Which GPU-based Algorithms Cut It?

Jörn Loviscach

GAME RESEARCH

GEMS

GCDC WORKSHOP LEIPZIG TUESDAY 22 AUGUST 2006
What GPUs could do in games

Some examples ...
What GPUs could do in games

20 G samples at 60 fps, no popping, no hick-ups:

- LOD pyramid; nested grids around the viewer
- Morphing between grids; continuous reloading, decompression

What GPUs could do in games

Ambient occlusion:
- Sum (groups of) polygons as disk-like blockers
- Multi-pass to take care of blockers blocking blockers

Bunnel. Dynamic ambient occlusion and indirect lighting. GPU Gems 2
What GPUs could do in games

Shadows maps without jaggies:
- Render depth map plus shadow silhouettes
- Pixel shader: connect the dots

Sen, Cammarano, Hanrahan. Shadow silhouette maps. SIGGRAPH 2003
What GPUs could do in games

OK, but why don’t GPUs actually do this in games?
What GPUs *could* do in games

Completely reorganize assets and render loop?

How to create and manage blockers?

Trade instructions for bandwidth?
Outline

- What’s needed?
- What’s available?
- So, which algorithms do cut it?
- Call to action
- Epilog
What’s needed?

That is:

What are the requirements of game development practice?
What’s needed?

Buzzwords for the press?
What’s needed?

Great results with lightweight methods!

Which means ...
What’s needed?

... hard requirements on:

- **Timing**
  (# of instructions, state changes, dependent tex reads, multithreading, ...)

- **Memory**
  (textures, vertices, off-screen rendering, ...)
What’s needed?

Robustness: no tweaking needed, just works

- Unexpected things may happen in a game (shadow acne?)
- Unexpected ideas may crop up during its design (100 NPCs in a swimming pool?)
What’s needed?

Flexibility: h/w ranges from a shared-memory GPU to a multicore CPU with two graphics cards, XGA

- Scalability (e.g., vertex/pixel workload balance)
- Fallbacks
What’s needed?

Integration
with game engine and/or existing code:

- Shadows?
- Collision, physics?
- Resource Management?
- ...

J. Loviscach: Which GPU-based algorithms cut it?
What’s needed?

Workflow integration (1):
Reuse of content; asset management; integration with DCC tools
Workflow integration (2):
Testing: algorithms, benchmarks, gameplay, ...
on a range of hardware!
What’s needed?

Workflow integration (3):
Division of labor between artists and programmers
What’s needed?

In short:
- Lightweight processing
- Robustness
- Flexibility
- Run-time integration
- Workflow integration
What’s available?

That is:

What are the results of computer graphics research?
Focus on solitary algorithms:

- One graphical (or non-graphical) effect
- Not 1000 different things to happen at 60 fps
- No integration or systems
What’s available?

Publish or perish:

- Least Publishable Unit? (Owen)
- Applied and inter-disciplinary work disencouraged as “soft”
What’s available?

Papers on applications:
- Bunches of block diagrams
- Often sketchy
What’s available?

Need to do something arcane and/or sophisticated because all simple things have been done.

Really?
What’s available?

Some of those “simple” things:

- Artifact-free noise?
- Sharp textures?
- Volume-preserving soft skinning?
- Carefree shadows?
What’s available?

Attempts to make the GPU do tasks it wasn’t designed for:

- Is this going to bite back?
- Ugly hack, till the next chip generation comes around? (Testing? Maintaining?)
Wrap-up: Research vs. Practice

No more gems? ;-)

J. Loviscach: Which GPU-based algorithms cut it?
So, which algorithms do cut it?

Some examples with their benefits and issues:

- Relief mapping
- Mipmapping normal maps
- Bi-level textures
- BRDF-Shop
- PRT
So, which algorithms do cut it?

Replace normal maps by virtual geometry:
- Ray casting in the pixel shader
- Convert normal maps to height maps

cf. Parallax Occlusion Mapping etc.

Policarpo, Oliveira. Rendering surface details in games with relief mapping using a minimally invasive approach. ShaderX4
So, which algorithms do cut it?

Relief mapping (1):
- Convert normal to height map: automatic step in the build process
- Doom3 demo implementation
- Easily switchable option
- Extensible: multilayer, curved base
So, which algorithms do cut it?

Relief mapping (2):
- Best if surface pushed back: expanded models (Keep two versions?)
- Aliasing near, far, at grazing angles
- Computational load: approx. 150 PS instructions
- z set for intersections; texkill for silhouettes: early-z switched off
So, which algorithms do cut it?

Relief mapping:
- Lean processing  ❓
- Robustness  ❓
- Flexibility  ✓
- Run-time integration  ✓
- Workflow integration  ✓
So, which algorithms do cut it?

Better MIP-mapping for normal maps:
- Denormalization of interpolated normals indicates their local divergence
- Model by a Gaussian distribution
So, which algorithms *do* cut it?

Mipmapping normal maps:
- Fast and simple: just one additional 2D texture retrieval
- Issues with locally varying NTB frames
- Helper texture depends on shinyness
  - No local variation?
  - Asset management?
So, which algorithms *do* cut it?

Mipmapping normal maps:
- Lean processing ✓
- Robustness ✓
- Flexibility ✓
- Run-time integration ✓
- Workflow integration ✓
So, which algorithms do cut it?

Vector-quality textures without the cost:
- Apply soft thresholding
- Optimize textures offline for best result

Loviscach. Rendering road signs sharply. Game Programming Gems 6
So, which algorithms do cut it?

Bi-level textures:
- Jaggies and MIP-mapping handled
- Runs everywhere:
  - 12 PS instructions, 1 tex read
- Compare to Perfect Hashing (SIGGRAPH 2006): 40 instructions, up to 5 tex reads
- Some manual adjustments with optimizer software required
- Manage hi-res and optimized textures
So, which algorithms do cut it?

Bi-level textures:
- Lean processing  ✓
- Robustness  ✓
- Flexibility  ✓
- Run-time integration  ✓
- Workflow integration  ?
So, which algorithms *do* cut it?

Define complex reflective behavior by painting:

- Specialized tools
- Option: restrictions from optical physics

Colbert, Pattanaik and Křivánek. BRDF-Shop: Creating physically correct bidirectional reflectance distribution functions. IEEE CG&A 26, 2005
So, which algorithms do cut it?

BRDF-Shop:
- Integrated into Maya, real-time preview
- How to create spatially varying behavior, i.e., textures?
- No run-time component provided?
So, which algorithms do cut it?

BRDF-Shop:
- Lean processing  
- Robustness  
- Flexibility  
- Run-time integration  
- Workflow integration  

- ✓  
- ×
So, which algorithms *do* cut it?

Encode diffuse light interreflections and SSS into a 3D model

- Spherical harmonics (SH) describe low-frequency variations
- Processing can mostly be done directly in SH base

Sloan, Kautz, Snyder. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. SIGGRAPH 2002
So, which algorithms *do* cut it?

Precomputed Radiance Transfer:
- Precomputation:
  - Software included
  - Adaptive tessellation
- Authoring & runtime software
  - Part of DirectX 9, but what about other platforms?
  - Fallback if performance gets critical?
- About 100 VS instructions;
  need enough triangles
- US patent applications
So, which algorithms do cut it?

Precomputed Radiance Transfer:

- Lean processing ✓
- Robustness ✓
- Flexibility ?
- Run-time integration ✓
- Workflow integration ✓
So, which algorithms do cut it?

Conclusion:

Even algorithms that look promising have their share of issues.
Call to action
Call to action

For the scientists:

- Think about systems. Create better ones?
- Use standard DCC software and game engines
- Handle the ugly details
- Publish “Lessons learned,” i.e., negative results
Call to action

For the developers:

- What are the day-to-day issues of game production?
- How would you like to create games for tomorrow’s platforms?

Let the researchers know!
Call to action

For the developers (cont’d):

- Efficient apps of GPUs need more than geometry: advanced linear algebra, PCA, harmonic functions, ...
- This is where researchers shine!
Epilog

GPU programming: To where from here?

- Z buffer till the end of days?
- Ray tracing?
- Point-Based Rendering?

Botsch, Hornung, Zwicker, Kobbelt. High quality surface splatting on today’s GPUs. EG Symp. on Point-Based Graphics 2005

Epilog

- How to cope with radical changes?
- Long-term investment: start abstracting today to save your code and content
- Today’s hack may be tomorrow’s built-in feature
Thanks for your attention.

Questions?
Notice required by the manufacturer of the clip art:

Copyright © 2006 Jörn Loviscach
und dessen Lizenzgeber.
Alle Rechte vorbehalten.