Rendering Radiance Animations on the GRID

Vibhor Aggarwal, Kurt Debattista, Gavin Ellis and Alan Chalmers

University of Bristol
University of Delhi
High-Fidelity Rendering

- Physically-based rendering
  - Physically based quantities/materials
  - Global illumination
  - Animations
Motivation

- Speedup Rendering
  - Shared computational resources
  - Radiance animations can take a long time to render
- Distributed method
  - GRID-like systems
Example: Mine Tunnel Ride

- 2964 * 768 HQ
- 2 hours per frame
- 2,310 frames
Computing on GRID-like systems

- Distributively-owned multi-programmed computing resources
- Large pool of potentially unutilised computational resources
  - Many unused at certain times
    - Offices
    - Labs
    - Clusters
    - Screen-saver time
- Workload management systems
  - Condor
    - Job queuing and monitoring
Computing on GRID-like systems

- GRID computing is a different animal from distributed computing on a cluster

Issues

- Dynamic change of resources
  - Multi-programmed/multi-user environments
- Minimise (no) control/data communication
  - Deadlock is easy
- Fault Tolerance
A first approach

- A queue of frames as jobs
  - Simple approach
  - Little implementation
    - Simple script
First pass
Example
Problems

- Issues
  - Artefacts produced from different IC samples
  - Does not take advantage of coherence
Irradiance Cache

- Acceleration data structure [Ward et al. 88]
  - Distributed ray tracing
  - Accelerates rendering by an order of magnitude

- Algorithm
  - Caches indirect diffuse samples
  - Interpolates/extrapolates from previous samples within radius
IC – Normal Animation

- Diffuse only
- Pure specular
- Diffuse and specular
- Diffuse only
IC – 2nd Frame Normal

Diffuse only

Pure specular

Diffuse and specular

Diffuse only
IC – 2\textsuperscript{nd} Frame GRID

- Diffuse only
- Pure specular
- Diffuse and specular
- Diffuse only
Our solution

- Two pass approach
  - First pass
    - Shoot random rays
    - Many caches
    - Merge cache
  - Second pass
    - Distribute merged cache
    - Render animation frames
First Pass & Merging

Irradiance Caches

Merged Cache
First Pass

Selection of the Frames for the first pass:
- Selecting every $n^{th}$ frame
  - Total number of frames corresponds to total number of processors
- Selecting frames after parsing the view file
  - Weigh according to change in position and direction
First Pass

Selection of the Frames for the first pass:

- Selecting every $n^{th}$ frame
  - Total number of frames corresponds to total number of processors
- Selecting frames after parsing the view file
  - Weigh according to change in position and direction
First Pass

Selection of the Frames for the first pass:

- Selecting every $n^{th}$ frame
  - Total number of frames corresponds to total number of processors
- Selecting frames after parsing the view file
  - Weigh according to change in position and direction
First Pass

Selection of the Frames for the first pass:

- Selecting every $n^{th}$ frame
  - Total number of frames corresponds to total number of processors
- Selecting frames after parsing the view file
  - Weigh according to change in position and direction
First Pass: Sampling

- Render using pseudo random sampling
  - Good hierarchy
  - Progression
  - Distribution
  - (0,2) quasi-random sequence

- Render until
  - Time runs out
  - IC hit/miss ratio threshold
  - Progressive aspect ensures we can stop whenever we want
(0,2) sequence hierarchy – 2
(0,2) sequence hierarchy – 4
(0,2) sequence hierarchy – 8
(0,2) sequence hierarchy – 16
(0,2) sequence - Distribution
Merging the Cache

- The Irradiance cache
  - Merged
  - Shared

Irradiance Caches

Merged Cache
Fault Tolerance - problem
Fault Tolerance (WIP)
Second Pass

- Same as standard method
  - Simplicity
    - Can be used with our various guises of Radiance
    - Potentially newer versions of Radiance

- Issues
  - Redo frame if not done

- Future work
  - Include fault tolerance at this stage
    - Compromise simplicity
Fault Tolerance – Second Pass
Implementation

- First pass
  - Modified version of rpict
    - Could have used rtrace

- Second pass
  - Standard Radiance rpict and variants

- Job distribution and management
  - Condor system
  - Shell scripts
    - Automate process
Examples

- Kalabsha
- Art Gallery (-ab 3)
- Corridor
Kalabsha merged
Art Gallery (-ab 3) unmerged
Art Gallery (-ab 3) merged
Timings – 100 procs

Corridor: ca. 320 frames
Merged Cache - 1hr 12 min
Unmerged Cache - 1hr 40 min

Art Gallery –ab 3: ca. 220 frames
Merged Cache – 4hr 35 min
Unmerged Cache – 5hr 7 min

Kalabsha: ca. 90 frames
Merged Cache - 31 min
Unmerged Cache - 48 min
Conclusions and Future Work

- Underutilised computational resources can be put to good use
  - Radiance for animations
    - Faster
    - Little cost
- Learnt a lot about GRID-like computation
- Future work
  - Design GRID specific parallel rendering algorithms
    - Better fault tolerance
    - Minimum sharing
Thank You!

Contact: Kurt.Debattista@bristol.ac.uk

Acknowledgments:
CG Bristol
Veronica Sundstedt for Kalabsha and Corridor models