Comparing Reyes and OpenGL on a Stream Architecture

John D. Owens    Brucek Khailany
Brian Towles    William J. Dally

Computer Systems Laboratory
Stanford University
Motivation

- Special-purpose processors: great performance
- General-purpose processors: great flexibility
- As graphics architects, how do we get both?
OpenGL and Reyes

- **OpenGL**: Prevalent in real-time graphics systems today
  - Designed for high-performance real-time implementations

- **Reyes**: Architecture to implement RenderMan
  - Designed for high-quality, non-real-time rendering

- **Streaming framework for rendering enables implementation and comparison of both pipelines**

Frame from *Monsters Inc.*
© Pixar Animation Studios, 2001
Summary

- Rendering in streaming framework provides flexibility and performance
- OpenGL and Reyes are both streaming apps
  - Focus of paper: comparison between two
- Contributions of paper:
  - Streaming framework for rendering
    - Enables multiple pipelines / hybrid pipelines
  - Algorithms for streaming implementation
  - Quantitative comparison between OpenGL and Reyes
Previous Work

- OpenGL: Segal and Akeley ’99
- Reyes: Cook/Carpenter/Catmull ’87
  - RenderMan: Upstill ’90, Apodaca and Gritz ’00
- Programmability and rendering:
  - Shade trees: Cook ’84
  - Programmable pipeline: Olano ’98 (dissertation)
  - RenderMan on OpenGL: Peercy et al. ’00
  - OpenGL with streams: Owens et al. ’00
  - Real-Time Shading Language: Proudfoot et al. ’01
  - Smash: McCool ’01
The Stream Programming Model

- **Streams**
  - Ordered sets of data elements of the same datatype
  - Datatype can be compound

- **Kernels**
  - Perform computation
  - Inputs/outputs are streams
  - Typical operation: apply function to each element in stream
  - Can be chained together
  - Expose parallelism
Programming Model Details

Limited control flow
- Goal: Exploit data-level parallelism
  - SIMD (single-instruction, multiple-data)
- Requirement: Simple control
  - Primary control structure: loop
  - No branches
- Conditional streams allow data-dependent operation

Kernels operate only on local data
- Goal: Fast kernel execution
- Requirement: Data must be close to functional units
  - Must structure program to avoid global accesses within kernels
  - No pointers, no global arrays within kernels
The Imagine Stream Processor

[Khailany et al., IEEE Micro Mar/Apr ’01]
Implementation

• Input stream divided into “batches”
• Batch loaded from memory to SRF
• Series of kernels run on input batch
• Output written back to memory

[Owens et al., Graphics Hardware ’00]
Why Stream Processing?

- Graphics tasks are stream tasks
  - Exploit parallelism, producer-consumer locality
  - Stream hardware is designed to:
    - Support lots of computation
    - Deliver high data bandwidth
    - SIMD nature of Imagine matches OpenGL/DirectX and Reyes shading models
- Goal: Design of efficient algorithms for the stream model results in efficient implementations in special-purpose hw
Stream Framework for Rendering

Object Space
- Application
- Command
- Per-Surface
- Tessellation
- Per-Vertex
- Primitive Assembly
- Per-Primitive
- Rasterization
- Per-Fragment
- Image Composition
- Per-Pixel
- Display

Image Space
- FB
- Pixel Ops

Texture Spaces
- Per-Texel

Texture Memory

From Akeley and Hanrahan, *Real-Time Graphics Architectures*
OpenGL in Streaming Model

- Application
- Command
- Per-Surface
- Tessellation
- Per-Vertex
- Primitive Assembly
- Per-Primitive
- Rasterization
- Per-Fragment
- Image Composition

- Vertex Program
- Primitive Assembly
- Clip/Project
- Rasterization
- Fragment Program
- Image Composition
- Display
Reyes in Streaming Model

- Application
- Command
- Per-Surface
- Tessellation
- Per-Vertex
- Primitive Assembly
- Per-Primitive
- Rasterization
- Per-Fragment
- Image Composition
- Per-Pixel
- Display

Subdivision (Dice/Split)
Vertex (Quad) Program
Sample
Image Composition
Summary of Pipeline Differences

- **Shading and texturing**
  - OpenGL: 2 shaders, 2 coordinate spaces
  - Reyes: Single shader, single coordinate space

- **Sampling vs. rasterization**
  - OpenGL: rasterizes arbitrary-sized triangles
  - Reyes: samples bounded-sized quads

- **Tessellation**
  - OpenGL: tessellates in host or at compile time
  - Reyes: tessellates dynamically as part of pipeline
Shading and Texturing

- **OpenGL**
  - Vertex shading: eye space
  - Fragment shading: screen space
    - Textures require filtering (mipmapping, 8 samples/access)
    - Imagine OpenGL: mipmapped scenes are > 2x slower than point-sampled
  - Factoring advantageous for large triangles, but must support two shading units

- **Reyes**
  - Vertex/quad shading: eye space
    - Coherent access textures: samples are properly filtered
  - Gain ability to shade before pixel coverage calculation: motion blur, depth of field
Sampling vs. Rasterization

- Sampling quads is simpler than rasterizing triangles
  - Quads have bounded size
  - Triangles can have arbitrary size

- Imagine implementations:
  - 8 Gouraud shaded primitives w/ identical coverage
  - Reyes sample: 100 cycles, 548 ops
  - OpenGL rasterize: 565 cycles, 2276 ops
    - More complex shaders need lots of interpolants
Tessellation

- **OpenGL**: compile time, or on host
- **Reyes**: runtime
  - Adaptive subdivision
  - Catmull-Clark subdivision surfaces
- Goals
  - Keep data structure on-chip
  - No global knowledge (i.e. binary dicing)
  - $O(\log n)$ storage for $n$ quads (depth first traversal)
- Most traditional subdivision algorithms inapplicable
  - Typically limit subdivision differences between levels
- Biggest problem: Ensuring no holes in surface
Ensuring Hole-Free Subdivision

- Quad on right: complete, and output
- Quad on left: must be subdivided
- Potential crack?
- Freeze edges once they fall beneath threshold
- Edges represented as edge equations
Performance

- Reyes scenes order of magnitude slower than OpenGL scenes

- OpenGL scenes:
  - Triangle sizes 2-12 pixels/triangle

- Why?
Runtime Results

- Avg. of 82% of Reyes runtime in subdivision
- Of remainder, about half in shading
- Subdivision produces many zero-frag quads
Reyes: Refining Subdivision

- Possible improvements
  - High-level backface culling
  - Intelligent splitting (x or y, not both)
  - Early quad kill

- Subdivision spectrum - adaptive to fixed
  - Our algorithm: fully adaptive
  - Non-adaptive “oracle” subdivision test:
    - Subdivision takes 10% of runtime
  - Ideal algorithm?
Conclusions

• Streaming is a natural way to describe programmable pipeline
  • Matches pipeline flow
  • Exploits concurrency and locality

• OpenGL and Reyes both fit into streaming framework
  • Framework supports either pipeline, or hybrid

• Reyes has several algorithmic advantages ...
  • Bounded size primitives, single shader, coherent textures, potential for more sophisticated effects ...

• ... but subdivision remains a challenge
Thanks to ...

- Stanford Flash Graphics group
- NVIDIA architecture group
- Kurt Akeley and Pat Hanrahan
- Kekoa Proudfoot and Bill Mark
- Matt Pharr
- Funding agencies: DARPA, Intel Foundation, MARCO