Hardware Shading: State-of-the-Art and Future Challenges

Hans-Peter Seidel
Wolfgang Heidrich

Max-Planck-Institut für Informatik
Saarbrücken, Germany
Graphics Hardware

**Hardware is now fast enough**
- for complex geometry
- for multipass rendering

**Multipass rendering:**
- render objects multiple times with different shading parameters and textures to achieve complex shading
- hardware treated as a SIMD processor
- graphics pipeline becomes complex instruction set
- e.g. Quake 3: sometimes 5 passes and more
New hardware features

- speeding up rendering by reducing number of passes
  - e.g. multitexturing reduces total bandwidth
  - make complex algorithms feasible: e.g. extensions for bump mapping

- sometimes completely new possibilities
  - e.g. dependent textures/pixel textures for environment mapped bump mapping
Applications of Complex, Interactive Shading

**Engineering and design**
- light planing, interior design, luminary design

**Marketing and (e-)commerce**
- product and material catalogs, online shops

**Entertainment**
- movie special effects, games

**Visualization**
- medical imaging, scientific visualization
Questions

**What features are necessary?**
- what does it take to support the typical rendering algorithms?

**What features are useful?**
- significant performance improvements for a variety of algorithms

**Programming support**
- make it easy to develop for graphics hardware
Overview

Hardware shading algorithms
- materials, shadows, bump maps, volume rendering
- discussion of shading algorithms

Recent hardware developments
- increased flexibility for geometry and fragment processing

Shading languages for graphics hardware
- motivation, required features, open problems

Hardware support for shading languages
Hardware Shading Algorithms

Examples for multipass rendering:

- realistic materials for local and global illumination
- shadow maps
- volume rendering and participating media
- bump mapping
Rendering of Realistic Materials

**Local illumination (Heidrich ’99, Kautz’99):**

- use textures to store sampled BRDFs and reflection models
- break down high-dimensional tables to lower-dimensional terms (textures)
- make sure texture coordinates are easy to compute
Example:
Torrance-Sparrow Model

Torrance-Sparrow (’67):
- \( f_r = F(\theta)D(\delta) \cdot G(\alpha, \beta)/(\pi \cos \alpha \cos \beta) \)

Sample factors
- store as 2D textures

Texture coordinates:
- reparameterize terms over cosines of angles
- allows for hardware-based texture coordinate generation
Torrance-Sparrow Model
Banks Model

Anisotropic model (Banks ’94, Heidrich ’98)
Sampled BRDFs

Factorization of sampled BRDFs (Kautz ’99)

- measured/simulated BRDF data
- rendering similar to analytic models
Sampled BRDFs With Textures

Combining BRDFs with diffuse textures
Environment Maps With Complex Materials:
Hardware Shading Algorithms

Examples for multipass rendering:
- realistic materials for local and global illumination
- shadow maps
- volume rendering and participating media
- bump mapping
Shadow Maps

Shadow maps using the alpha test
Shadow Maps

*Alpha shadow map:*
Shadow Maps

*Shadow map applied as projective texture*
Shadow Maps

$Light\ source\ z\ as\ 1D\ alpha\ texture$
Shadow Maps

Subtract projective texture image from 1D texture image
Hardware Shading Algorithms

Examples for multipass rendering:

- realistic materials for local and global illumination
- shadow maps
- volume rendering and participating media
- bump mapping
Volume Rendering

Texture-based volume rendering
(Cabral ’94, Westermann ’98, Salama’00)

- back-to-front rendering of textured volume slices
Volume Rendering

Texture-based volume rendering
(Cabral ’94, Westermann ’98, Salama’00)

Images: Hastreiter ’98
Participating Media

Volume rendering for streaks of light (Everitt ’00, Dobashi ’00)

Image: Everitt ’00

Image: Dobashi ’00
Examples for multipass rendering:

- realistic materials for local and global illumination
- shadow maps
- volume rendering and participating media
- bump mapping
**Bump Mapping**

*Tangent space bump mapping (Peercy ’97):*
- store normal textures
- use with interpolated local coordinate frame (Schilling ’97)
- illumination is function of dot products

*Environment mapped bump mapping*
- needs dependent texture lookups

*Space-varying BRDFs (Kautz ’00)*
Bump Map Shadows

Horizon maps (Max ’88)
- pre compute and store horizon for each point in a height field

Hardware implementation (Sloan ’00)
- original horizon map data structure
- several rendering passes

Single pass multi-texturing (Heidrich ’00)
- using different horizon representation (ellipse)
Bump Map Shadows

center \((c_x, c_y)\)
axis\((a_x, a_y)\)
r_x, r_y

Lit
directions

Shadowed
directions

Wolfgang Heidrich
Bump Map Shadows

Project light direction into tangent plane

Transform projected point into new coordinate system

- given by center and main axes of ellipse \((l_x', l_y')\)

Shadow test:

\[
1 - \frac{(l_x')^2}{r_x^2} = \frac{(l_y')^2}{r_y^2} \geq 0
\]
Discussion

**Common theme: multipass rendering**

- for texture generation
  - e.g. generating shadow or environment maps
  - typically: simulation of global effects
- for texture application
  - combining results of different textures and shading algorithms in the frame buffer
  - typically: complex local effects
Discussion - Implications

**Bandwidth**
- multiple transmission of geometry
- multiple read/writes to framebuffer
- sometimes read back of framebuffer

**Geometry**
- different passes require different per-vertex params.
  - *color*, *texture coordinate*, *normal*...
- require flexible way of generating these parameters
Recent Hardware Development

**Increased flexibility of rendering pipeline**
- mostly to reduce number of passes

**Rasterization stage: multi-texturing**
- reduces total required bandwidth for most algorithms requiring multiple textures
- but becomes bottleneck for fragment processing stage of rendering pipeline if many textures are used
- texture shaders (McCool '99), register combiner (NVIDIA '99): use the available time to perform more complex operations with the looked up values
Recent Hardware Development

Diagram: NVIDIA

Wolfgang Heidrich
Recent Hardware Development

**Geometry stage**

- texture coordinate generation
- programmable geometry (DirectX 8)
  
  - [http://www.gamasutra.com/features/index_gdc.htm](http://www.gamasutra.com/features/index_gdc.htm)
Dependent Texture Lookups

Useful for:
- environment mapped bump mapping
- light transport for indirect illumination (Heidrich '00)
- line integral convolution (LIC, Heidrich '99)
- ...

Wolfgang Heidrich
Problems With New Hardware Features

**Problem:**
- extensions hard to program for
- no debugging support
- non-standard: capabilities vary with specific hardware
  - have to adapt algorithms to individual graphics boards

**Solution:** high-level shading language
Shading Languages: RenderMan

**RenderMan shading language** *(Hanrahan '90)*

- standard in offline applications
- e.g. Toy Story

**RenderMan on OpenGL** *(Peercy '00)*

- RenderMan shading language with standard OpenGL+ pixel textures*
  floating point framebuffer

Images: Peercy '00
Feasible Languages

For hardware shading:

- use sampled representations of complex functions
- shading languages for combining textures

Existing systems:

- Quake 3 shading language
- Stanford programmable shading project
  - http://graphics.stanford.edu/projects/shading
- SGI interactive shading language (ISL)
  - http://www.sgi.com/software/shader
Research Issues

**Immediate mode vs. scene graph**
- problem: no widely accepted scene graphs available

**Compiler technology**
- code generation for very complex instruction sets

**Working around the brick wall**
- managing partial results and multiple passes
Beyond RenderMan?

Management of texture generation passes
- language for describing generation of environment maps, shadow maps, light maps...

New kinds of interesting shaders
- cellular automata (e.g. clouds)
- reaction-diffusion shaders (e.g. animal skin)
- require side effects (textures evolving over time)
- desirable: hardware convolution
Hardware Support for Procedural Shading

What can hardware developers do?

- orthogonal feature sets
- cost/precision models for hardware operations
- avoiding the brick wall
  - managing intermediate results
  - micro-threading (Mark '00, McCool '00)
Summary

Complex hardware shading is now possible
- multi-pass algorithms and new hardware features

Recent hardware developments
- reduce number of passes, increase performance
- hard to program

Shading languages for graphics hardware
- make programming easier
- a lot of research to be done
- need hardware support
Acknowledgments

**MPI rendering group:**
- Stefan Brabec, Katja Daubert, Jan Kautz, Philipp Slusallek

**Stanford shading group:**
- Bill Mark, Kekoa Proudfoot, Pat Hanrahan

**SGI procedural shading group:**
- Marc Olano, Marc Peercy

**Images from:**
- Michael McCool, Mark Kilgaurd, Cass Everitt, Yoshinori Dobashi