Real-Time Reyes
Programmable Pipelines and Research Challenges

Anjul Patney
University of California, Davis
This talk

• Parallel Computing for Graphics: In Action

• What does it take to write a programmable pipeline?
  – Many questions
  – Some answers

• We will focus on the Reyes pipeline
Graphics on parallel devices

- Over the years
  - Increasing performance
  - Increasing programmability

- How is that useful for real-time graphics?
  - Improve existing pipeline
  - Redesign the pipeline
Redesign the pipeline

• An Exploration
  – May not be the answer for everyone

• My Goals
  – Interactive performance
  – High visual quality

• How should I choose a pipeline?
My real-time pipeline

• An improvement in real-time rendering
  – Build around shading
  – Remove existing rendering artifacts

• Desired features
  – High-quality anti-aliasing
  – Realistic motion-blur, depth-of-field, volume effects
  – Global Illumination
  – Order-independent transparency
Reyes

• Introduced 1987

• Photorealistic rendering
  – Smooth surfaces
  – Complex shading, lighting
  – Depth of field, motion blur
  – Order-independent blending

• Designed for offline use
  – But favors SIMD
Real-Time Reyes

Input

Geometry

Shading

Sampling

Composition

Final pixels
Step 1: Geometry

Convert input surfaces to micropolygons

Input

Geometry

Shading

Sampling

Composition

Final pixels
Step 2: Shading

Decide colors for grid micropolygons
Step 3: Sampling

Collect stochastic samples of micropolygons
Step 4: Image Composition

1. **Input**
2. **Geometry**
3. **Shading**
4. **Sampling**
   - Blend samples to get colors
   - Combine colors to get pixels
5. **Composition**
   - Final pixels
Step 1: Geometry

Convert input surfaces to micropolygons
Input

- Higher-order surfaces
  - Bézier surfaces
  - NURBS
  - Subdivision surfaces
- Displacement-mapped
- Animated

Hand image courtesy: Tamy Boubekeur, Christophe Schlick
Task – Split and Dice

- Adaptively subdivide the input surface
- Tessellate when small enough
- Rinse and repeat
Challenges

• Recursive, irregular computation
  – Bad for parallelism, SIMD

• Too many micropolygons
  – Limited memory
Ideas

• Breadth-first Traversal

• Bucket Rendering
Works in real-time!

- Killeroo: 11532 Patches
- Split and dice in 9.8 ms
- 29.69 fps at 512 x 512
- Parametric surfaces only
- Subdivision cracks

Patney and Owens, 2008

Killeroo Model Courtesy Headus Inc.
Geometry Output – Unshaded Grids

1 Grid

Micropolygons
Step 2: Shading

Decide colors for grid micropolygons

Input

Geometry

Shading

Sampling

Composition

Final pixels
Task

• Run shader(s) for each grid
  – Displacement
  – Surface
  – Light
  – Volume
  – Imager

• Good behavior
  – Highly parallel, SIMD friendly
  – Good locality behavior

Image courtesy: Saty Raghavachary
Challenges

• Massively parallel is great
  – But is it good enough?

• Shaders can be complex
  – Too many instructions, conditionals
  – Global illumination
  – File I/O
  – Arbitrary texture fetches
Ideas

- Cache redundant computation
  - Across a grid
  - Across frames

- Architectural support
  - Virtual memory
Interactive Relighting

- **Lpics** [Pellacini 2005]
  - Cache image-space samples
  - Interactive feedback
  - Manual pre-processing

- **Lightspeed** [Ragan-Kelley 2007]
  - Shader caching
  - Interactive preview
  - Slow pre-processing

Images belong to respective paper authors
Output – Shaded Grids
Bust: Many micropolygons
Step 3: Sampling

Collect stochastic samples of micropolygons

Input

Geometry

Shading

Sampling

Composition

Final pixels
Task
Samples

1
2
3
4
5
Challenges

• Generate samples
  – Jittered grid
  – Parallel Poisson sampling [Wei 2008]

• For each sample, find all intersecting micropolygons
  – Raycast or Rasterize?

• Output: A (depth-sorted?) list of samples
Step 4: Image Composition

- Blend samples to get colors
- Combine colors to get pixels

Input

Geometry

Shading

Sampling

Composition

Final pixels
Task 1: Blend
Task 2: Filter to get pixel colors
Challenges

- Represent the irregular work-list
  - Traditionally: linked-list per sample (arbitrary size)

- Sort and Reduce
  - Unequal work-items

- Generate and apply filter kernels
  - Box
  - Gaussian
Stencil-Routed A-buffer

Myers and Bavoil, NVIDIA, 2007
Summary: What is easy?

- Can be parallelized well
- Highly Parallel, SIMD friendly
- Good locality behavior
- Parallel Poisson Sampling

Diagram:
- Input
  - Geometry
  - Shading
  - Sampling
  - Composition
  - Final pixels
Summary: What is hard?

Subdivision Surfaces, cracks
Long shaders
File I/O, arbitrary textures
Raycast or Rasterize?
Sort and reduce irregular work-lists

Input

Geometry

Shading

Sampling

Composition

Final pixels
Conclusion

• Reyes is promising for real-time
  – Enables natural high-quality rendering
  – Portions map well to current hardware

• But there are challenges
  – Everything isn’t easy to implement
  – Architecture limitations

• Lots of interesting questions
Thanks to

- Course organizers
- Prof. John Owens, Shubho Sengupta
- Tim Foley
- Per Christensen
- Matt Pharr
Realistic Effects using Reyes

- Motion-blur
  - A stochastic time for each sample
  - Move micropolygons accordingly

- Depth-of-field
  - A stochastic lens position for each sample
  - Render micropolygons accordingly

- Take many samples to ensure quality
- Adjust screen bound during subdivision
Global Illumination with Reyes

- Traditional: shadow maps, environment maps

- Raytracing [Christensen et al. 2006]
  - Multi-level geometry cache
  - Ray-differentials to select appropriate resolution

- Effects taken care of
  - Shadows and reflections
  - Ambient Occlusion
My version of the world - today

- Many SIMD Cores (16-32)
- Precious memory bandwidth
- Data-parallel (SPMD)
My version of the world - tomorrow

- More cores, still SIMD (8-16)
- Memory bandwidth still precious
  - But flexible access behavior
- Data-Parallel and Task-Parallel