# Level-of-Detail Techniques and Cache-Coherent Layouts

**Sung-Eui Yoon**

**Lawrence Livermore National Laboratory**

Note: this talk is not supported or sanctioned by DoE, UC, LLNL, CASC

# Collaborators

- **Prof. Dinesh Manocha**
  - ♦ **Primary investigator**

- **Russ Gayle**
- **Christian Lauterbach**
- **Brandon Lloyd**
- **Brian Salomon**

# Goal

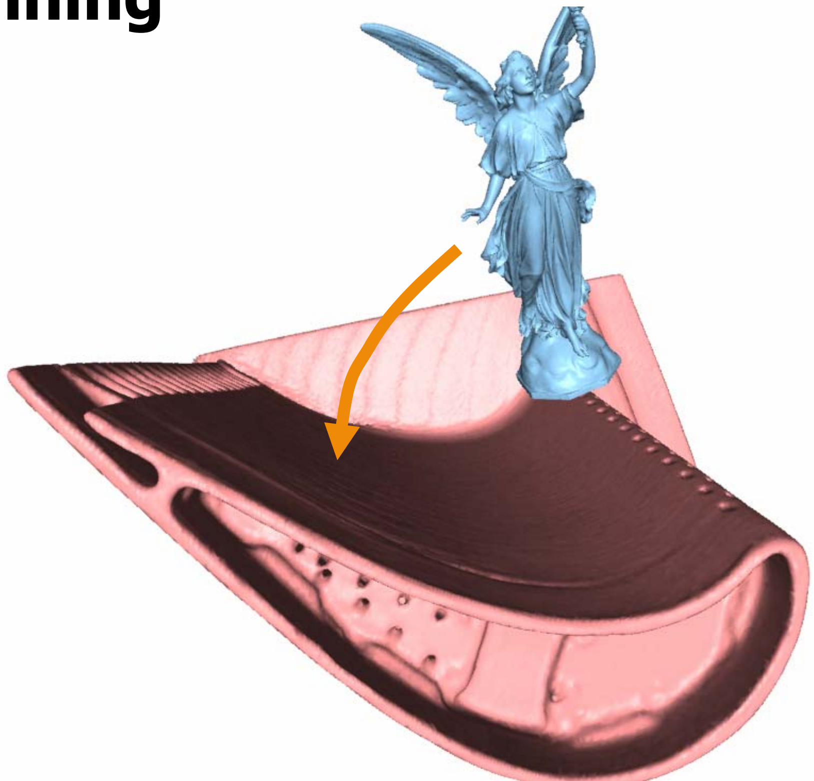
- **Efficient algorithms for:**
  - ♦ **Interactive visualization (rasterization and ray tracing)**
  - ♦ **Collision detection**
  - ♦ **Other geometric applications**

# Interactive Visualization

- **Walkthrough**
  - ♦ **large man-made structures**
- **Investigate scientific simulation data**

# Collision Detection

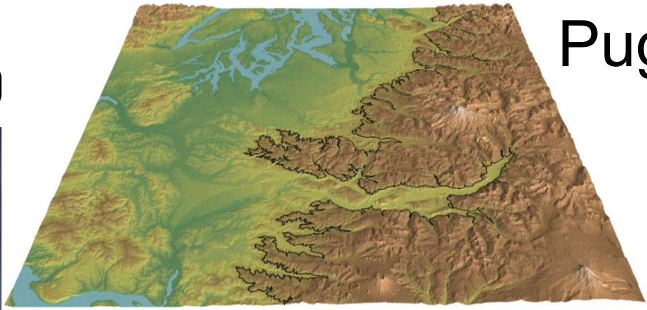
- **Main component of:**
  - ♦ **Dynamic simulation**
  - ♦ **Navigation and path planning**
  - ♦ **Haptic rendering**
  - ♦ **Virtual prototyping**



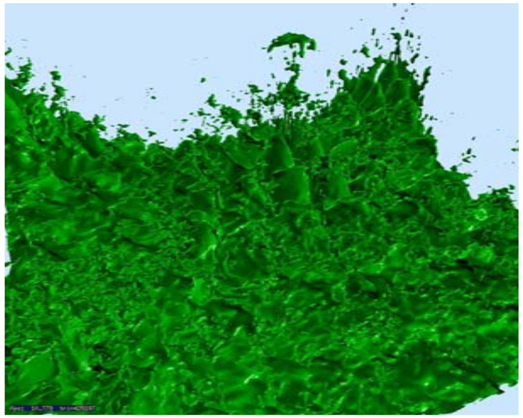
# Challenges

- **Complex and massive models**
  - ♦ **Ever-increasing model complexity**

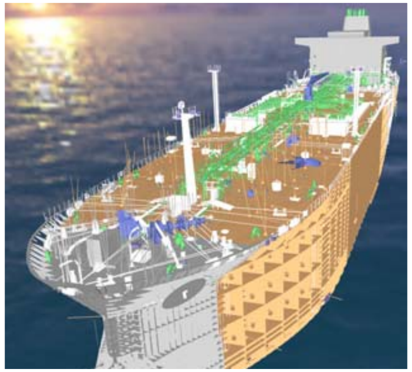
St. Matthew, 372M (10GB)



Puget sound, 400M+



Power plant, 12M



Double eagle tanker, 82M

**Isosurface (472M) from a turbulence simulation**

# Issues and Our approaches

- **Huge amount of data**
  - ♦ **Take tens of giga-bytes in disk and memory**
- **Data access time**
  - ♦ **Major bottleneck**
- **Orthogonal approaches**
  - ♦ **Levels-of-detail (LODs) techniques**
  - ♦ **Cache-coherent layouts**

# Orthogonal Approaches

- **LOD approaches**

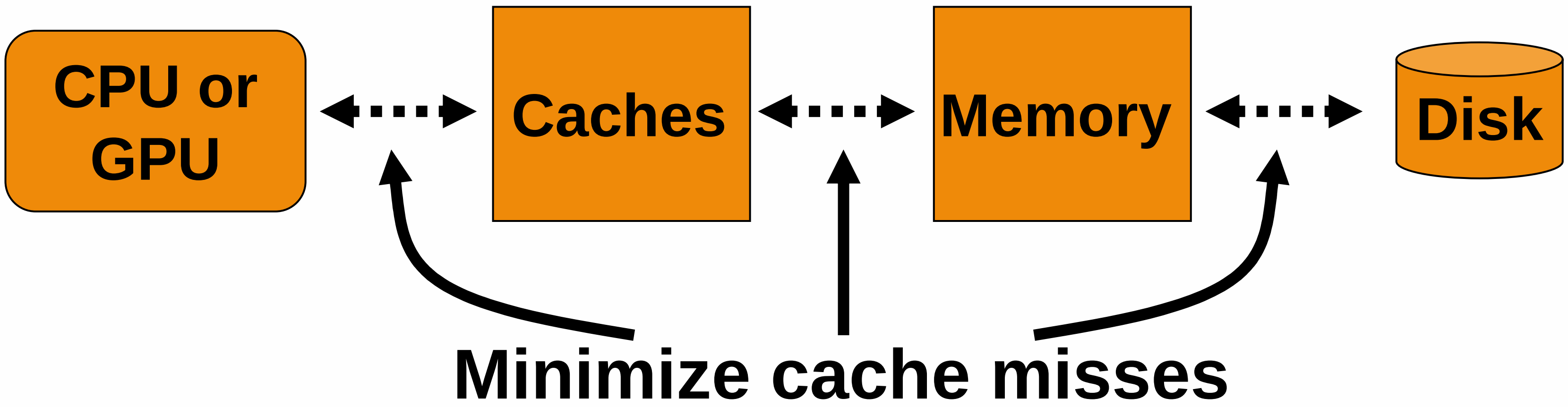
**Use simplification given an error** →

Reduce the amount of necessary data!

- **Cache-coherent layouts**

# Orthogonal Approaches

- LOD approaches
- Cache-coherent layouts




Reduce expensive I/O accesses!

# Orthogonal Approaches

- **LOD approaches**
  - ♦ Dynamic simplification for rasterization
  - ♦ Static LODs for ray tracing
- **Cache-coherent layouts**
  - ♦ Cache-efficient layouts of meshes and graphs
  - ♦ Cache-efficient layouts of BVHs

# Outline

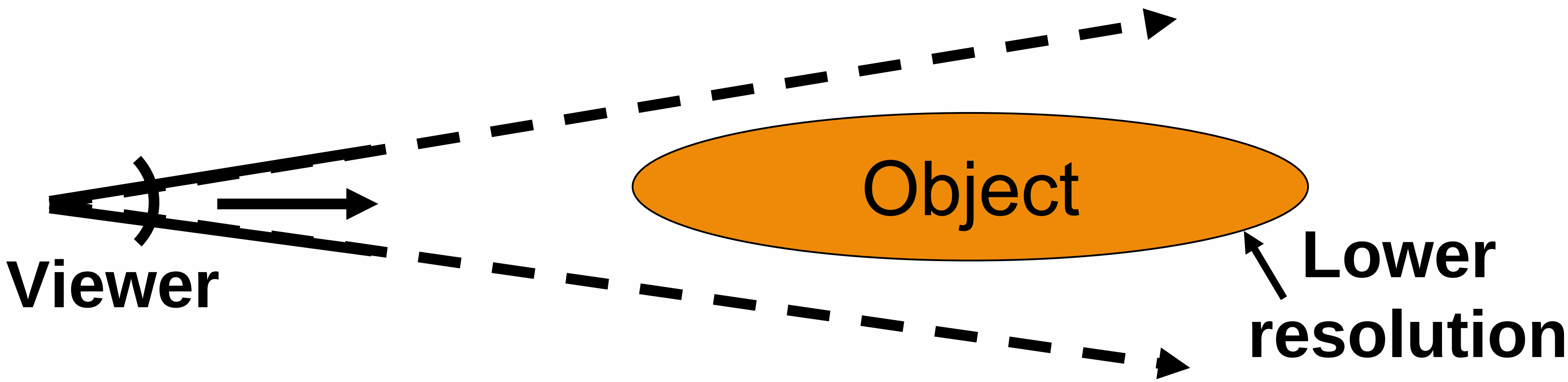
- Dynamic simplification for rasterization
- LOD-based ray tracing
- Cache-coherent layouts
- Conclusion

# Outline

- Dynamic simplification for rasterization
- LOD-based ray tracing
- Cache-coherent layouts
- Conclusion

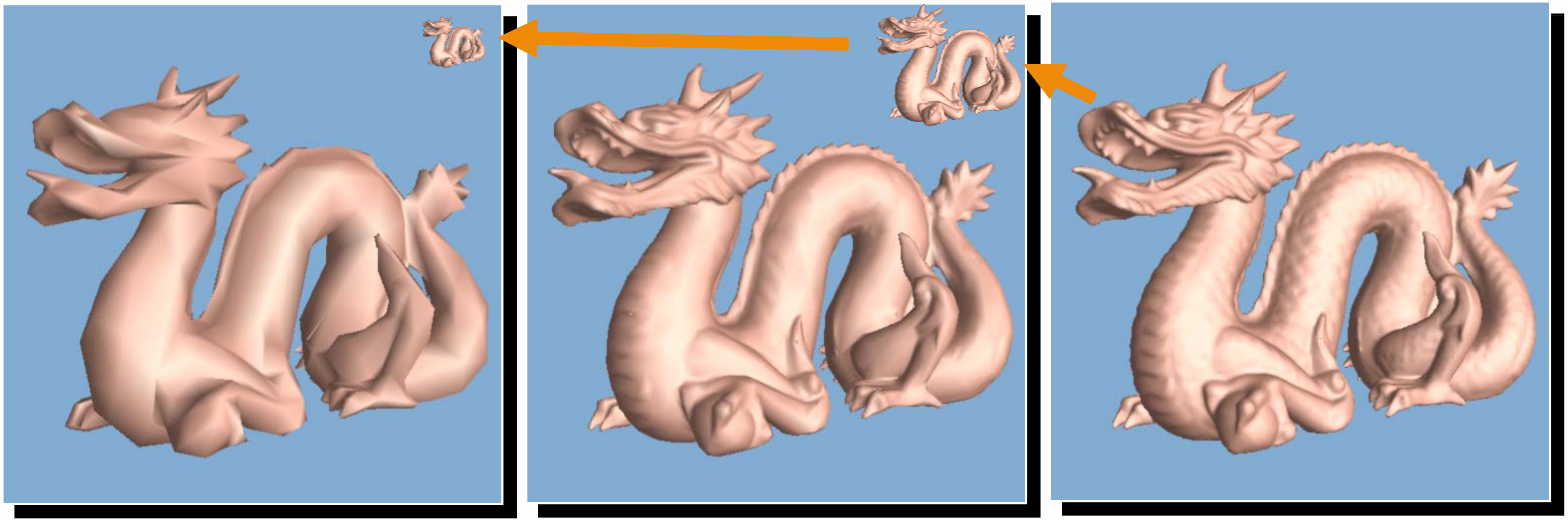
# View-Dependent Rendering

- [Clark 76, Funkhouser and Sequin 93]




- Static levels-of-detail (LODs)
- Dynamic (or view-dependent) simplification

# Static LODs



2,000 faces 10,000 faces 50,000 faces

pop pop

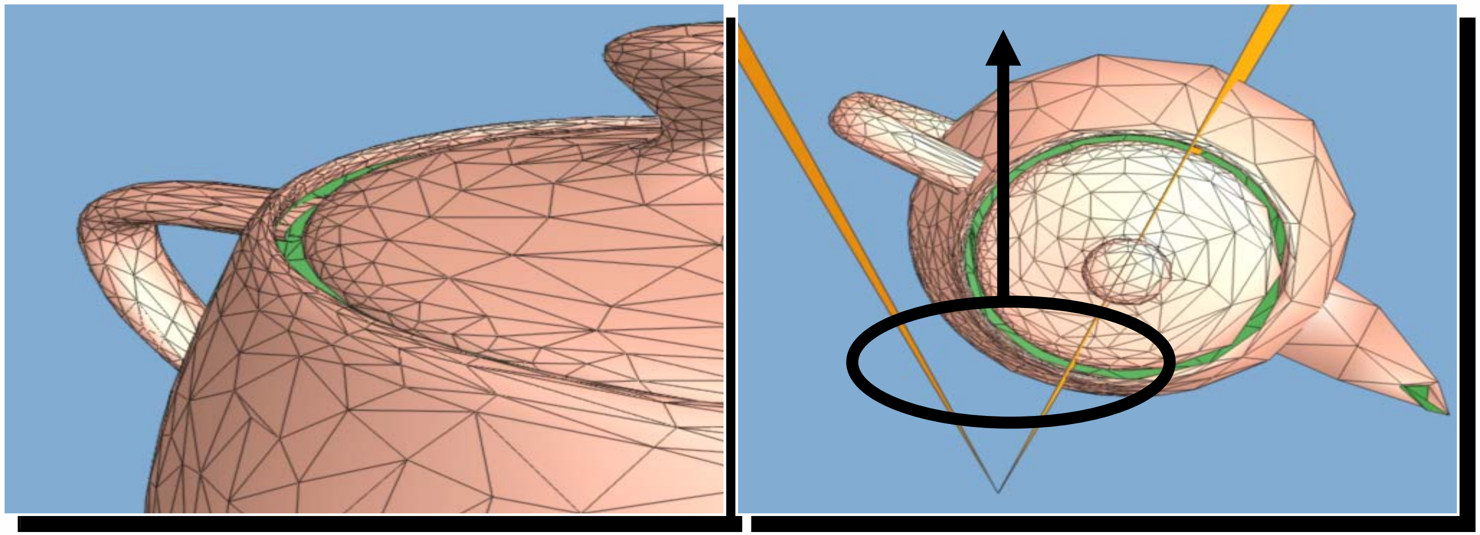


Courtesy of [Hoppe 97]

# Dynamic Simplification

- **Provides smooth and varying LODs over the mesh** **[Hoppe 97]**

1st person's view 3rd person's view





Courtesy of [Hoppe 97]

# Dynamic Simplification: Issues

- **Representation**
  - ♦ **High CPU usages**
- **Runtime computation and rendering**
  - ♦ **Low cache-utilization**
- **Construction**
  - ♦ **Out-of-core computations**

# Toward Scale-able Dynamic Simplification Method

- **View-dependent rendering [Yoon et al. Vis 04]**
  - New multi-resolution hierarchy (CHPM)
  - Out-of-core construction
  - Applied to collision detection [Yoon et al. SGP 04] and shadow computation [Lloyd et al. EGSR 06]

- **Cache-coherent layouts [Yoon et al. SIG 05]**
  - Higher GPU utilization

# Live Demo – View-Dependent Rendering


**20 Pixels of error**

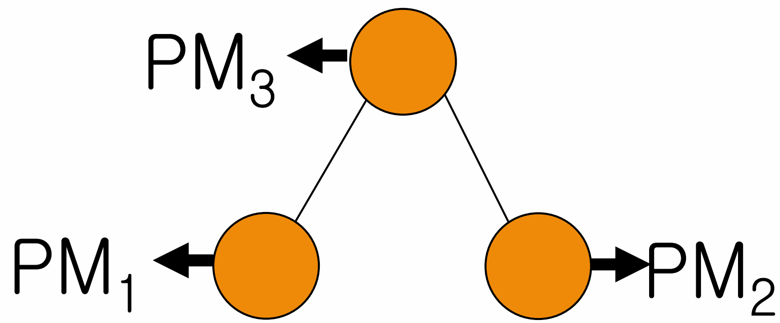
**Pentium 4**

**GeForce Go 6800 Ultra**

**1GB RAM**

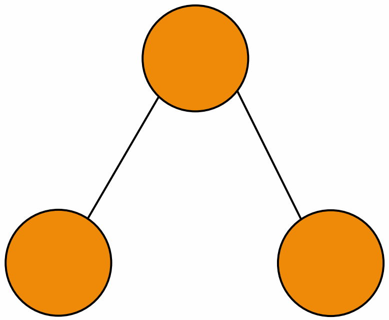
# Clustered Hierarchy of Progressive Meshes (CHPM)

- **Novel dynamic simplification representation**
  - ♦ **Cluster hierarchy**
  - ♦ **Progressive meshes**



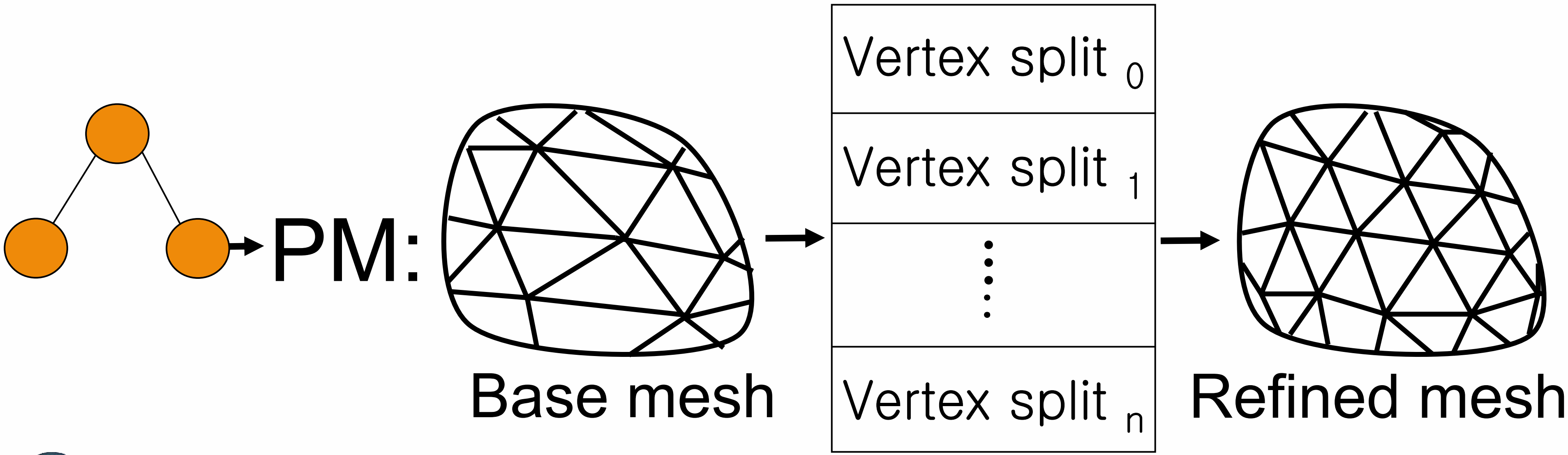
# Clustered Hierarchy of Progressive Meshes (CHPM)

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



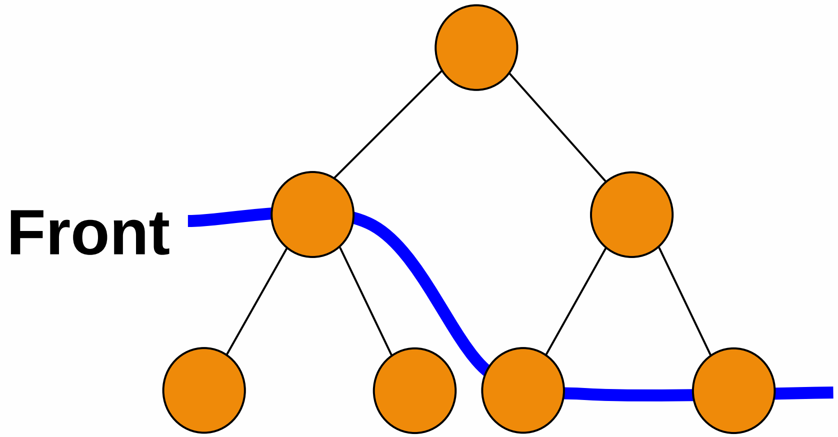
# Clustered Hierarchy of Progressive Meshes (CHPM)

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



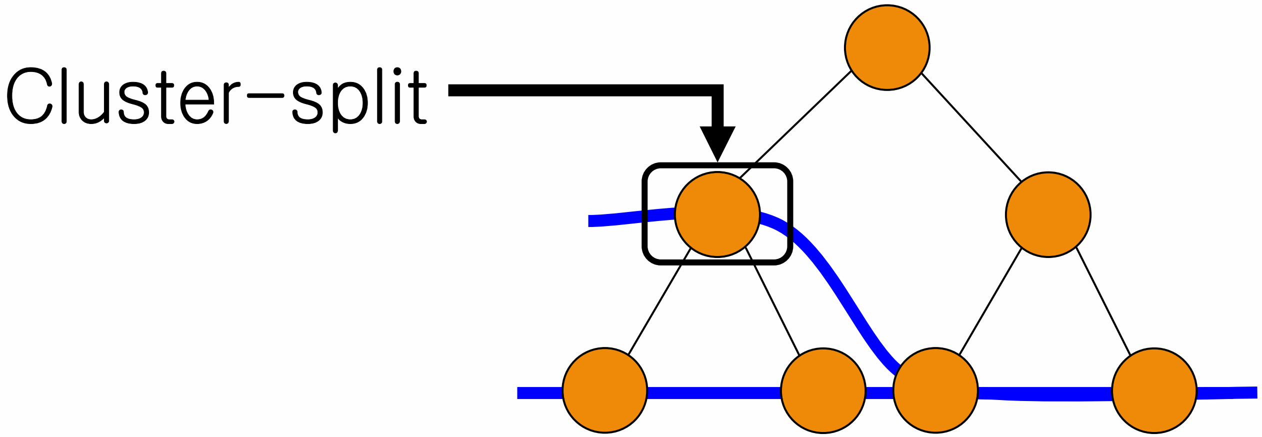
# Two-Levels of Refinement at Runtime

- **Coarse-grained view-dependent refinement**
  - **Provided by selecting a front in the cluster hierarchy**
  - **Inter-cluster level refinements**



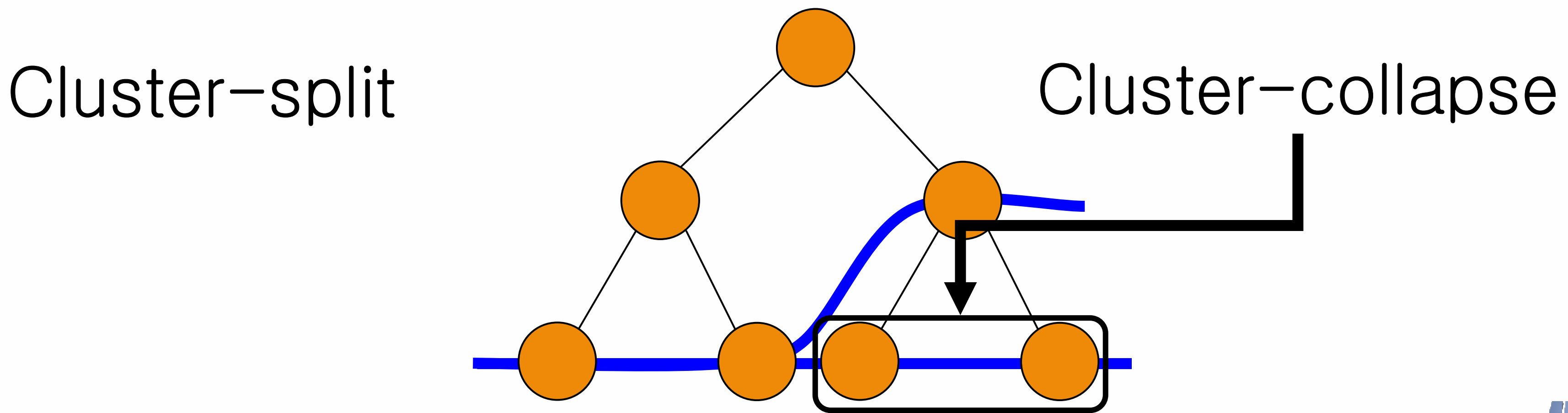
# Two-Levels of Refinement at Runtime

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



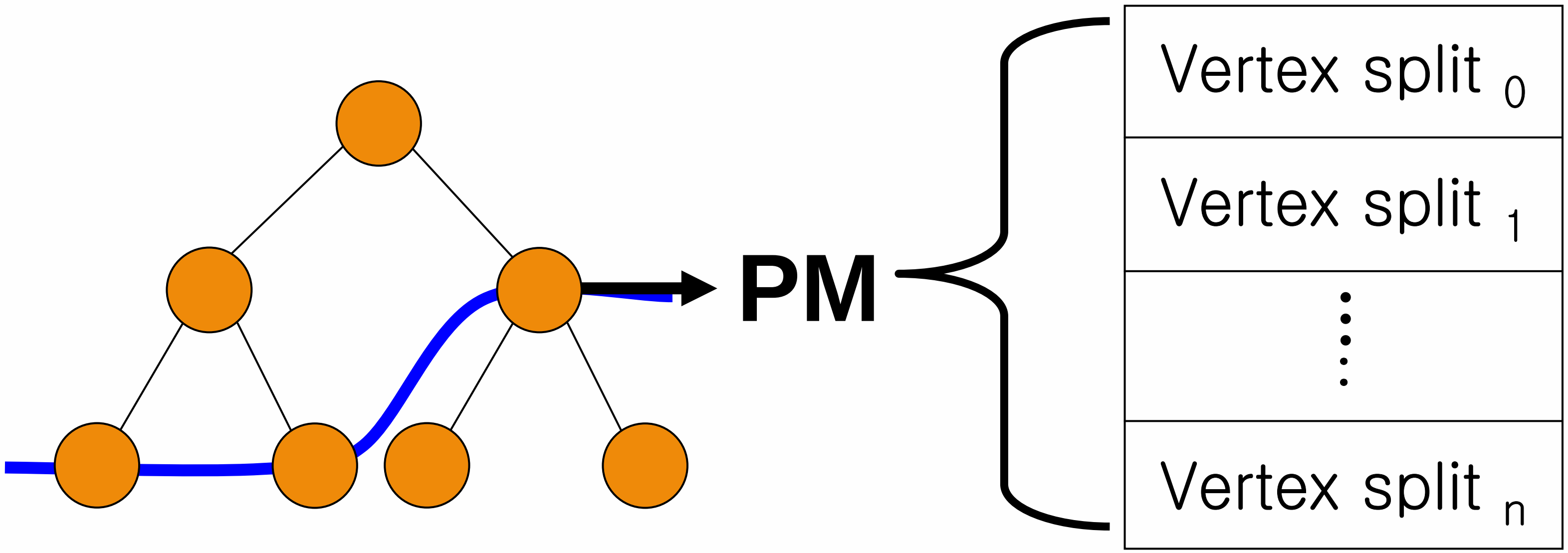
# Two-Levels of Refinement at Runtime

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



# Two-Levels of Refinement at Runtime

- **Fine-grained local refinement**
  - ♦ **Supported by performing vertex splits in PMs**
  - ♦ **Intra-cluster refinements**



# 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

# Overview of Building a CHPM

Input model

↓

**Cluster decomposition**

↓

**Cluster hierarchy generation**

↓

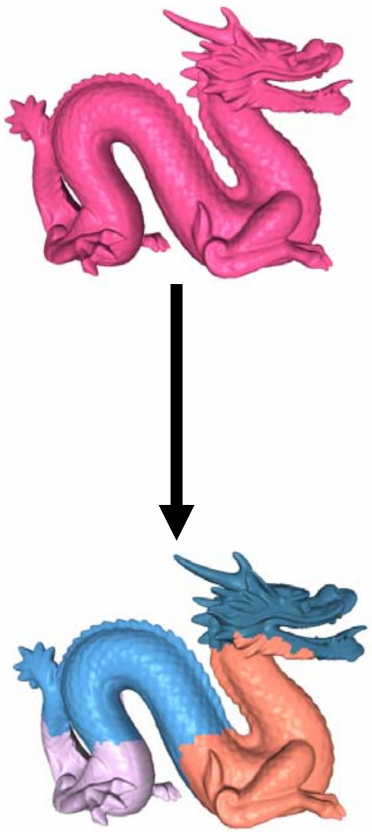
**Hierarchical simplification**

↓

CHPM

**Performed out-of-core**

# Overview of Building a CHPM



Input model

↓

**Cluster decomposition**

↓

**Cluster hierarchy generation**

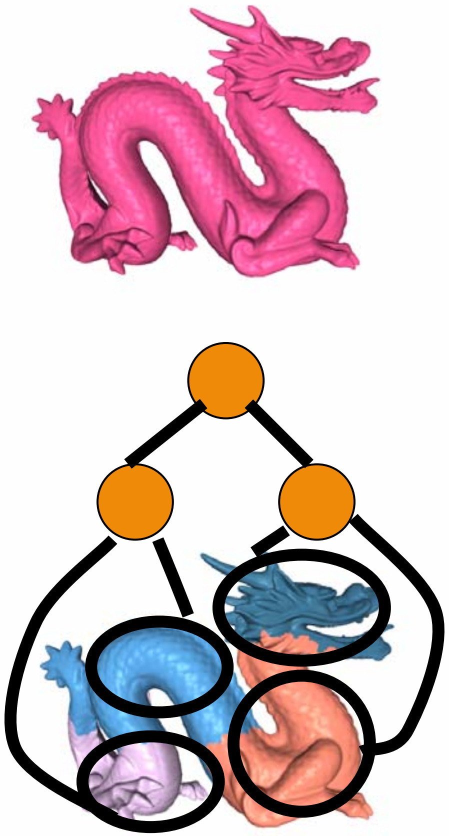
↓

**Hierarchical simplification**

↓

CHPM

# Overview of Building a CHPM



Input model

↓

**Cluster decomposition**

↓

**Cluster hierarchy generation**

↓

**Hierarchical simplification**

↓

CHPM

# Overview of Building a CHPM

Input model

↓

**Cluster decomposition**

↓

**Cluster hierarchy generation**

↓

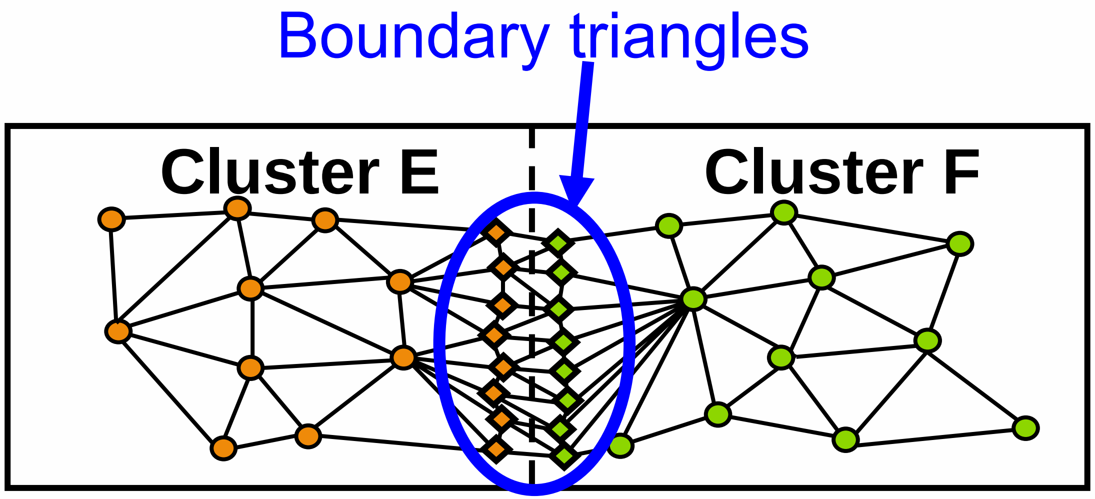
**Hierarchical simplification**

↓

CHPM

# Boundary Constraints

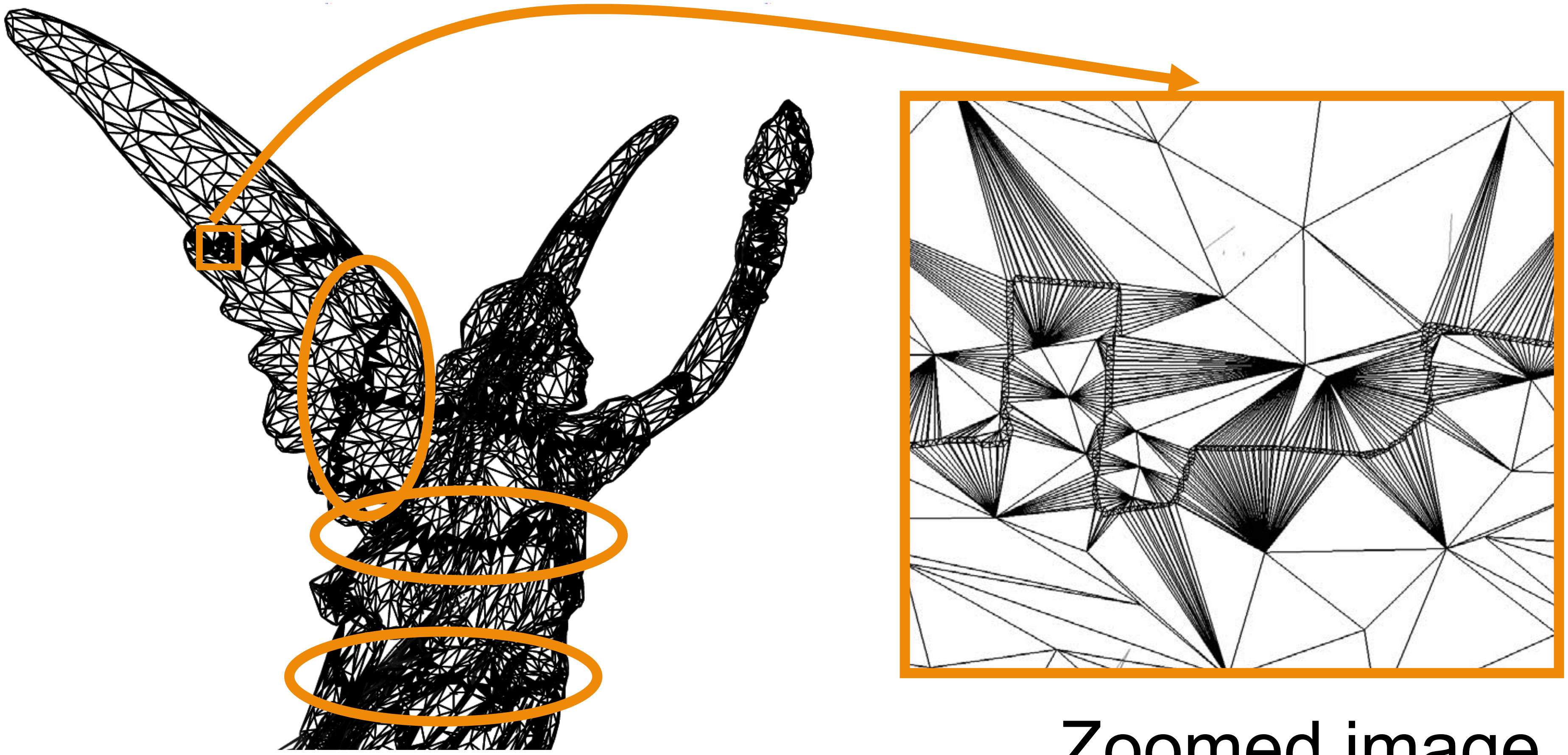
- **Do not simplify boundary triangles**
  - ♦ **Guarantee crack-free boundaries**




- **Common problem in many hierarchical simplification algorithms**
  - ♦ **[Hoppe 98; Prince 00; Govindaraju et al. 03]**

# Boundary Constraints

# Boundary Constraints



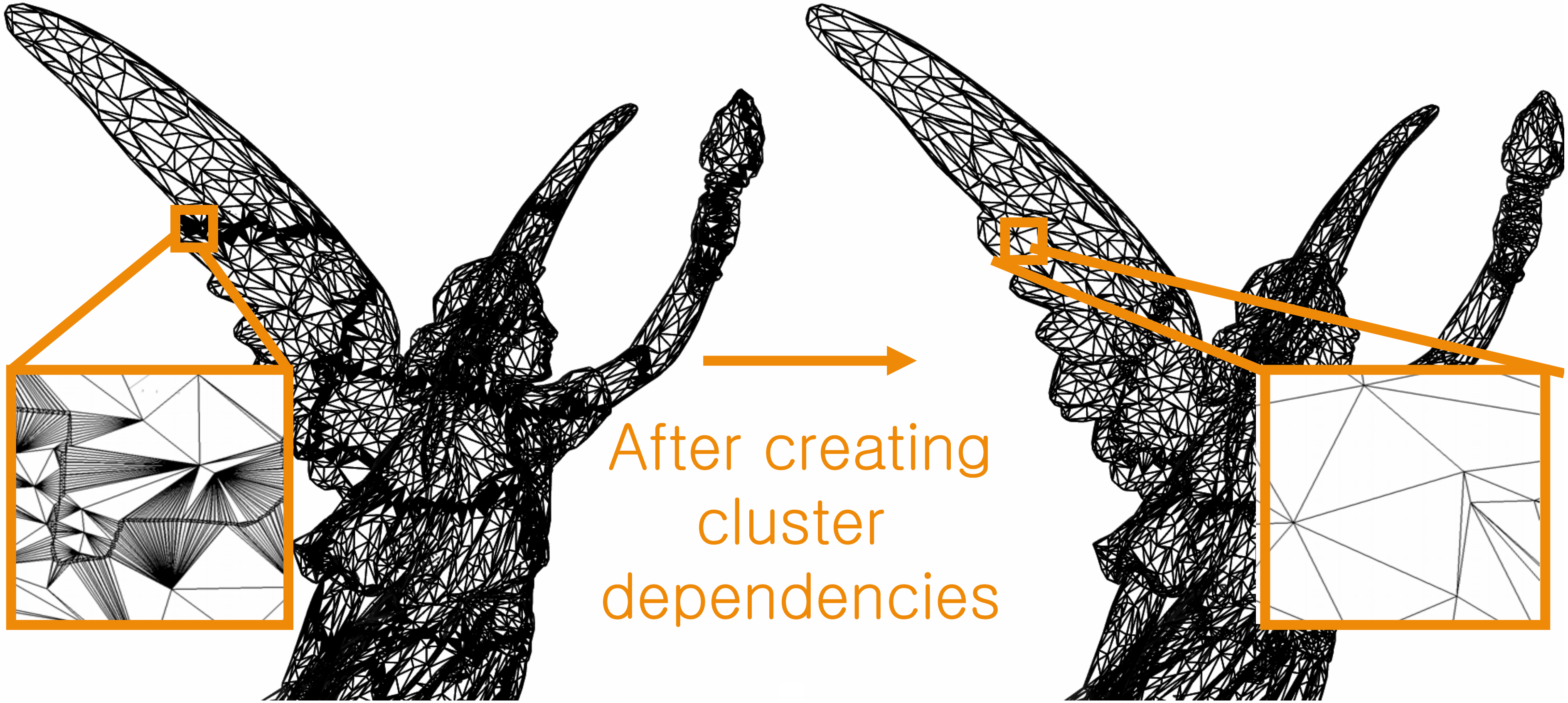
Zoomed image

# Cluster Dependencies

- Replaces preprocessing constraints with runtime dependencies
  - Simplify boundary triangles
  - Consider them at runtime with dependencies

# Cluster Dependencies

227K triangles | 92% triangles reduced | 19K triangles



# Runtime Performance

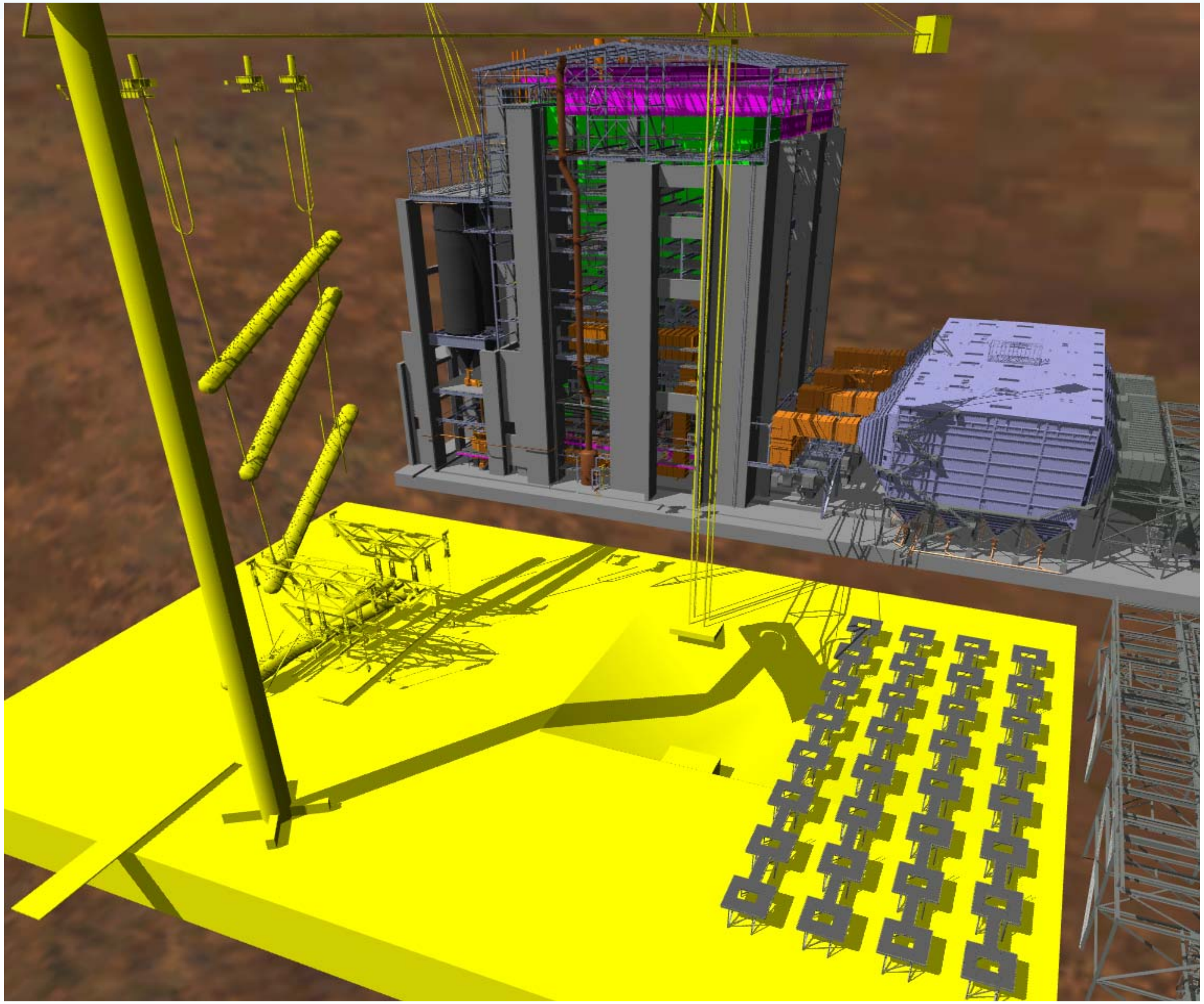
| Model | Pixels of error | Frame rate | Mem. footprint | Model size |
|---|---|---|---|---|
| Power plant | 1 | 28 | 400MB | 1GB |
| St. Matthew | 1 | 29 | 600MB | 13GB |

**512x512 image resolution, GeForce 5950FX**

# Applications

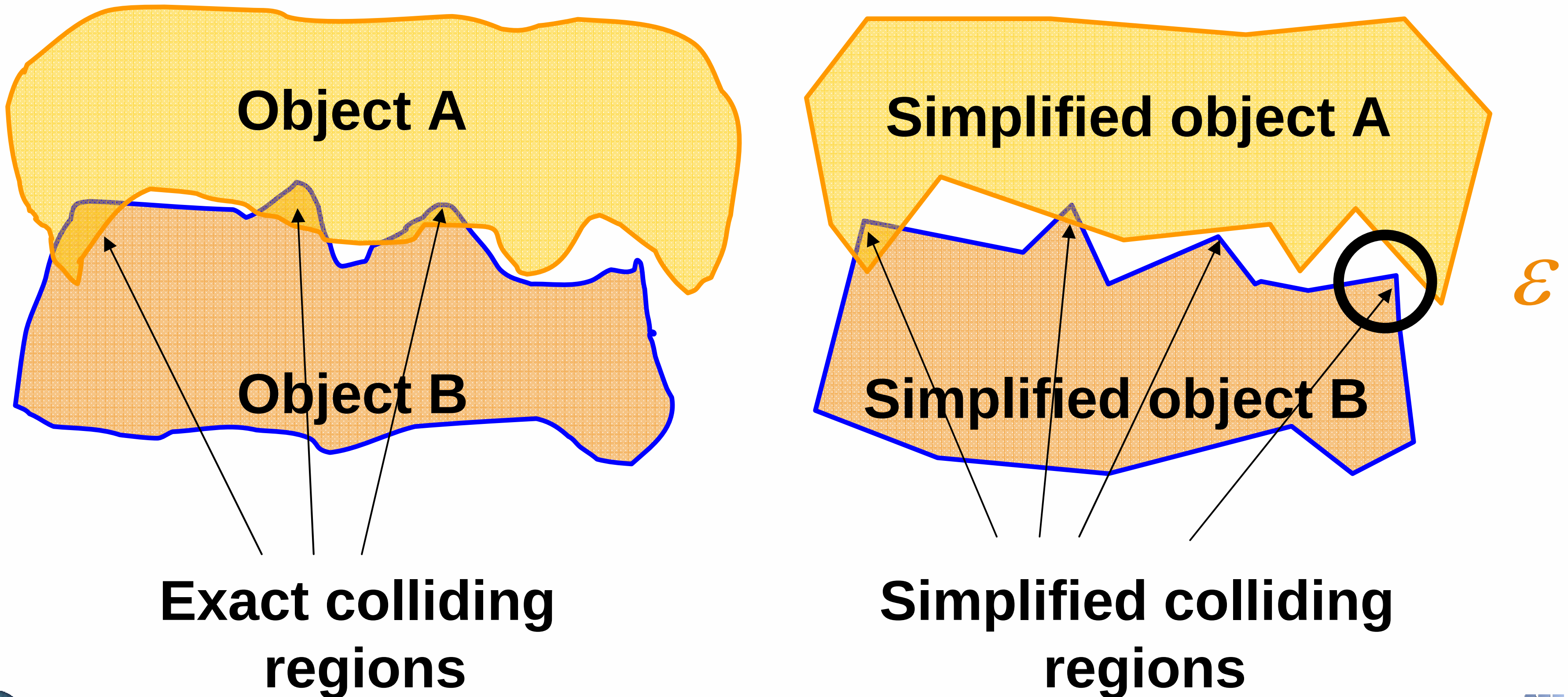
- **Shadow computations**
- **Approximate collision detection**

# Interactive View-Dependent Shadow Generation [Lloyd et al. EGSR 06]



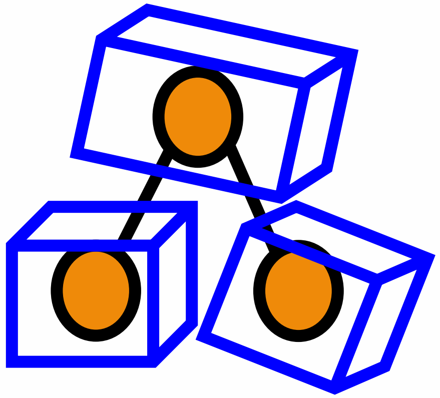
# Approximate Collision Detection [Yoon et al. SGP 04]

- **Perform approximate query based on a simplified mesh**



# CHPM Representation

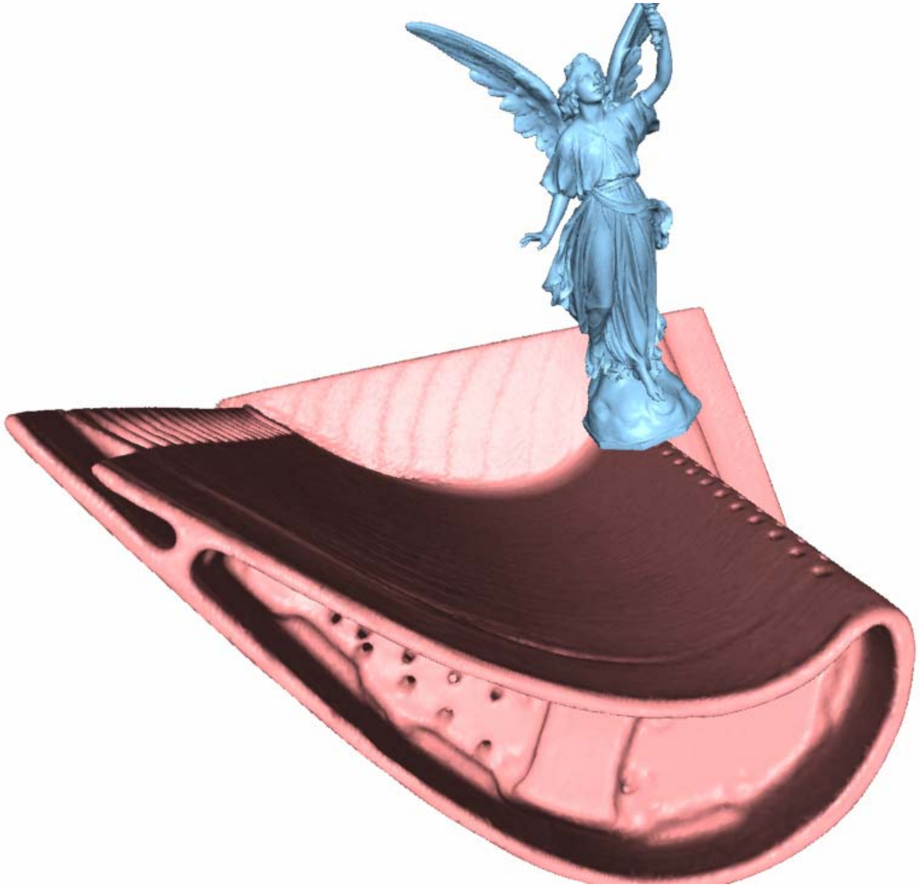
- Serve as a dual hierarchy for collision detection
  - ♦ LOD hierarchy
  - ♦ Bounding volume hierarchy



- Unified representation for:
  - ♦ Rendering and collision detection

- Advantages
  - ♦ Improve the performance
  - ♦ Alleviates simulation discontinuities

# Benchmark Models – Dynamic Simulation



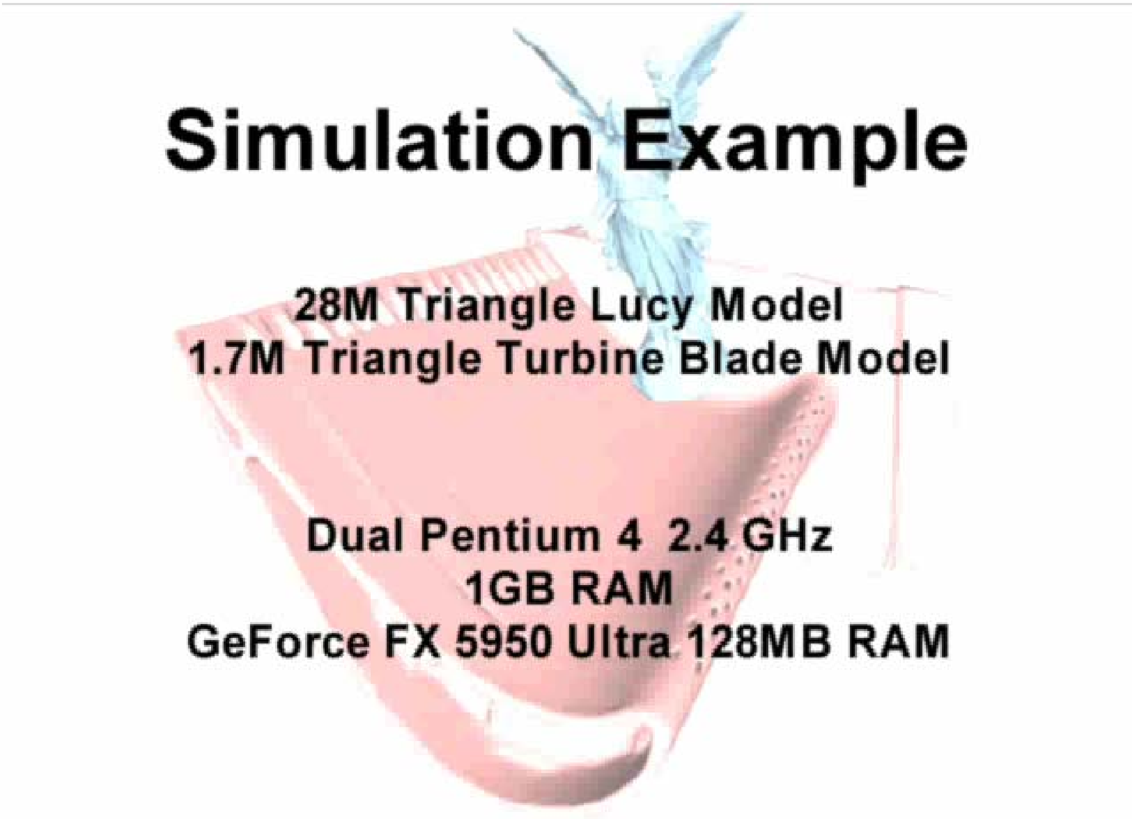
**Lucy model: 28M triangles**

**Turbine model: 1.7M triangles**

**Impulse based rigid body simulation**

**[Mirtich and Canny 1995]**

# Live Demo – Rigid Body Simulation




**Error bound:**
**0.1% of width**
**of Lucy model**

**Average query time:**
**18ms**

# Outline

- Dynamic simplification for rasterization
- **LOD-based ray tracing**
- Cache-coherent layouts
- Conclusion

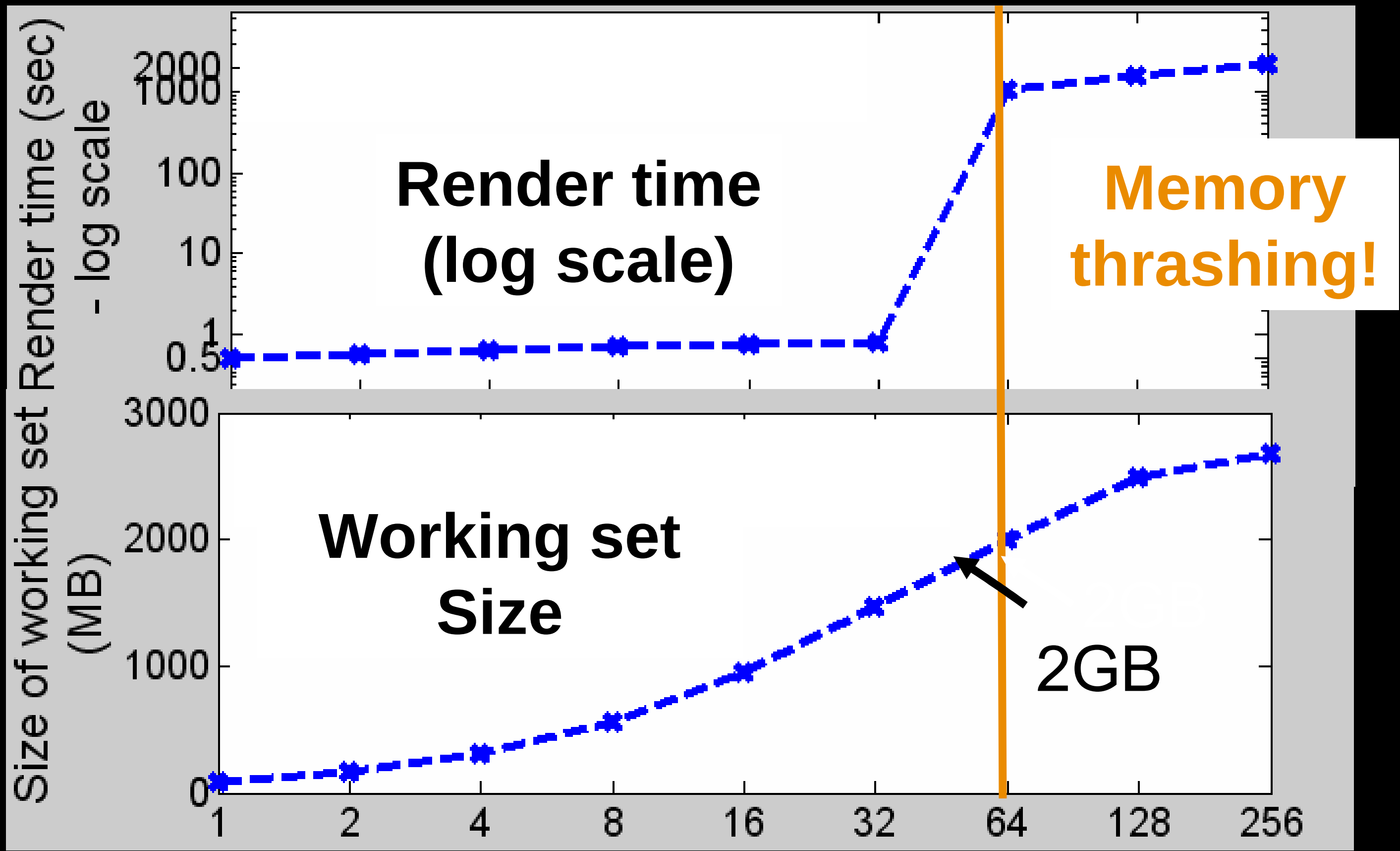
# Ray Tracing

- **Well researched for 25+ years**

- **Slower than rasterization**
- **But: asymptotic performance**
  **~ logarithmic**
  - **Good choice for massive models?**
  - **Observed only in in-core cases**

# Ray Tracing: Performance

- Measured with 2GB main memory




Model complexity (M tri) - log scale

# Incoherent Memory Accesses

- Model with 370M triangles
- Assuming 512x512 resolution
  - Hundreds of triangle per pixel
  - At most <1% of triangles visible
  - Each triangle likely in different area of memory

Scan of Michelangelo's St.Matthew:

# Our approach

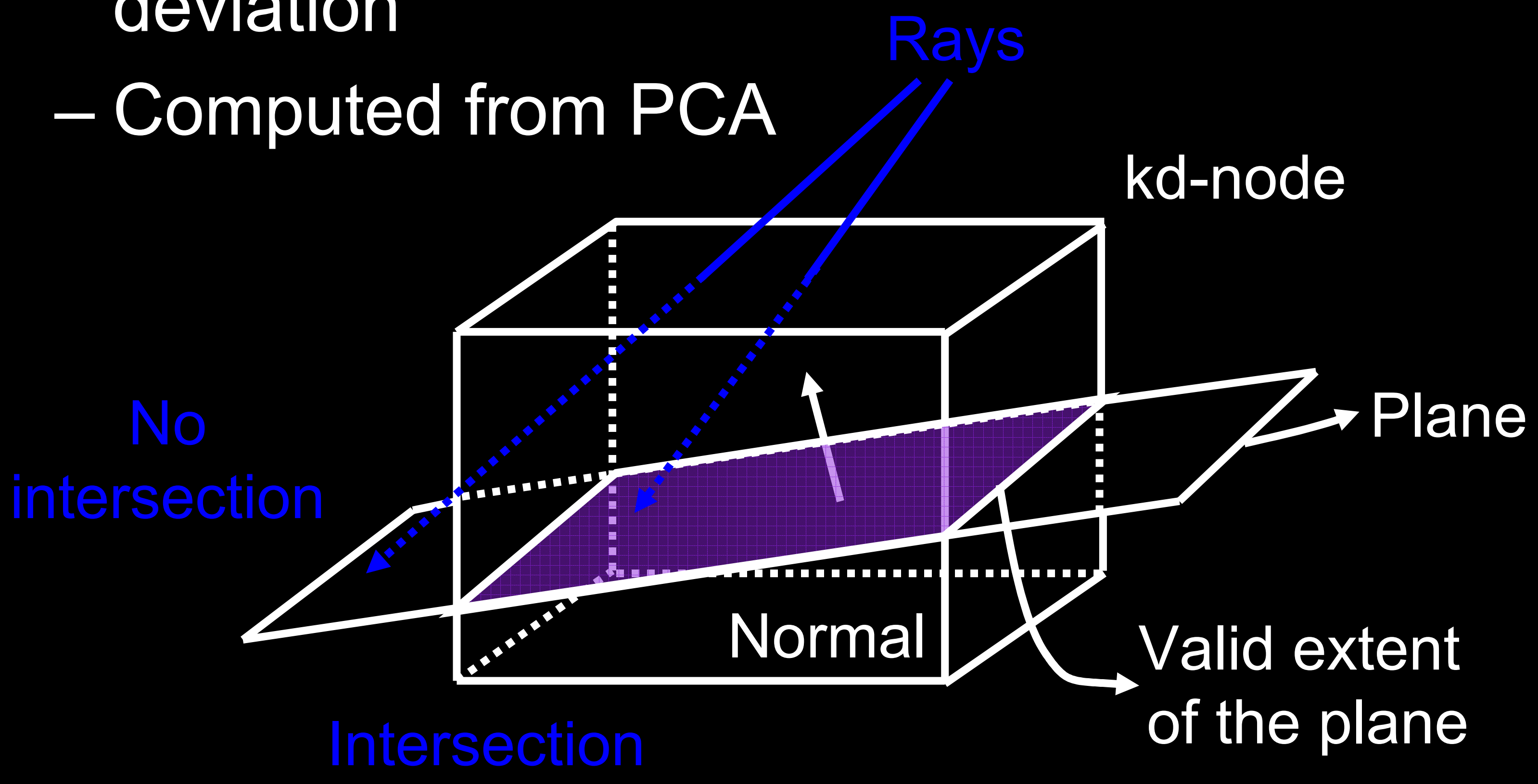
- Add levels-of-detail to ray tracing
  - *LOD*: simplified versions of geometry
- Selection according to *LOD metric*
  - Rasterzation: selection per object
  - Ray tracing: selection per ray
- Main benefit:
  - Reducing working set size
  - Improved memory coherence

# Our approach

- R-LODs [Yoon et al. PG 06]
  - Highly integrated with kd-tree [Wald et al. 05]
  - Can also be integrated with BVHs
- Simple but fast LOD metric
  - Works with shadows, reflections
- Integrates ray and cache coherences

# R-LOD Representation

- Tightly integrated with kd-nodes
  - A plane, material attributes, and surface deviation
  - Computed from PCA



# LOD-based Runtime Traversal

- Modification of efficient kd-tree traversal
  - [Wald 04]
- Traverse, evaluate metric at each node
- If satisfies, intersect with plane instead
  - if it hits, we're done
  - if not, go back up, try other sub tree
- In any case: don't need to go deeper!

# Properties of R-LODs

- Compact and efficient LOD representation
  - Add only 4 bytes to (8 bytes) kd-node
- Drastic simplification
  - Useful for performance improvement

# Properties of R-LODs

- Error-controllable LOD rendering
  - Error is measured in a screen-space in terms of pixels-of-error (PoE)
  - Provides interactive rendering framework

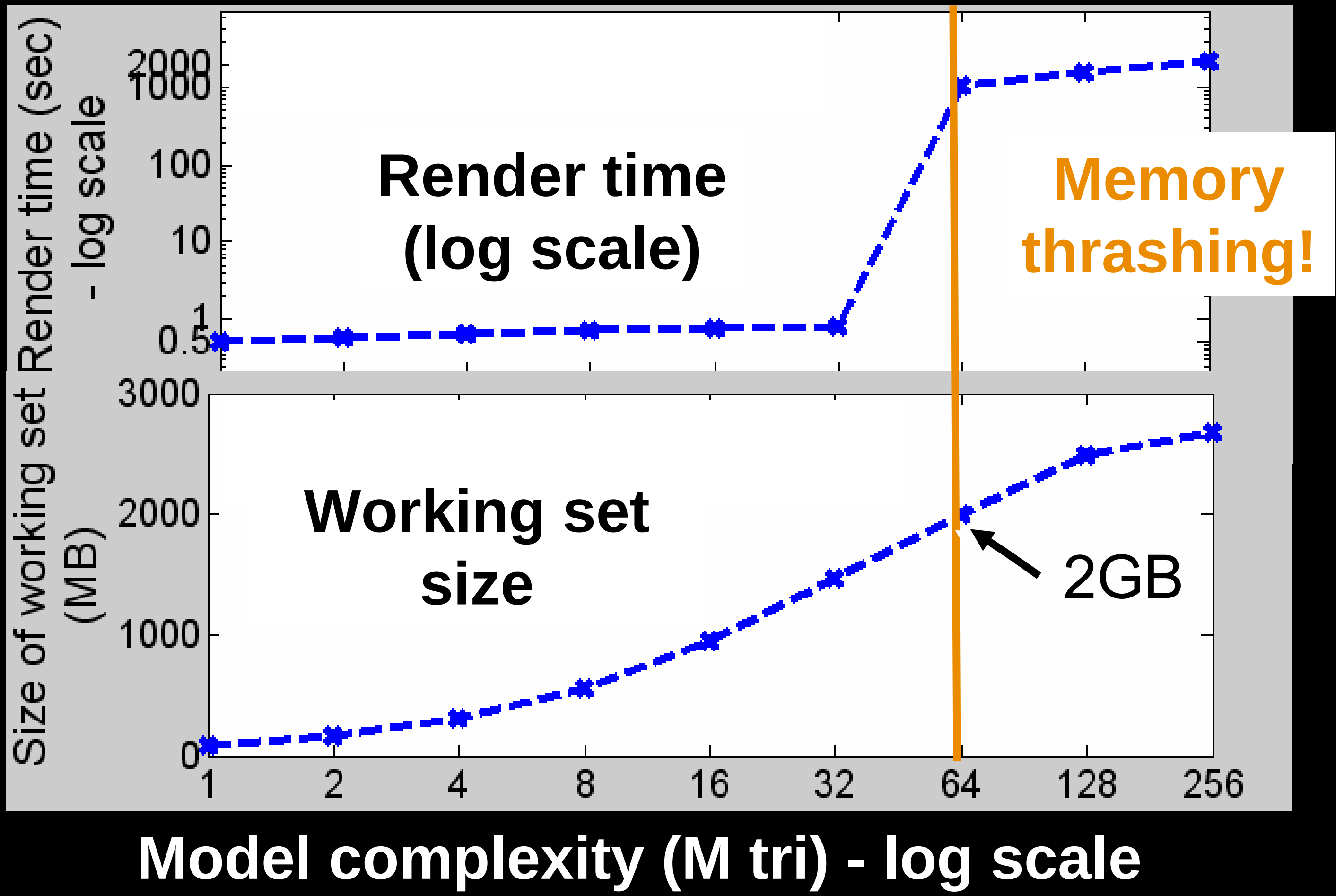
# R-LODs with Different PoE Values

**PoE: Original 1.85 5 10**

**(512x512, no anti-aliasing)**

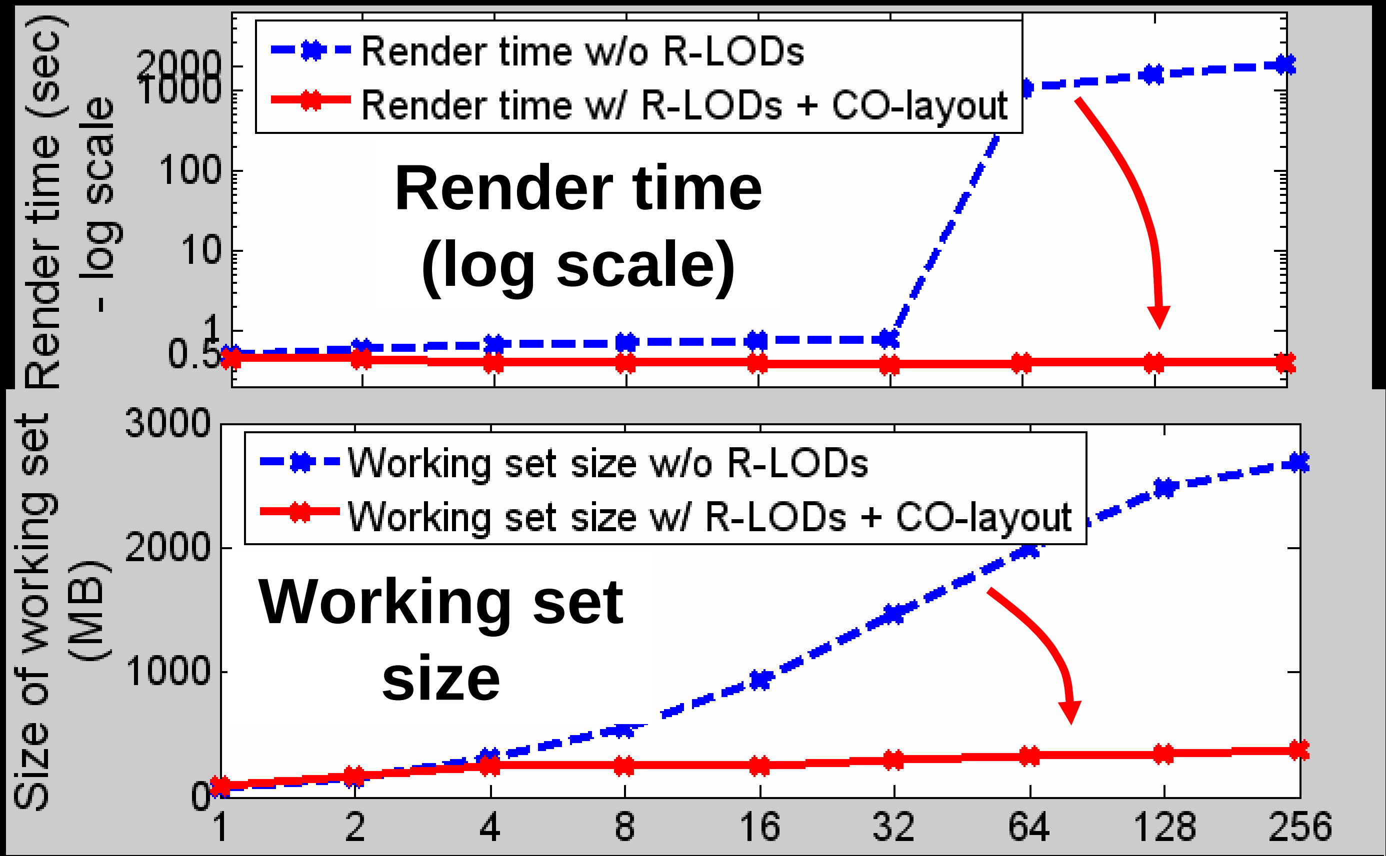
# Ray Tracing: Performance

- Measured with 2GB main memory



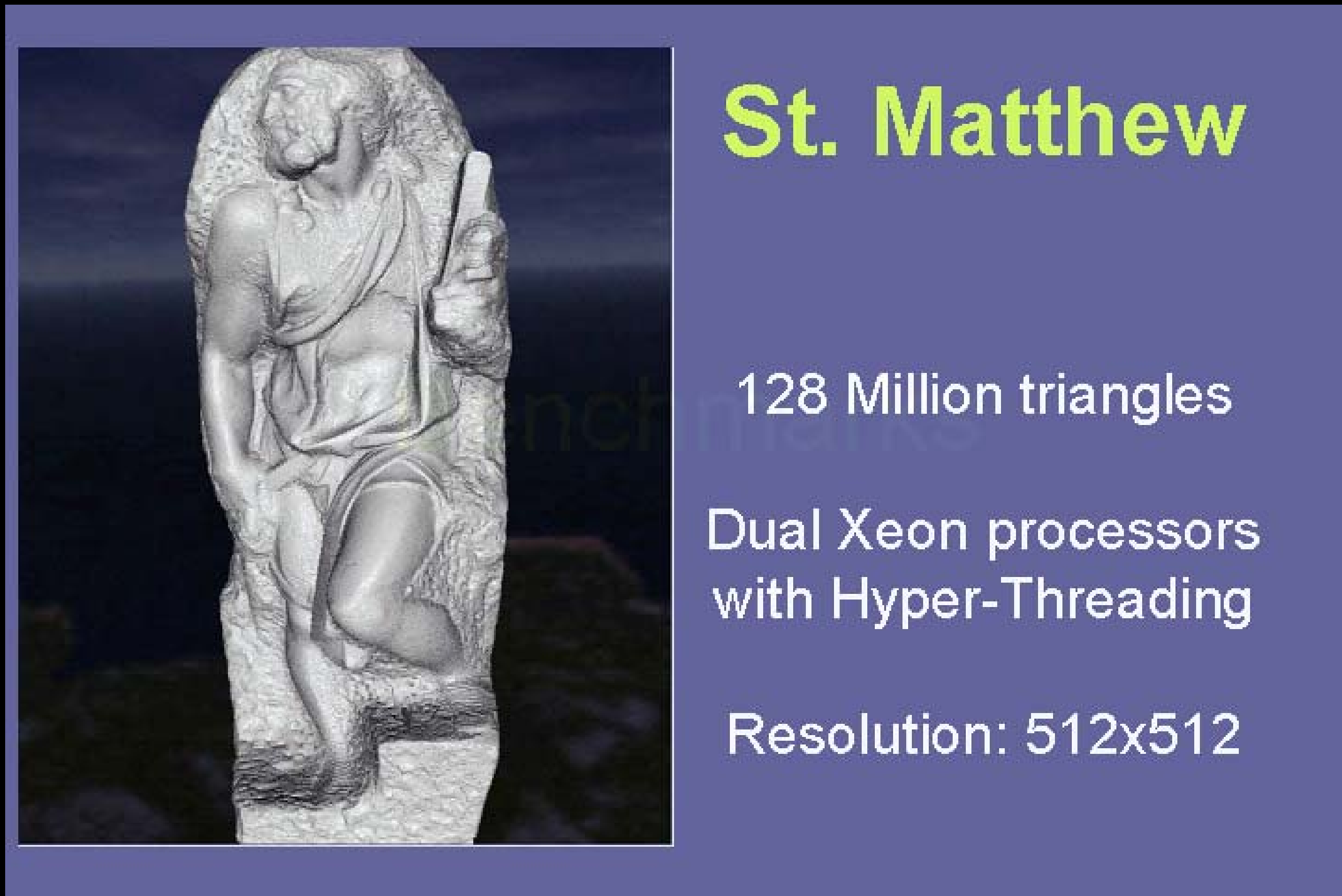
# Ray Tracing: Performance

**Achieved up to three order of magnitude speedup!**



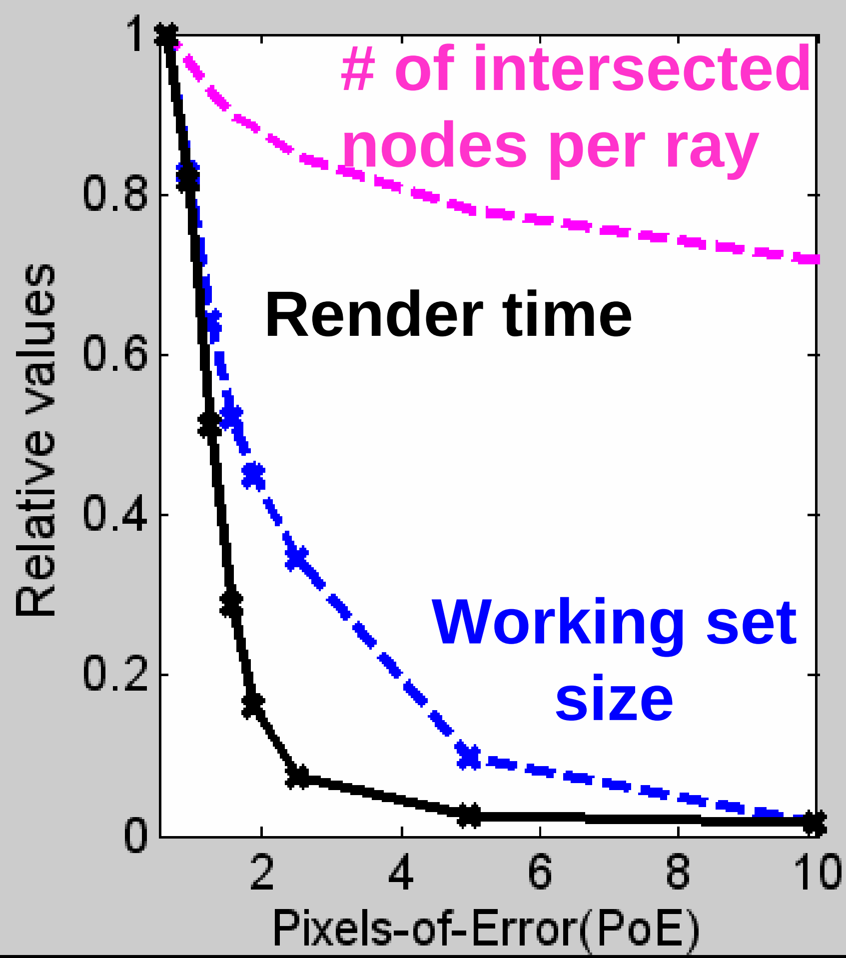

**Model complexity (M tri) - log scale**

# Real-time Captured Video – St. Matthew Model

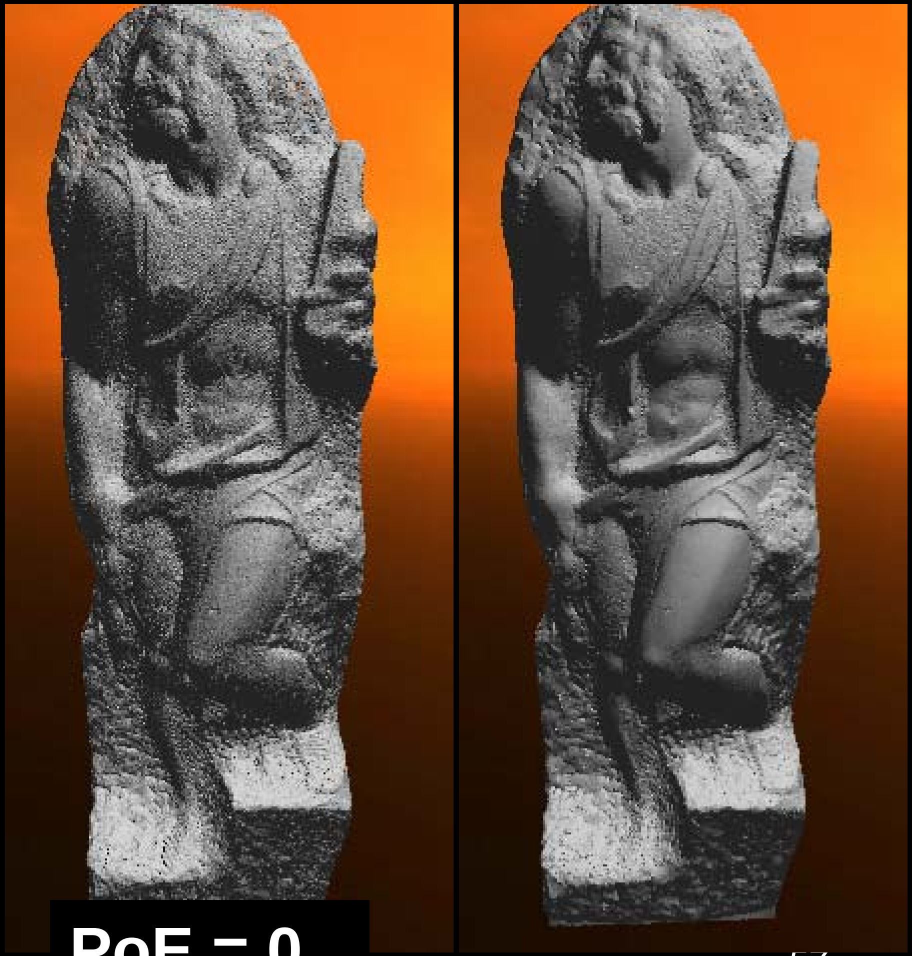



**512 by 512 and 2x2 super-sampling, 4 pixels-of-error**

# Impacts of R-LODs




10X speedup

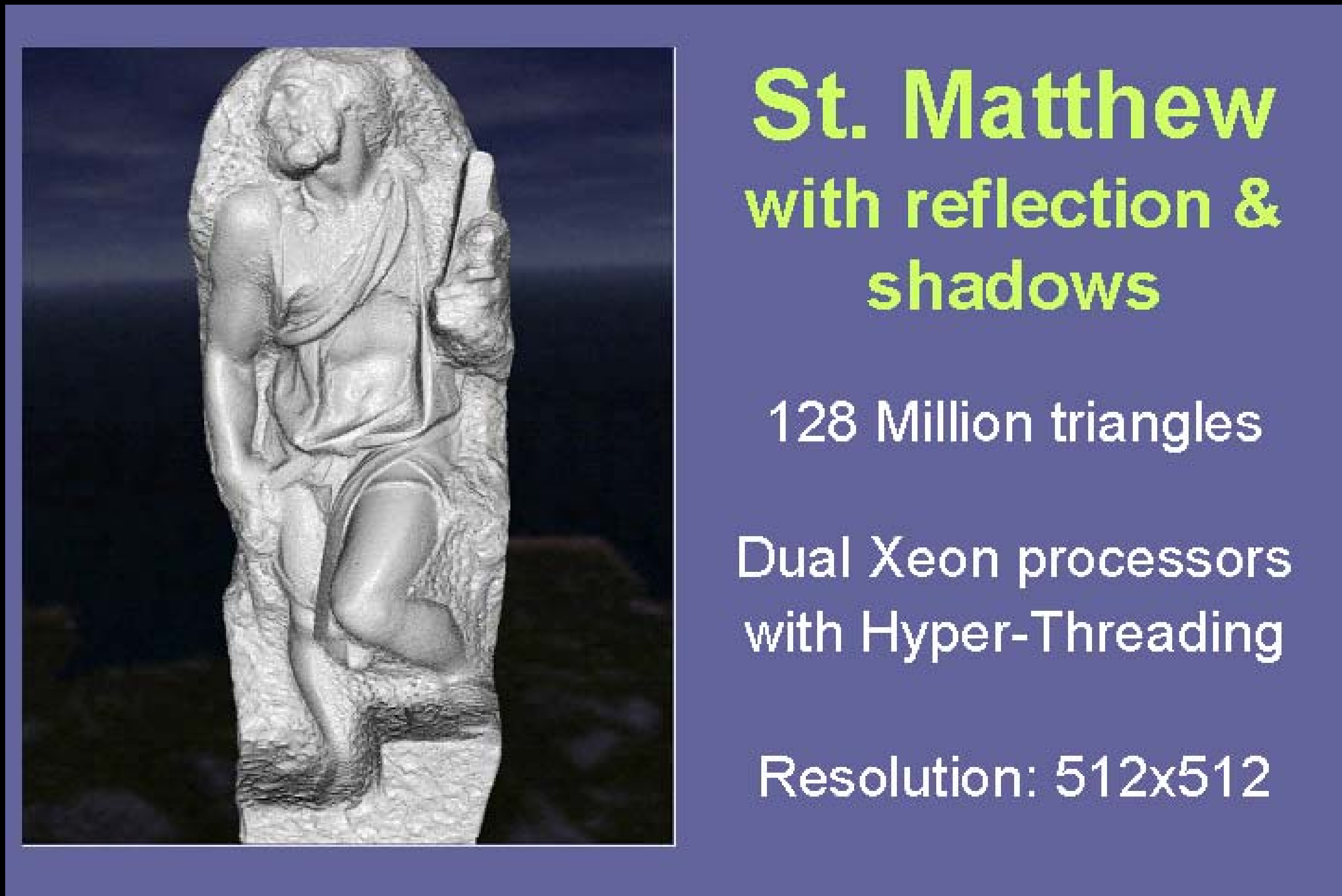


PoE = 0 (No LOD)

PoE = 2.5

# Real-time Captured Video – St. Matthew Model

512 x 512, 2 x 2 anti-aliasing, PoE = 4



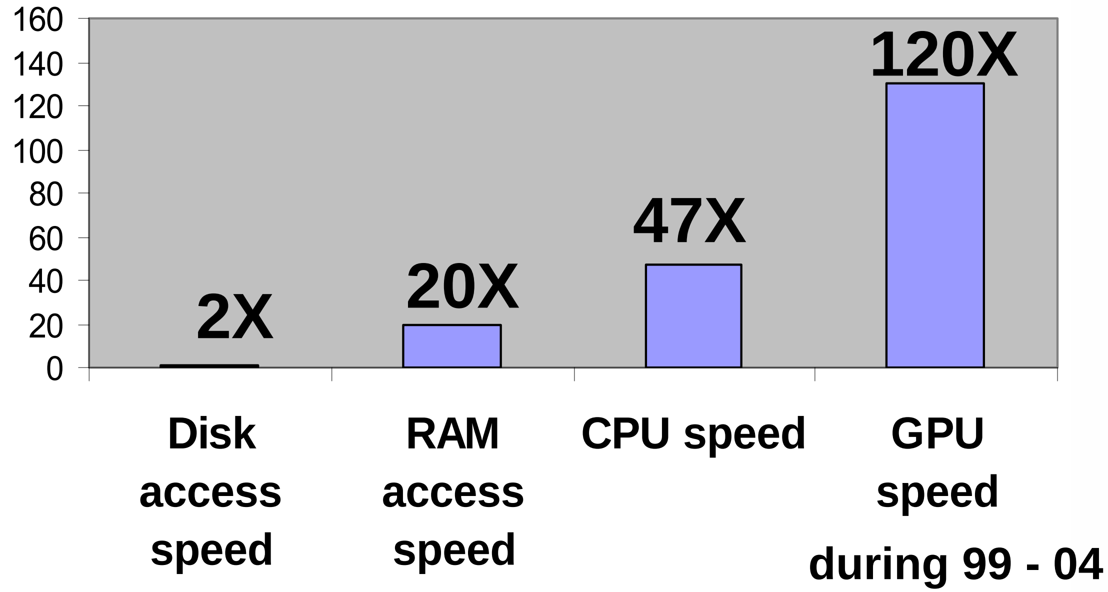
# Outline

- Dynamic simplification for rasterization
- LOD-based ray tracing
- Cache-coherent layouts
- Conclusion

# Motivation

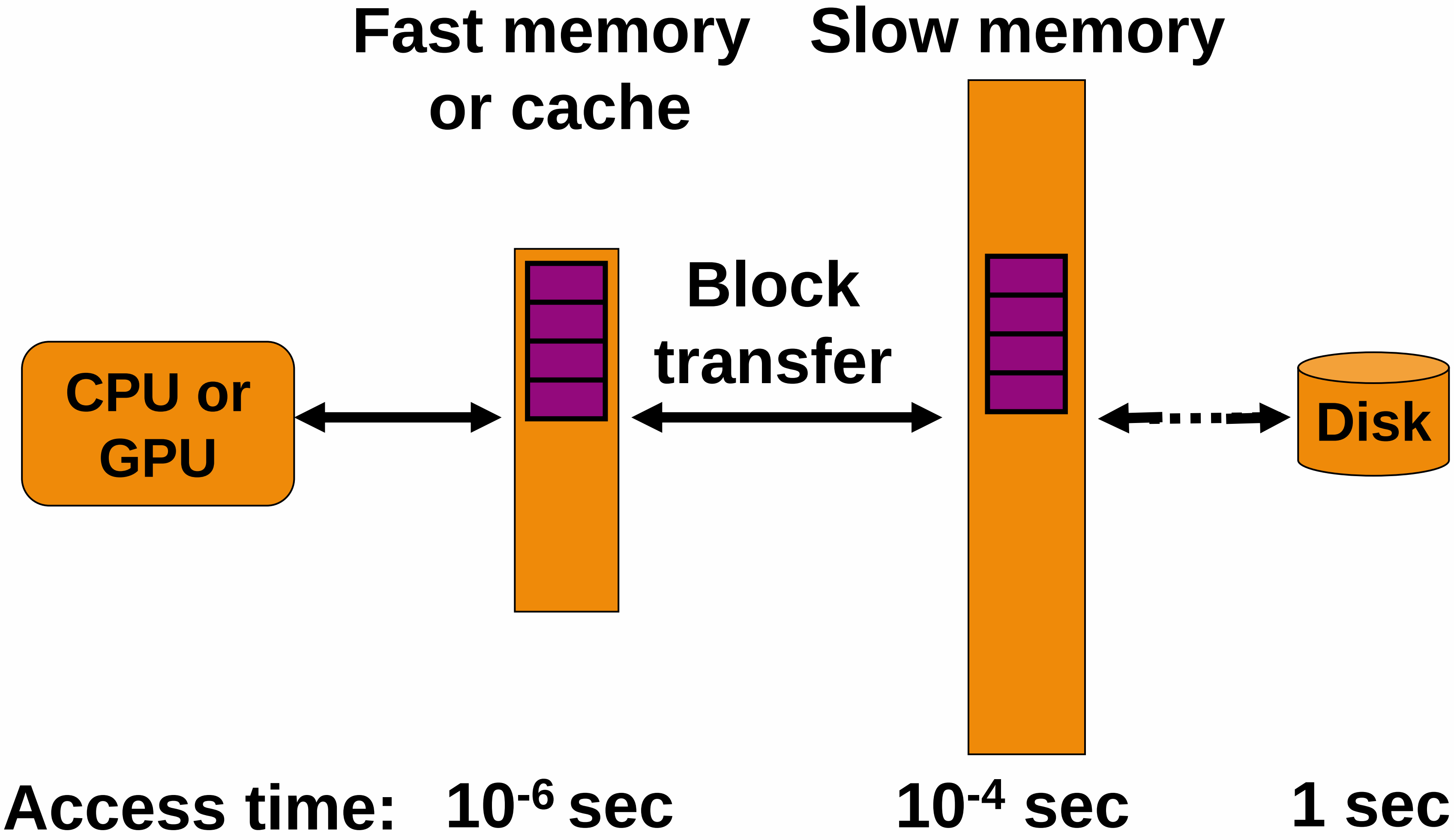
- **Lower growth rate of memory access time**

Growth rate during 1993 - 2004




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

# Block-based I/O Model [Aggarwal and Vitter 88]



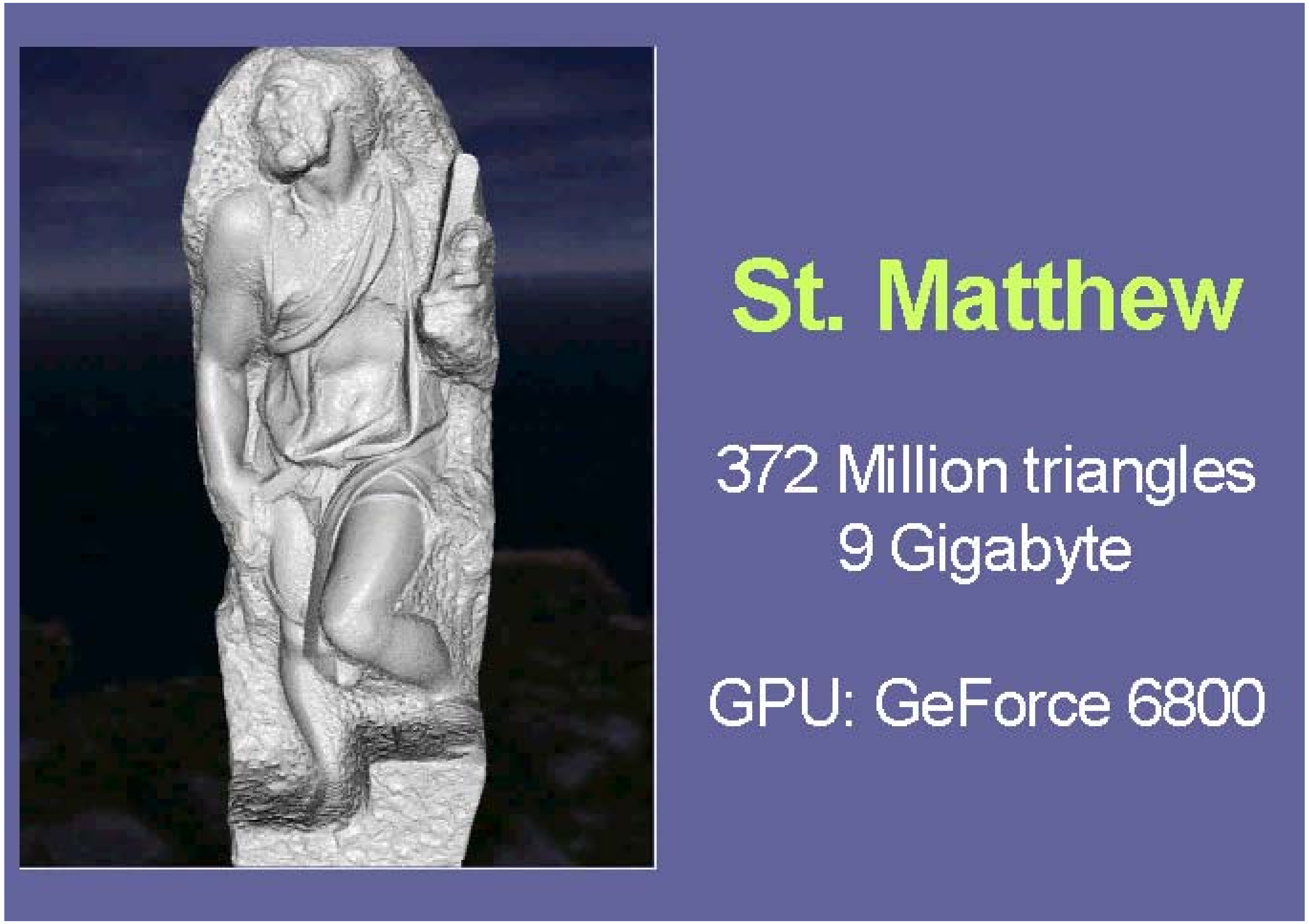
# Cache-Coherent Layouts

- **Cache-aware layouts**
  - ♦ Optimized for particular cache parameters (e.g., block size)
- **Cache-oblivious layouts**
  - ♦ Minimize data access time without any knowledge of cache parameters
  - ♦ Even work with various hardware and memory hierarchies

# Our Approaches

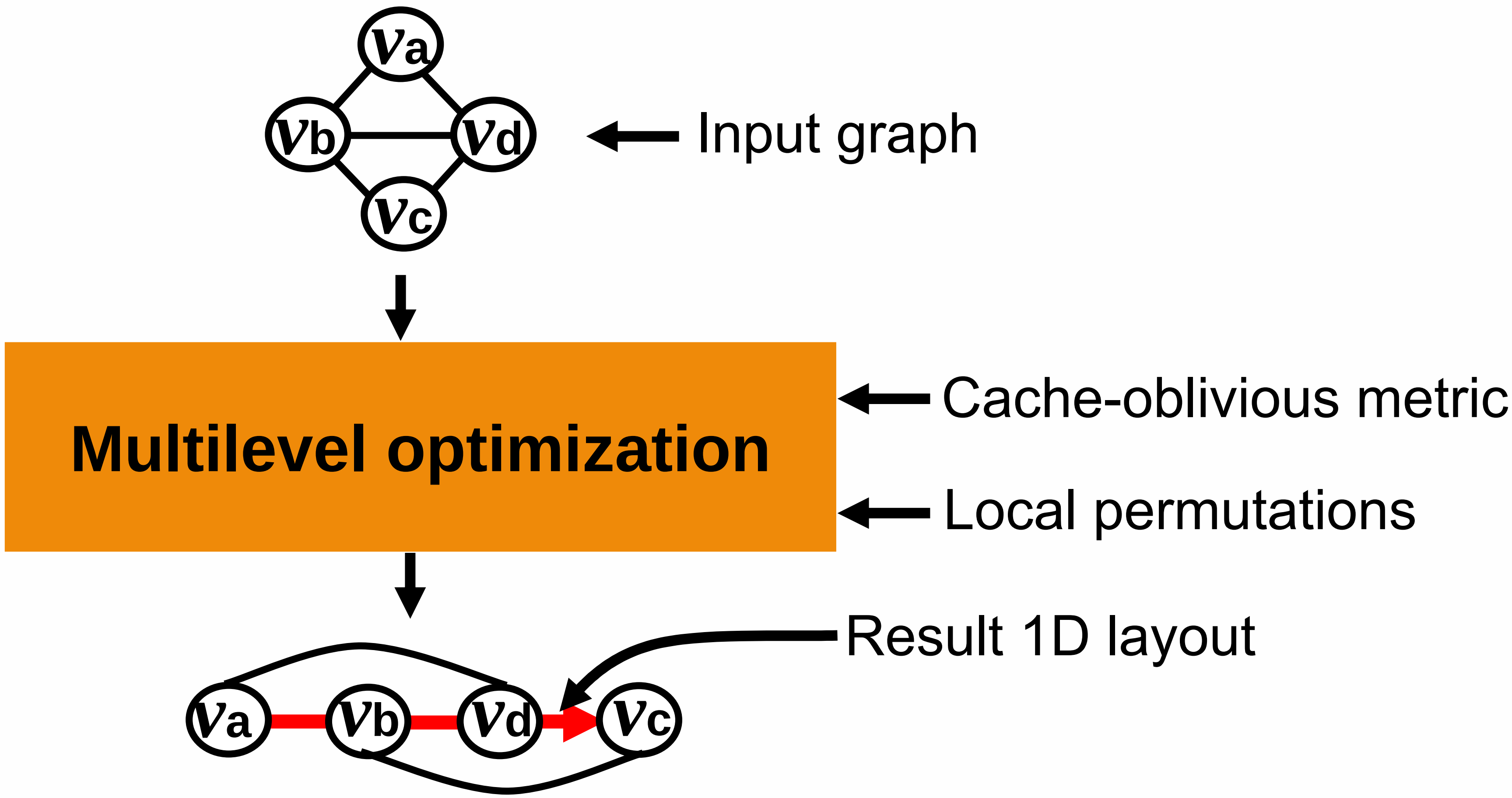
- **Algorithms to compute cache-aware and cache-oblivious layouts [Yoon et al., SIG 05, Yoon and Lindstrom, Vis 06]**
  - ♦ **Cache-aware and cache-oblivious metrics**
  - ♦ **Multi-level optimization framework**
  - ♦ **Specialization for bounding volume hierarchies [Yoon and Dinesh, Euro 06]**

# Realtime Captured Video – Rendering Throughput of Dynamic Simplification




GeForce
6800

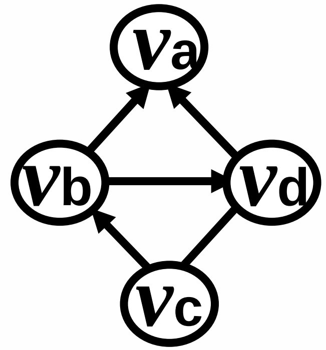
# Overview



Va
Vb
Vd
Vc
Input graph
Multilevel optimization
Cache-oblivious metric
Local permutations
Result 1D layout
Va
Vb
Vd
Vc

# Graph-based Representation

- Directed graph, G = (V, E)
  - ♦ Represents access patterns of applications
- Vertex
  - ♦ Data element
  - ♦ (e.g., mesh vertex or mesh triangle)

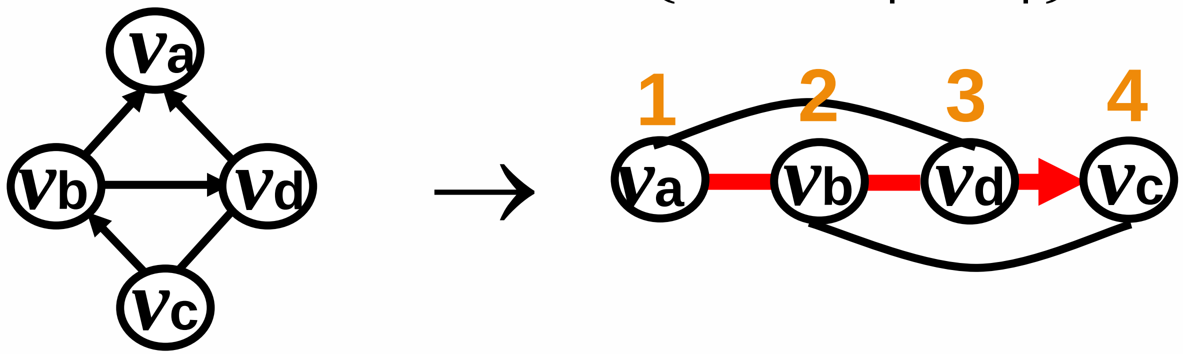



- Edge
  - ♦ Connects two vertices if they are likely to be accessed sequentially

# Problem Statement

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

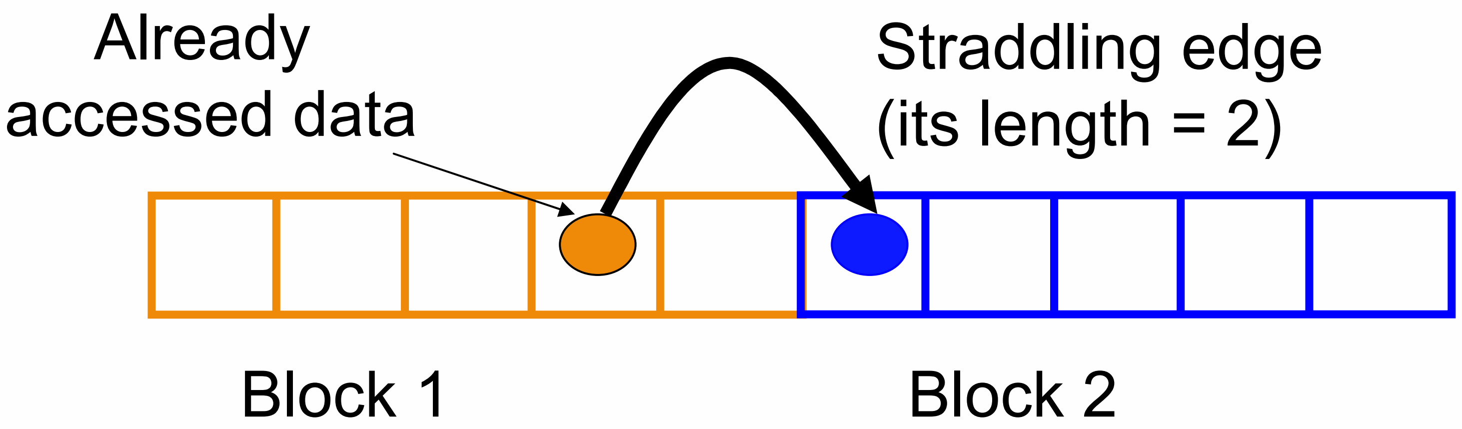
$$\varphi : V \rightarrow \{1, \dots, |V|\}$$




- **Compute a $\varphi$ that minimizes the expected number of cache misses**

# Cache-Aware Metric, One Cache Block

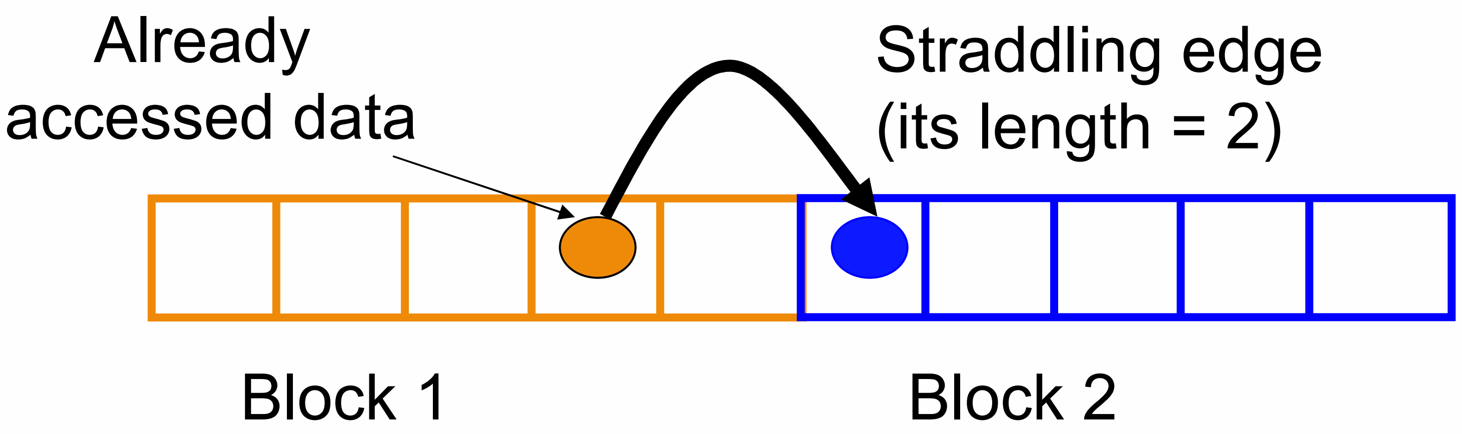
- **Cache misses when a cache holds only one block**




- **Layout computation**
  - ♦ **Minimize the number of straddling edges**
  - ♦ **Graph partitioning**

# Cache-Aware Metric, Multiple Cache Blocks

- **What if a cache can hold multiple blocks?**




- **Approximated with cache-aware metric of a single cache block**
  - ♦ **Has strong correlation**

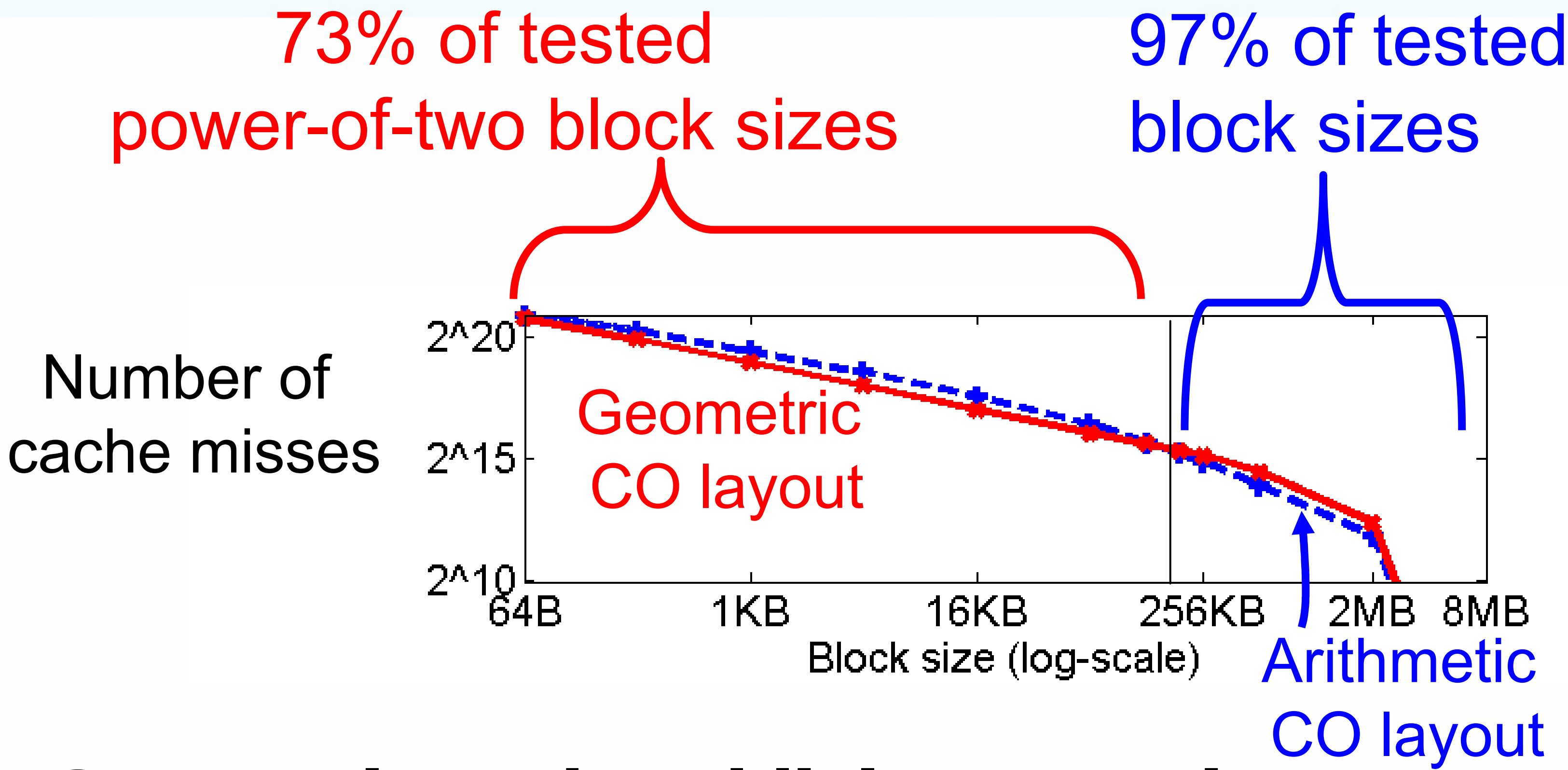
# Cache-Oblivious Metrics

- **Assuming arithmetic block sizes (e.g., 1, 2, 3, ..)**
  - **Mean of edge lengths, $\Sigma|x - y|$**
  - **Arithmetic mean**

x and y are indices of two vertices of an edge in the layout

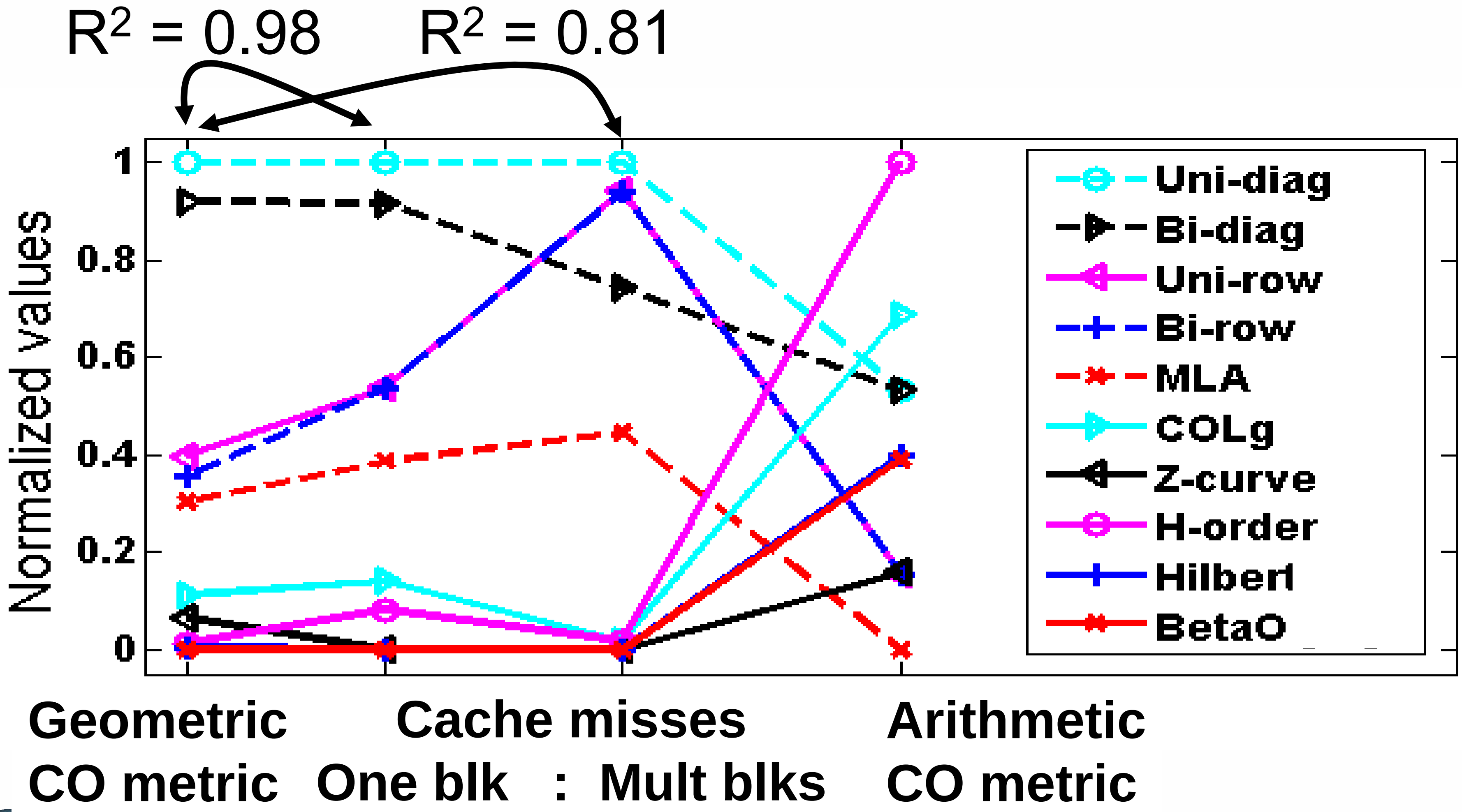
- **Assuming geometric block sizes (e.g., 1, 2, 4, 8, ..)**
  - **Mean of log of edge lengths, $\Sigma \log |x - y|$**
  - **Geometric mean**

# Validation for Cache-Oblivious (CO) Metrics




- **Geometric cache-oblivious metric**
  - ♦ **Practical and useful**

# Correlations with Observed Number of Cache Misses



# Layout Optimization

- **Find an optimal layout that minimizes our metric**
  - ♦ **Combinatorial optimization problem [Diaz et al. 2002]**

- **Employ multi-level construction method**
  - ♦ **Construct layouts that consider geometrically increasing blocks sizes**
  - ♦ **A good heuristic for geometric cache-oblivious metric**

# Applications

- View-dependent rendering
- Collision detection
- Ray tracing
- Isocontour extraction

# View-Dependent Rendering

- **Layout vertices and triangles of CHPM**
  - ♦ **Reduce misses in GPU vertex cache**

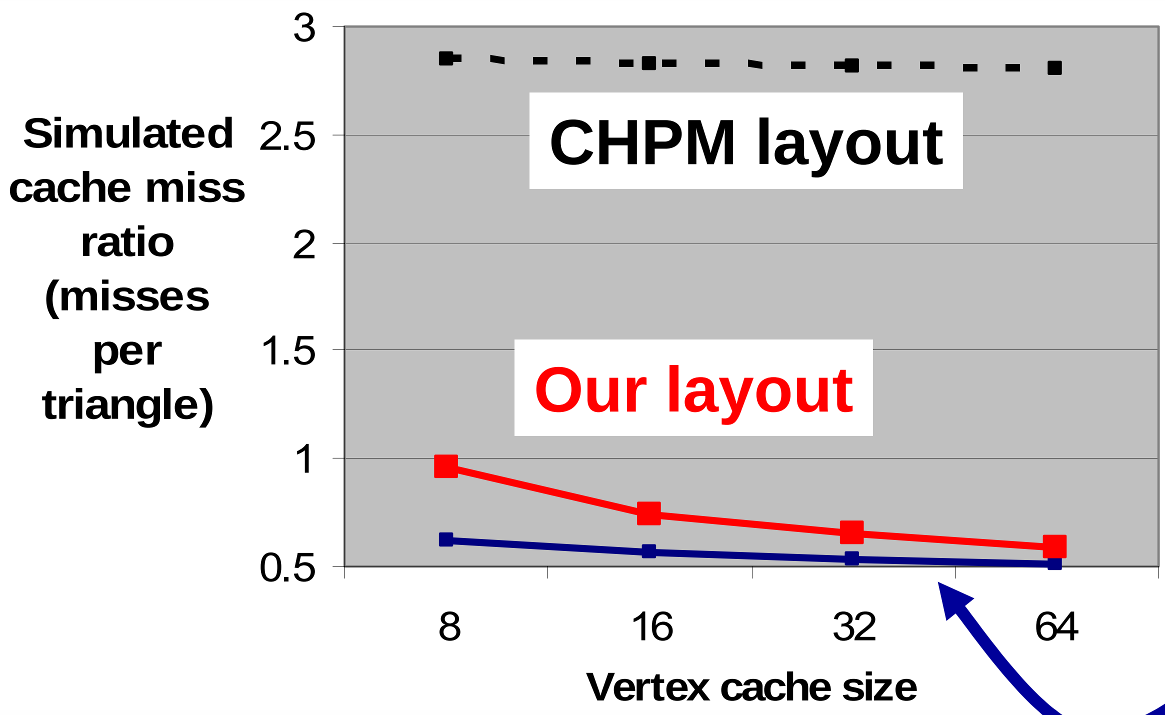
**Peak performance: 145 M tri / s on GeForce 6800 Ultra**

| Models | # of Tri. | Our layout | CHPM layout |
|---|---|---|---|
| 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

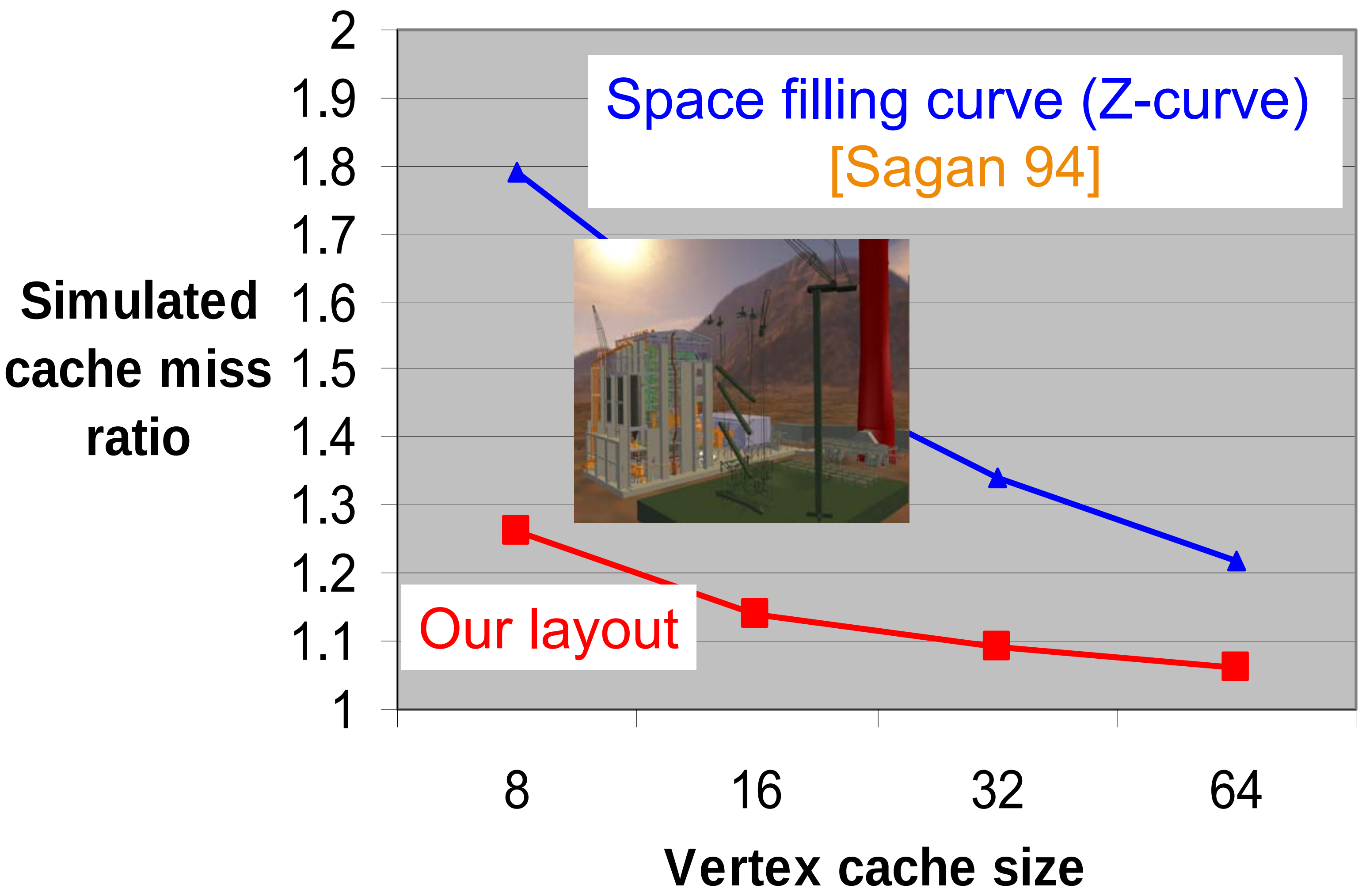
# Comparison with Optimal Cache Miss Ratio




**Lower bound of optimal cache miss ratio [Bar-Yehuda and Gotsman 96]**

**Test model: Bunny model**

# Comparison with Space Filling Curve on Power Plant Model



# Collision Detection and Ray Tracing

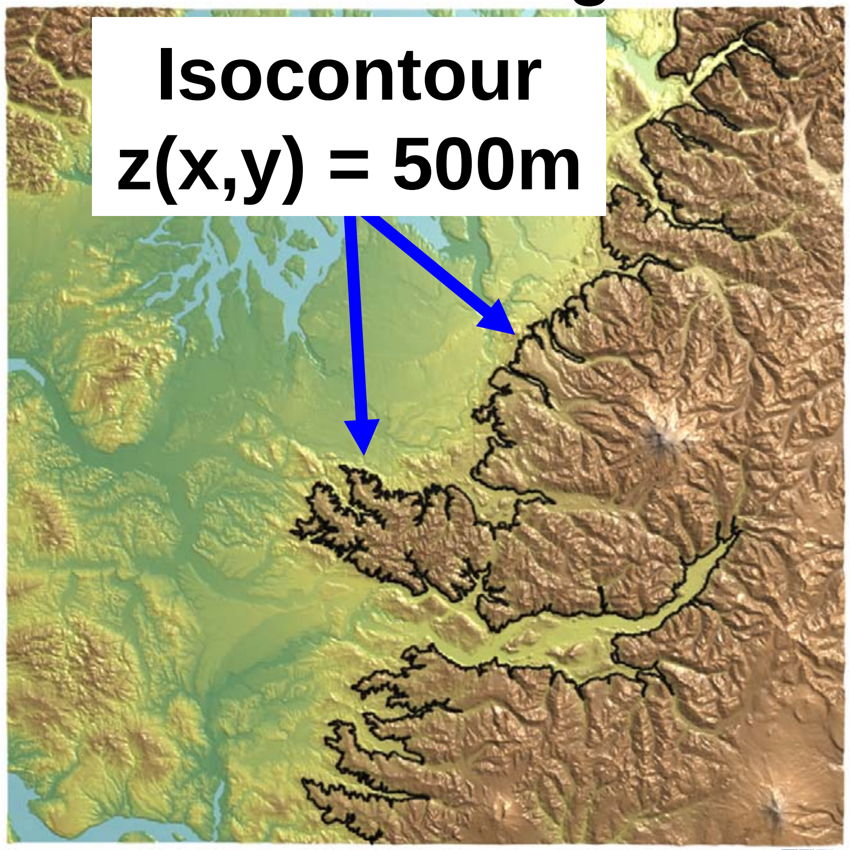
- **Bounding volume hierarchies [Yoon and Manocha Euro 06]**
  - ♦ **Consider geometric relationship to capture runtime access patterns**
  - ♦ **Achieve 30% ~ 300% performance improvement**


Dynamic simulation

# Isocontour Extraction

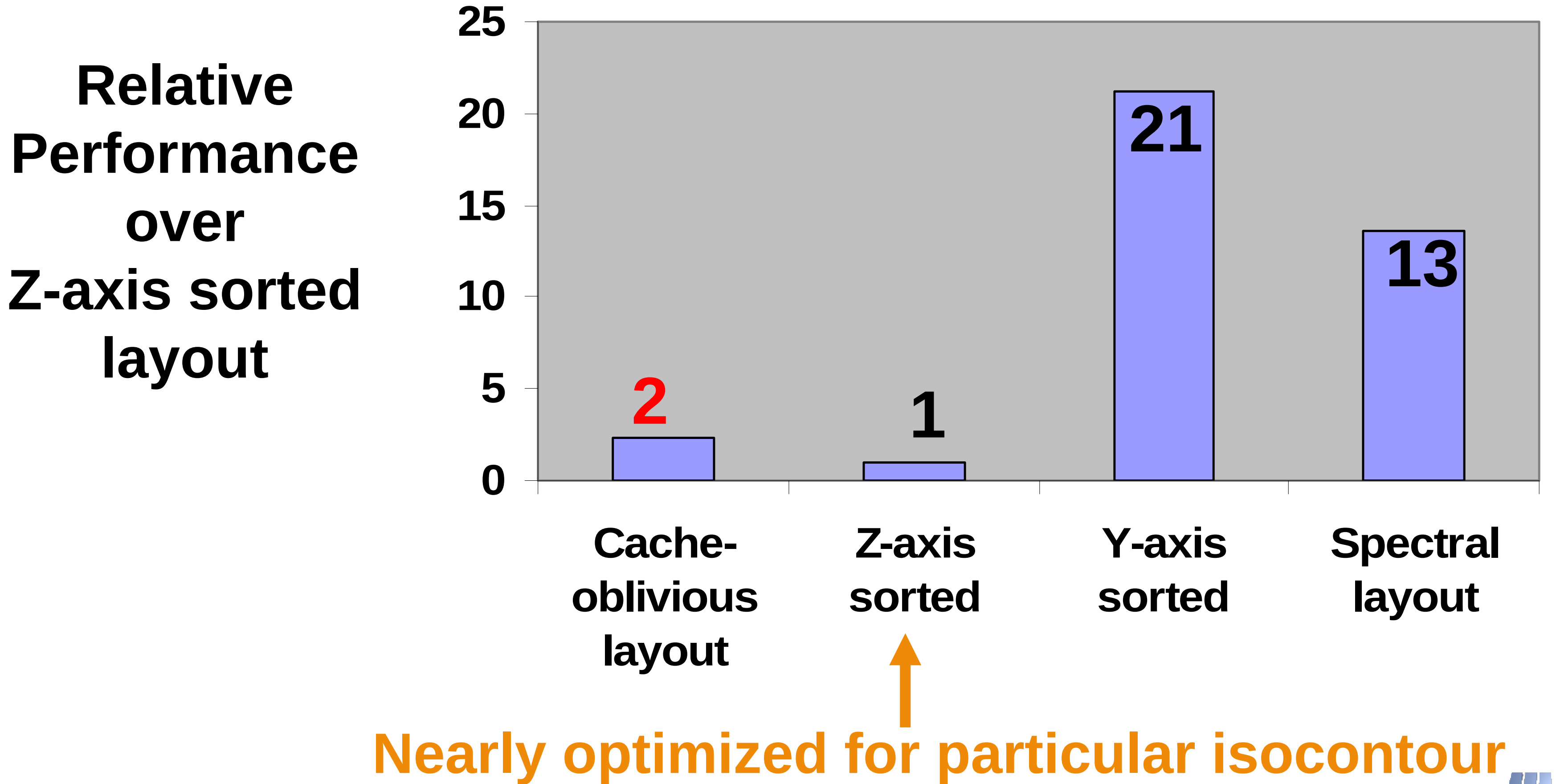
- Uses contour tree [van Kreveld et al. 97]
- Use mesh as the input graph
- Extract an isocontour that is orthogonal to z-axis

Puget sound,
134 M triangles

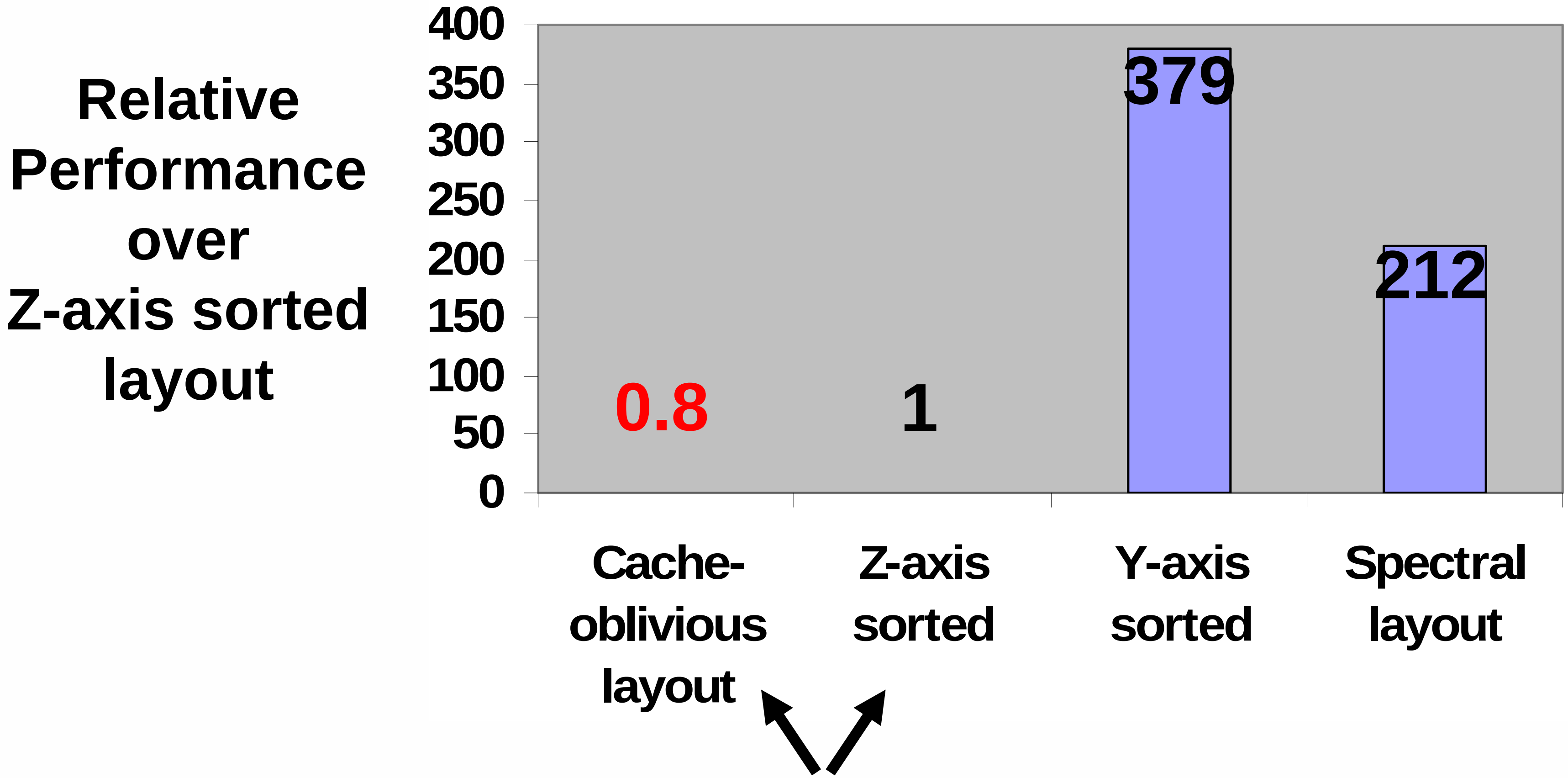


# Comparison – First Extraction of Z(x,y) = 500m

Disk access time is bottleneck



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




Memory and L1/L2 cache access times are bottleneck

# Advantages

- **General**
  - ♦ Applicable to all kinds of polygonal models
  - ♦ Works well for various applications
-

**Source codes are available**
**as a library called**
**OpenCCL**

- **No modification of runtime applications**
  - ♦ Only layout computation

# Conclusion

- **Huge amount (giga-bytes) of data**
  - ♦ **Limited L1/L2 cache and memory sizes**

- **Data access time**
  - ♦ **Major bottleneck**

# Conclusion

- **Orthogonal approaches**
  - ♦ **Levels-of-detail (LOD) techniques**
  - ♦ **Cache-coherent layouts**

- **Applications**
  - ♦ **Visualization and geometric processing**

- **Achieved interactive performance**

# Questions?

# UCRL-PRES-223537

This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-ENG-48.

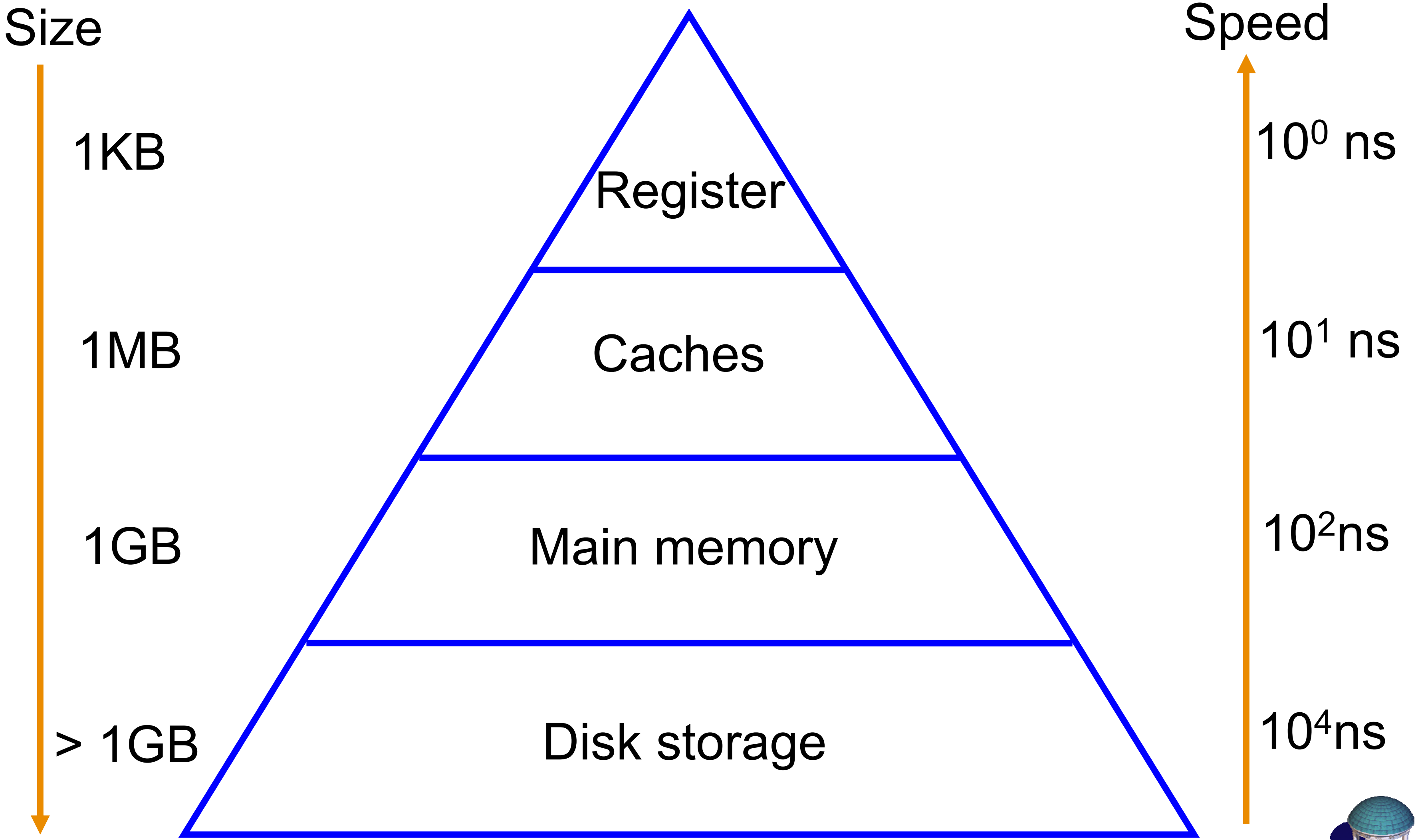
# Additional slides

# Main Requirements

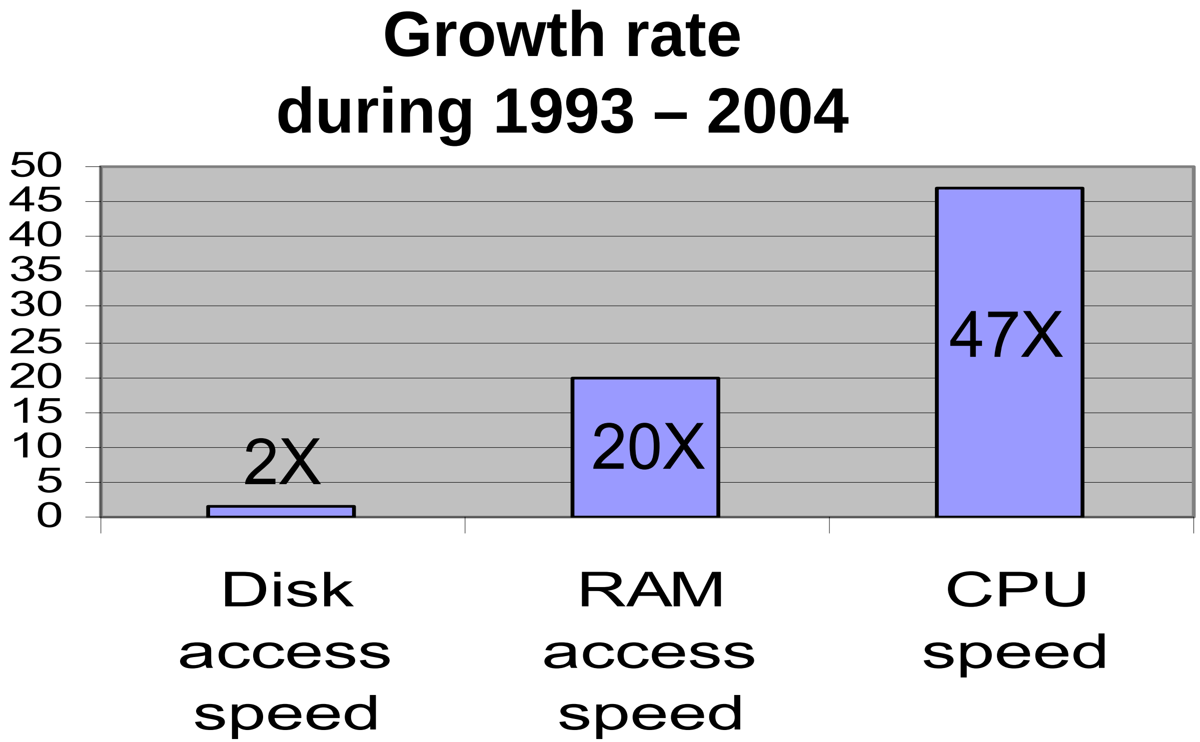
- **Generality**
  - ♦ **Handle any kind of polygonal models**
  - ♦ **(e.g., CAD, scanned, isosurface models)**

- **Interactivity**
  - ♦ **Provide at least 10 frames per second**

# Memory Hierarchies



# Low Growth Rate of Memory Bandwidth




**Recent hardware improvements may**

**not provide an efficient solution to our problem!**

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

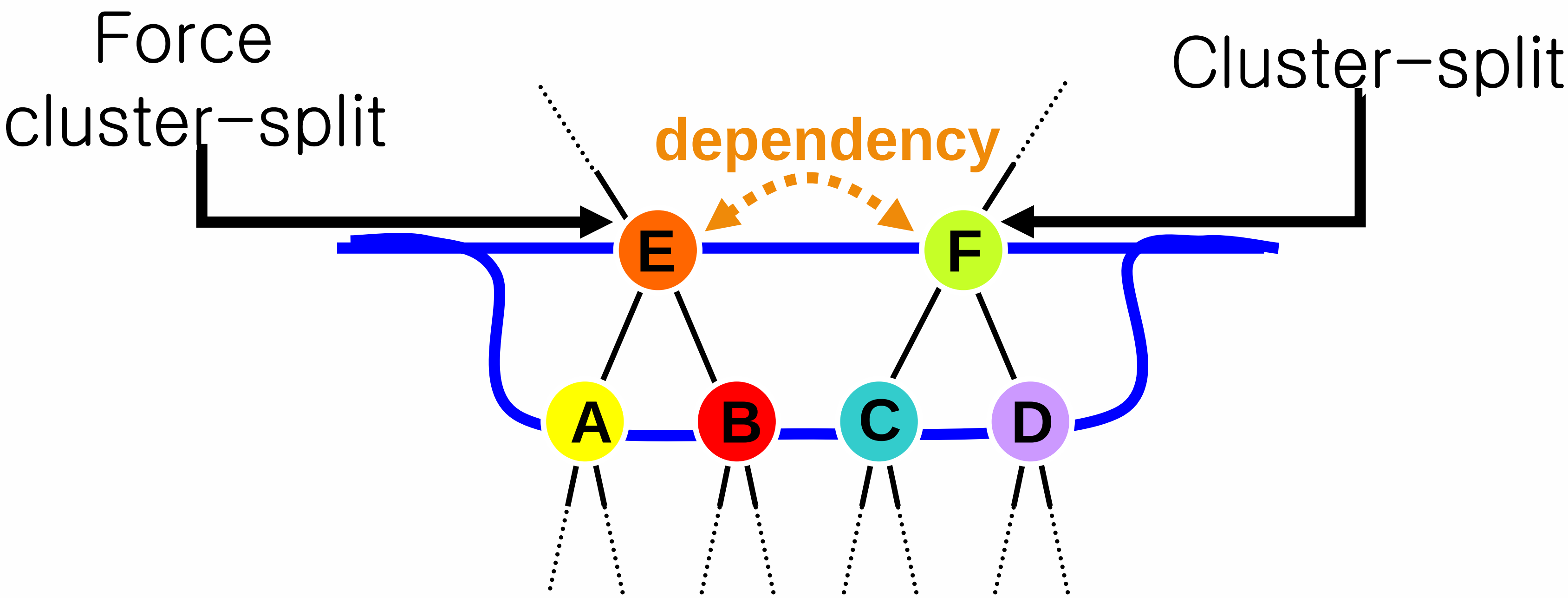
# Ongoing and Future Work

- **What is an optimal cluster size?**
  - **Performance depends on computed clusters [Yoon and Manocha EG 06]**
- **How can we efficiently deal with dynamic models?**
  - **Require efficient data structure updates and rebuilding [Lauterbach et al. IEEE RT 06]**

# Summary

- **Dynamic simplification representation (CHPM)**
  - **Low refinement time**
- **Out-of-core construction method**
- **Tested with different applications**

# Cluster Dependencies at Runtime




Cluster hierarchy

# Approximate Collision Detection

- **Uses dynamic simplification**
  - ♦ **CHPM representation**

- **Conservative error metric**
  - ♦ **Approximate collision results introduces only epilson distance error**

- **Two lemmas**
  - ♦ **Guarantees that our runtime LOD selection method satisfies the metric**

- **Employ GPU-based collision detection**

# Image Quality Comparison – Forest Model (32M Triangles)

**4 X speedup**

PoE = 0 (No LOD)

PoE = 4 and cache-oblivious layout of kd-tree

Shading difference

# Ongoing and Future Work

- **Investigate dynamic simplification to improve visual quality**
- **Extend to global illumination**

# Acknowledgements

- **Model contributors**
- **Funding agencies**
  - ♦ Army Research Office
  - ♦ Defense Advance Research Projects Agency
  - ♦ Ilju foundation
  - ♦ Intel company
  - ♦ Lawrence Livermore National Laboratory
  - ♦ National Science Foundation
  - ♦ Office of Naval Research

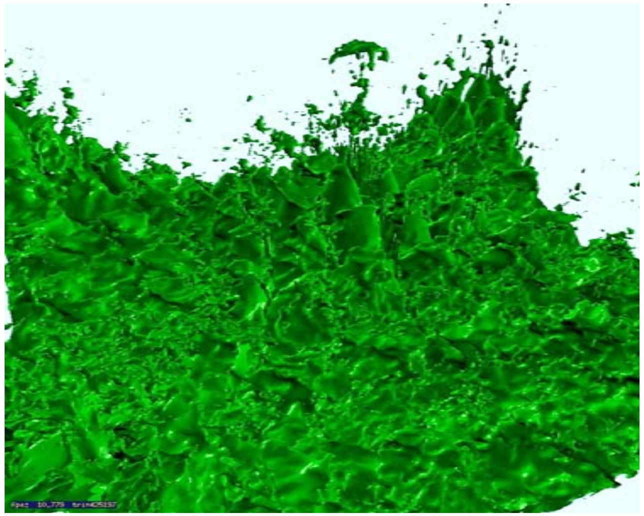
# New Results

- **Dynamic simplification method**
  - ♦ **CHPM representation**
  - ♦ **Out-of-core construction method**
  - ♦ **Application to collision detection**
- **Cache-oblivious layout algorithm**
  - ♦ **Cache-oblivious metric**
  - ♦ **Multilevel minimization**

# Future Work on Visualization

- **Achieve end-to-end interactivity**
  - ♦ **Requires no or minimal preprocessing**

- **Handle time-varying geometry**

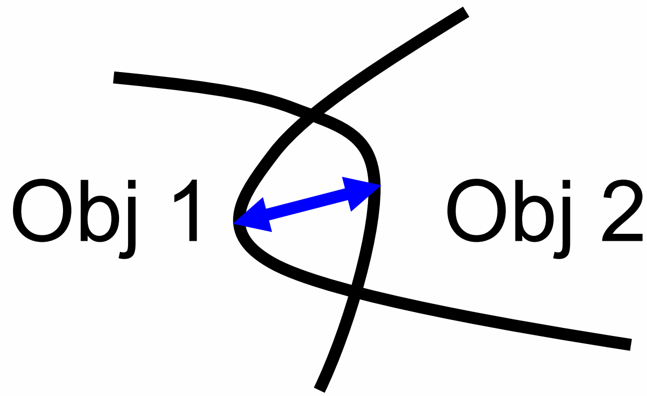


Just one instance
among 27K time steps
during simulation

# Future Work on Collision Detection

- **Handle dynamically deformable models (e.g. cloth simulation)**
  - **Requires no or minimal preprocessing**

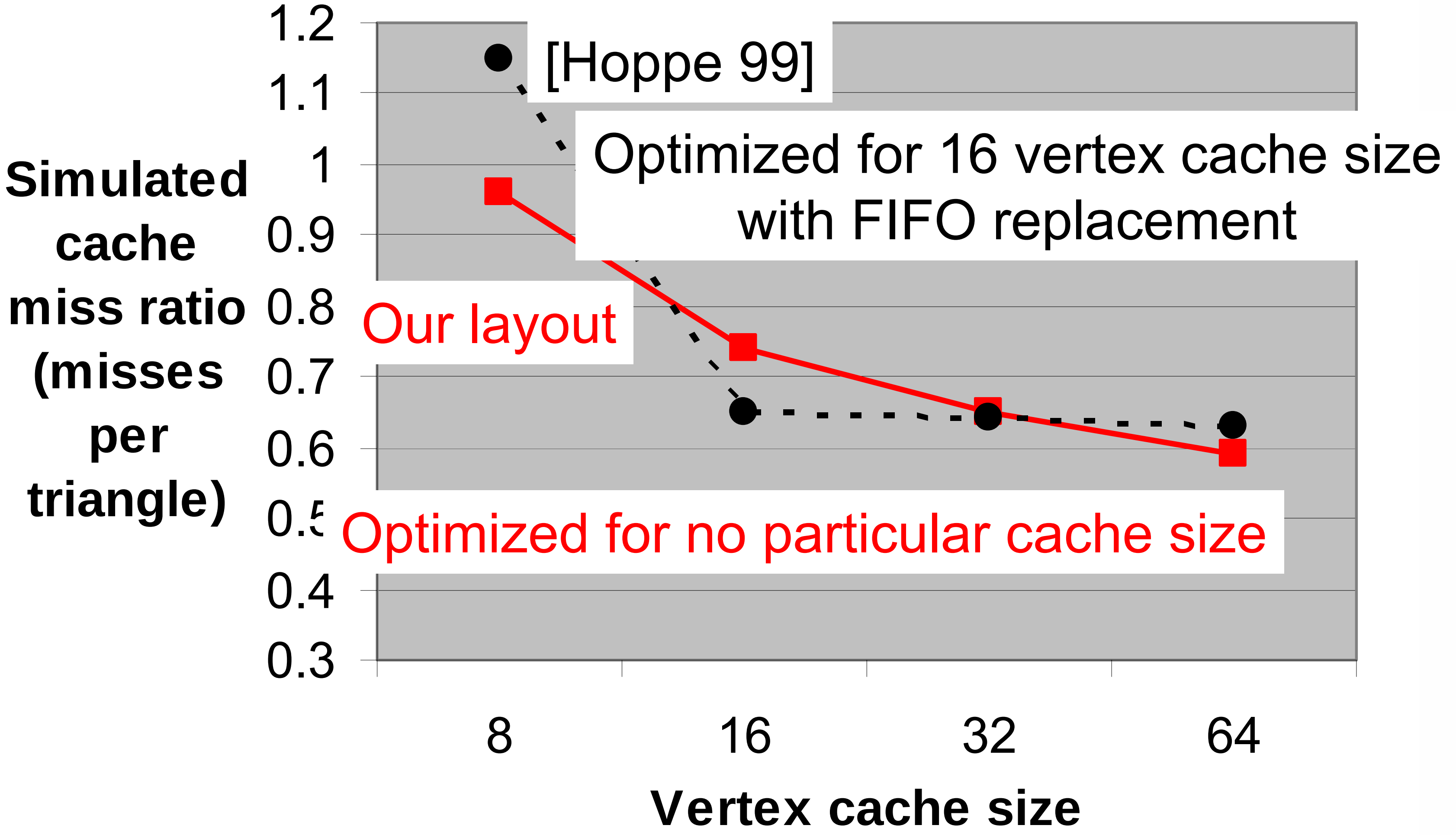
- **Support penetration depth computations**



# Future Work on Cache-Coherent Layouts

- Develop cache-aware layouts
- Investigate optimality
- Apply to other applications and other representations
  - Shortest path computation, etc.
- Provide multiresolution functionality from layouts
  - [Pascucci and Frank 01]

# Comparison with Hoppe's Rendering Sequence




**Test model: Bunny model**

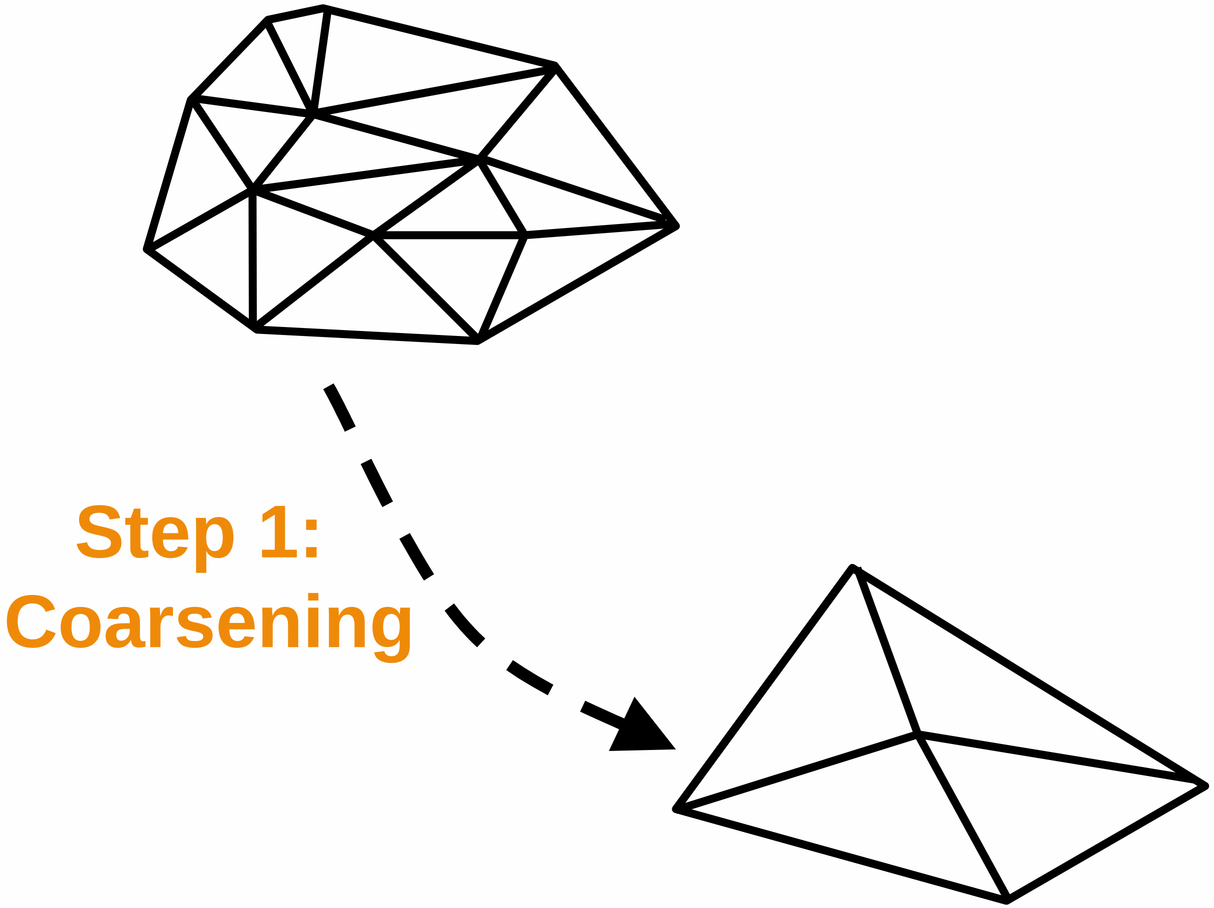
# Limitations

- **Monotonicity assumption**
  - ♦ May not work well for all applications
- **Does not compute global optimum**
  - ♦ Greedy solution

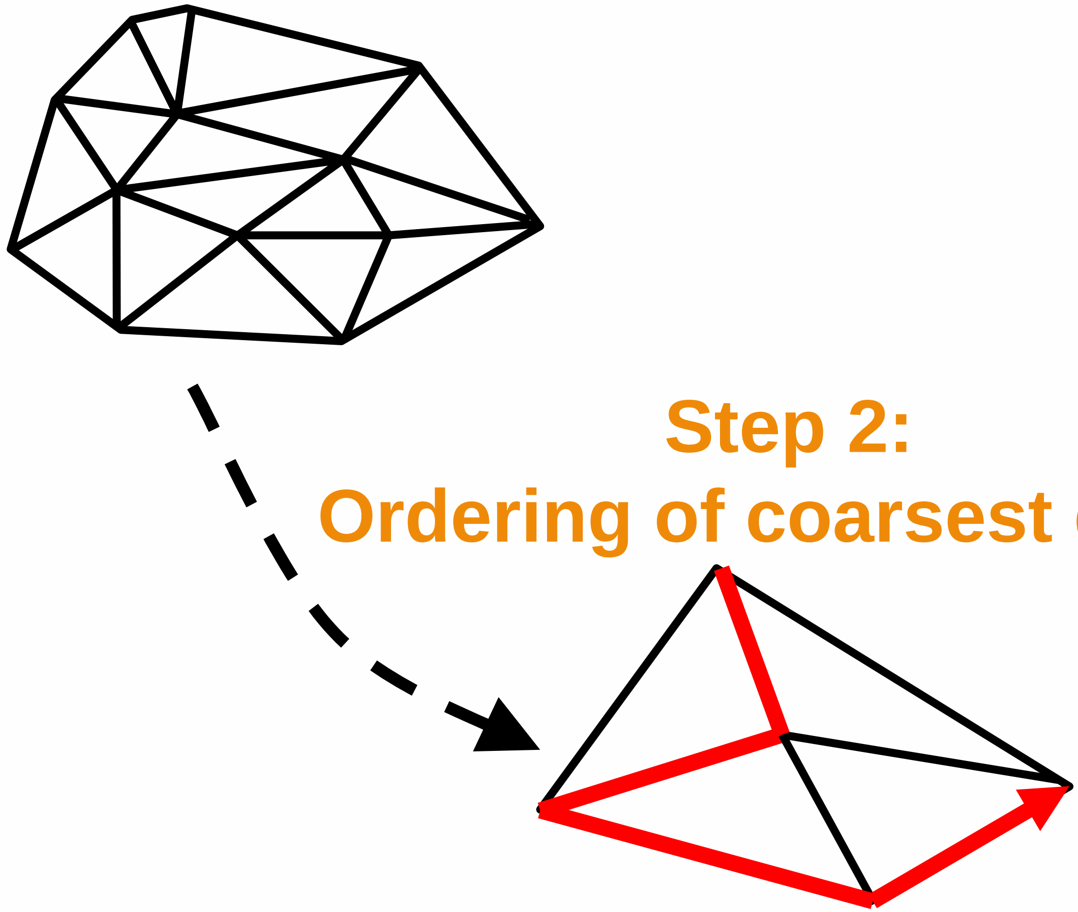
# Conclusion

- LOD techniques and cache-efficient layouts
  - ♦ Applied them to visualization and collision detection
  - ♦ Demonstrated with a wide variety of polygonal models
  - ♦ Achieved interactive performance on commodity hardware

# Multilevel Minimization

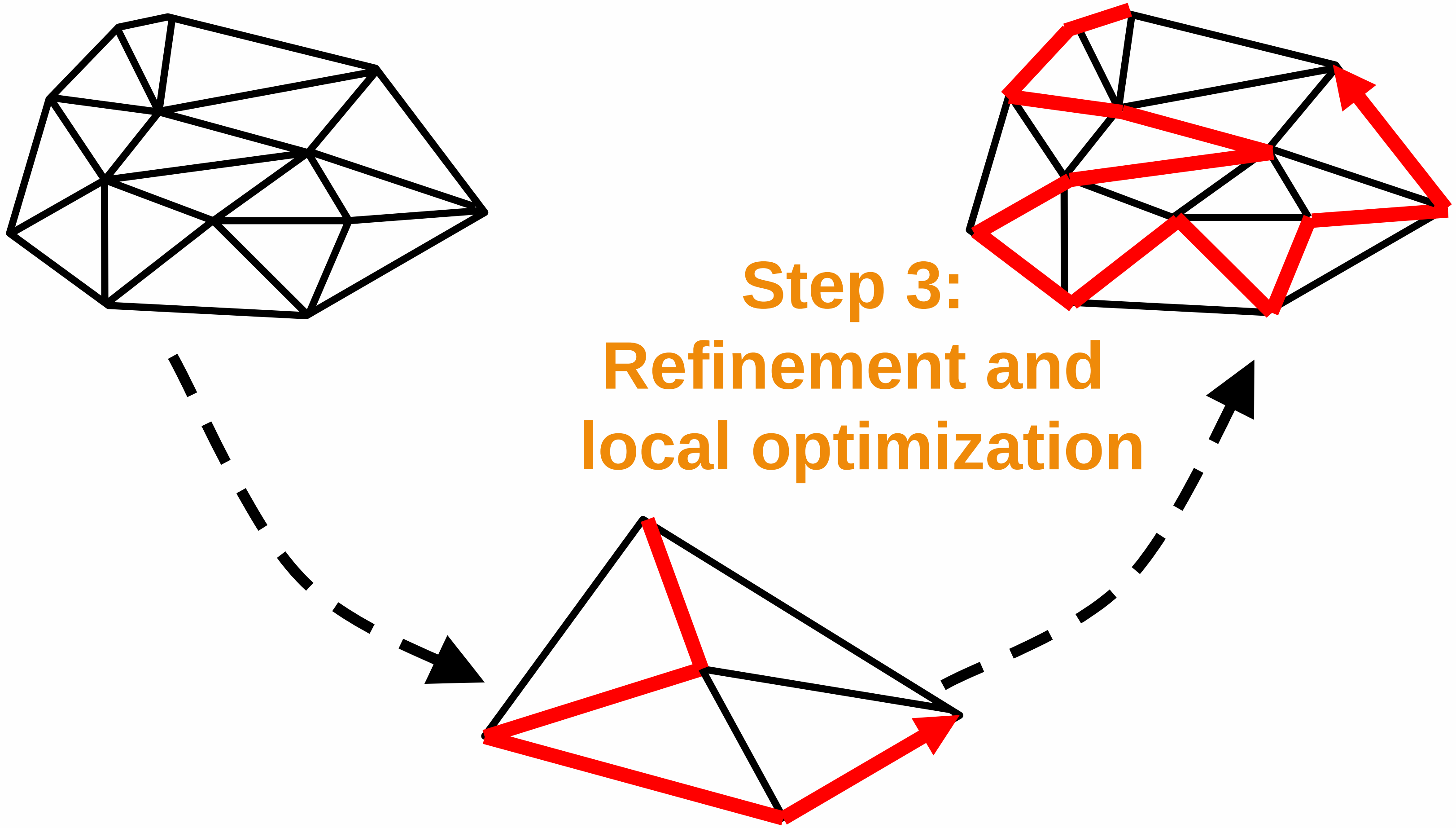


# Multilevel Minimization



Step 2:
Ordering of coarsest graph

# Multilevel Minimization



# Dynamic Simplification: Issues

- **Representation**
  - ♦ **High CPU usages**
- **Runtime computation and rendering**
  - ♦ **Low cache-utilization**
- **Construction**
  - ♦ **Out-of-core computations**

# Dynamic Simplification: Issues

- **Representation**
  - ♦ **High CPU usages**

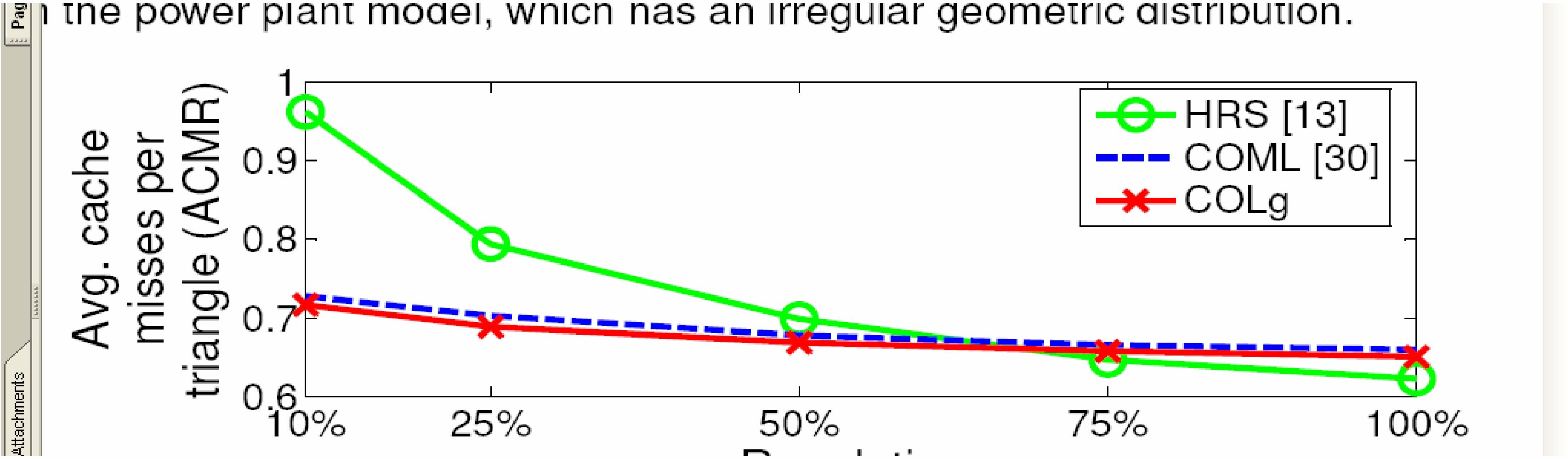
- **Runtime computation and rendering**
  - ♦ **Low cache-utilization**

- **Construction**
  - ♦ **Out-of-core computations**

# Low Computation Speed

- **Rendering throughput**
  - ♦ GPU capable of 100M+ triangles per sec
  - ♦ Only achieved 20M triangles per sec
- **Low cache utilization**
  - ♦ Cannot efficiently use triangle strips for dynamically generated geometry

# Comparison with Hoppe's Rendering Sequence




Highest resolution

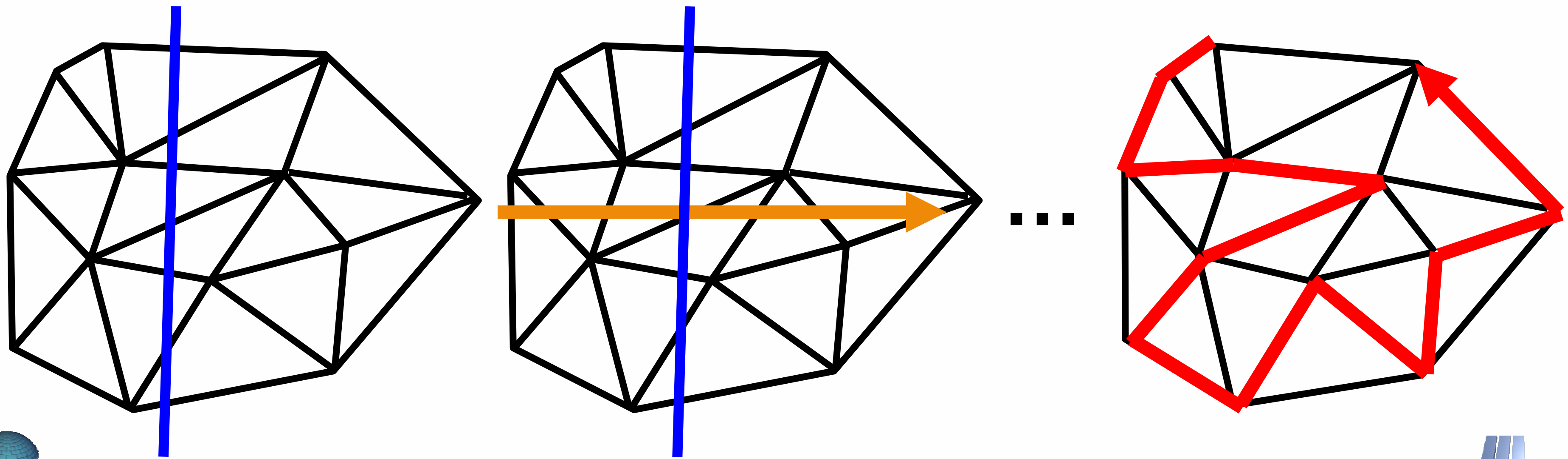


**Test model: Bunny model**

# Multilevel Construction Method

- **Heuristic**
  - ♦ **Optimize a layout for geometrically increasing block sizes**
  - ♦ **Well suited for a multi-level method**

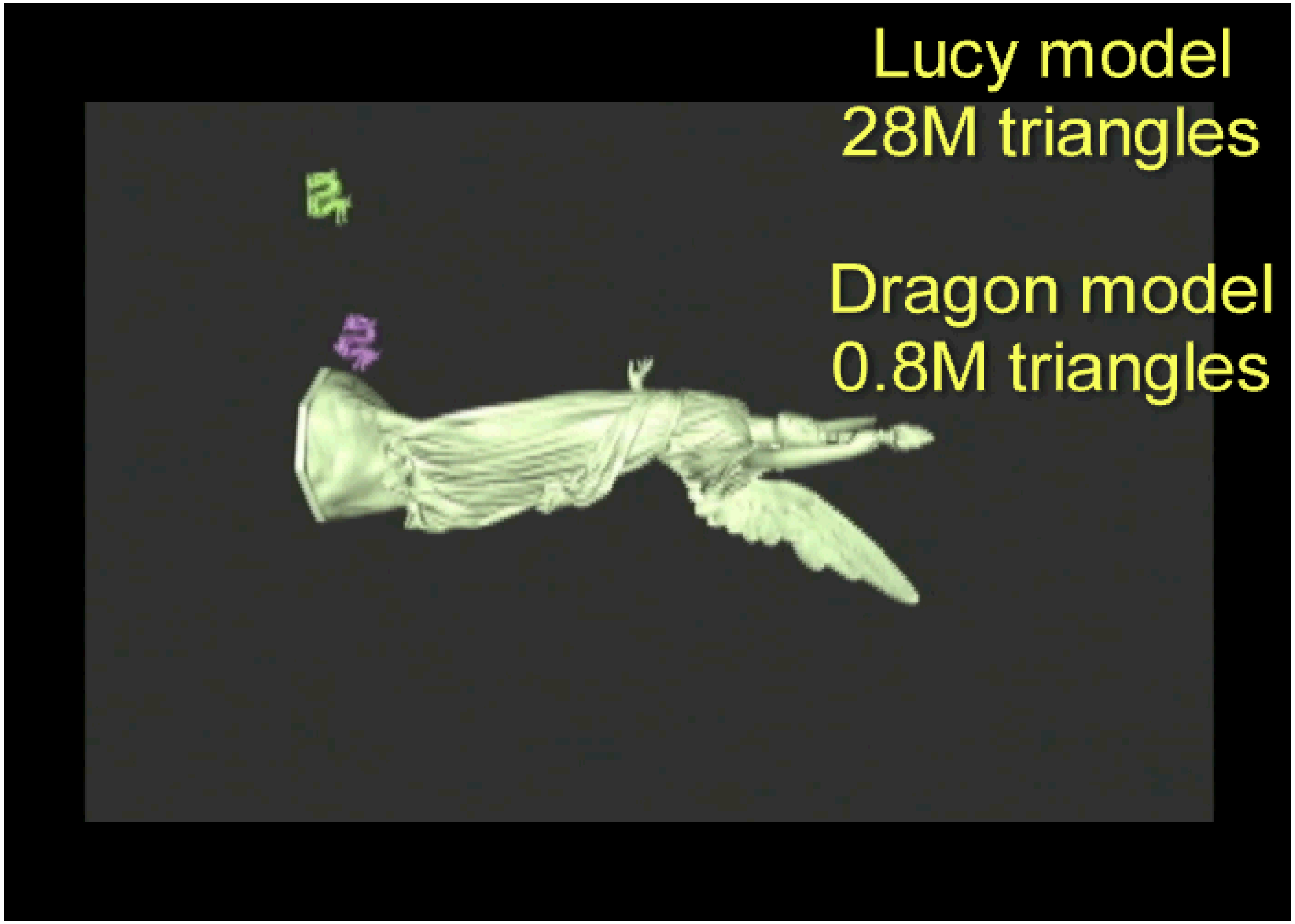
1. Partition    2. Lay out



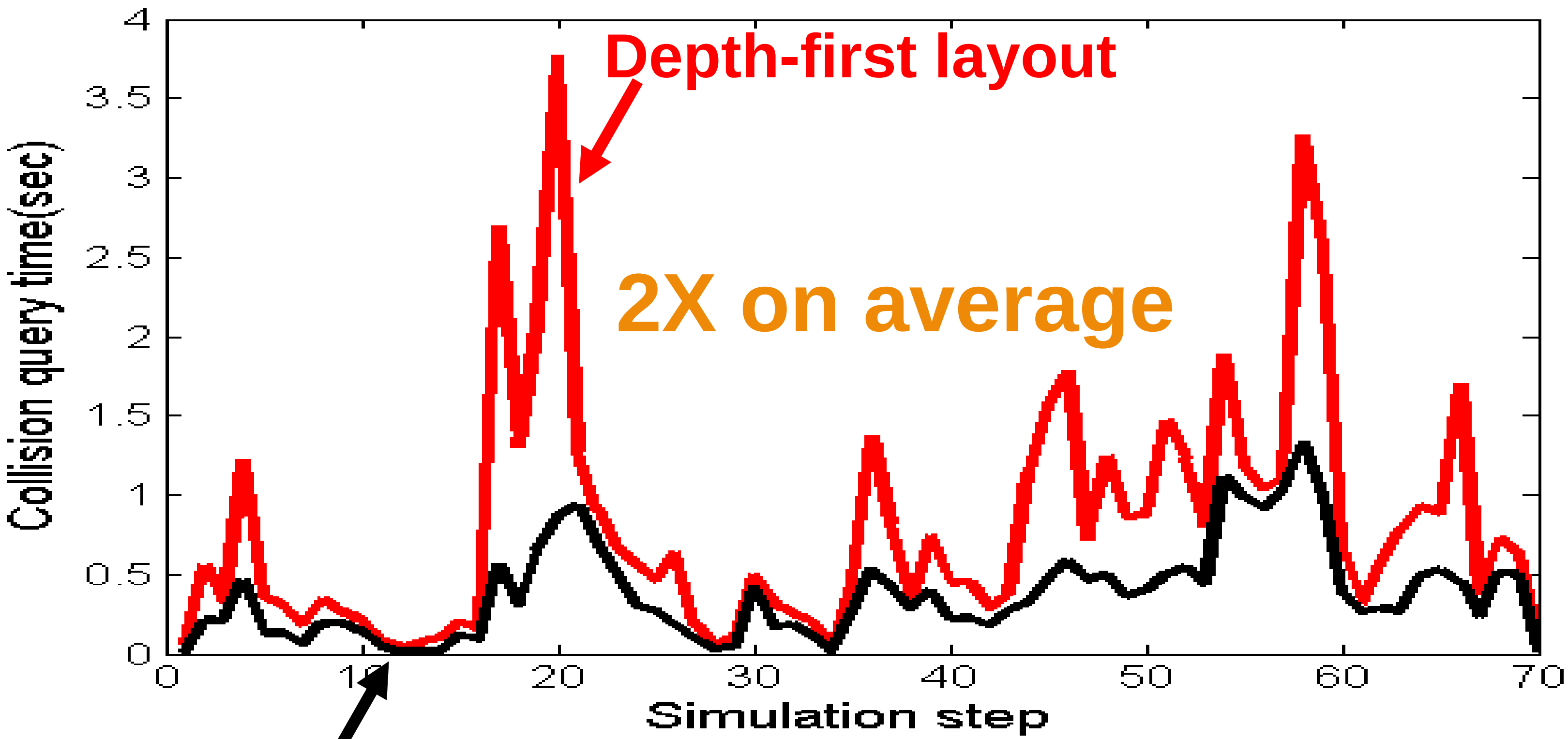
# Goal

- **Compute cache-coherent layouts of polygonal meshes**
  - ♦ **For visualization and collision detection**
  - ♦ **Handle any kind of polygonal models (e.g., irregular geometry)**

# Rigid Body Simulation



# Collision Detection Time




Cache-oblivious layout

# Runtime Performance

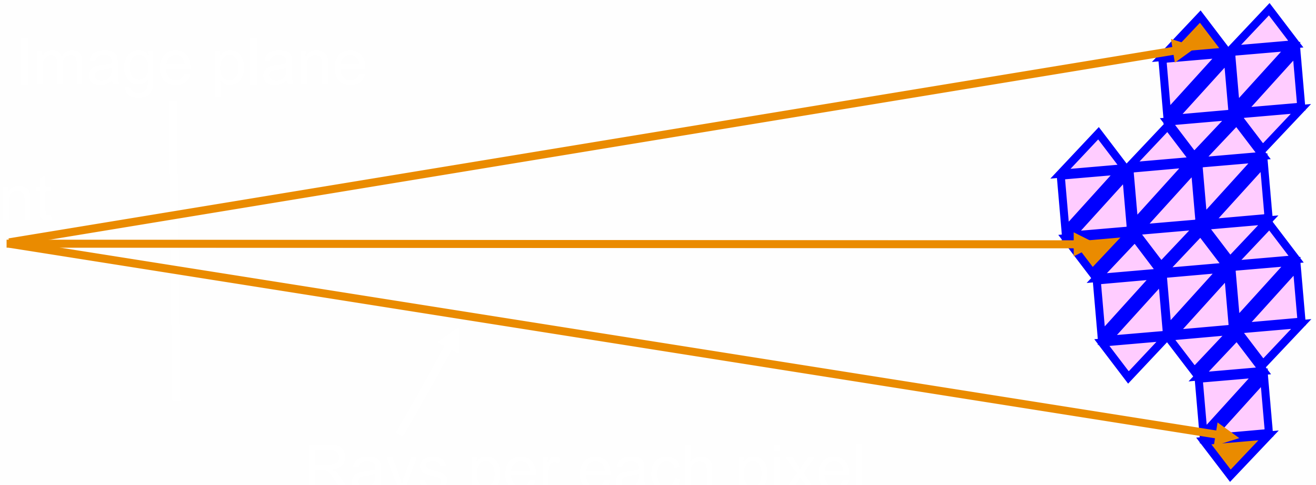
**512x512 image resolution, GeForce 5950FX**

| Model | Pixels of error | Frame rate | Mem. footprint | Refinement time |
|---|---|---|---|---|
| Power plant | 1 | 28 | 400MB | 1% |
| St. Matthew | 1 | 29 | 600MB | 2% |

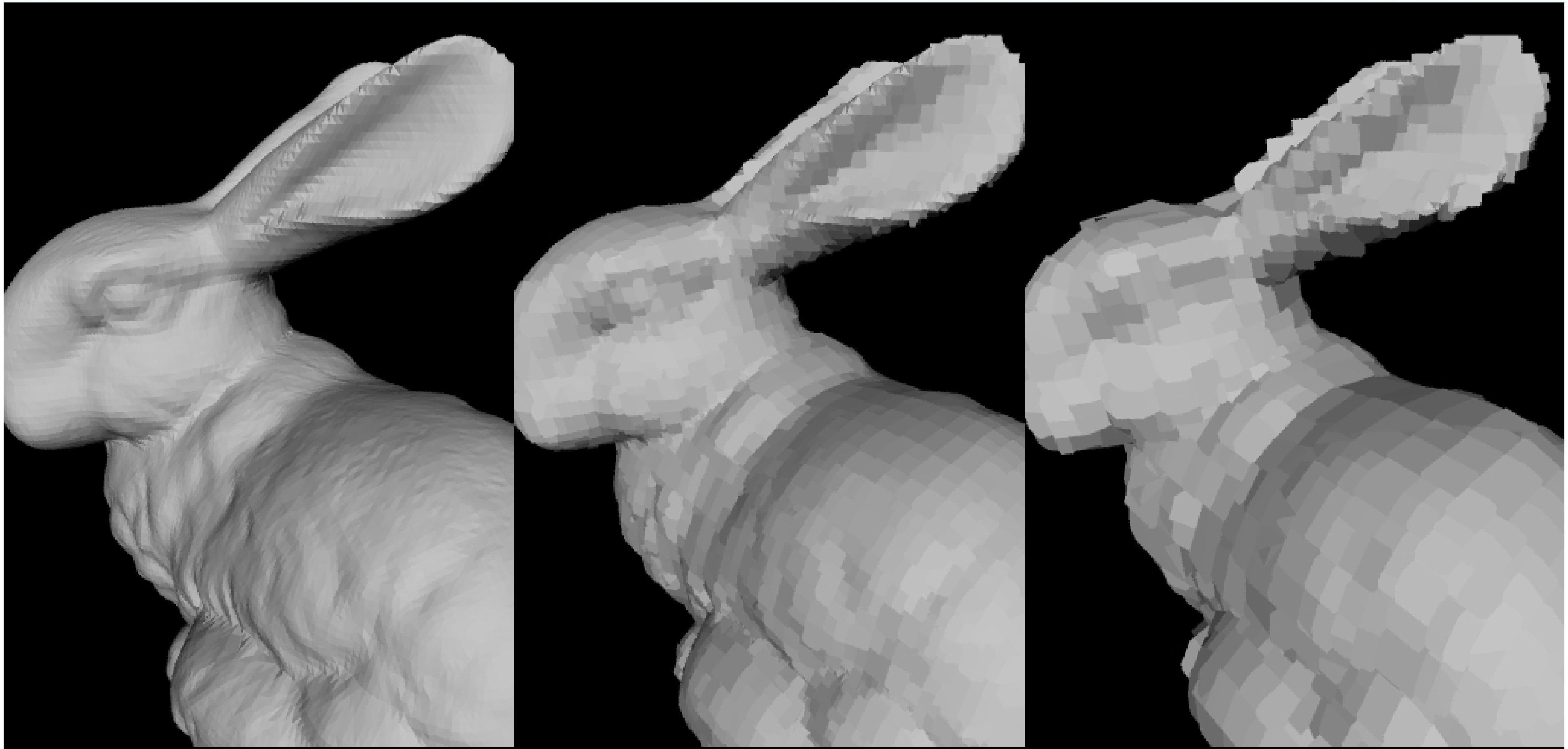
# Ray Coherence Techniques

- **Assume coherences between rays**
  - ♦ **Works well with CAD or architectural models**

- **Highly-tessellated models**
  - ♦ **Not much coherence between rays**



# R-LODs with Different PoE Values



PoE: Original 40 80