Adaptive Real-Time Rendering

Fredo Durand
MIT
Digression: Photography & Video

- Enhancement
  - Contrast management
  - Flash photography
  - Capture style & skills from professionals

- Tools
  - Non-linear filtering
  - Gradient domain
  - Statistical analysis

- High Computational cost
- The Image is a stream
e.g. Contrast management

- Real world: high range of intensity (often 1: 100,000)
- Display or print have a limited contrast (1:50)
Live demo

- 1.6 GHz Pentium 4
Adaptive Real-Time Rendering

Fredo Durand
MIT
Real Time Rendering: Context

- Send geometry: mostly polygons
- Rasterization
- Visibility (z buffer)

- Appearance
  - Programmable “shaders”
Quality: amazing
Rules of the game

- Real time is Important
- Highly parallelizable
- We can degrade quality
-Multiple platforms/architectures
  - PC, PlayStation, GameCube, Xbox
- Various levels of parallelism
- Before an application (e.g. game) is shipped, a lot of optimization is affordable
Goals in Real-time graphics

• Better
  – Nicer appearance, better lighting

• Faster
  – Culling (do not draw what is hidden)
  – Simplification (for distant objects)
  – Optimizations (hardware specific)

• Easier
  – For the programmer
  – For the artists
Better: e.g. Fake soft shadows

With Eric Chan

17 ms  shadow map

142 ms  bicubic filter

22 ms  Our method (\(t = 0.02\))

24 ms  Our method (\(t = 0.08\))
Faster

- Simplification: cheaper model for distant objects

- Culling
  - Do not waste resources on hidden objects

- Optimization
  - hardware specific
Hot Topic: Shader Simplification

- Adapt shader to hardware:
- Manipulation of expression tree
- Lossless: Hardware virtualization
  - [Chan et al. 02]
- Lossy: Degrade shader quality
  - Find simplification operations (peephole)
  - Predict impact of simplification
Simplification metric

• Objective: Geometric, L2
• Subjective: perceptual
  – Use psychophysics
  – Just Noticeable difference
  – Masking (frequency content)
  – Saliency
  – Ad hoc developer judgment
Faster

- Simplification: cheaper model for distant objects

- Culling
  - Do not waste resources on hidden objects

- Optimization
  - hardware specific
Visibility culling – many choices

- point-based / From region (cells)
- Occluders / Portals
- Object space / Image space
Additional degrees of freedom

• Spatial hierarchy
  – Which hierarchy?
  – Which depth?

• Latency issues
  – E.g. occlusion queries:
    • Ask graphics hardware if object is visible
    • Large delay
  – Importance of scheduling
Faster

- Simplification: cheaper model for distant objects
- Culling
  - Do not waste resources on hidden objects
- Optimization
  - hardware specific
Faster: Low-level optimization

• Improve bandwidth/caching
  – Triangle strips
  – Vertex cache
  – Vertex arrays

• Avoid context switch
  – Sort by material
  – But conflicts with spatial hierarchy
Dealing with complexity

- Better and faster conflict with easier

=> Put more intelligence in the system
  - Inspiration from compilation, optimization, linear algebra packages
Adaptive real-time rendering

• Problem
  – Writing a fast rendering engine is a black art
  – Performances depend on
    • The hardware configuration (CPU, GPU bandwidth, memory)
    • The scene properties
  – It is impossible to optimize for all configurations

• Solution: automatic optimization and self-adaptive systems
Adaptive real-time rendering

- High-level
  - Choose acceleration strategies
  - Optimize parameters
  - Scheduling, latency (e.g. culling queries)
- Low level
  - Optimize how geometry is sent
  - Sort by material, find a smart order of triangles for better caching
- Hardware level
  - Reconfigure hardware
  - E.g. shadows in Doom 3 make most of the programmable transistors idle
Rules of the game

- Real time is Important
- Very repetitive computation
- We can degrade quality
- Multiple platforms/architectures
  - PC, PlayStation, GameCube, Xbox
- Various levels of parallelism
- Before an application (e.g. game) is shipped, a lot of optimization is affordable
Thanks
Invitation

- Opportunities for much architecture and compiler research
- One big difference: quality can be degraded
Real-time shaders

• Capabilities vary tremendously
  – Some hardware is not programmable
  – Different set of instructions
  – Different control structures
  – Different speed

• Hard to develop for all platforms
• Developers target for 1 or 2 platforms
Goal

• Systems that can adapt
  – To the hardware resources
  – To the scene
• Real-time
  – Set the minimal frame rate
• Adaptation
  – Tune the parameters
  – Choose the algorithms
  – Static and dynamic
• Longer-term: distributed context
Degrade the image to reach real time

- Frame rate is more important than image quality
- Generalize the notion of levels of details
- Study precisely how framerate varies
- Prediction of rendering time
- Control problem
- Perceptual metric to estimate image degradation
Pervasive computing makes it harder!

- Very different resources
  - PDA, laptop, desktop
- Distributed
  - Maybe the framebuffer is on one machine, the display on the other machine, etc.
  - Bandwidth and latency must be taken into account
- Load varies
  - Dynamically adapt to load variation
Challenges

- Flexible architecture
- More flexible acceleration techniques
- Shader simplification and levels of detail
- Transitions between levels of details
- Speed prediction (statistics, law)
- Optimize the algorithms and parameters
High-performance compilation

- Better than scientific computing ;-) 
- Industry demand 
- Performance matters 
- Programs are smaller 
- Analysis and profiling 
- Result can be changed
Our approach

• 2-scale decomposition of intensity
Feedback & optimization

- If we know the final image, we can optimize for it
- Reduction operators
- Delay streams and others
  - It is easier to optimize when you know the results
- Adaptive, perception, masking
  - Masking
  - Gaze
What is the situation?

- Everything needs to be precisely targeted
- Usually choose one or two target platform and optimize manually
- Each programmer have their favorite algorithms
- Tedious
- Sub-optimal for most platforms
- Real-time is not ensured
Simplification: Billboard clouds

(Decoret, Durand, Sillion and Dorsey)

- Approximate shape by a set of plane
- Project model on these planes => textures
Faster: Low-level optimization

- Improve bandwidth/caching
  - Triangle strips
  - Vertex cache
  - Vertex arrays
- Avoid context switch
  - Sort by material
- Electronic Art uses an art-asset compiler