail Compression

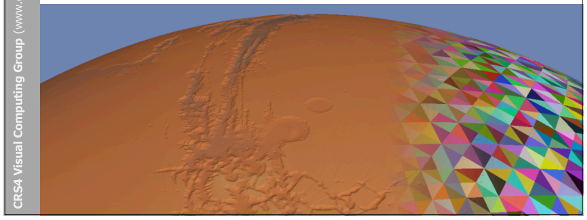
- Possibility to apply various compression algorithms to each patch encoded detail.
- High compression ratio up to a desired error with:
  - Quantization of detail coefficients
  - Entropy coding of quantized values
- Example result: PLANET MARS
  - 1.5 Giga points
  - Compressed to 75 MB
  - Max error: 50 meters



Planet Mars

Conclusions

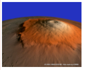
- Integrated aggressive compression in a general output sensitive framework
  - BDAM + Wavelets
- See EG paper this year (C-BDAM)



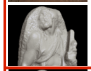
Our contributions

GPU-friendly output-sensitive techniques

BDAM - Local Terrain Models
Gobbetti/Marton (CRS4),
Cignoni/Ganovelli/Ponchio/Scopigno (CNR)
EUROGRAPHICS 2003

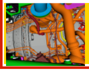


P-BDAM - Planetary terrain models
Gobbetti/Marton (CRS4),
Cignoni/Ganovelli/Ponchio/Scopigno (CNR)
IEEE Visualization 2003

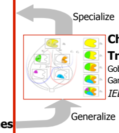


Adaptive Tetrapuzzles – Dense meshes
Gobbetti/Marton (CRS4),
Cignoni/Ganovelli/Ponchio/Scopigno (CNR)
SIGGRAPH 2004

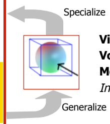
Layered Point Clouds – Dense clouds
Gobbetti/Marton (CRS4)
SPBG 2004 / Computers & Graphics 2004



Far Voxels – General
Gobbetti/Marton (CRS4)
SIGGRAPH 2005



Chunked Multi-Triangulations
Gobbetti/Marton (CRS4), Cignoni/Ganovelli/Ponchio/Scopigno (CNR)
IEEE Viz 2005

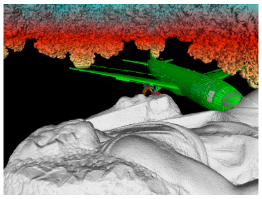


View-dep. Volumetric Model
In progress

Our contributions

Far Voxels – General 3D models

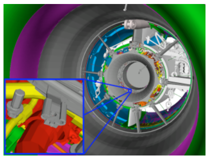
- Far Voxels: High performance visualization of arbitrary 3D models
  - Mixed model
  - Seamless integration of occlusion culling with out-of-core data management and multiresolution rendering

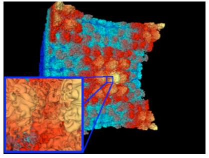


Gobbetti and Marton.
Far Voxels – A nultiresolution Framework for Interactive Rendering of Huge Complex 3D Models on Commodity Graphics Platforms.
ACM Transactions on Graphics, 23(4), August 2005
(Proc. SIGGRAPH 2005).

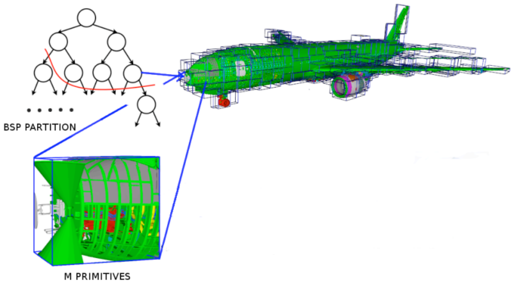
Our contributions
Far Voxels – General 3D models

- Classic multiresolution models
  - Error measured on boundary surfaces
  - LOD construction based on local surface coarsening/simplification operations
  - Visibility culling decoupled from multiresolution
- Hard to apply to models with high detail and complex topology and high depth complexity!

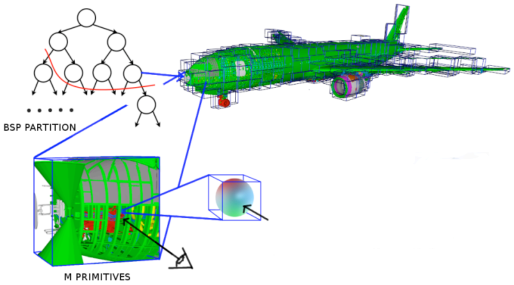




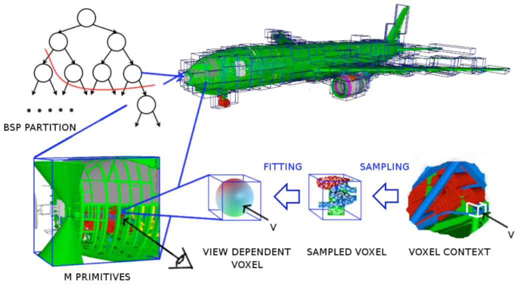
Our contributions
Far Voxels – General 3D models



Our contributions
Far Voxels – General 3D models

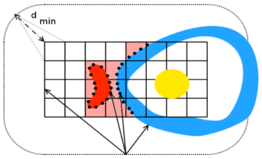


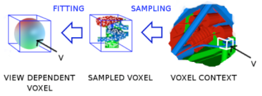
Our contributions
Far Voxels – General 3D models



Our contributions
Far Voxels – General 3D models

- Off-line: Reconstruction = sampling + fitting
  - Sampling
    - Raycasting
      - extract n,BRDF = f(ray)
    - Occlusion culling!
      - Sample from distance d_{min} dictated by maximum possible projected voxel size
  - Fitting
    - Choose best voxel representation among selected parameterized shaders
    - Error minimization
- On-line: Rendering
  - Refine until projected voxel size < desired accuracy
  - Exploit GPU for shader evaluation and on-line occlusion culling

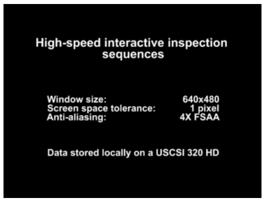




$$Shader_i(v,l) = BRDF_i(v,l)(n(v).l)_+$$

Our contributions
Far Voxels – General 3D models

- **Far Voxels**: High performance visualization of arbitrary 3D models
  - Mixed model
  - Seamless integration of occlusion culling with out-of-core data management and multiresolution rendering
  - … work in progress



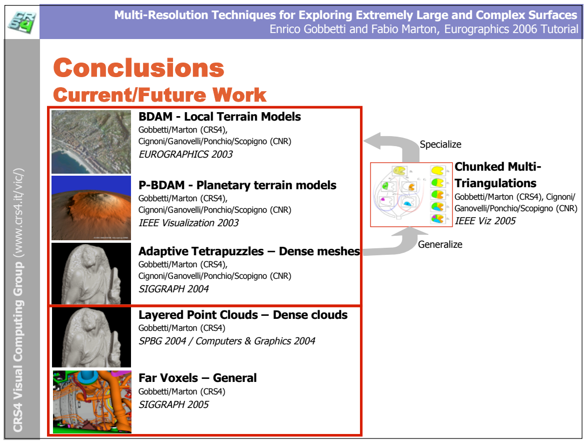
High-speed interactive inspection sequences

Window size: 640x480
Screen space tolerance: 1 pixel
Anti-aliasing: 4X FSAA

Data stored locally on a USCSI 320 HD

Intel Xeon 2.4GHz 1GB
GeForce 6800GT AGP8X

Gobbetti and Marton.
Far Voxels – A nultiresolution Framework for Interactive Rendering of Huge Complex 3D Models on Commodity Graphics Platforms.
ACM Transactions on Graphics, 23(4), August 2005
(Proc. SIGGRAPH 2005).



Conclusions
Current/Future Work

BDAM - Local Terrain Models
Gobbetti/Marton (CRS4),
Cignoni/Ganovelli/Ponchio/Scopigno (CNR)
EUROGRAPHICS 2003

P-BDAM - Planetary terrain models
Gobbetti/Marton (CRS4),
Cignoni/Ganovelli/Ponchio/Scopigno (CNR)
IEEE Visualization 2003

Adaptive Tetrapuzzles – Dense meshes
Gobbetti/Marton (CRS4),
Cignoni/Ganovelli/Ponchio/Scopigno (CNR)
SIGGRAPH 2004

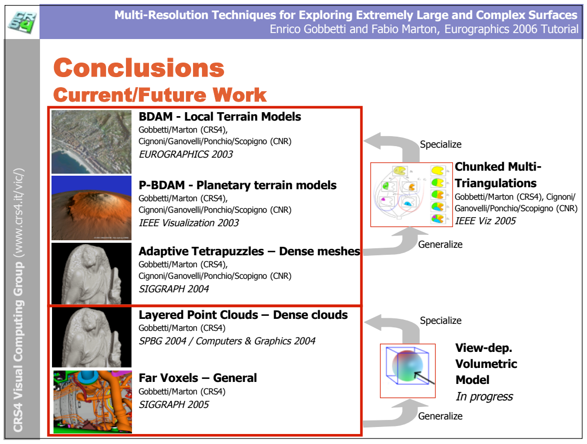
Layered Point Clouds – Dense clouds
Gobbetti/Marton (CRS4)
SPBG 2004 / Computers & Graphics 2004

Far Voxels – General
Gobbetti/Marton (CRS4)
SIGGRAPH 2005

Specialize

Chunked Multi-Triangulations
Gobbetti/Marton (CRS4), Cignoni/Ganovelli/Ponchio/Scopigno (CNR)
IEEE Viz 2005

Generalize



Conclusions
Current/Future Work

BDAM - Local Terrain Models
Gobbetti/Marton (CRS4),
Cignoni/Ganovelli/Ponchio/Scopigno (CNR)
EUROGRAPHICS 2003

P-BDAM - Planetary terrain models
Gobbetti/Marton (CRS4),
Cignoni/Ganovelli/Ponchio/Scopigno (CNR)
IEEE Visualization 2003

Adaptive Tetrapuzzles – Dense meshes
Gobbetti/Marton (CRS4),
Cignoni/Ganovelli/Ponchio/Scopigno (CNR)
SIGGRAPH 2004

Layered Point Clouds – Dense clouds
Gobbetti/Marton (CRS4)
SPBG 2004 / Computers & Graphics 2004

Far Voxels – General
Gobbetti/Marton (CRS4)
SIGGRAPH 2005

Specialize

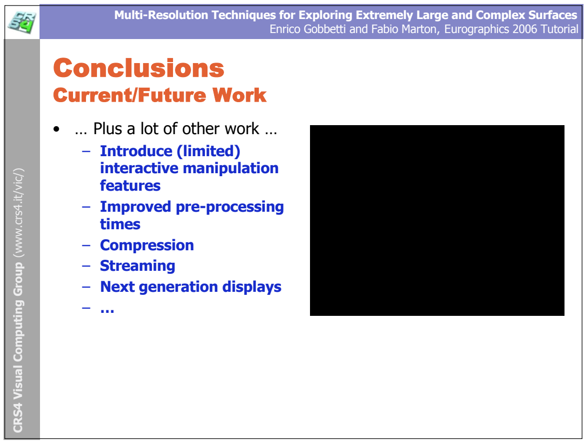
Chunked Multi-Triangulations
Gobbetti/Marton (CRS4), Cignoni/Ganovelli/Ponchio/Scopigno (CNR)
IEEE Viz 2005

Generalize

Specialize

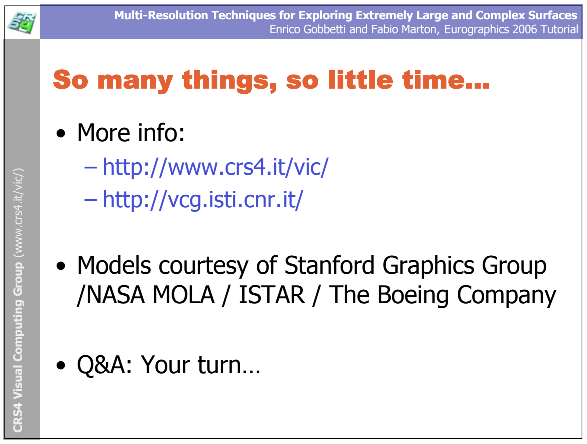
View-dep. Volumetric Model
In progress

Generalize



Conclusions
Current/Future Work

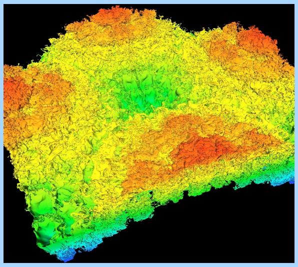
- … Plus a lot of other work …
  - **Introduce (limited) interactive manipulation features**
  - **Improved pre-processing times**
  - **Compression**
  - **Streaming**
  - **Next generation displays**
  - **…**



So many things, so little time…

- More info:
  - http://www.crs4.it/vic/
  - http://vcg.isti.cnr.it/
- Models courtesy of Stanford Graphics Group /NASA MOLA / ISTAR / The Boeing Company
- Q&A: Your turn…

Interactive Out-Of-Core Visualization of Large Datasets on Commodity PCs



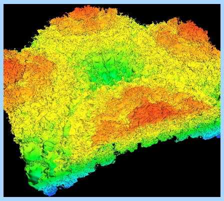
Wagner Corrêa
Research Staff Member
IBM Watson Research Center

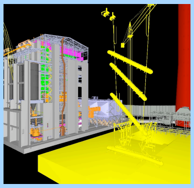
Goal

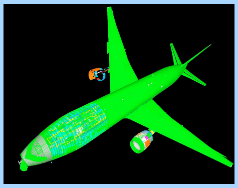
- Interactive visualization of large datasets on inexpensive PCs
  - interactive: 10 or more frames per second
  - large: larger than main memory
  - inexpensive: under $2,000 per PC

Motivations

- Large datasets have many applications
  - CAD
  - modeling and simulation
  - virtual training







Motivations (cont.)

- PCs are good alternative to high-end workstations
  - better price/performance
  - easier to upgrade

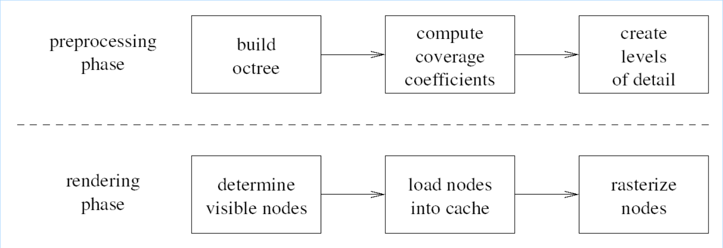
Challenges

- Datasets are larger than main memory
- High I/O latency and low I/O bandwidth
- Only one graphics pipe per PC
- Low screen resolution

Solutions

- Out-of-core preprocessing algorithms
  - spatialization, visibility precomputation, and simplification
- Out-of-core rendering algorithms
  - approximate visibility and prefetching
  - hardware-assisted conservative visibility
- Out-of-core parallel rendering algorithms
  - rendering on multi-tile screen using PC cluster

Preprocessing and Rendering



preprocessing phase

build octree

compute coverage coefficients

create levels of detail

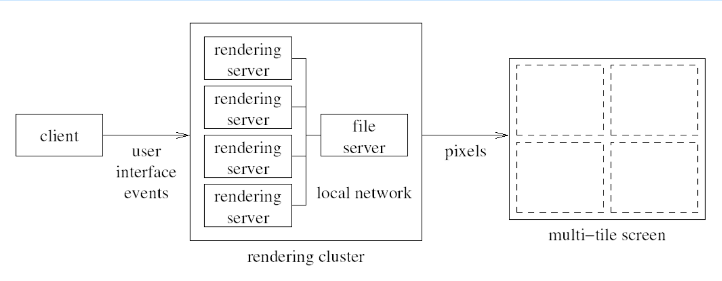
rendering phase

determine visible nodes

load nodes into cache

rasterize nodes

Parallel Rendering



client

user interface events

rendering server

rendering server

rendering server

rendering server

file server

local network

rendering cluster

pixels

multi−tile screen

Talk Outline

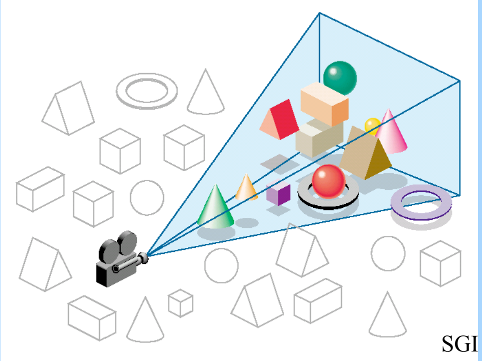
- Out-of-core preprocessing
- Out-of-core rendering
- Out-of-core parallel rendering
- Conclusions

Out-Of-Core Preprocessing

- Build an octree
  - Hierarchical frustum culling
  - Working set management
- Compute visibility coefficients
  - Occlusion culling
  - Prefetching
- Create simplified versions
  - Level-of-detail control

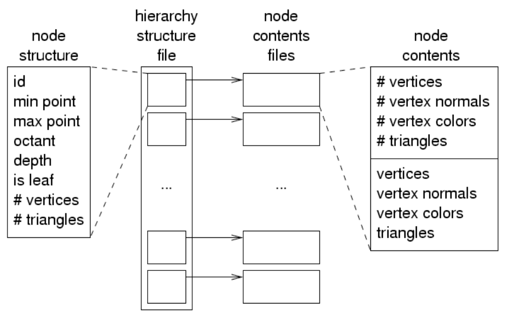
View-frustum Culling

- Clark76



SGI

Building an Octree



node structure

id
min point
max point
octant
depth
is leaf
# vertices
# triangles

hierarchy structure file

...

node contents files

...

node contents

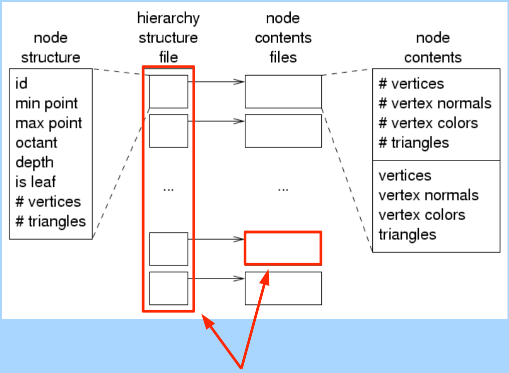
vertices
vertex normals
vertex colors
triangles

vertices
vertex normals
vertex colors
triangles

Building an Octree

- Break model in sections that fit in memory
- For each section
  - read hierarchy structure (HS) file
  - perform fake insertions
  - for each touched node
    - read old contents
    - merge old + new
    - update contents on disk
  - update HS file on disk

Building an Octree



In memory at once

Advantages of Our Spatialization Algorithm

- Out-of-core
  - we need memory for the section, the HS file, and the contents of one leaf
- Incremental
  - only updates regions touched by the section
  - important for 3D scanning
- Efficient
  - only reads a modified node once per section

Computing Visibility Coefficients

- For each node, for each viewing direction
  - compute coefficient:
    projected area of data/projected area of bbox
- Used to determine node priority at runtime

Detail Culling

- Avoid rendering unimportant details
- Also known as level-of-detail management
- LOD switching approaches
  - based on distance from viewer
  - optimized (Funkhouser93)
    - maximize image-quality (benefit)
    - given time and geometry constraints (cost)
  - based on visibility information

Creating Levels of Detail

- Several static LODs per octree node
  - uses vertex clustering [Rossignac and Borrel 93]
  - limitations: popping, different levels between adjacent nodes
- Possible improvements:
  - dynamic LODs (slower, less suitable for HW)
  - hysteresis (don't switch LODs too often)

Advantages of Vertex Clustering

- Fast and robust
- Only needs to traverse the data once
- Produces good enough approximations
- Has an intuitive, user-controlled accuracy dial
- Does not need topological adjacency graph

Preprocessing Tests

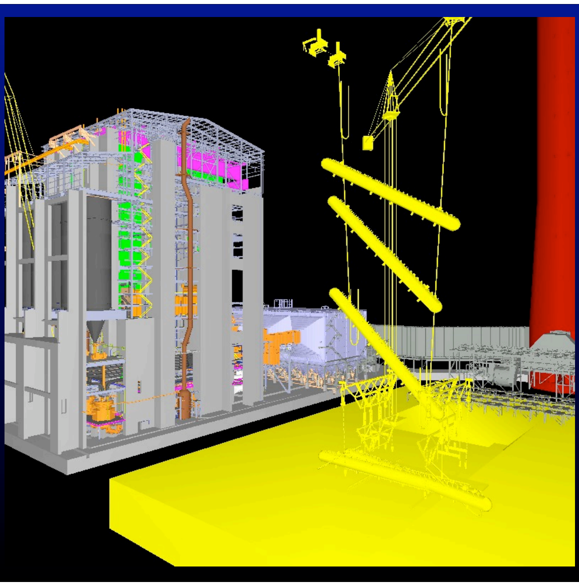
- Measure time to preprocess datasets
- Study tradeoff between spatialization granularity and octree size
- Assess quality of approximations

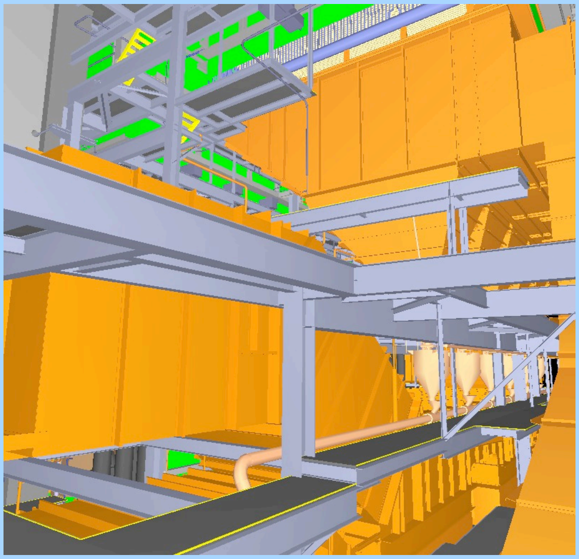
Test Datasets

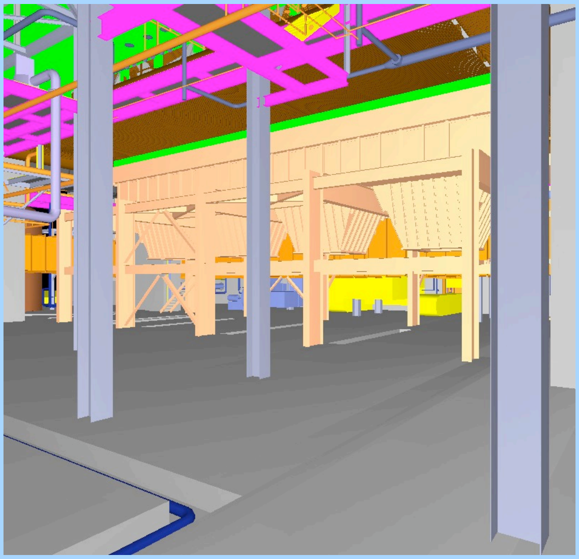
- UNC power plant
- LLNL isosurface
- Boeing 777

UNC Power Plant

- CAD model
- 13 million triangles
- High depth complexity
- 363 MB of raw data
- 1GB after preprocessing

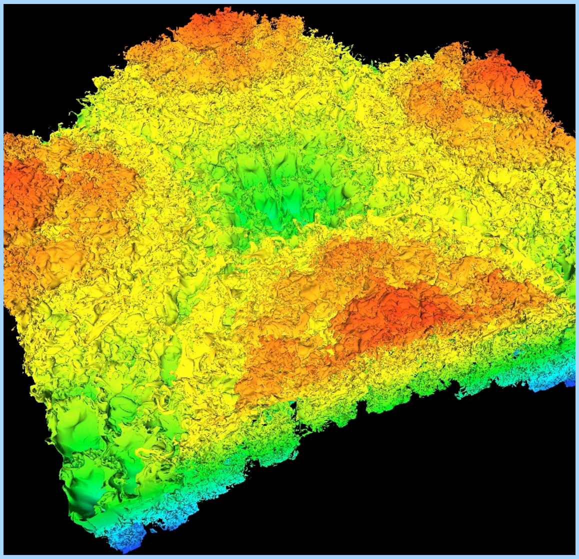






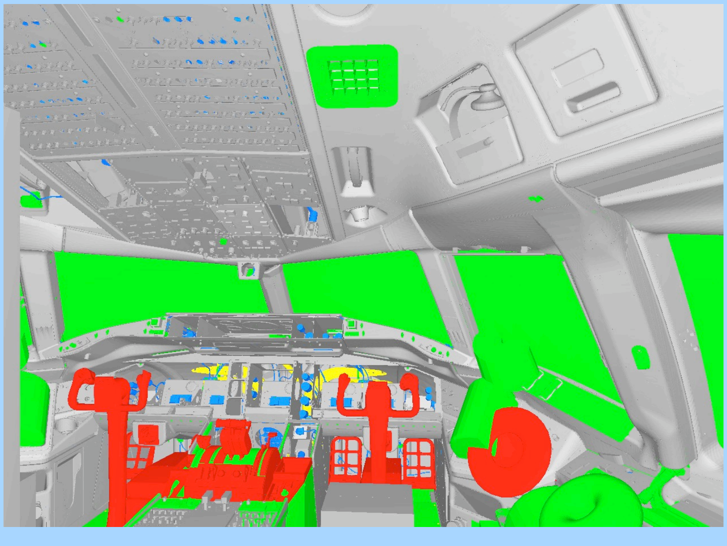
LLNL Isosurface

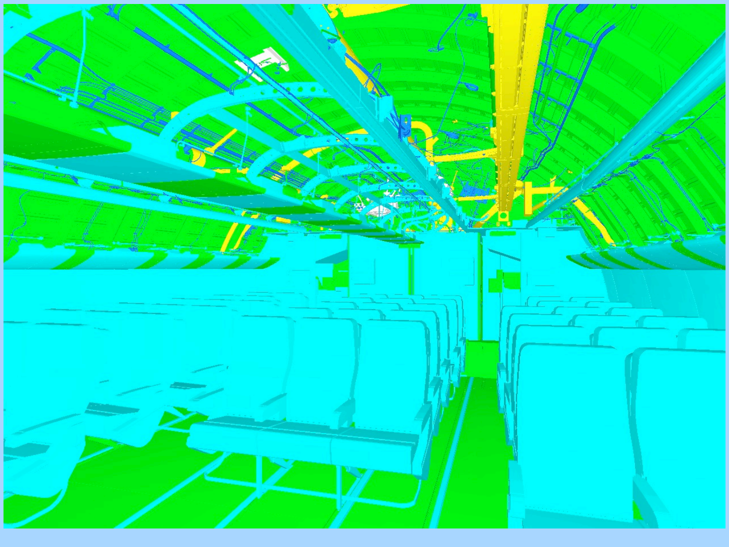
- Isosurface of turbulent boundary between two mixing fluids
- 473 million triangles
- 10GB of data



Boeing 777

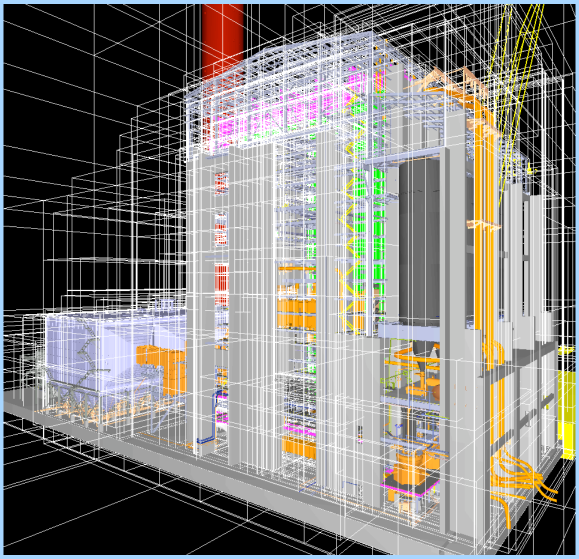
- CAD model
- 13,525 parts
- 352 million triangles
- 5GB of data





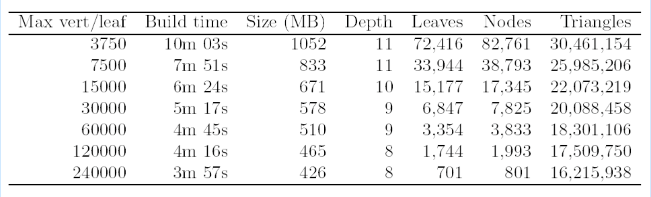
Test Machine

- 2.4 GHz Pentium IV
- 512 MB RAM
- 250 GB IDE disk
- NVIDIA GeForce Quadro FX 500 graphics
- Red Hat Linux 8.0
- Cost: about $1,000



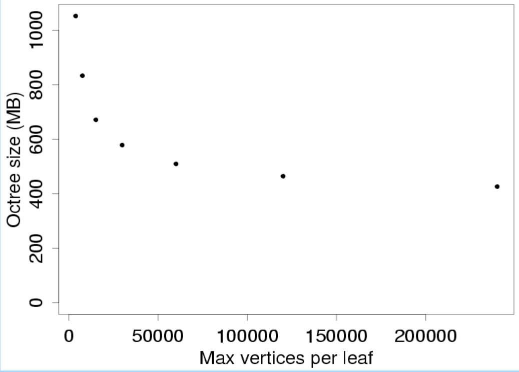
Power Plant Results

- Effect of spatialization granularity



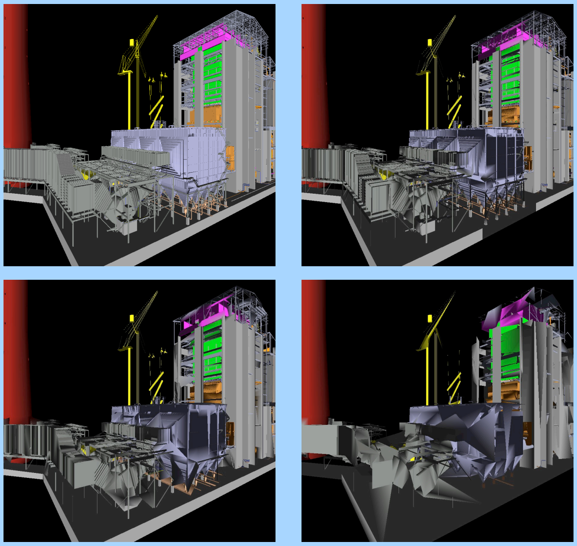
Max vert/leaf	Build time	Size (MB)	Depth	Leaves	Nodes	Triangles
3750	10m 03s	1052	11	72,416	82,761	30,461,154
7500	7m 51s	833	11	33,944	38,793	25,985,206
15000	6m 24s	671	10	15,177	17,345	22,073,219
30000	5m 17s	578	9	6,847	7,825	20,088,458
60000	4m 45s	510	9	3,354	3,833	18,301,106
120000	4m 16s	465	8	1,744	1,993	17,509,750
240000	3m 57s	426	8	701	801	16,215,938

Power Plant Results



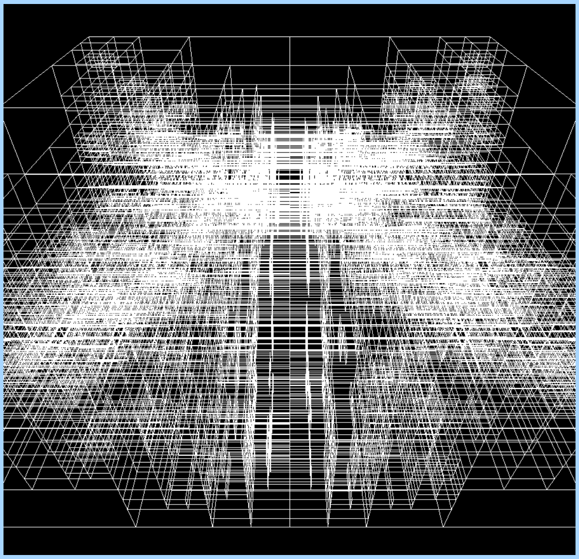
Power Plant Results

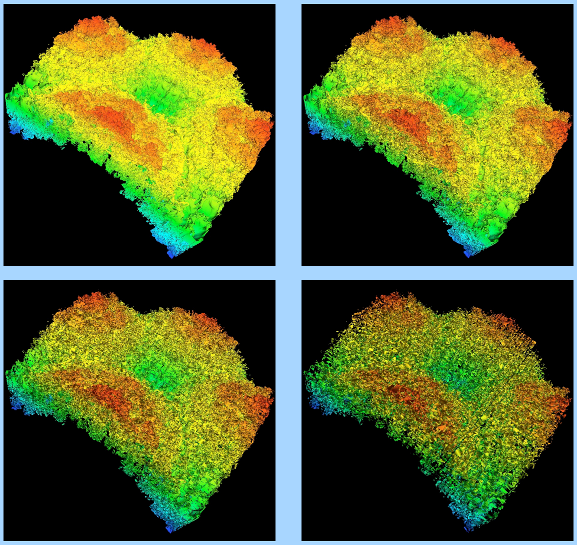
- Octree (15,000 triangles per leaf)
  - 6m 24s, 15,177 leaves
  - 3.4 MB for structure, 671 MB for data
- Visibility coefficients (20 dirs, 64x64 window)
  - 2m 36s, 711KB
- Levels of detail (up to 5 levels, 1/4 each time)
  - 8m 5s, 268 MB
- Total: about 17m and 1GB of data



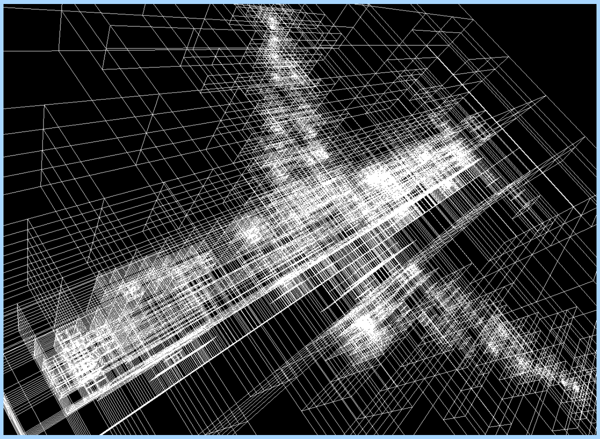
LLNL Isosurface Results

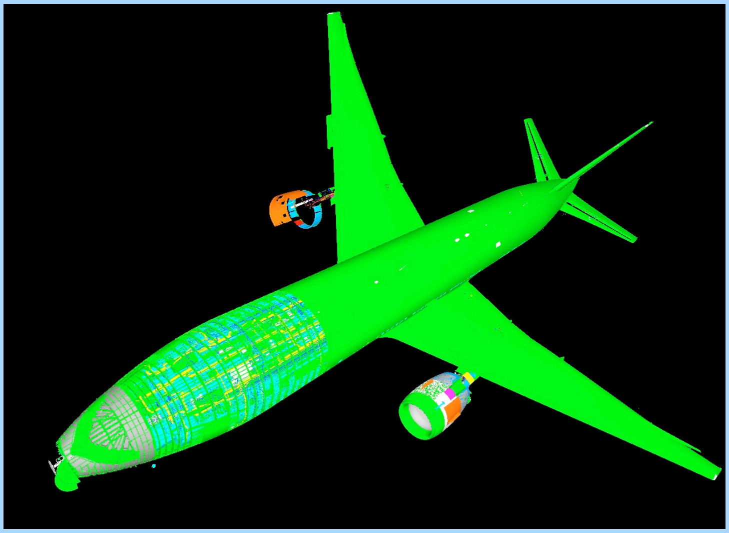
- Octree (480,000 triangles per leaf)
  - 1h 24m, 6,469 leaves
  - 1.3 MB for structure, 10 GB for data
- Visibility coefficients (20 dirs, 64x64 window)
  - 26m, 303 KB
- Levels of detail (up to 5 levels, 1/4 each time)
  - 1h 16m, 2.3 GB
- Total: about 3h and 12 GB





Boeing 777 Results





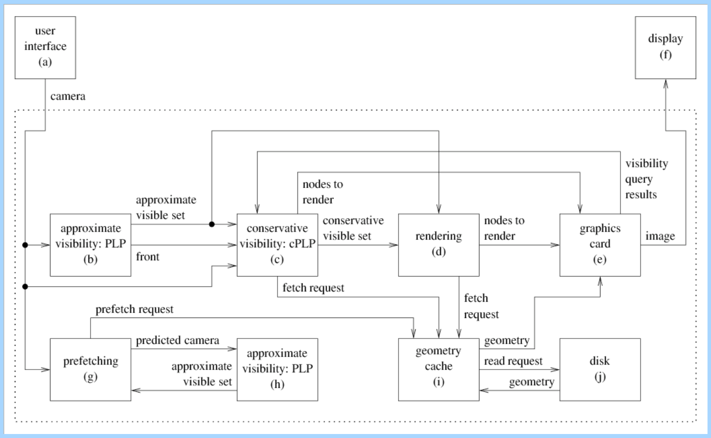
Summary of Preprocessing Results

- Spatialization
  - 5X faster than best similar approach (Wald01)
- Visibility precomputation
  - negligible time and storage requirements
- Simplification
  - fast, good enough, low storage requirements

Out-Of-Core Rendering

- Load the visible nodes on demand
- Multiple threads (as opposed to processes)
  - visibility computation
  - cache management
  - prefetching
  - rasterization

The iWalk System



user interface (a)

display (f)

camera

approximate visible set

nodes to render

visibility query results

approximate visibility: PLP (b)

front

conservative visibility: cPLP (c)

conservative visible set

rendering (d)

nodes to render

graphics card (e)

image

fetch request

fetch request

prefetch request

predicted camera

prefetching (g)

approximate visibility: PLP (h)

approximate visible set

geometry cache (i)

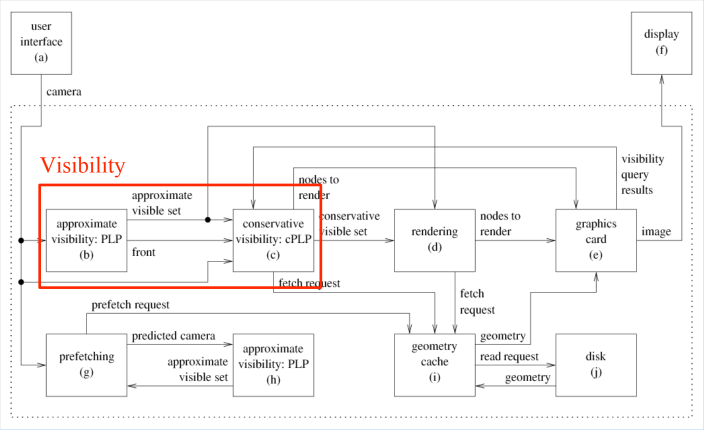
geometry

read request

geometry

disk (j)

The iWalk System



Visibility

user interface (a)

display (f)

camera

approximate visible set

nodes to render

visibility query results

approximate visibility: PLP (b)

front

conservative visibility: cPLP (c)

conservative visible set

rendering (d)

nodes to render

graphics card (e)

image

fetch request

fetch request

prefetch request

predicted camera

prefetching (g)

approximate visibility: PLP (h)

approximate visible set

geometry cache (i)

geometry

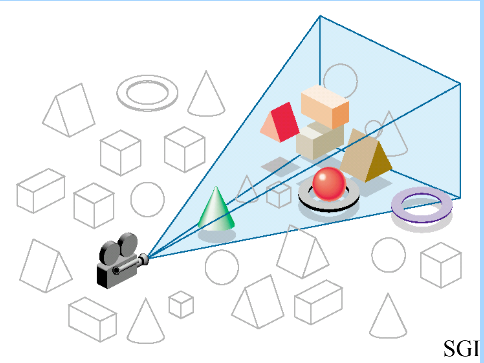
read request

geometry

disk (j)

Occlusion Culling

- Teller91, Greene93, Zhang97, Durand99, Klosowski99, Wonka99, Cohen-or02 Hall-Holt03



SGI

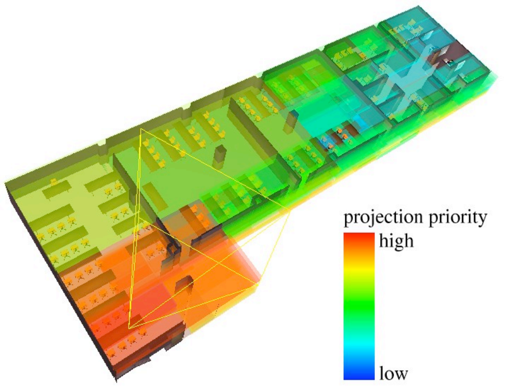
Occlusion Culling

- Classification criteria for occlusion culling algorithms
  - from-point vs. from-region
  - precomputed vs. online
  - object space vs. image space
  - conservative vs. approximate

The PLP Algorithm

- Approximate volumetric visibility
- Keeps the octree nodes in a priority queue called the *front*
- First visits nodes most likely to be visible
- Stops when a budget is reached
- **Doesn't need to read the geometry**
  - estimates the visible set from the hierarchy structure (HS) file

The PLP Algorithm

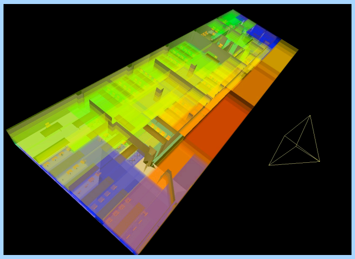


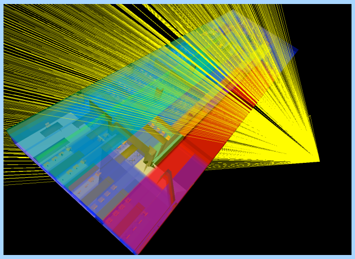
The cPLP Algorithm

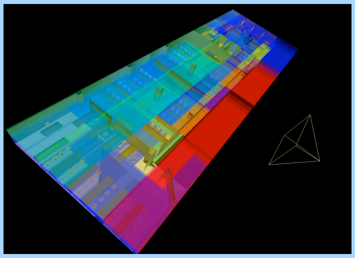
- Conservative extension of PLP
- Uses PLP to compute initial guess
- Adds nodes to guarantee correct images
- Unlike PLP, needs to read geometry
  - can't determine visible set from HS file only
- Three implementations
  - item buffer, HP test, NV occlusion query

Improving the Accuracy of PLP

- Use precomputed visibility coefficients to estimate node's opacity for current view
- Shoot rays from user's viewpoint to estimate projection priority of octree nodes
- Ray contribution is initialized to 1
- Attenuate contribution based on opacity of nodes hit along ray path







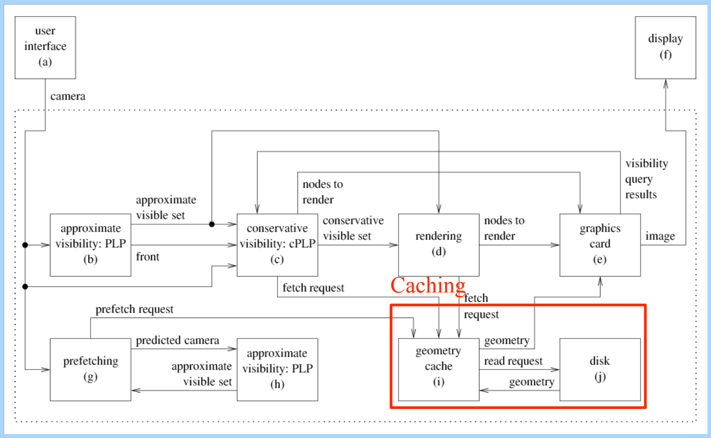
Advantages of Improved Heuristic

- Better images in approximate mode
- Better frame rates in conservative mode
  - less work for cPLP
- Better prefetching
  - less cache pollution
  - fewer cache misses
- Better visibility-based LOD selection

Improving the Running Time of cPLP

- Item buffer
  - slow, multiple tests at a time, int result
- HP occlusion test
  - fast, one test at a time, boolean result
- NV occlusion query
  - fast, many tests at a time, int result

The iWalk System



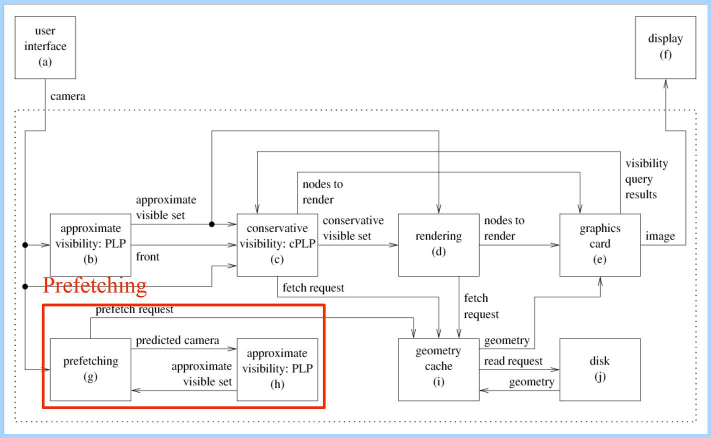
Geometry Caching

- Keep bulk of data on disk
- Bring data into memory on demand
- Keep in memory the least recently used data

The Geometry Cache

- User-defined maximum size
- Blocks of variable size
- Global lock
- Busy flag per block
- Work queue of fetch requests
- Work queue of prefetch requests
- LRU replacement policy

The iWalk System



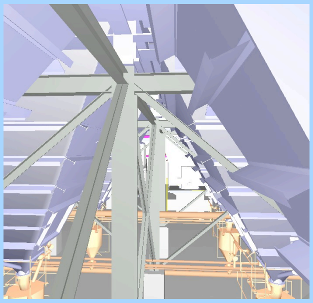
Geometry Prefetching

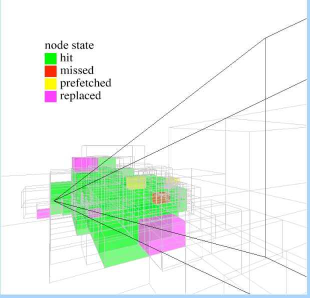
- Guess what data will be needed next
- Read data ahead of time
- Hides I/O latency

From-Point Prefetching

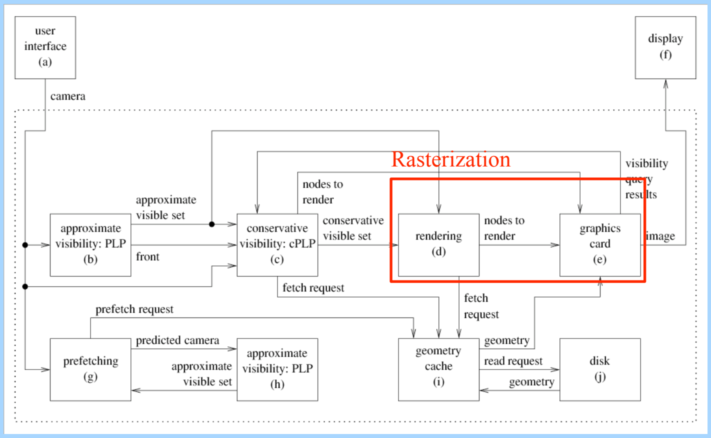
- Improves frame rate by hiding I/O latency
- Uses PLP (approximate visibility algorithm)
  - fine, because prefetching is speculative
- Doesn't need geometry (good for out-of-core)
- Doesn't need graphics pipe (good for PCs)
- Needs less preprocessing than from-region
- Tighter estimate than from-region (less I/O)

The Geometry Cache





The iWalk System



user interface (a)

display (f)

camera

Rasterization

approximate visible set

nodes to render

visibility query results

approximate visibility: PLP (b)

front

conservative visibility: cPLP (c)

conservative visible set

rendering (d)

nodes to render

graphics card (e)

image

fetch request

prefetch request

fetch request

predicted camera

geometry

prefetching (g)

approximate visible set

approximate visibility: PLP (h)

geometry cache (i)

read request

geometry

disk (j)

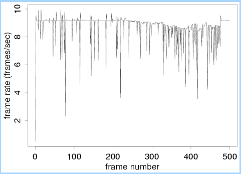
Rasterization

- Pass geometry to the graphics card
  - OpenGL rendering
  - Gouraud shading
- Vertex array per octree node
  - more memory efficient than display lists

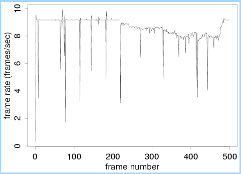
Rendering Results

- Measure frame rates
- Assess image quality
- Evaluate effect of multi-threading and prefetching
- Study the importance of frame-to-frame coherence
- Assess how much better the improved visibility heuristic is

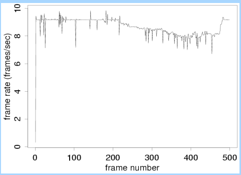
Multi-threading Improves Frame Rates



sequential fetching and rendering

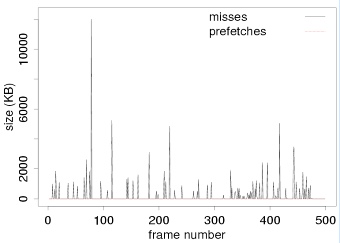


concurrent fetching and rendering

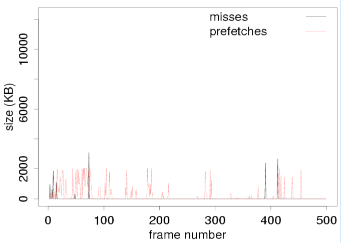


concurrent fetching, rendering, and prefetching

Prefetching Amortizes the Cost of I/O Operations

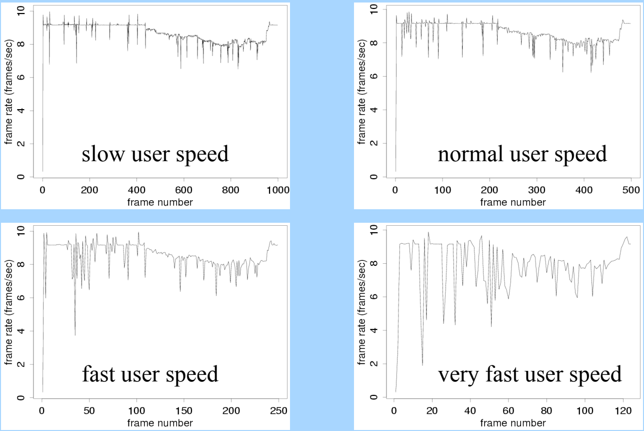


without prefetching



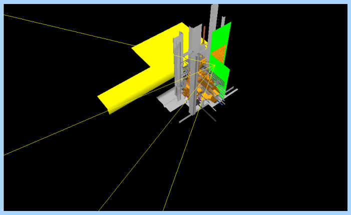
with prefetching

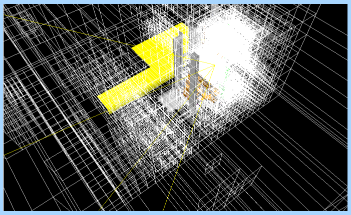
Importance of Frame Coherence

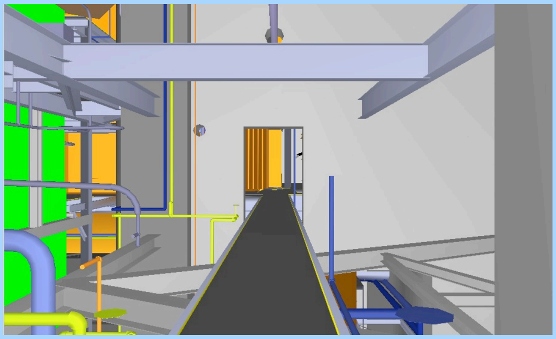


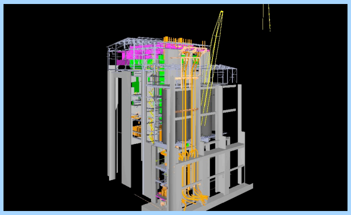
How Much Better is the Improved Visibility Heuristic

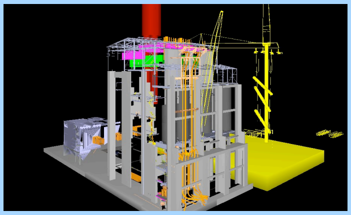
- For interior views
  - not much
- For exterior views
  - quite a bit

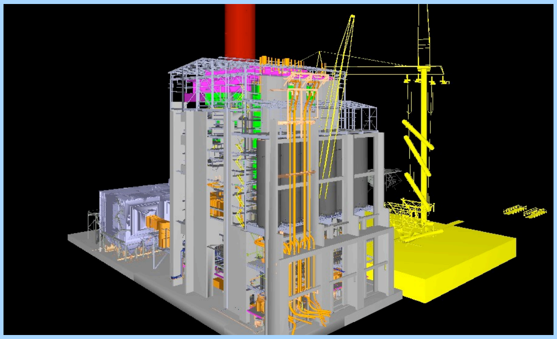




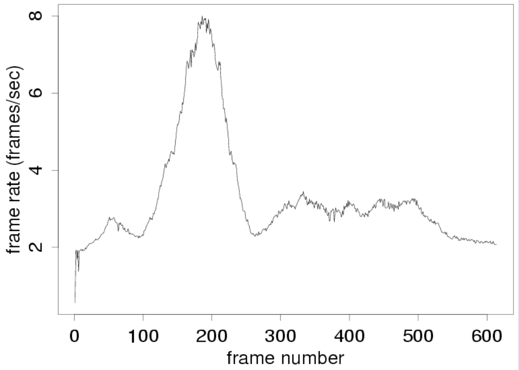








LLNL Isosurface Rendering Results



Summary of Rendering Results

- We can render a model 20 times larger than main memory at interactive frame rates and acceptable quality on a cheap PC
- Performance is heavily dependent on frame-to-frame-coherence
- Sparse ray tracing helps visibility estimation significantly without much overhead

Out-Of-Core Parallel Rendering

- So far
  - single PC
  - low resolution images (1024x768)
  - interactive frame rates
- Now
  - display wall driven by a cluster of PCs
  - high resolution images (4096x3072)
  - same or faster frame rates

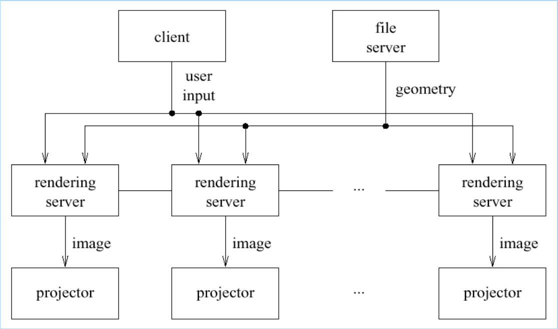
Parallel Rendering

- Sort-first
  - distribute object-space primitives
  - each processor is assigned a screen tile
- Sort-middle
  - distribute image-space primitives
  - geometry processors and rasterizers
- Sort-last
  - distribute pixels
  - rendering and compositing processors

Choosing the Parallelization Strategy

- Why sort-first?
  - each processor runs entire pipeline for a tile
  - exploits frame-to-frame coherence well
- Why *not* sort-middle?
  - needs tight integration between geometry processing and rasterization
- Why *not* sort-last?
  - needs high pixel bandwidth
  - prevents us from using image occlusion queries

The Out-Of-Core Sort-First Parallel Architecture



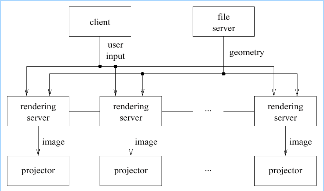
The Out-Of-Core Sort-First Parallel Architecture

- Separate rendering server for each tile
- Client does almost no work, and can be as lightweight as a hand-held computer
- MPI to start and synchronize the servers
- Options: distributed vs. centralized data

UNC Power Plant Tests

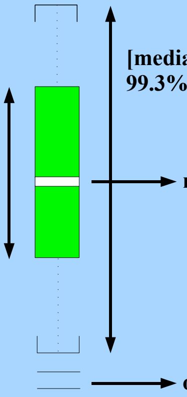
- Pre-recorded 500-frame camera path
- Cluster sizes
  - 1, 2, 4, 8, and 16
- Disk type
  - local and network

Old Cluster



- Rendering servers
  - 900 MHz Athlon, 512 MB of RAM
  - GeForce2, IDE disk
- Client: 700 MHz Pentium III
- File server: 400 GB SCSI disk array
- Network: gigabit Ethernet
- Software: Red Hat Linux 7.2, MPI/Pro 1.6.3

Box Plots



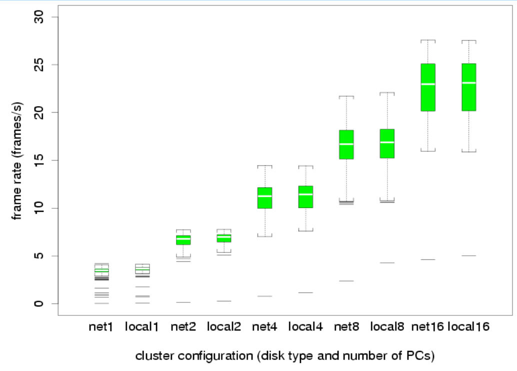
[median – 1.5 IQD, median + 1.5 IQD]: 99.3% of the data (if Gaussian)

interquartile distance (IQD): spread

median: center

outliers

Results for Approximate Visibility

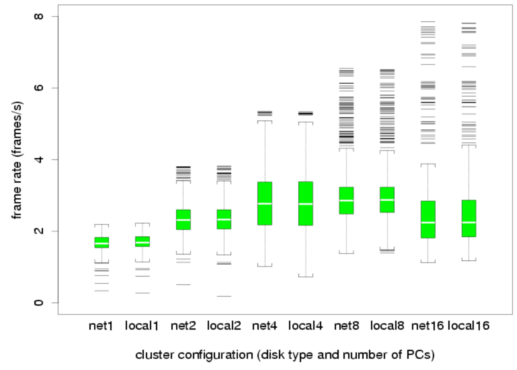


- Median frame rates improve with cluster size
- Disk type makes no difference

Obstacles for Perfect Scalability

- Duplication of effort
  - primitives may overlap multiple tiles
- Communication overhead
  - barrier at the end of each frame
- Load imbalance
  - primitives may cluster into regions

Results for Conservative Visibility Without LODs



- Median frame rates remain almost constant
- Disk type makes no difference
- Additional obstacle: visible geometry increases with resolution

Summary of Power Plant Parallel Rendering Results

- 1 PC (1024x768 images)
  - median frame rate: 9.1 frames per second
- 16 PCs (4096x3072 images)
  - median frame rate: 10.8 frames per second
  - cap on frame rate
    - gives prefetching better chance to run
    - reduces frame rate variance

New Cluster

- 8 rendering servers:
  - 2.8 GHz Pentium IV, 512 MB RAM
  - 35 GB SCSI disk
  - NVIDIA Quadro 980 XGL graphics card
- File server
  - same plus 200 GB SCSI disk
- Gigabit Ethernet
- Red Hat Linux 8.0, MPICH 1.2.5

LLNL Isosurface Parallel Rendering Results

- Conservative visibility and LOD
- 8 x 1280 x 1024 (10 megapixels)
- For outside views
  - 3-5 frames per second
- For inside views
  - 8-10 frames per second
- Frame rates using shared disk almost the same as frame rates using local disks

Summary of Parallel Rendering Results

- We can scale the resolution of an application without any loss in performance
- Caching and prefetch exploit coherence well: even with centralized file server, usually limited by rendering

Comparison to Other Parallel Rendering Systems

- Better frame rates than Humphreys02, but we do need to change the source code
- Faster frame rates and higher resolution than Wald01, but lower image quality
- Similar frame rates to Moreland01, plus image occlusion queries

Conclusions

- iWalk system is practical and scalable
- Out-of-core techniques are fast and effective
- PCs are an attractive, cost-effective alternative to high-end machines
- The system can help to bring visualization of large datasets to a broader audience

Research Contributions

- Efficient out-of-core algorithm to build octree
- Extensions of the PLP visibility algorithm
  - ray-tracing based approximate heuristic
  - hardware-assisted conservative extension
- Out-of-core, from-point prefetching algorithm
- Out-of-core sort-first architecture

Future Work

- Support for different types of scenes
  - textures, volumes (working prototype), dynamics
- Efficiency
  - add geometry and appearance quantization
  - eliminate geometry replication
- Analysis
  - develop analytic model for system parameters
  - optimize system parameters automatically

Acknowledgements

- Financial support
  - CNPq (Brazilian research funding agency)
  - Princeton University
  - AT&T Research
  - Oregon Graduate Institute
  - IBM Research
- Datasets
  - UNC Chapel Hill, UC Berkeley, 3rdTech, LLNL, Boeing

Visibility-Based Prefetching for Interactive Out-Of-Core Rendering

Wagner T. Corrêa[*] Princeton University
James T. Klosowski[†] IBM Research
Cláudio T. Silva[‡] University of Utah

Abstract

We present a new visibility-based prefetching algorithm for interactive out-of-core rendering of large models on an inexpensive PC. Using an approximate visibility technique, we can very accurately and efficiently determine which geometry will be visible in the near future and prefetch that geometry from disk before it must be rendered. Our prefetching algorithm is a key part of a visualization system capable of rendering a 13-million triangle model with 99% accuracy at interactive frame rates. Our prefetching algorithm is the first of its kind to be based on a from-point visibility technique, and enables interactive rendering on a commodity PC, as opposed to expensive high-end graphics workstations or parallel machines.

CR Categories: I.3.2 [Graphics Systems]: Computer Graphics—Computing Methodologies; I.3.3 [Picture/Image Generation]: Computer Graphics—Computing Methodologies

Keywords: out-of-core rendering, interactive rendering, commodity PCs, occlusion culling, prefetching, walkthrough

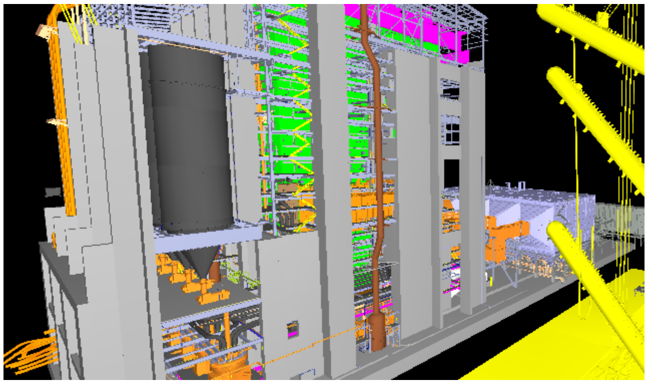


Figure 1: Our out-of-core rendering system can preprocess a 13-million-triangle model in 3 minutes, and then render it with 99% accuracy at 10 frames per second on an inexpensive PC.

1 Introduction

In this paper, we present a new visibility-based prefetching algorithm for retrieving out-of-core 3D models and rendering them at interactive rates on an inexpensive PC. Interactive rendering of large models has applications in many areas, including computer-aided design, engineering, entertainment, and training. Traditionally, interactive rendering of large models has required triangle throughput only available on high-end graphics workstations or parallel machines that cost hundreds of thousands of dollars. Recently, with the explosive growth in performance of PC graphics cards that cost a few hundred dollars, inexpensive PCs are becoming an attractive alternative to high-end machines. Although inexpensive PCs can match the triangle throughput of high-end machines, inexpensive PCs have much less main memory than high-end machines. Today a typical high-end machine has 16 GB of main memory, while a typical inexpensive PC has 512 MB (32 times less). Thus, a challenge in exploiting the performance of PC graphics cards is designing rendering systems that work under tight memory constraints.

Our rendering system, iWalk, overcomes the memory constraints of an inexpensive PC by using an out-of-core preprocessing algorithm and a new multi-threaded out-of-core rendering approach to overlap rendering, visibility computation, and disk operations. The preprocessing algorithm breaks the geometry of a model into manageable pieces, and creates on disk a spatial subdivision of the geometry. At runtime, the system maintains in memory a cache of the most recently used geometry.

The system can run in two visibility modes: approximate or conservative. In approximate mode, the system tries to maximize the quality of the rendered images, given a user-defined budget of geometry per frame. This budget is based on the hardware capabilities (such as the graphics card triangle throughput and the disk bandwidth) and the target frame rate. In conservative mode, for those instances that cannot tolerate any errors in the rendered images, the system determines and renders all the visible geometry, potentially at lower frame rates. In either visibility mode, as the viewpoint changes, the visible geometry changes, and the geometry cache has to load from disk the visible geometry that is not in the cache. Even small changes in the viewpoint can cause large changes in visibility. This problem manifests itself as abrupt drops in frame rates because of bursts of disk operations.

The prefetching algorithm we present in this paper addresses this problem. The goal of prefetching is to have the geometry already in memory by the time it is needed. The prefetching algorithm runs as a separate thread, and is orthogonal to the visibility mode. The prefetching thread uses a from-point visibility algorithm to find the geometry the user is likely to see in the near future, and sends prefetch requests to the geometry cache. If the geometry cache is busy loading geometry needed for the current frame, it ignores prefetch requests; otherwise, it loads the requested geometry from disk. By amortizing the cost of bursts of disk operations over frames with few disk operations, prefetching improves the performance of the system in either visibility mode.

The main contribution of this paper is a multi-threaded out-of-core rendering approach which to our knowledge is the first to combine speculative prefetching with a from-point visibility algorithm.

[*] Department of Computer Science, Princeton University, 35 Olden St., Princeton, NJ 08540; wtcorrea@cs.princeton.edu.

[†] Visual Technologies, IBM T. J. Watson Research Center, PO Box 704, Yorktown Heights, NY 10598; jklosow@us.ibm.com.

[‡] Work performed while at AT&T. Currently at Scientific Computing and Imaging Institute, School of Computing, University of Utah, 50 S. Central Campus Dr., Salt Lake City, UT 84112; csilva@cs.utah.edu.

Our prefetching algorithm plays a critical role in our iWalk system, which is capable of rendering a model with tens of millions of polygons at interactive frame rates on an inexpensive PC (Figure 1).

2 Related Work

Researchers have studied the problem of rendering complex models at interactive frame rates for many years. Clark [1976] proposed many of the techniques for rendering complex models used today, including the use of hierarchical spatial data structures, level-of-detail (LOD) management, hierarchical view-frustum and occlusion culling, and working-set management (geometry caching). Garlick et al. [1990] presented the idea of exploiting multiprocessor graphics workstations to overlap visibility computations with rendering. Airey et al. [1990] described a system that combined LOD management with the idea of precomputing visibility information for models made of axis-aligned polygons.

Funkhouser et al. [1992] described the first published system that supported models larger than main memory, and performed speculative prefetching. Their system was based on the from-region visibility algorithm of Teller and Séquin [1991], which required long preprocessing times, and was limited to models made of axis-aligned cells. Our system is based on the from-point visibility algorithm of Klosowski and Silva [2000; 2001], which requires very little preprocessing, and can handle any 3D polygonal model.

Aliaga et al. [1999] presented the Massive Model Rendering (MMR) system, which employed many acceleration techniques, including replacing geometry far from the user's point of view with imagery, occlusion culling, LOD management, and from-region prefetching. MMR was the first published system to handle models with tens of millions of polygons at interactive frame rates, although it did require an expensive high-end multi-processor graphics workstation.

Wald et al. [2001] developed a ray tracing system that used a cluster of 7 dual-processor PCs to render low-resolution images of models with tens of millions of polygons at interactive frame rates. Avila and Schroeder [1997] and El-Sana and Chiang [2000] developed systems for interactive out-of-core rendering based on LOD management, but these systems did not perform occlusion culling. Varadhan and Manocha [2002] describe a system for out-of-core rendering that uses hierarchical LODs [Erikson et al. 2001] and prefetching, but their system does not perform occlusion culling, and their preprocessing step is in-core.

Wonka et al. [2001] employed a from-point visibility algorithm and used two processors to overlap visibility computation and rendering at runtime (similarly to the idea introduced by Garlick et al. [1990]), but they only reported results for 2.5D environments that were smaller than main memory. Many other researchers have also developed systems for out-of-core rendering, but without focusing on achieving interactive frame rates [Chiang and Silva 1997; Chiang et al. 1998; Cox and Ellsworth 1997; Pharr et al. 1997; Shen et al. 1999; Sutton and Hansen 2000].

3 Out-Of-Core Preprocessing

Recall that our main goal is to render a large model using an inexpensive PC with small memory. To accomplish this, we must first construct an out-of-core hierarchical representation for the model during a preprocessing step, and then at runtime load on demand the hierarchy nodes that the user sees. Our current algorithm builds an out-of-core octree [Samet 1990] whose leaves contain the geometry of the model. To store the octree on disk, we save the geometric contents of each octree node in a separate file, and create a *hierarchy structure* (HS) file, which stores information about the spatial relationship of the nodes in the hierarchy, and for each node it contains the node's bounding box and auxiliary data needed for visibility culling. The HS file is the main data structure that our system uses to control the flow of data and is assumed to fit in memory. For the 13-million triangle model used throughout this paper, the HS file was only 3 MB.

An in-core approach to build an octree for a model would process the entire model in one pass, using a machine with large enough memory to hold both the model and the resulting octree. We avoid this brute-force approach because we do not want to use a separate expensive machine with large memory just to build the octree. Our out-of-core algorithm builds an octree for a model directly on machines with small memory. The algorithm first breaks the model in sections that fit in main memory, and then incrementally builds the octree on disk, one pass for each section, keeping in memory only the section being processed. Our preprocessing algorithm requires no user intervention and is very fast, often orders of magnitude faster than previous approaches. It constructs the octree for the UNC power plant model [UNC 1999] in just 3 minutes, while the MMR system [Aliaga et al. 1999] required over two weeks, and the system by Wald et al. [2001] spent 2.5 hours preprocessing the same model.

Our preprocessing is similar in nature to several other construction algorithms [Cignoni et al. 2002; Ueng et al. 1997; Wald et al. 2001]. It is most akin to the algorithm of Cignoni et al. [2002], and therefore has comparable preprocessing times. Other recent construction algorithms are presented in [Durand et al. 2000; Schaufler et al. 2000; Wonka et al. 2000; Wonka et al. 2001]. As our construction algorithm is not the main focus of this paper, we refer the reader to [Corrêa et al. 2002] for a thorough examination of the differences between all of these algorithms.

4 Out-of-Core Rendering

Figure 2 shows a diagram of iWalk's rendering approach. The user interface (a) keeps track of the position, orientation, and field-of-view of the user's camera. For each new set of camera parameters, the system computes the visible set — the set of octree nodes that the user sees. According to the user's choice, the system can compute an approximate visible set (b), or a conservative visible set (c). To compute an approximate visible set, iWalk uses the prioritized-layered projection (PLP) algorithm [Klosowski and Silva 2000]. To compute a conservative visible set, iWalk uses cPLP [Klosowski and Silva 2001], a conservative extension of PLP. For each node in the visible set, the rendering thread (d) sends a fetch request to the geometry cache (i), which will read the node from disk (j) into memory. The rendering thread then sends the node to the graphics card (e) for display (f). To avoid bursts of disk operations, the look-ahead thread (g) predicts where the user's camera is likely to be in the next frame. For each predicted camera, the look-ahead thread uses PLP (h) to estimate the visible set, and then sends prefetch requests to the geometry cache (i).

To better understand our rendering approach, we need to briefly review the visibility algorithms that iWalk uses. PLP is an approximate, from-point visibility algorithm that may be thought of as a set of modifications to the traditional hierarchical view frustum culling algorithm [Clark 1976]. First, instead of traversing the model hierarchy in a predefined order, PLP keeps the hierarchy leaf nodes in a priority queue called the *front*, and traverses the nodes from highest to lowest priority. When PLP visits a node, it adds it to the *visible set*, removes it from the front, and adds the unvisited neighbors of the node to the front. Second, instead of traversing the entire hierarchy, PLP works on a budget, stopping the traversal after a certain number of primitives have been added to the visible set. Finally, PLP requires each node to know not only its children, but also all of its neighbors.

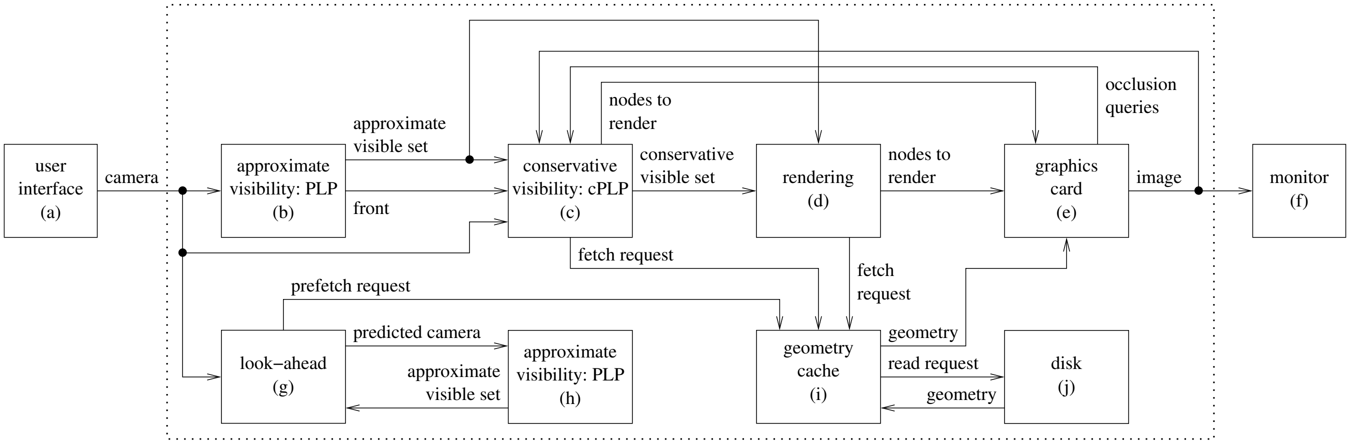


Figure 2: The multi-threaded out-of-core rendering approach of the iWalk system. For each new camera (a), the system finds the set of visible nodes using either approximate visibility (b), or conservative visibility (c). For each visible node, the rendering thread (d) sends a fetch request to the geometry cache (i), and then sends the node to the graphics card (e). The look-ahead thread (g) predicts future cameras, estimates the nodes that the user would see then (h), and sends prefetch requests to the geometry cache (i).

In addition to being time-critical, another key feature of PLP that iWalk exploits is that PLP can generate an approximate visible set based on just the information stored in the hierarchy structure file created at preprocessing time. In other words, PLP can estimate the visible set *without* access to the actual scene geometry.

An implementation of PLP may be simple or sophisticated, depending on the heuristic to assign priorities to each node. Several heuristics precompute for each node a value between 0.0 and 1.0 called *solidity*, which estimates how likely it is for the node to occlude an object behind it. At run time, the priority of a node is found by initializing it to 1.0, and attenuating it based on the solidity of the nodes found along the traversal path to the node (Figure 3).

Although PLP is in practice quite accurate for most frames, it does not guarantee image quality, and some frames may show objectionable artifacts. To avoid this potential problem, the system may use cPLP [Klosowski and Silva 2001], a conservative extension of PLP that guarantees 100% accurate images. However, cPLP cannot find the visible set from the HS file only, and needs to read the geometry of all potentially visible nodes. These additional disk operations may make cPLP much slower than PLP. Our implementation of cPLP can use either an item-buffer technique that is portable to any platform that supports OpenGL, or occlusion query extensions (such as the HP test [Severson 1999] and the nVidia occlusion query [Rege 2002]) when they are available. Thus, cPLP needs to fetch geometry from the geometry cache, and read pixels or occlusion queries from the graphics card.

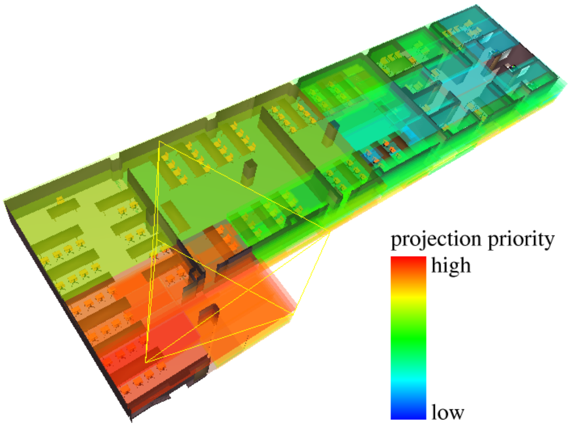


Figure 3: A section of the UC Berkeley Soda Hall model. At runtime, the iWalk system uses the prioritized-layered projection (PLP) algorithm to estimate the nodes potentially visible from the current view frustum (outlined in yellow). The transparent color of each node indicates the projection priority of the node.

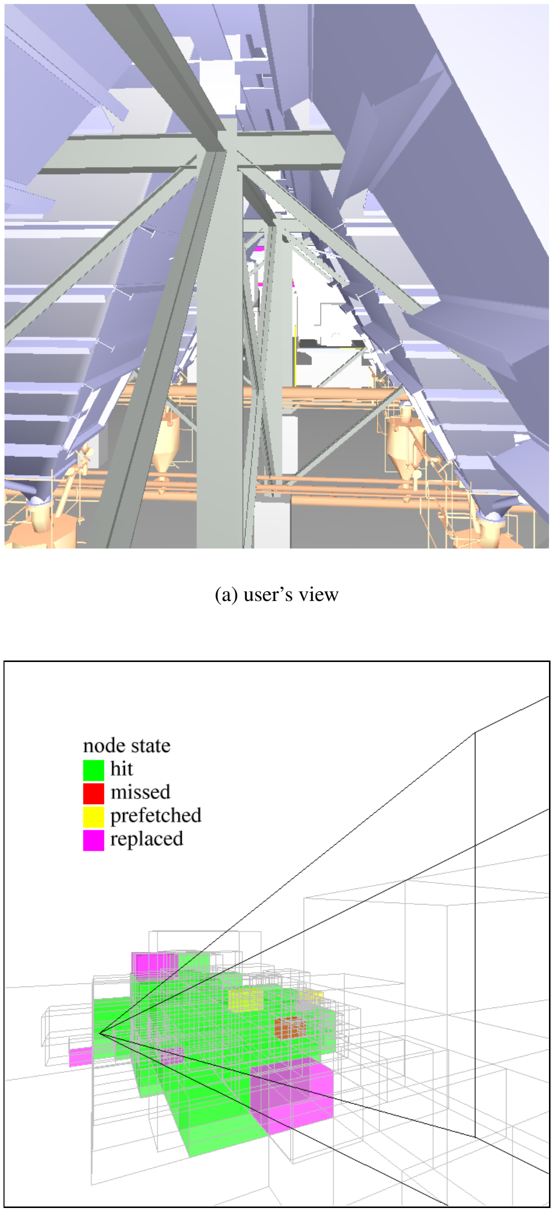
5 From-Point Visibility-Based Prefetching

The idea behind prefetching is to predict a set of nodes that the user is likely to see next, and bring them to memory ahead of time. Ideally, by the time the user sees those nodes, they will be already in the geometry cache, and the frame rates will not be affected by the disk latency. Systems researchers have studied prefetching strategies for decades [Gindele 1977; Przybylski 1990], and many previous rendering systems [Aliaga et al. 1999; Funkhouser 1996; Funkhouser et al. 1992; Varadhan and Manocha 2002] have used prefetching successfully. To our knowledge, all previous prefetching methods that employ occlusion culling have been based on from-region visibility algorithms, and were designed to run on multiprocessor machines. Our prefetching method works with from-point visibility algorithms, and runs as a separate thread in a uniprocessor machine.

Our prefetching method exploits the fact that PLP can very quickly compute an approximate visible set. Given the current camera (Figure 2a), the look-ahead thread (Figure 2g) predicts the next camera position by simply extrapolating the current position and the camera's linear and angular speeds. More sophisticated prediction schemes could consider accelerations and several prior camera locations. For each predicted camera, the look-ahead thread uses PLP (Figure 2h) to determine which nodes the predicted camera is likely to see. For each node likely to be visible, the look-ahead thread sends a prefetch request to the geometry cache (Figure 2i). The geometry cache puts the prefetch requests in a queue and a set of prefetch threads process the requests. If there are no fetch requests pending, and if the maximum amount of geometry that can be prefetched per frame has not been reached, a prefetch thread will pop a request from the prefetch queue, and read the requested node from disk (if necessary) (Figure 2j). If the cache is full, the least recently used nodes are evicted from memory. The requested nodes are then placed in a queue of nodes that are ready to be rendered.

```
fetch(node, ready_queue)
{
    lock cache;
    while (node is busy)
        wait until node is free;
    mark node as busy;
    if (node is valid) {
        miss = false;
        update node position;
    } else {
        miss = true;
        allocate memory;
    }
    unlock cache;

    if (miss)
        read node;

    lock cache;
    if (miss)
        add node to cache;
    if (no fetches pending)
        broadcast no fetches pending;
    unlock cache;
    add node to ready_queue;
}

prefetch(node, ready_queue)
{
    lock cache;
    while (there are fetch requests pending)
        wait until no fetch requests pending;
    while (node is busy)
        wait until node is free;
    mark node as busy;
    if ((node is valid)
        || (reached max prefetch amount per frame)
        || (reached max prefetch request age))
        can_read = false;
    else {
        can_read = true;
        allocate memory;
    }
    unlock cache;

    if (can_read) {
        read node;
        lock cache;
        add node to cache;
        unlock cache;
    }
    add node to ready_queue;
}

release(node)
{
    lock cache;
    mark node as free;
    if (node is valid)
        broadcast memory available;
    broadcast node is free;
    unlock cache;
}
```

Figure 4: Pseudo-code for the main cache routines.



(a) user's view

(b) cache view

Figure 5: A sample frame inside the power plant model. (a) The image that the user sees. (b) The state of the nodes in the cache.

Figure 4 shows the pseudo-code for the main routines run by the threads in the cache. When a client makes a fetch request, a thread executes the fetch routine (and similarly for a prefetch request). When the client is done using that node, it must call the release routine. These routines have to be very careful about sharing the cache data structures. To guarantee mutual exclusion, there is a lock to access the cache, and each node has a flag indicating whether it is free or busy. This scheme is similar to the one used in the UNIX buffer cache [Bach 1986]. Figure 5a shows the user's view of the UNC power plant model [UNC 1999] during a walkthrough session, and Figure 5b shows the state of the octree nodes in the cache.

Unlike our from-point prefetching method, from-region prefetching methods decompose the model into cells, and precompute for each cell the geometry that the user would see from any point in the cell. At runtime, from-region methods guess in which cell the user will be next, and load the geometry visible from that cell ahead of time. Our from-point prefetching method has several advantages over from-region prefetching methods. First, from-region methods typically require long preprocessing times (tens of hours), while our from-point method requires little preprocessing (a few minutes). Second, the set of nodes visible from a single point is typically much smaller than the set of nodes visible from any point in a region. Thus, our from-point prefetching method avoids unnecessary disk operations, and has a better chance than a from-region method of prefetching nodes that actually will be visible soon. Third, some from-region methods require that cells coincide with axis-aligned polygons in the model. Our from-point method imposes no restriction on the model's geometry. Finally, the nodes visible from a cell may be very different from the nodes visible from a neighbor of that cell. Thus, a from-region method may cause bursts of disk activity when the user crosses cell boundaries, while a from-point method better exploits frame-to-frame coherence.

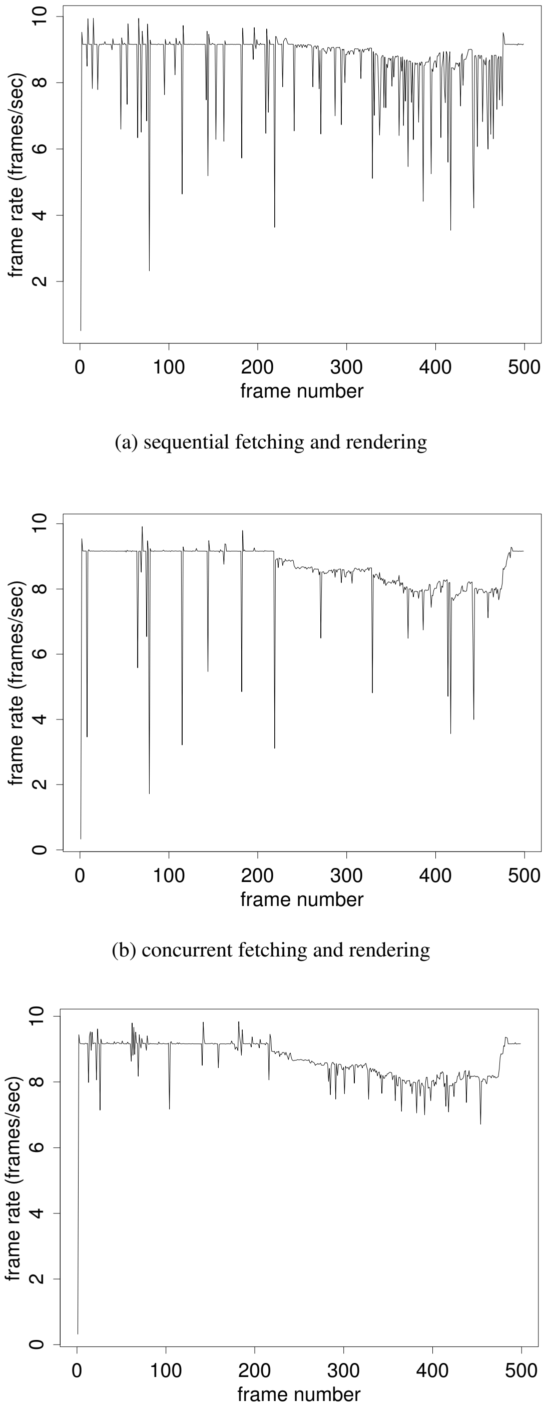
Since the cost of disk read operations is high, most systems try to overlap all of these operations with other computations by running several processes on a multiprocessor machine [Aliaga et al. 1999; Funkhouser 1996; Garlick et al. 1990], or on a network of machines [Wald et al. 2001; Wonka et al. 2001]. Along these same lines, our system uses multiple threads on a single processor machine to overlap disk operations with visibility computations and rendering.

6 Experimental Results

To evaluate our system, and in particular our prefetching algorithm, we experimented with the 13-million-triangle UNC power plant model [UNC 1999]. The raw model was roughly 600 MB in size; after our preprocessing step, the model size increased to 1 GB. To our knowledge, no other system has been able to render this model at interactive rates on a single PC. Our system ran Red Hat Linux 8.0, had a 2.8 GHz Pentium IV CPU, 512 MB of main memory, a 35 GB SCSI disk, and an nVidia Quadro 980 XGL card. Using `top`, we found that the operating system and related utilities used roughly 64 MB of main memory. For our test machine, we found that the following configuration worked well: 256 MB of geometry cache, 8 fetch threads, 1 prefetch thread, a maximum of 2 MB of prefetched geometry per frame, approximate visibility with a budget of 280,000 triangles per frame, a target frame rate of 10 fps, and image resolution of 1024×768.

To analyze the overall performance of our system, we measured the frame rates achieved when walking through the power plant model along several predefined paths (which enabled repeatable conditions for our experiments). Note that our algorithms made no assumptions on the paths being known beforehand; therefore, complete camera interactivity is always available to the user. The first path used has 36,432 viewpoints, visits almost every part of the model, and requires fetching a total of 900 MB of data from disk. Using the above configuration, our system rendered the frames along that path in 74 minutes. Only 95 frames (0.26%) caused the system to achieve less than 1 fps. The mean frame rate was 9.2 fps, and the median frame rate was 9.3 fps. To analyze the detailed performance of our system, it is easier to use shorter paths. For this purpose, we used a 500-frame path which required 210 MB of data to be read from disk. If fetched independently, the maximum amount of memory necessary to render any given frame in approximate mode would be 16 MB.

To study how multiple threads improve the frame rates, we ran tests using three different configurations. The first configuration



(a) sequential fetching and rendering

(b) concurrent fetching and rendering

(c) concurrent fetching, rendering, and prefetching

Figure 6: Using multiple threads to improve frame rates. We measured the frame rates during a 500-frame walkthrough of the power plant model under three configurations: (a) using one thread for fetching and rendering; (b) using multiple threads to overlap fetching and rendering; and (c) using multiple threads to overlap fetching, rendering, and prefetching. Concurrent fetching eliminates some downward spikes, and adding concurrent speculative prefetching eliminates almost all of the remaining spikes. The first spike happens because the cache is initially empty. The three configurations produce identical images.

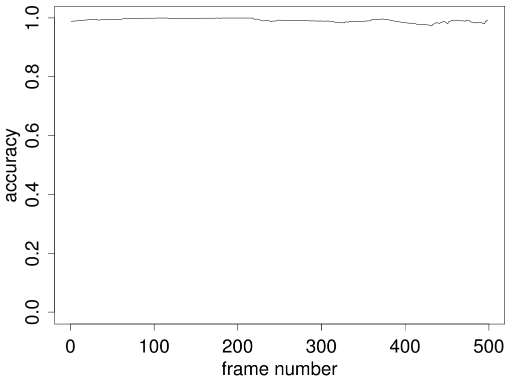
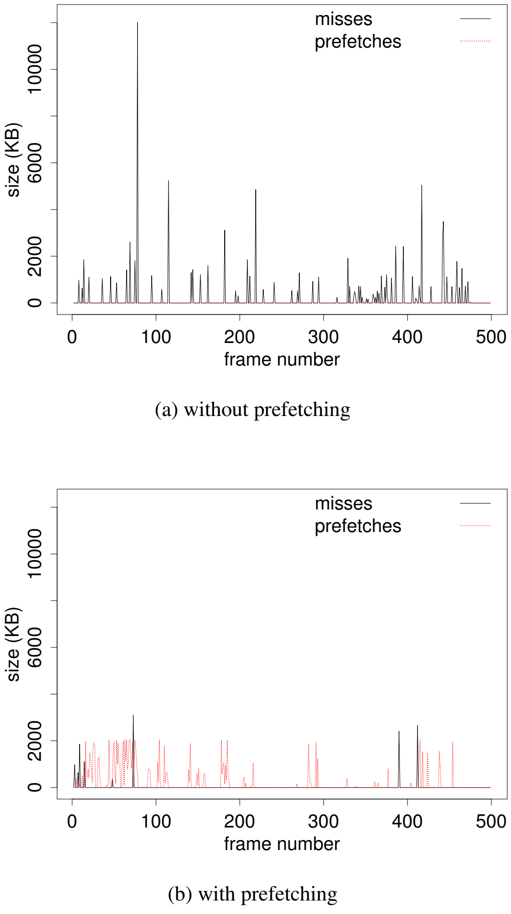


Figure 7: Image accuracy for a 500-frame walkthrough of the power plant model when using approximate visibility. The vertical axis represents the fraction of correct pixels in the approximate images in comparison to the conservative images. The minimum accuracy was 97.3%, and the median accuracy was 99.2%.



(a) without prefetching

(b) with prefetching

Figure 8: Using prefetching to amortize the cost of disk operations. We measured the amount of geometry fetched per frame without prefetching (a) and with prefetching (b). Prefetching amortizes the cost of bursts of disk operations over frames with few disk operations, thus eliminating most frame rate drops. The system was configured to prefetch at most 2 MB per frame.

is entirely sequential: a single thread is responsible for computing visibility, performing disk operations, and rendering. The second configuration adds asynchronous fetching to the first configuration, allowing up to 8 fetch threads. The third configuration adds an extra thread for speculative prefetching to the second configuration, allowing up to 2 MB of geometry to be prefetched per frame. Figure 6 shows the frame rates achieved by these three configurations for the 500-frame path. For the purely sequential configuration, we see many downward spikes that correspond to abrupt drops in frame rates, which are caused by the latency of the disk operations, and spoil the user's experience (The first spike happens because the cache is initially empty). When we add asynchronous fetching, many of the downward spikes disappear, but too many still remain. The user's experience is much better, but the frame rate drops are still disturbing. When we add speculative prefetching, all significant downward spikes disappear, and the user experience is smooth. Note that the gain in interactivity comes entirely from overlapping the independent operations. The three configurations achieve exactly the same image accuracy (Figure 7).

Figure 8 shows why prefetching improves the frame rates. The charts compare the amount of geometry that the system reads from disk per frame for the second and third configurations described above. Prefetching greatly reduces the need to fetch large amounts of geometry in a single frame, and thus helps the system to maintain higher and smoother frame rates.

Figure 9 shows that the user speed is another important parameter in the system, and has to be adjusted to the disk bandwidth. When the user speed increases, the changes in the visible set are larger. In other words, as the frame-to-frame coherence decreases, the amount of data the system needs to read per frame increases. Thus, caching and prefetching are more effective if the user moves at speeds compatible with the disk bandwidth. The figure also indicates that higher disk bandwidth should improve frame rates.

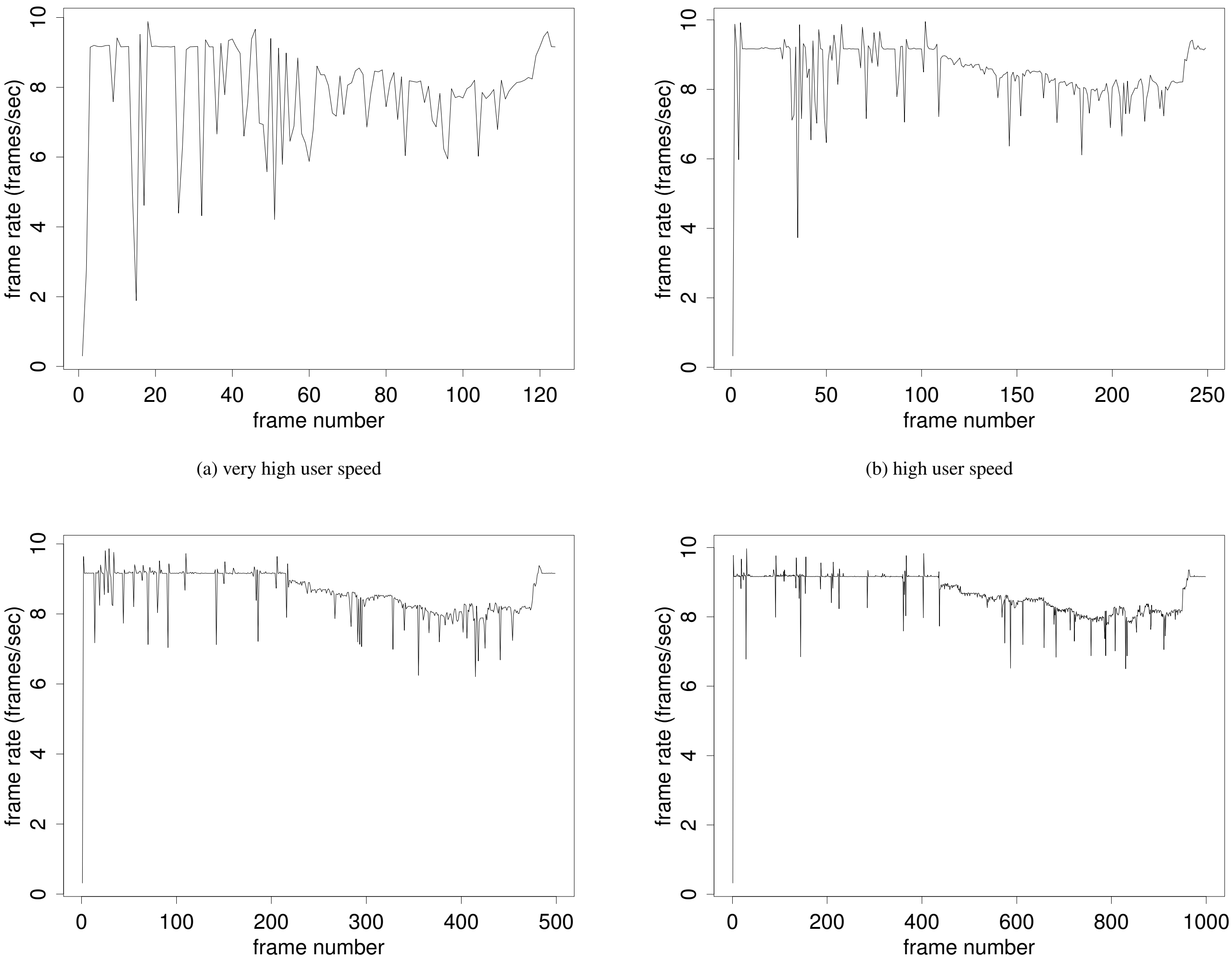
7 Conclusion

We have presented a system for rendering large models on machines with small memory at interactive frame rates. A key component of our out-of-core rendering approach is a new prefetching algorithm based on a from-point visibility algorithm. The prefetch algorithm accurately and efficiently predicts what geometry will be visible in subsequent frames and prefetches them from disk. We believe our system is the first to be able to preprocess the 13-million triangle UNC power plant model and render it interactively on a single PC.

One area of future work is adding level-of-detail (LOD) management to our entire system. In approximate mode, our system may produce images with low accuracy if the camera sees the entire model. El-Sana et al. [2001] show how to integrate LOD management with PLP-based occlusion culling. Another possible area for future work is speeding up rendering in conservative mode, which currently can be much slower than rendering in approximate mode. Finally, we also would like to extend the system to support dynamic scenes.

Acknowledgments

We thank Daniel Aliaga, David Dobkin, Juliana Freire, Thomas Funkhouser, Jeff Korn, Patrick Min, and Emil Praun for suggestions and encouragement. We also thank the University of North Carolina at Chapel Hill and the University of California at Berkeley for providing us with datasets. This research was partly funded by CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico), Brazil.



(a) very high user speed

(b) high user speed

(c) normal user speed

(d) low user speed

Figure 9: Adjusting the user speed to the disk bandwidth. We measured the frame rates along a camera path inside the power plant model for different user speeds (or equivalently, for different number of frames in the path). If the user moves too fast, the frame rates are not smooth. The faster the user moves, the larger the changes in occlusion, and therefore the larger the number of disk operations.

References

AIREY, J. M., ROHLF, J. H., AND FREDERICK P. BROOKS, J. 1990. Towards image realism with interactive update rates in complex virtual building environments. *1990 ACM Symposium on Interactive 3D Graphics 24*, 2 (Mar.), 41–50.

ALIAGA, D., COHEN, J., WILSON, A., ZHANG, H., ERIKSON, C., HOFF, K., HUDSON, T., STÜRZLINGER, W., BAKER, E., BASTOS, R., WHITTON, M., BROOKS, F., AND MANOCHA, D. 1999. MMR: An interactive massive model rendering system using geometric and image-based acceleration. *1999 ACM Symposium on Interactive 3D Graphics* (Apr.), 199–206.

AVILA, L. S., AND SCHROEDER, W. 1997. Interactive visualization of aircraft and power generation engines. In *IEEE Visualization '97*, IEEE, 483–486.

BACH, M. J. 1986. *The Design of the UNIX Operating System*. Prentice Hall.

CHIANG, Y.-J., AND SILVA, C. T. 1997. I/O optimal isosurface extraction. *IEEE Visualization '97* (Nov.), 293–300.

CHIANG, Y.-J., SILVA, C. T., AND SCHROEDER, W. J. 1998. Interactive out-of-core isosurface extraction. *IEEE Visualization '98* (Oct.), 167–174.

CIGNONI, P., ROCCHINI, C., MONTANI, C., AND SCOPIGNO, R. 2002. External memory management and simplification of huge meshes. *IEEE Transactions on Visualization and Computer Graphics*.

CLARK, J. H. 1976. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM 19*, 10 (Oct.), 547–554.

CORRÊA, W. T., KLOSOWSKI, J. T., AND SILVA, C. T. 2002. iWalk: Interactive out-of-core rendering of large models. Technical Report TR-653-02, Princeton University. Available at: http://www.cs.princeton.edu/˜wtcorrea/papers/iwalk.pdf.

COX, M. B., AND ELLSWORTH, D. 1997. Application-controlled demand paging for out-of-core visualization. *IEEE Visualization '97* (Nov.), 235–244.

DURAND, F., DRETTAKIS, G., THOLLOT, J., AND PUECH, C. 2000. Conservative visibility preprocessing using extended projections. In *Proceedings of Siggraph 2000*, 239–248.

EL-SANA, J., AND CHIANG, Y.-J. 2000. External memory view-dependent simplification. *Computer Graphics Forum 19*, 3 (Aug.), 139–150.

EL-SANA, J., SOKOLOVSKY, N., AND SILVA, C. T. 2001. Integrating occlusion culling with view-dependent rendering. In *IEEE Visualization 2001*, 371–378.

ERIKSON, C., MANOCHA, D., AND BAXTER III, W. V. 2001. HLODs for faster display of large static and dynamic environments. In *2001 ACM Symposium on Interactive 3D Graphics*, 111–120.

FUNKHOUSER, T. A., SÉQUIN, C. H., AND TELLER, S. J. 1992. Management of large amounts of data in interactive building walkthroughs. *1992 ACM Symposium on Interactive 3D Graphics 25*, 2 (Mar.), 11–20.

FUNKHOUSER, T. A. 1996. Database management for interactive display of large architectural models. *Graphics Interface '96* (May), 1–8.

GARLICK, B. J., BAUM, D. R., AND WINGET, J. M. 1990. Interactive viewing of large geometric databases using multiprocessor graphics workstations. In *Siggraph Course: Parallel Algorithms and Architectures for 3D Image Generation*. ACM Siggraph, 239–245.

GINDELE, B. S. 1977. Buffer block prefetching method. *IBM Technical Disclosure Bulletin 20*, 2, 696–697.

KLOSOWSKI, J. T., AND SILVA, C. T. 2000. The prioritized-layered projection algorithm for visible set estimation. *IEEE Transactions on Visualization and Computer Graphics 6*, 2 (Apr.-June), 108–123.

KLOSOWSKI, J. T., AND SILVA, C. T. 2001. Efficient conservative visibility culling using the prioritized-layered projection algorithm. *IEEE Transactions on Visualization and Computer Graphics 7*, 4 (Oct.-Dec.), 365–379.

PHARR, M., KOLB, C., GERSHBEIN, R., AND HANRAHAN, P. 1997. Rendering complex scenes with memory-coherent ray tracing. *Proceedings of Siggraph 97* (Aug.), 101–108.

PRZYBYLSKI, S. A. 1990. *Cache and Memory Hierarchy Design: A Performance-Directed Approach*. Morgan Kaufmann.

REGE, A., 2002. Occlusion extensions. http://developer.nvidia.com/docs/-IO/2645/ATT/GDC2002_occlusion.pdf.

SAMET, H. 1990. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley.

SCHAUFLER, G., DORSEY, J., DECORET, X., AND SILLION, F. X. 2000. Conservative volumetric visibility with occluder fusion. In *Proceedings of Siggraph 2000*, 229–238.

SEVERSON, K., 1999. VISUALIZE workstation graphics for Windows NT. HP product literature.

SHEN, H.-W., CHIANG, L.-J., AND MA, K.-L. 1999. A fast volume rendering algorithm for time-varying fields using a time-space partitioning (TSP) tree. *IEEE Visualization '99* (Oct.), 371–378.

SUTTON, P. M., AND HANSEN, C. D. 2000. Accelerated isosurface extraction in time-varying fields. *IEEE Transactions on Visualization and Computer Graphics 6*, 2 (Apr.-June), 98–107.

TELLER, S. J., AND SÉQUIN, C. H. 1991. Visibility preprocessing for interactive walkthroughs. *Proceedings of Siggraph 91 25*, 4 (July), 61–69.

UENG, S.-K., SIKORSKI, C., AND MA, K.-L. 1997. Out-of-core streamline visualization on large unstructured meshes. *IEEE Transactions on Visualization and Computer Graphics 3*, 4 (Oct.-Dec.), 370–380.

UNC, 1999. Power plant model. http://www.cs.unc.edu/˜geom/-Powerplant/. The Walkthru Group at UNC Chapel Hill.

VARADHAN, G., AND MANOCHA, D. 2002. Out-of-core rendering of massive geometric environments. In *IEEE Visualization 2002*.

WALD, I., SLUSALLEK, P., AND BENTHIN, C. 2001. Interactive distributed ray tracing of highly complex models. *Rendering Techniques 2001*, 277–288.

WONKA, P., WIMMER, M., AND SCHMALSTIEG, D. 2000. Visibility preprocessing with occluder fusion for urban walkthroughs. In *Rendering Techniques 2000*, 71–82.

WONKA, P., WIMMER, M., AND SILLION, F. 2001. Instant visibility. *Computer Graphics Forum 20*, 3, 411–421.

Eurographics 2006

State-of-the-art in Real-time, Interactive Massive Model Visualization

Putting Theory into Practice

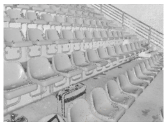
(Inigo Quilez, VRcontext)

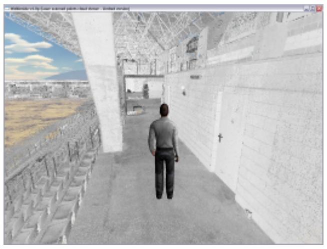
September 2006

Perspective

Lasser scanning - pointclouds

•About 100 billion points
•With collision detection

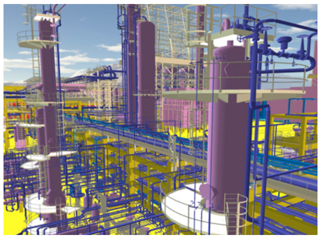


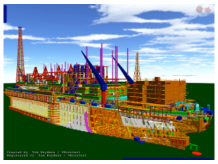


UEFA stadium, 2.5 billion points dataset – 200 scans

CAD File

•An FPSO with more than 1 billion polygons
•A refinery with 4 million objects (>1 billion polygons)
•A nuclear reactor room with more than 1 million primitives



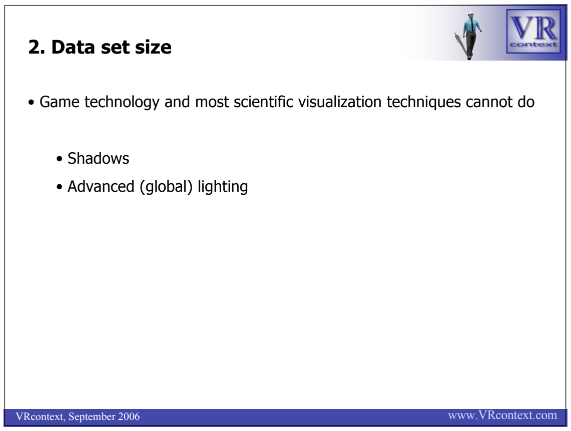


Problems

- 1. Most real data is bad formed
- 2. About data set size
  gamming technologie are unsuitable for massive models
- 3. About quallity
  massive model rendering techniques are not design for quallity
- 4. Other practial problems
- 5. Somehow incorrect assumtions on the research

1. Most real data is bad formed

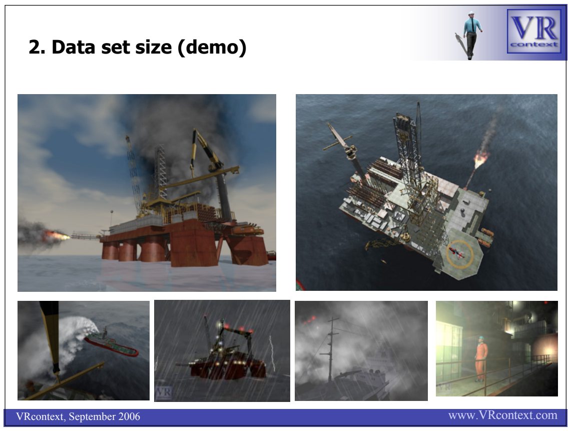
- CAD models:
  - T junctions
  - incorrectly oriented geometry
  - not orientable geometry
  - duplicated geometry
  - clashes
- Laser scanned point clouds
  - noise
- GIS
  - less than optimal triangulation



2. Data set size

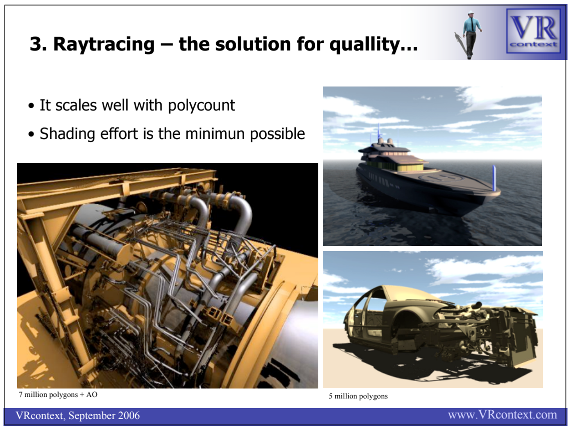
- Game technology and most scientific visualization techniques cannot do
  - Shadows
  - Advanced (global) lighting

VRcontext, September 2006 www.VRcontext.com



2. Data set size (demo)

VRcontext, September 2006 www.VRcontext.com



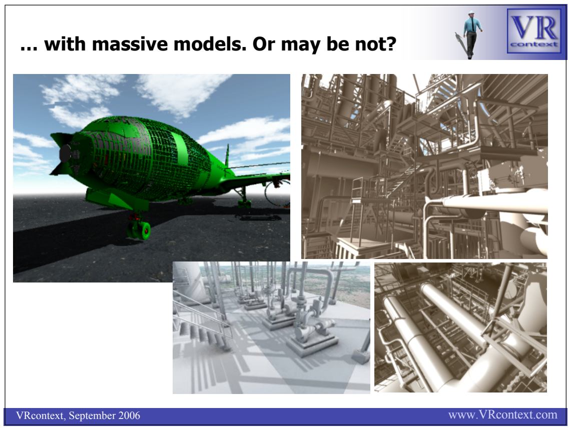
3. Raytracing – the solution for quallity…

- It scales well with polycount
- Shading effort is the minimun possible

7 million polygons + AO

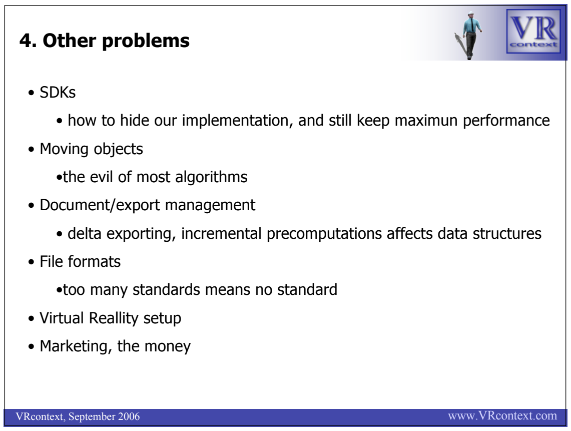
5 million polygons

VRcontext, September 2006 www.VRcontext.com



… with massive models. Or may be not?

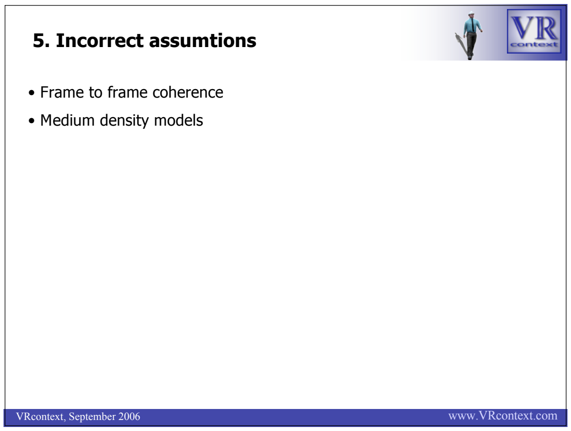
VRcontext, September 2006 www.VRcontext.com



4. Other problems

- SDKs
  - how to hide our implementation, and still keep maximun performance
- Moving objects
  - the evil of most algorithms
- Document/export management
  - delta exporting, incremental precomputations affects data structures
- File formats
  - too many standards means no standard
- Virtual Reallity setup
- Marketing, the money

VRcontext, September 2006 www.VRcontext.com



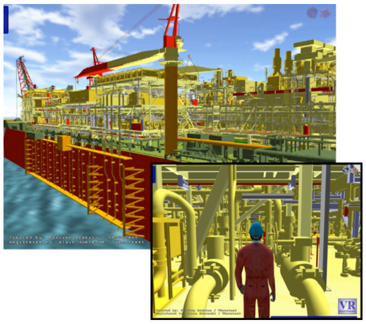
5. Incorrect assumtions

- Frame to frame coherence
- Medium density models

VRcontext, September 2006 www.VRcontext.com

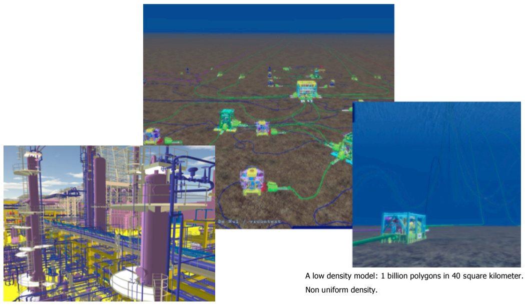
5. Density of the models

- High Density
  - D > 10k t/m^3
  - DR ~ $1:10^3$
- Medium Density
  - D ~ 100-1k t/m^3
  - DR ~$1:10^5$
- Low Density
  - D < 1tri/m^3
  - DR >$1:10^6$



A typical medium density model (600 million polygons)

Low Density models



A low density model: 1 billion polygons in 40 square kilometer. Non uniform density.

2 million objects in 5 square kilometers. Uniform density.

Low Density models

Can become Medium Density locally… if we cluster the geometry