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

http://www.sci.utah.edu/~abe/massive06/

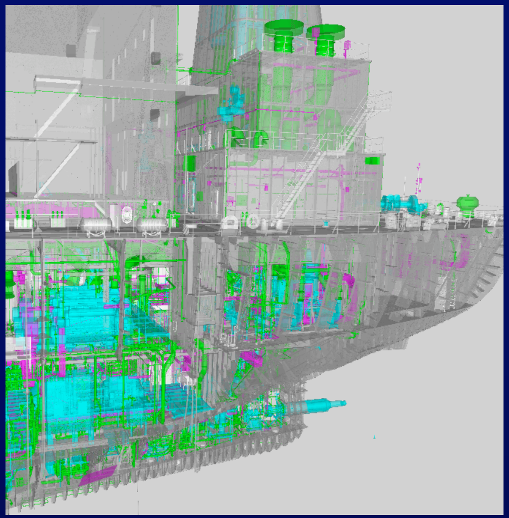
UNIVERSITY OF UTAH

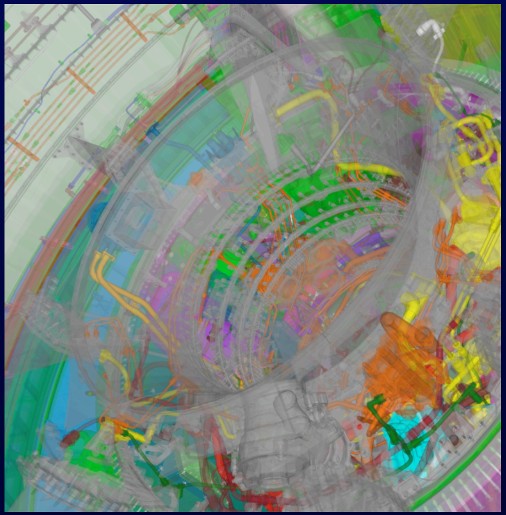
Overview

- Reasons for ray tracing massive models.
- Examine Multi-Core/Processor today.
  - Types of parallelism to look for.
  - Considerations for ray tracing.
- Describe Manta general purpose interactive ray tracing architecture.
- Miscellaneous Issues.
  - Simple parallel practices to adopt.
  - Remote/Collaborative Visualization.

Massive Model Ray Tracing

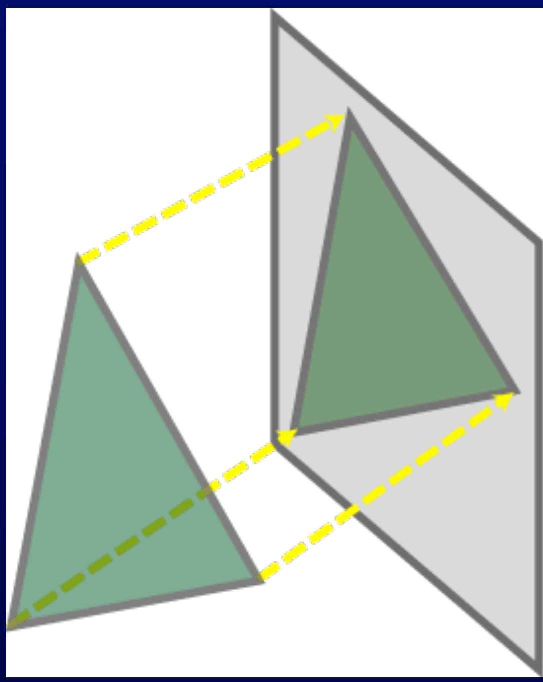
- Basic ray tracing shading and intersection technique shared with serial renderers.
- No precomputed visibility allows for transparent rendering, hiding or culling geometry on the fly.
- Largest datasets still well in-core on moderate sized machines.
- Parallel systems available on the desktop!

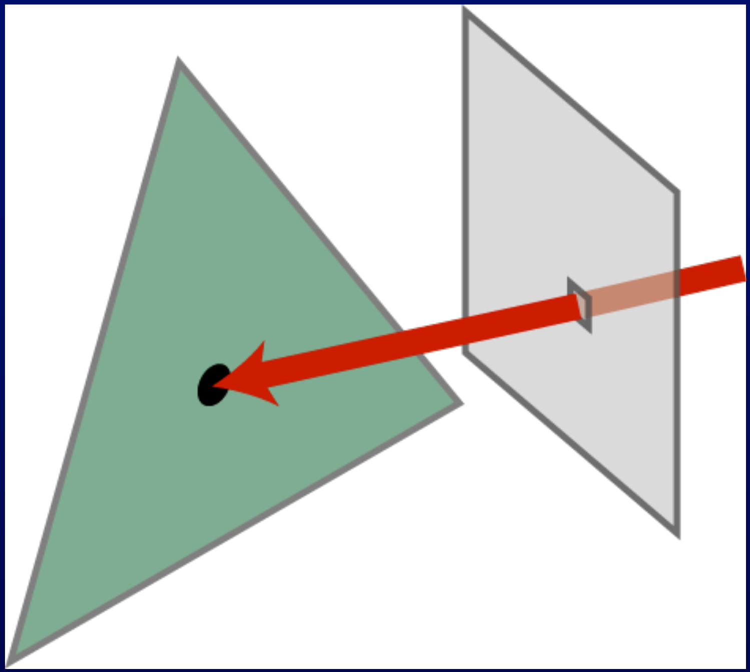




Tanker dataset courtesy Northrup Grumman Newport News Ship Building.
Boeing 777 dataset courtesy The Boeing Company.
All images rendered in Manta Interactive Ray Tracer

Ray Tracing in a nutshell

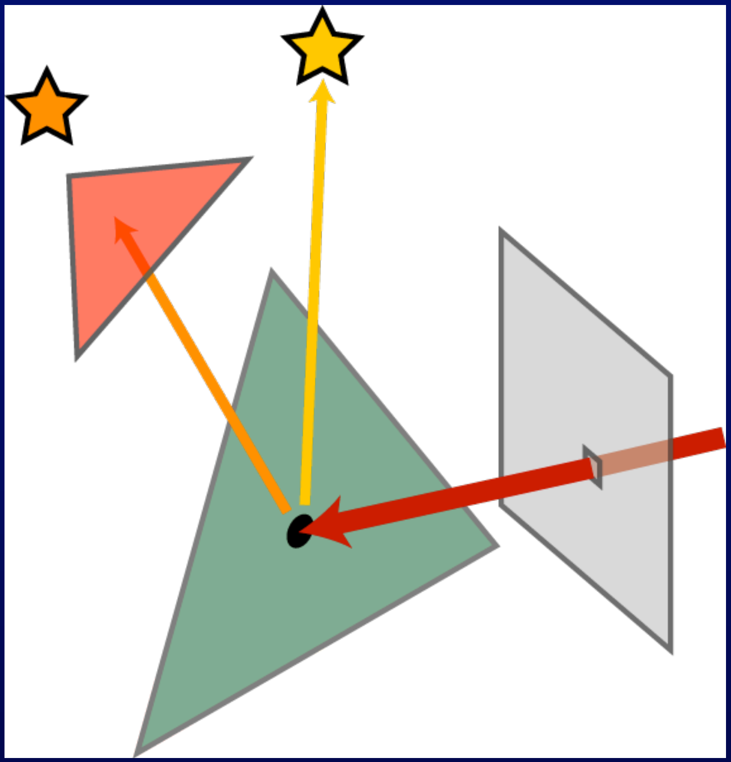


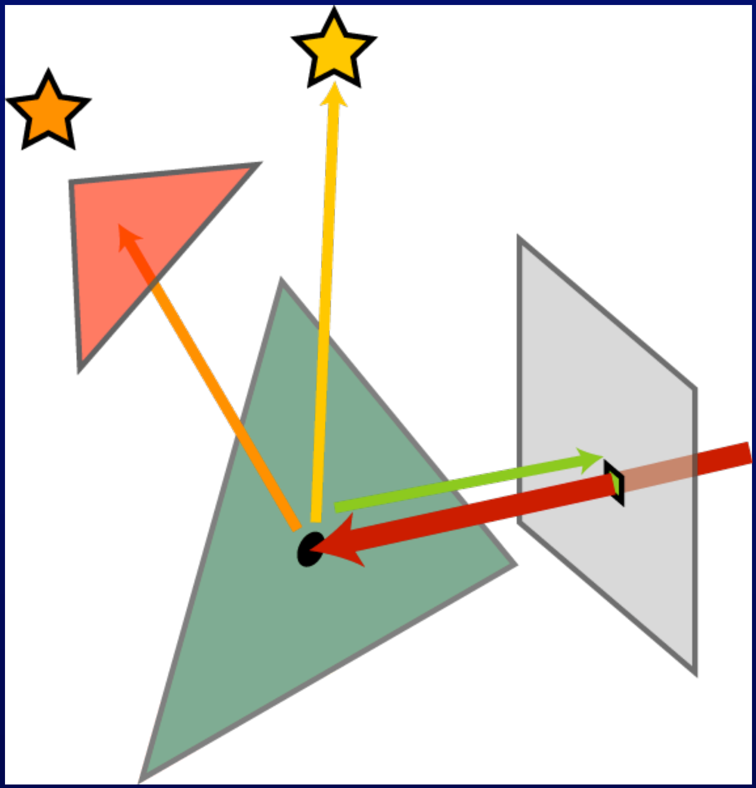


Find closest intersection to image along a ray.

- Rasterization uses projection

Ray Tracing in a nutshell



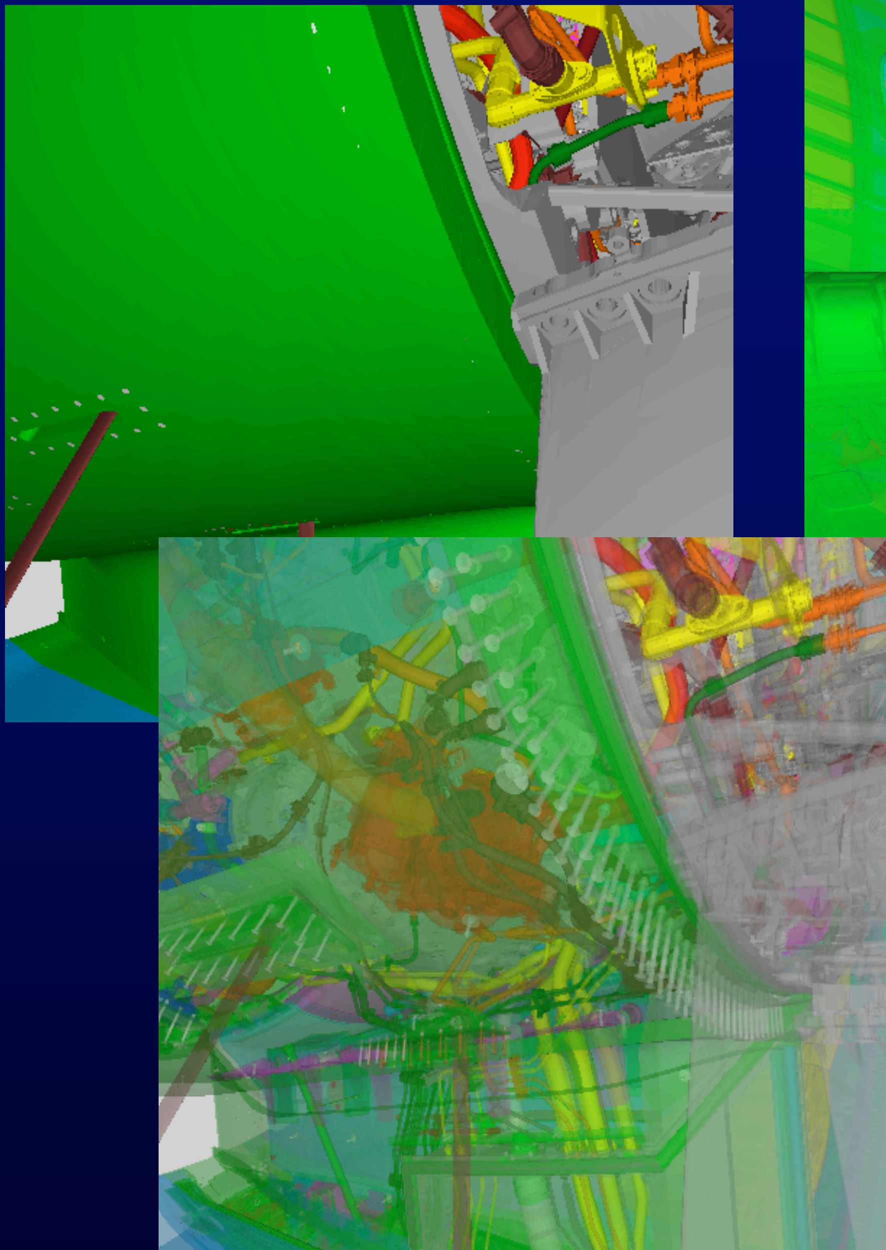


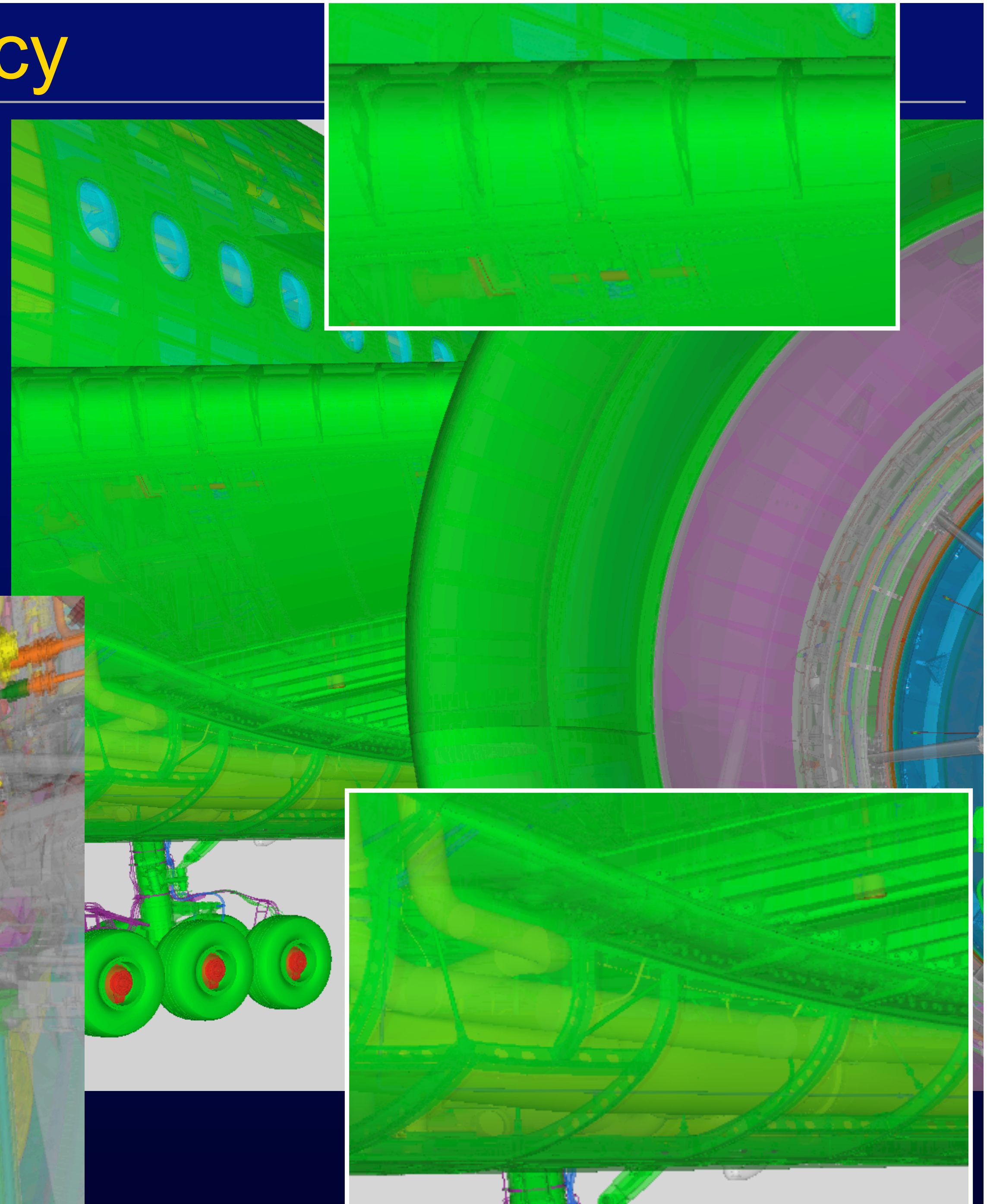
1. Find closest intersection
2. Invoke material shader on hit point.
   - Send shadow rays.
   - Send secondary rays.
   - Repeat.
3. Return sample color.

- Easy to change visibility.
- Easy shading effects.

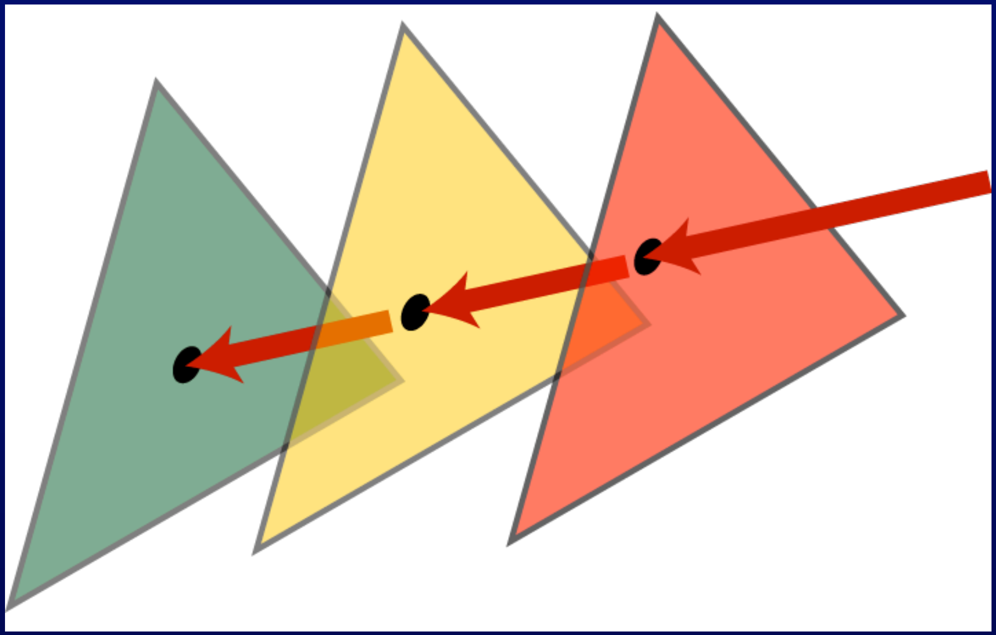
For example: Transparency

Example Transparency





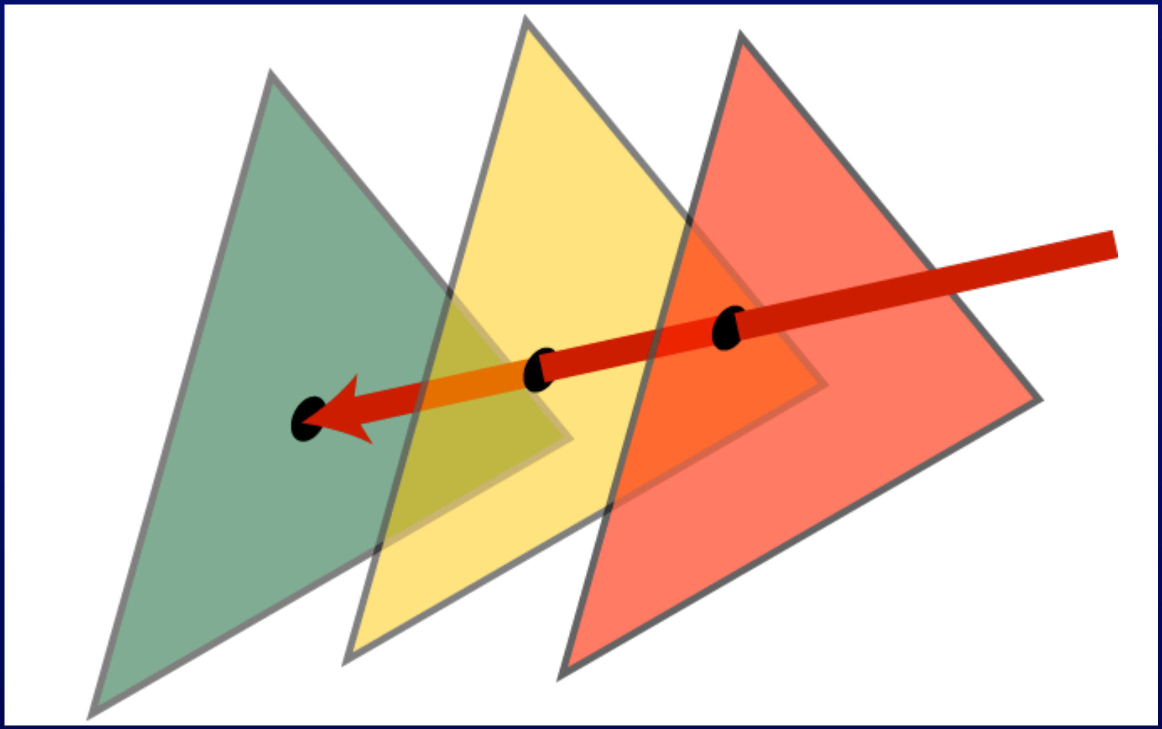
Transparency is easy in a ray tracer!



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)

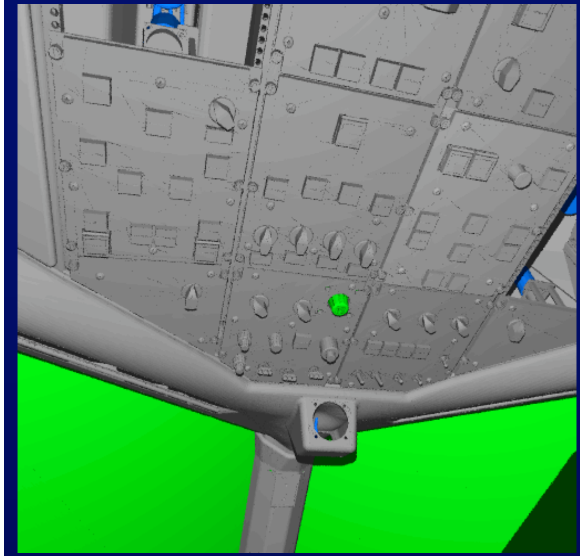
Sorting is necessary since triangles won't be intersected in order. (Each kdtree leaf contains several triangles.)

Other techniques?

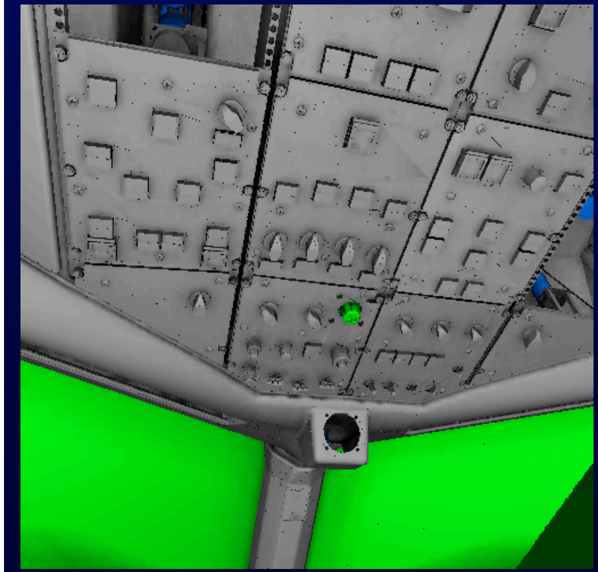
Along the ray transparency is only one example.

Other shading effects are possible with secondary rays, for example ambient occlusion.

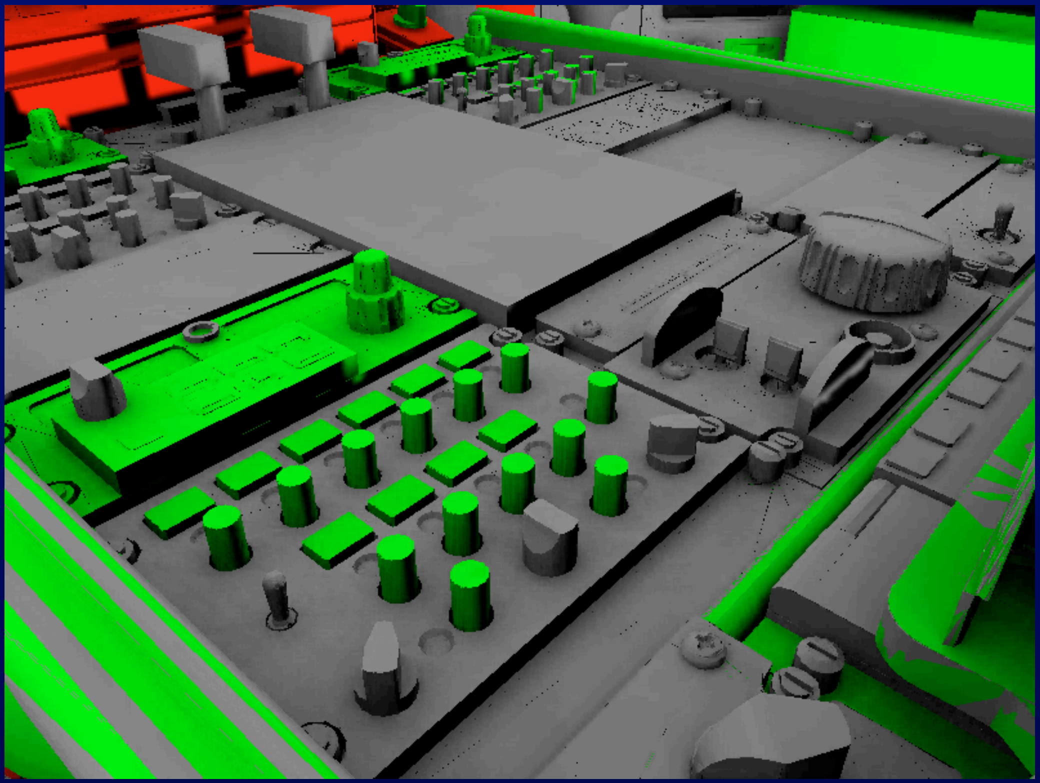
Ambient Occlusion



Lambertian w/ Shadows



Ambient Occlusion



Ambient Occlusion increases contrast In areas of fine detail.

Why multi-core systems?

- Large amount of processors and memory.
- The same system used for scientific computing and visualization.
- Becoming smaller and cost less.
- Faster multi-core clusters require fewer nodes.

16 core Opteron system. (top)
16 processor SGI Itanium (half rack).

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.

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.

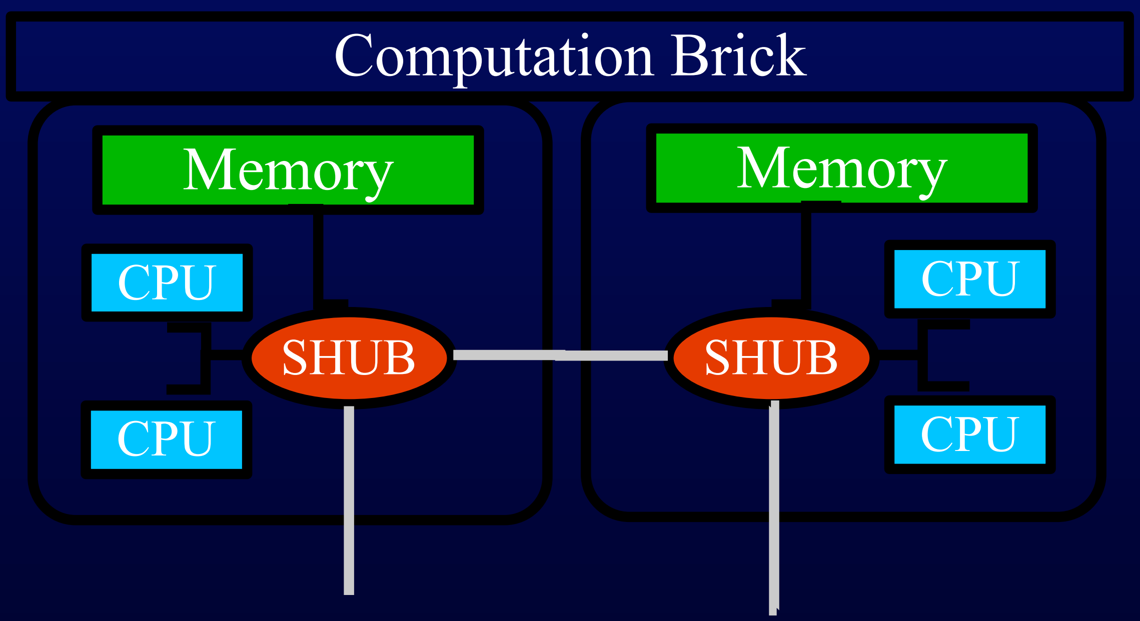
- e.g. AMD Hyper-Transport.
- Other devices connect to HT network.

Dense Multi-Core

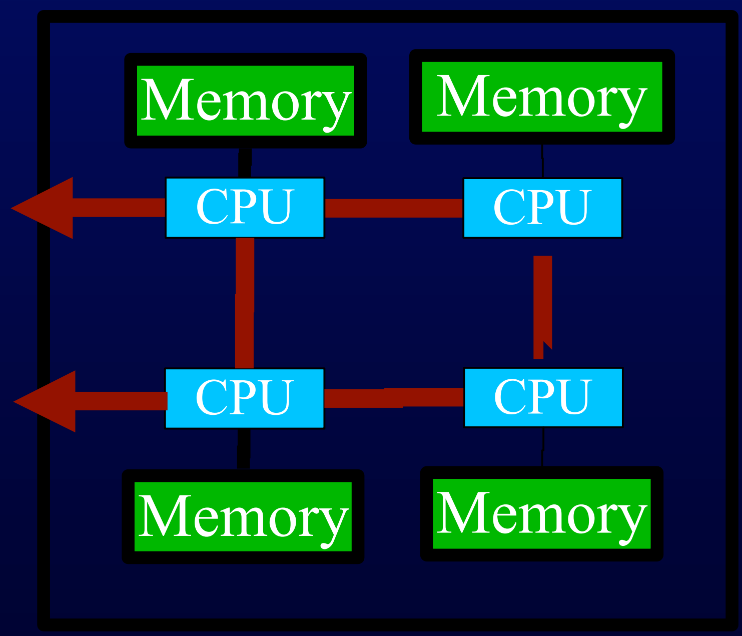
- 1 or 2 processors with many cores each.
- Multiple threads per core.
- Possible future direction.

Example multi-processor systems for ray tracing.

SGI Prism / Itanium2

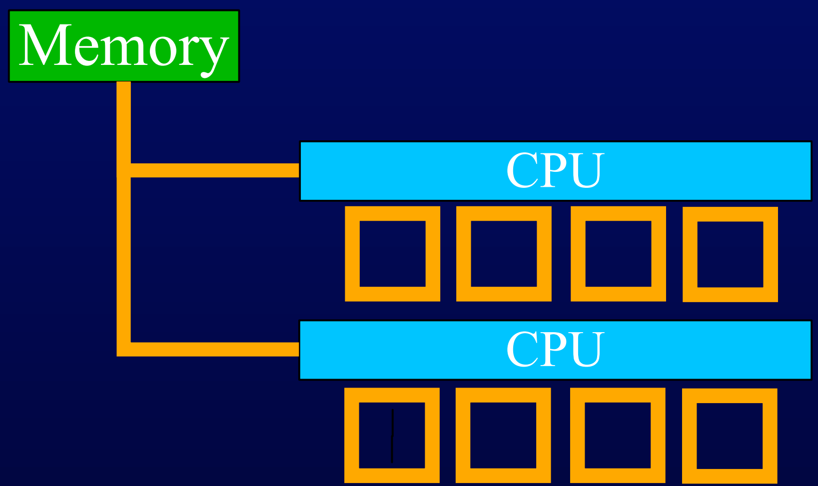


AMD HyperTransport / Opteron

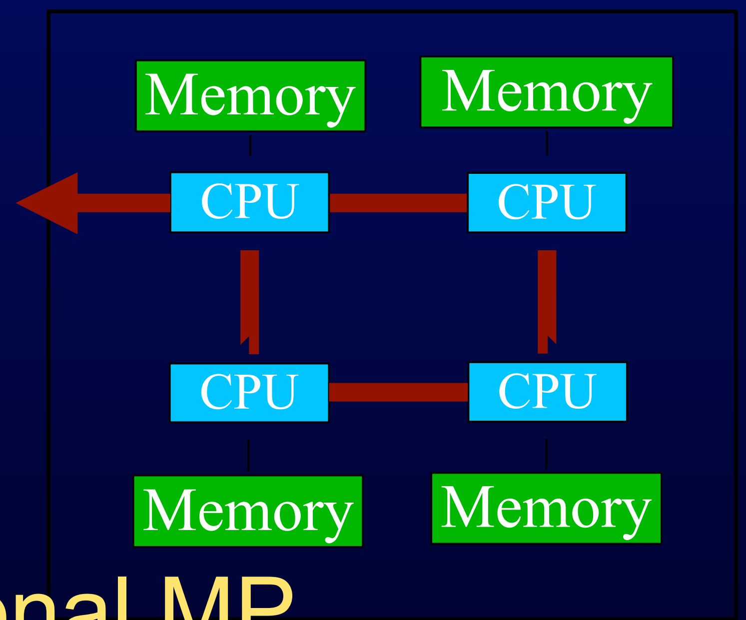


Example multi-processor systems for ray tracing.

Dense Multi-Core

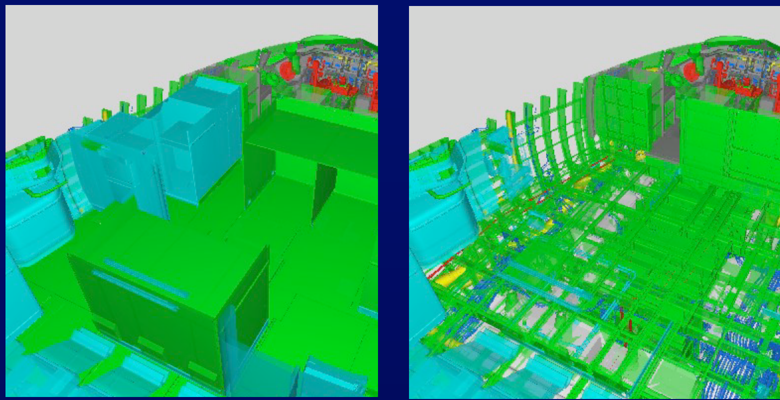


AMD HyperTransport / Opteron

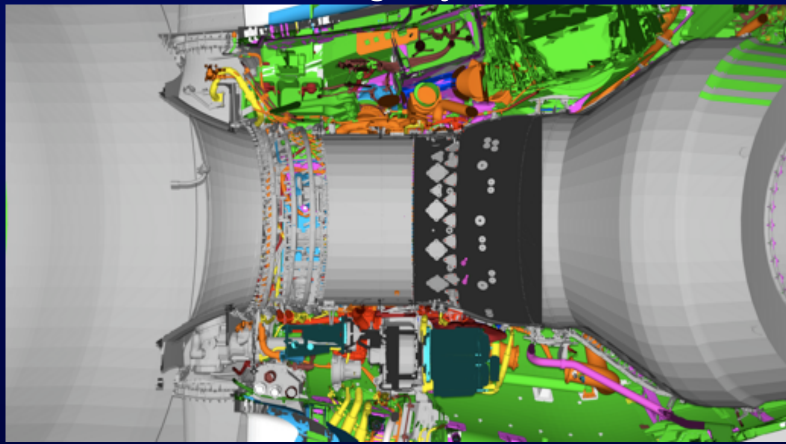


How will it scale with traditional MP workloads?

What can we implement for Massive Model vis?



Hiding Objects



Cutting Planes

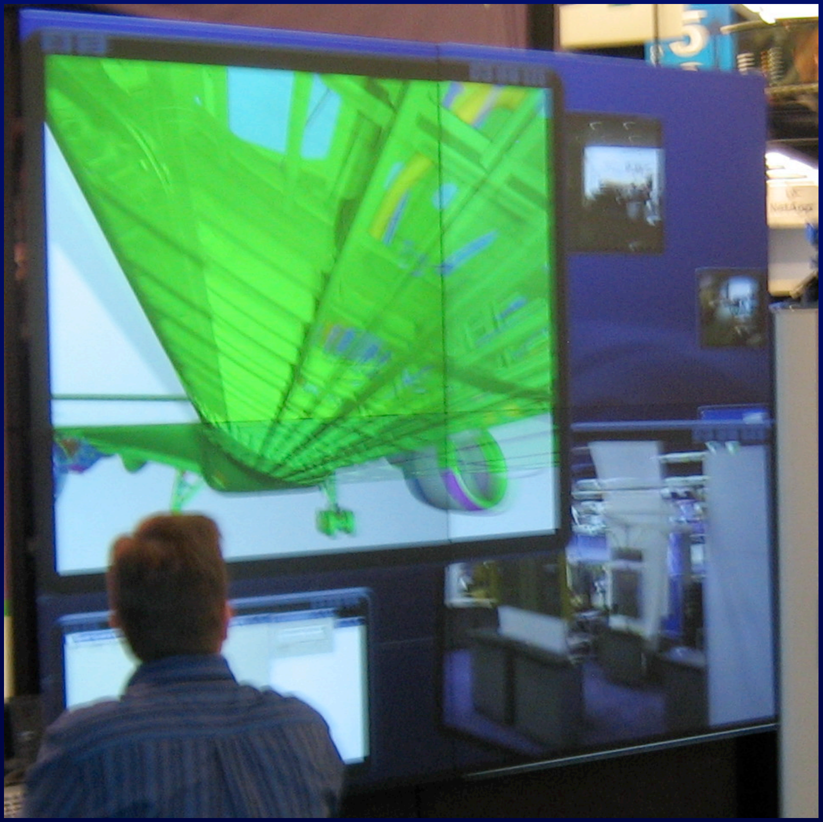
Massive Model Vis:

- Cutting Planes
- Hiding Objects
- Transparency

All with hundreds of millions of triangles.

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.

Application Scenario



Quality Engineers use ray tracer to visualize problems with aircraft assembly.

A. Stephens, S. Boulos, J. Bigler, I. Wald, and S. G. Parker *An Application of Scalable Massive Model Interaction using Shared Memory Systems* Proceedings of the Eurographics Symposium on Parallel Graphics and Visualization, 2006

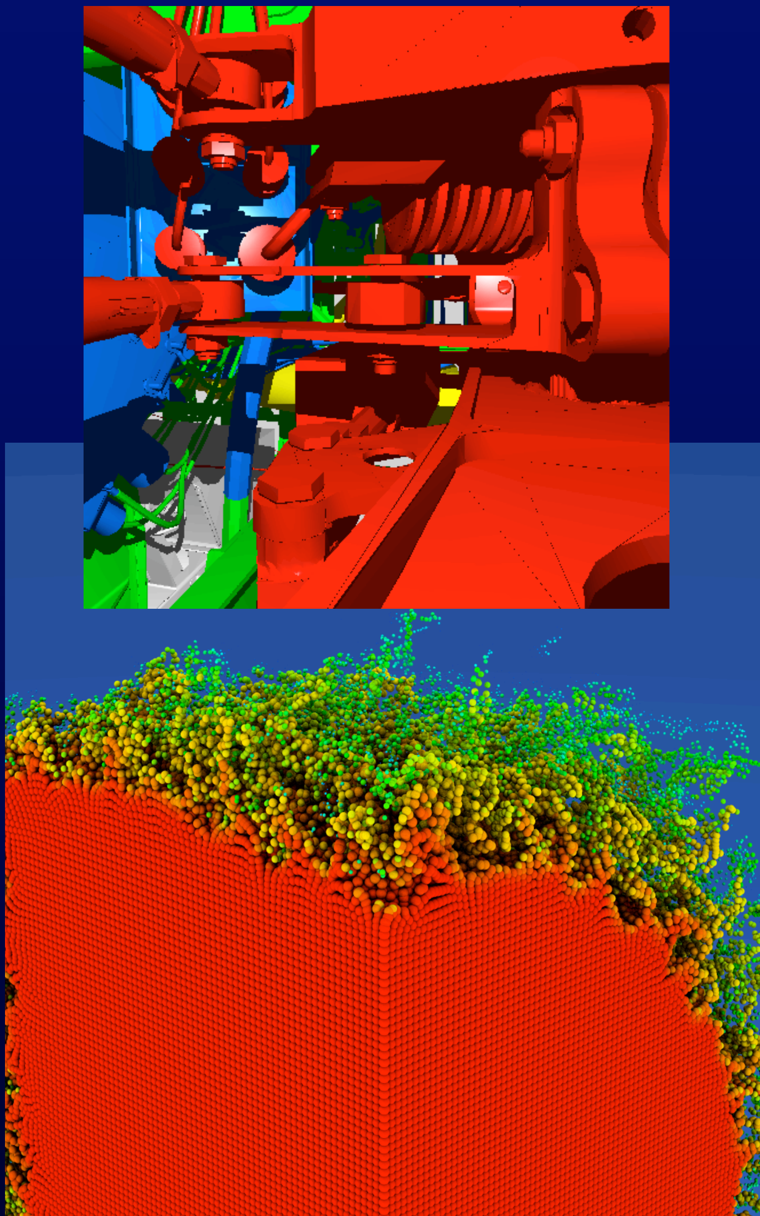
Manta Interactive Ray Tracer

- Platform for implementing different ray tracer applications.
- Take advantage of modern multi-core processors.
  - Multi-threads.
  - Single thread performance derived from specific instruction stream optimizations.
    - SIMD, special cases, etc.

Design Philosophy

- Obtain thread scalability:
  - Parallel Pipeline, communication constraints, state update w/ transactions.
- Facilities for good single thread performance:
  - SIMD data layout.
  - Wide ray packets, w/ properties.
  - Lazy evaluation & special case code.
- Build acceleration structures & shaders on top of these two goals.

More than just triangles.



Material Point Method Dataset

Modular Design

- Allows Manta to be embedded in other programs.

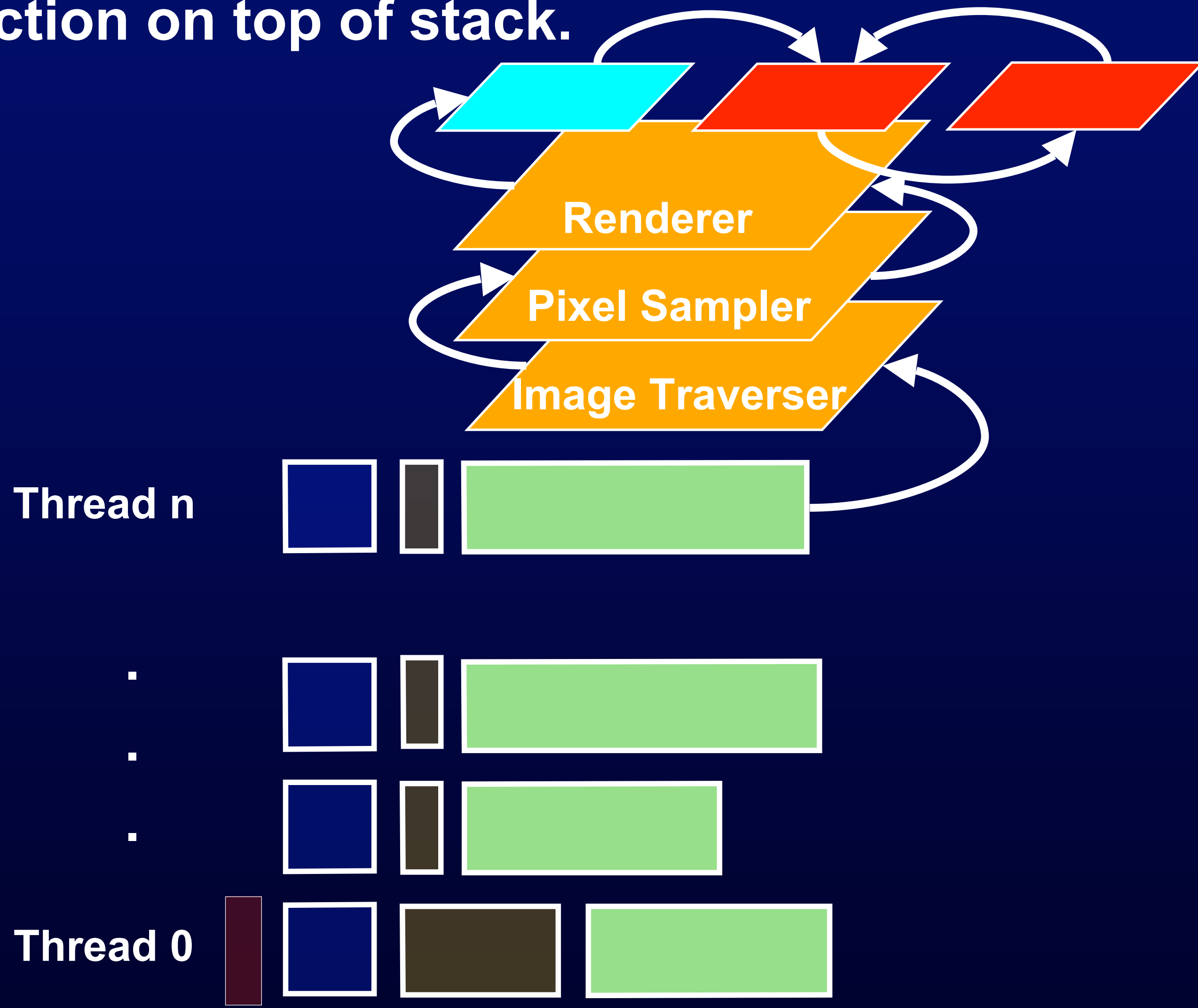
- Supports multiple primitives:
    - Massive triangle models.
    - Massive volumes.
    - Sphere glyph (MPM) rendering.

- Python front-end

Open Source
Highly Portable

Pipeline and Rendering Stack

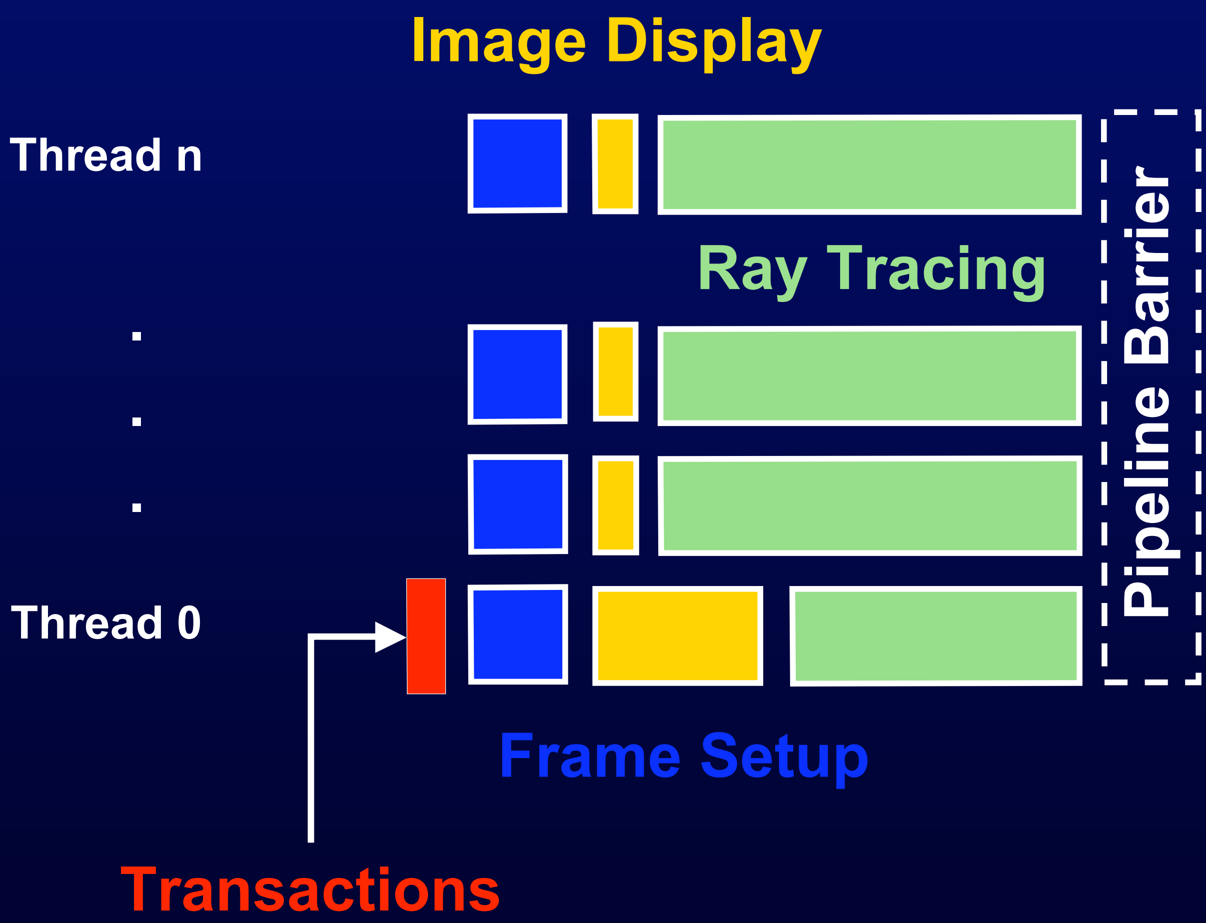
1. **Modular components vary by application.**
2. **Synchronized multi-thread pipeline**
3. **Asynchronous single thread rendering stack.**
4. **Scene intersection on top of stack.**



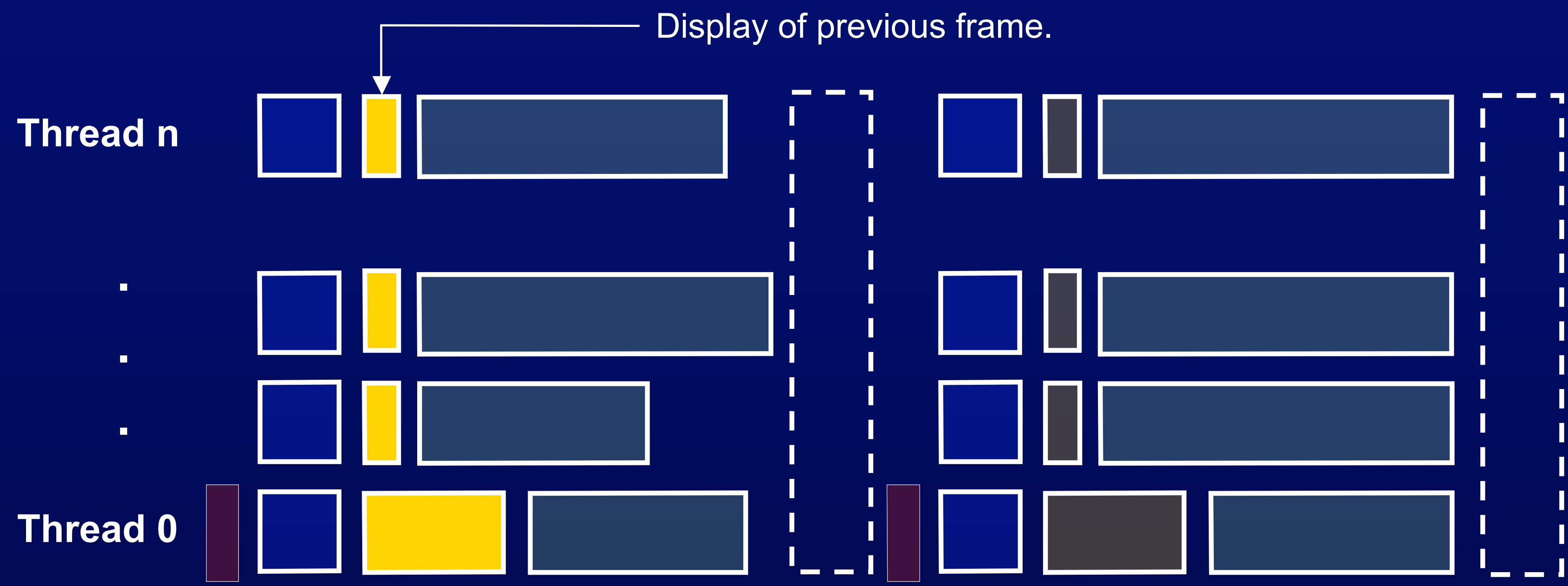
Parallel Pipeline

Manta Pipeline

- **Modular and extensible components.**
- **Transaction state changes applied each stage.**
- **Barrier synchronization between stages.**

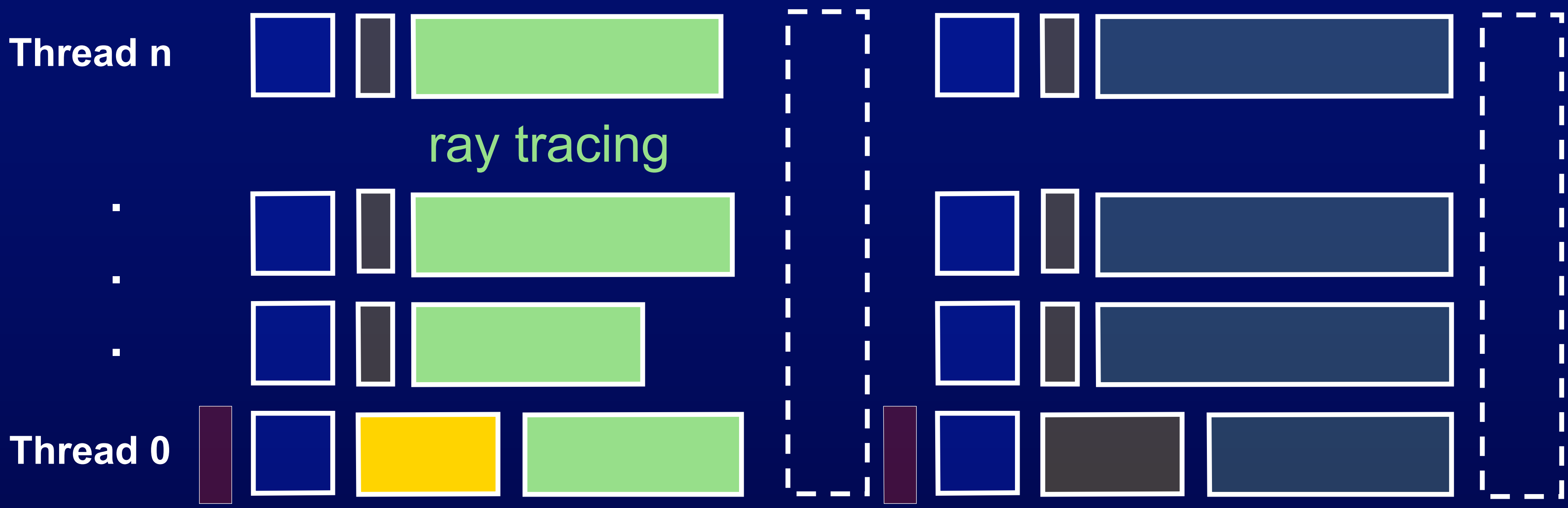


Parallel Pipeline



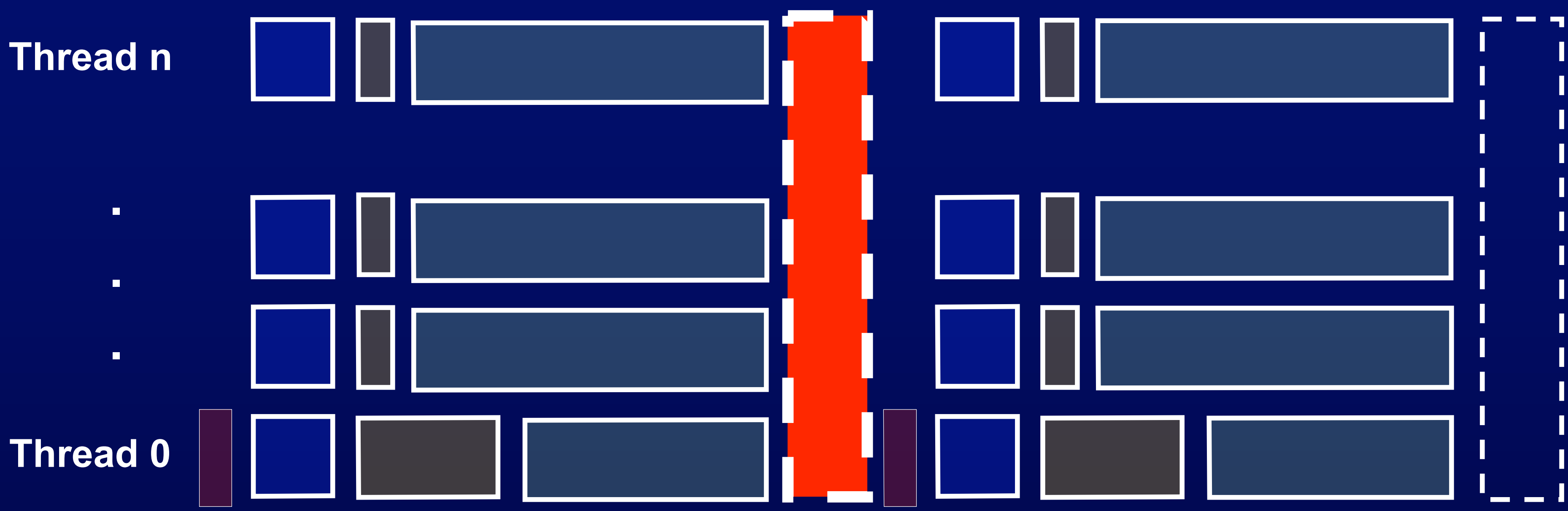
- Display frame *i-1*
  - Thread 0 calls opengl.
  - All others return immediately.

Parallel Pipeline



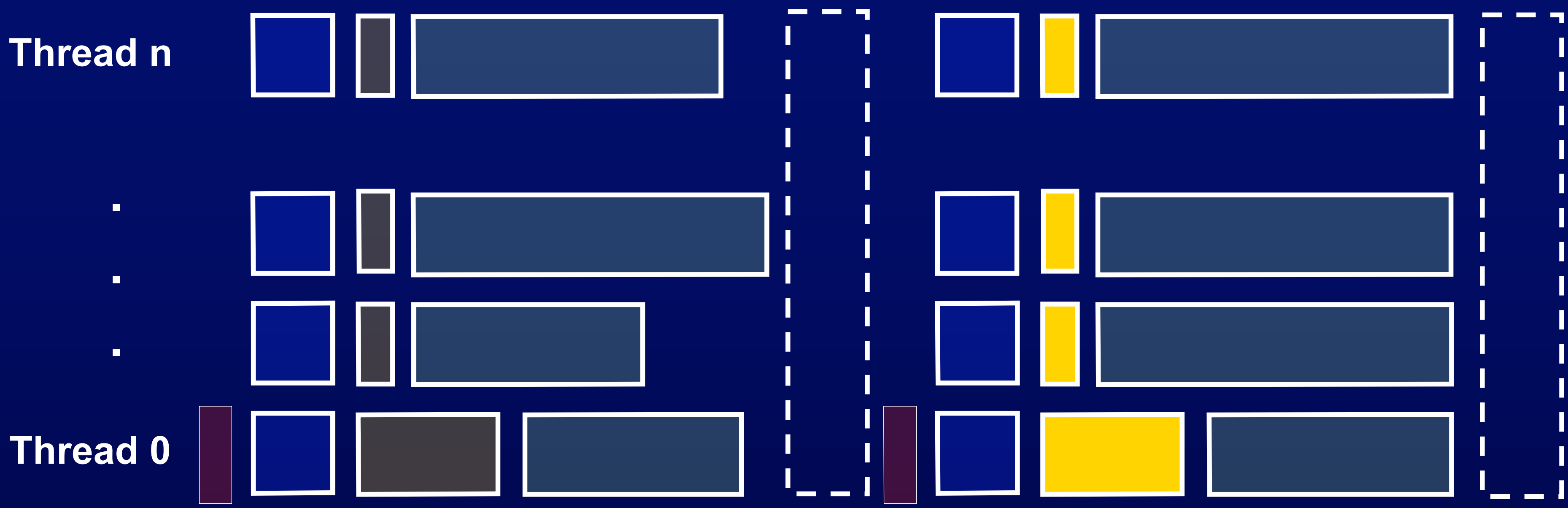
- While thread 0 is displaying frame *i-1*:
  - All other threads start rendering frame *i*.
- Thread 0 joins as soon as it finishes image display.

Parallel Pipeline



- 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.

Manta Pipeline Implementation

1. One thread per core (if non-hyper threaded).
2. Stages in pipeline differentiated by sub-functions.
3. Stages ordered by load balance characteristics.
4. One pipeline barrier.
5. Additional synchronization for transactions.

```
void Pipeline::inner_loop( int frame,
    int proc, int numProcs ) {
   // Global synchronization.
  pipeline_barrier.waitFor( numProcs );

  // Inherently load balanced.
  parallel_animation_callbacks();

  // Imbalanced.
  if (proc == display_proc)
    image_display->
      displayImage( buffer[frame-1] );

  // Dynamically balanced.
  image_traverser->
     render_image( buffer[frame], proc );
}
```

Implementing Transactions

Transactions permit manta to invoke a method in a foreign object to change some state at a “safe” time.

For example, using python:

```
def OnAutoView(self, event): # Called by python thread on GUI event.
        # Add the transaction. Unsafe to actually change or access renderer state!
        self.engine.addTransaction("auto view",
                                   manta_new(createMantaTransaction(self.MantaAutoView,() )))

def MantaAutoView(self): # Called by Manta thread.

        # Find the bounding box of the scene.
        bounds = BBox();
        self.engine.getScene().getObject().computeBounds( PreprocessContext(), bounds );

        # Invoke the autoview method.
        channel = 0
        self.engine.getCamera(channel).autoview( bounds )
```

Wide Ray Packets

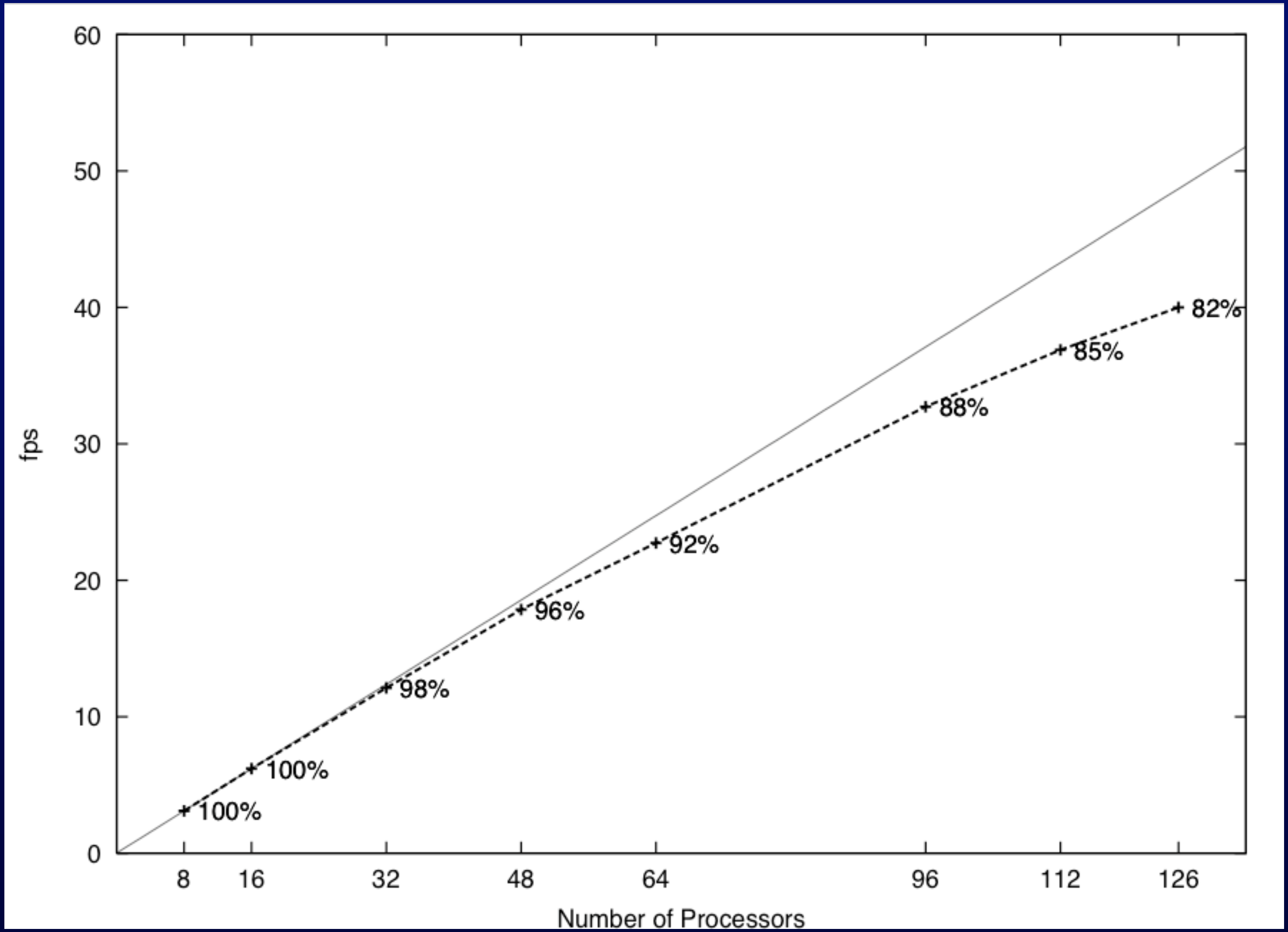
Contain:

- Ray origin, direction and hit info.
- Packet *properties* for special case, lazy, etc.
- Data layout for SIMD w/ vertical and horizontal accessors.

Packets are containers for data used to:

- Perform intersection.
- Store & access info about intersection.

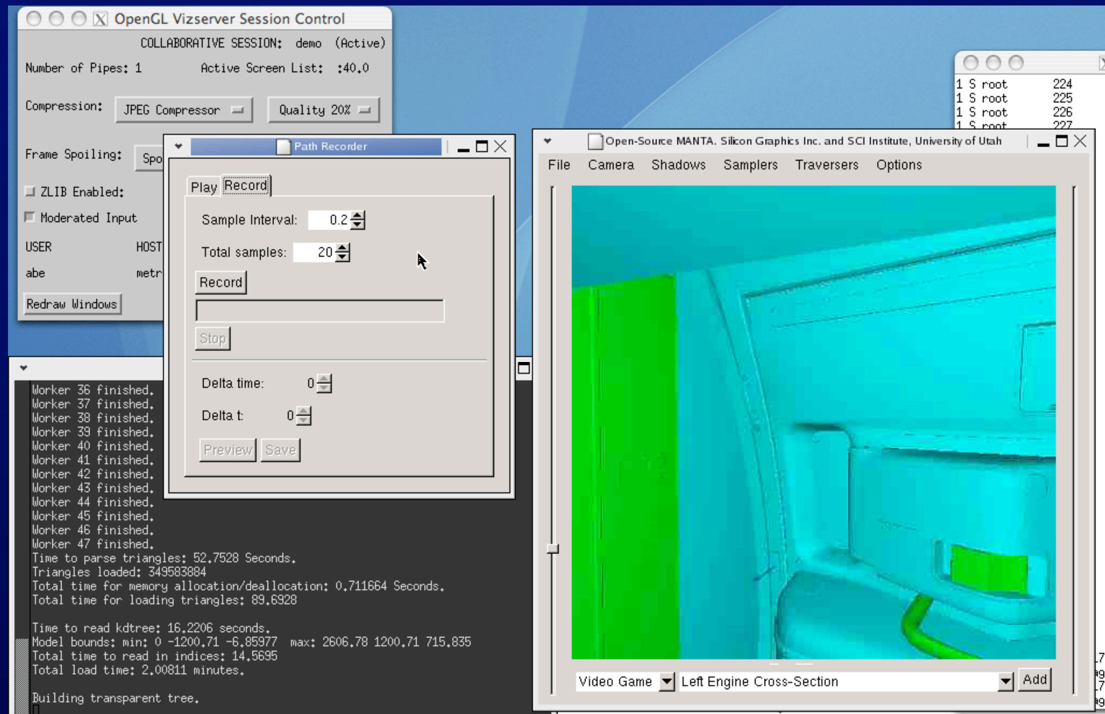
Manta Scaling



128 p 1.6 Ghz Itanium2

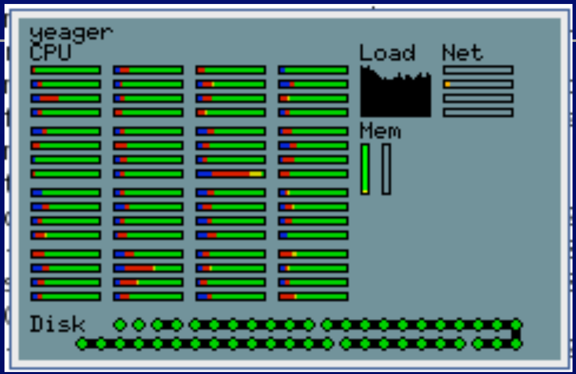
- 92% linear at 64p 82% at 126p
- Resolution 1024x768

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.

Load Balancing



Coarse Load Balancing.

- Choose a strategy for assigning tiles to threads.
- Implement it as efficient as possible (hw intrinsics)

Load Balancing

Fine-grained Task Assignment

- Share read data, avoid common write data.
- Complications: Lazy evaluation policies, Multi-thread/core scheduling.
- How does ray coherence effect each level of parallelism for read/write?

Acceleration Structure Build

Parallel KD-Tree build.

- Strategies for offline full SAH build
  - Multi-thread sorting and merging.
  - Evaluate split candidates in parallel.
  - Build sub-trees in parallel.

Reduced 777 build time from one day to several hours.

Recent approaches vary heuristic & parallelization with tree depth.

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 own pipeline stage.
- Per-thread copy of structure.
- Several threads share copy in neighborhood.
- Choice depends on application.

- Unsafe practice is not an option.

Bottom Line

To achieve scalable multi-thread performance:

- Use a parallel pipeline with limited synchronization points. (transactions)
- Use asynchronous display.

Optimize for single processor performance.

- Use packet properties for instruction optimization.

Use at least naïve sub-tree parallel kdtree build.

Questions?

A. Stephens, S. Boulos, J. Bigler, I. Wald, and S. G. Parker *An Application of Scalable Massive Model Interaction using Shared Memory Systems* Proceedings of the Eurographics Symposium on Parallel Graphics and Visualization, 2006

J. Bigler, A. Stephens, S. G. Parker. *Design for Parallel Interactive Ray Tracing Systems.* Scientific Computing and Imaging Institute, University of Utah. Technical Report No UUSCI-2006-027. (submitted)

This work is supported by:

U.S. Department of Energy through the Center for the Simulation of Accidental Fires and Explosions, under grant W-7405-ENG-48

Utah Center of Excellence for Interactive Ray-Tracing and Photo Realistic Visualization.

National Science Foundation.

Additional support through internships:

Silicon Graphics Inc.

Intel Corporation

Additional resources:

http://www.sci.utah.edu/~abe/