Massive Model Rendering with Super Computers

Abe Stephens

1:30 - 1:50pm

Speaker affiliations: SCI Institute, University of Utah and Intel Corporation.
Overview

Focus on shared-memory/multi-core software design.

• Massive models? Why use super computers?
• Challenges: parallel build & rendering.
• Manta architecture.
• Applications & conclusions.
Massive Model Visualization

- Hundreds of millions of primitives.
- Scientific data, CAD, architectural.
- Principle task is static inspection.

Massive Model Visualization

**Double Eagle Tanker**
85 M Triangles

**CSAFE Container**
2.8 million particles
2.1 voxel volume
450 timesteps

**Boeing 777**
350 M Triangles

**Richtmyer-Meshkov**
8 GB volume
272 timesteps
Application Scenario

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

Application Scenario
Why parallel computers?

- 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.
Parallel Acceleration Structure Build

- Example parallel KD-Tree build.
  - Strategies for offline 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.
Parallel Ray Tracing

• Easy to break ray tracing into parallel pieces.
• Parallel architecture must focus on scalability.
  – User input coordination.
  – Thread safe state changes.
  – Display overhead.
  – Acceleration structure update.

• Both thread level parallelism and instruction level parallelism effect design.

Processor utilization (green is unused capacity)
Manta Software Architecture

- Addresses both thread level parallelism and instruction stream optimization.
- Provides a scalable foundation to solve a variety of rendering problems.
- Modular software components and Python bindings.

http://code.sci.utah.edu/Manta
Open Source
Manta Parallel Pipeline

Image Display

Thread n

Ray Tracing

Pipeline Barrier

Thread 0

Transactions
Manta Parallel Pipeline

Display of previous frame.

Ray tracing, dynamically load balanced.
Manta Rendering Stack

- Stack of modular sampling and ray tracing components.
- Only global synchronization in pipeline.
- Threads execute stack asynchronously.
Load balancing

Load balancer tile division, requires thread safety.
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 );
}
void Raytracer::traceRays(const Context& context, RayPacket& rays) {
    context.camera->makeRays(rays);
    rays.resetHits();
    context.scene->getObject()->intersect(context, rays);

    for(int i = rays.begin(); i < rays.end(); i++){
        RayPacket subPacket(rays, i, end);
        if(rays.wasHit(i)){
            const Material* hit_matl = rays.getHitMaterial(i);
            int end = i+1;
            while(end < rays.end() && rays.wasHit(end) &&
                rays.getHitMaterial(end) == hit_matl)
                end++;
            hit_matl->shade(context, subPacket);
            i=end;
        } else {
            int end = i+1;
            while(end < rays.end() && !rays.wasHit(end))
                end++;
            RayPacket subPacket(rays, i, end);
            context.scene->getBackground()->shade(context, subPacket);
            i=end;
        }
    }
}
Scalability - 128 processors
Bottom Line

• To achieve scalable multi-thread performance:
  – Use a parallel pipeline with limited synchronization points.
  – Use asynchronous display.

• Optimize for single processor performance.
  – Use packet properties for instruction optimization.

• Not really “big iron” any more.
Questions?


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