Z-Buffer Optimizations

Patrick Cozzi

Analytical Graphics, Inc.

Overview


Z-Buffer Review


Hardware: Early-Z


Software: Front-to-Back Sorting


Hardware: Double-Speed Z-Only


Software: Early-Z Pass


Software: Deferred Shading

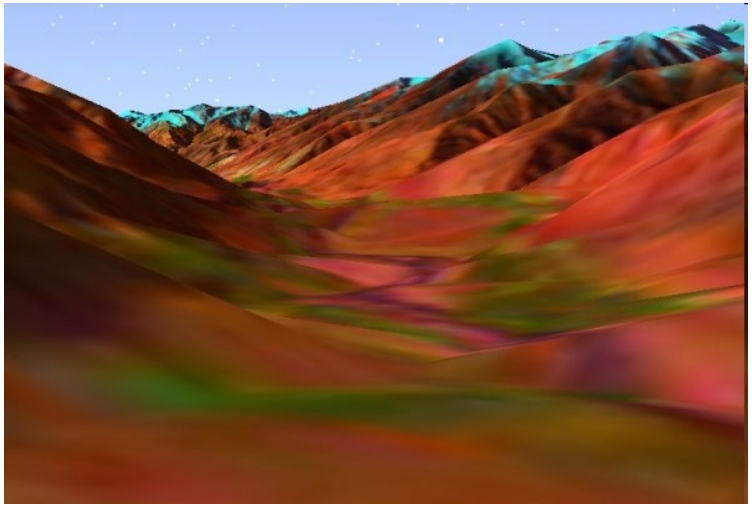

Hardware: Buffer Compression


Hardware: Fast Clear


Hardware: Z-Cull


Future: Programmable Culling Unit

Z-Buffer Review



- Also called Depth Buffer
- Fragment vs Pixel
- Alternatives: Painter's, Ray Casting, etc

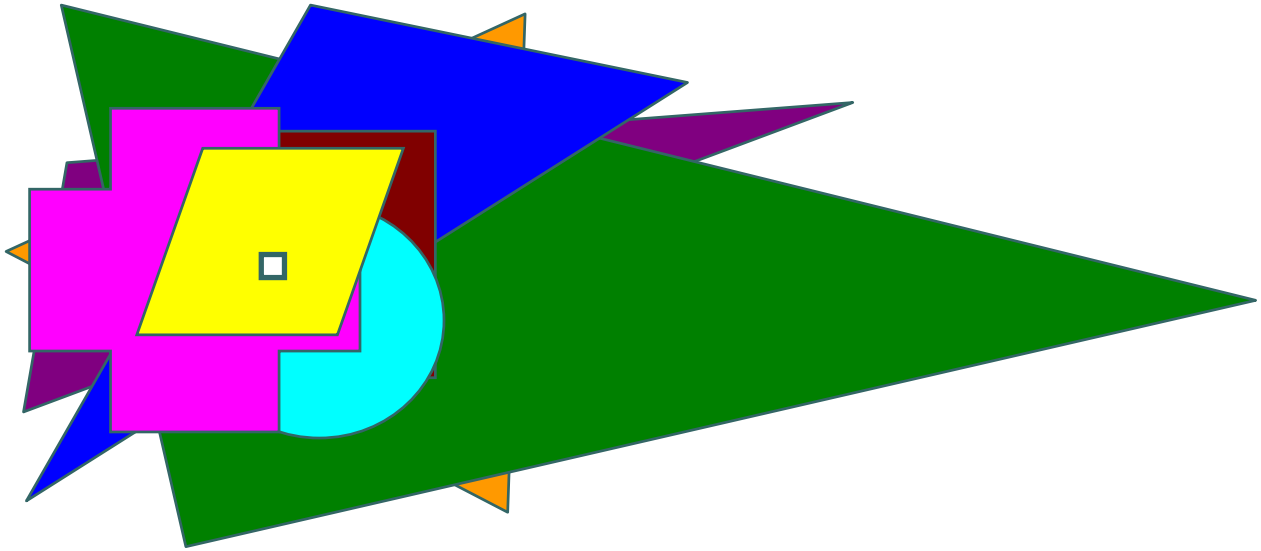
Z-Buffer History

- “Brute-force approach”
- “Ridiculously expensive”

- Sutherland, Sproull, and, Schumacker, “A Characterization of Ten Hidden-Surface Algorithms”, 1974

Z-Buffer Quiz

- 10 triangles cover a pixel. Rendering these in random order with a Z-buffer, what is the average number of times the pixel's z-value is written?



See Subtle Tools Slides: erich.realtimerendering.com

Z-Buffer Quiz

- 1st triangle writes depth
- 2nd triangle has 1/2 chance of writing depth
- 3rd triangle has 1/3 chance of writing depth

- 1 + 1/2 + 1/3 + …+ 1/10 = 2.9289…

See Subtle Tools Slides: erich.realtimerendering.com

Z-Buffer Quiz

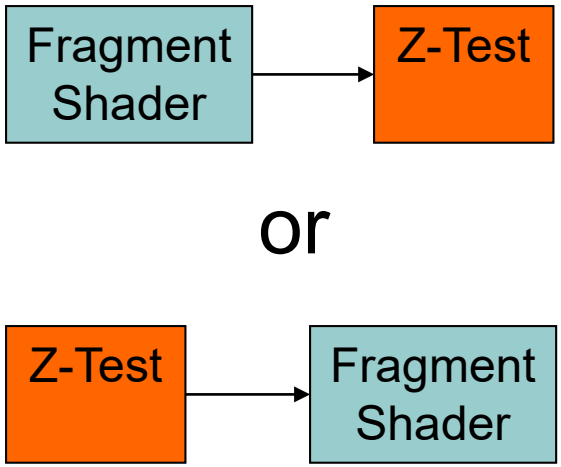
Harmonic Series

# Triangles	# Depth Writes
1	1
4	2.08
11	3.02
31	4.03
83	5
12,367	10

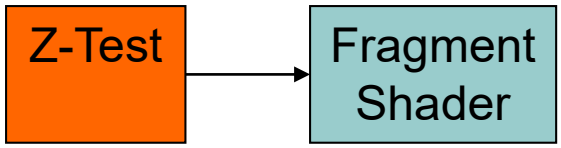
See Subtle Tools Slides: erich.realtimerendering.com

Z-Test in the Pipeline

- When is the Z-Test?

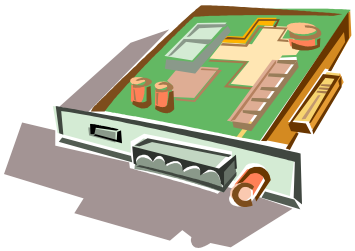


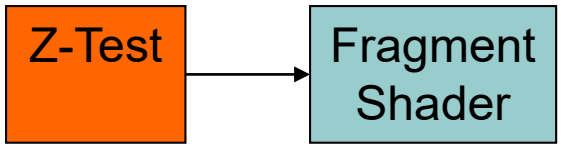
Early-Z



- Avoid expensive fragment shaders
- Reduce bandwidth to frame buffer
  - Writes not reads

Early-Z



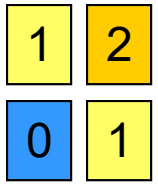


- *A*utomatically enabled on GeForce (8?) unless
  - Fragment shader discards or write depth
  - Depth writes and alpha-test are enabled
- Fine-grained as opposed to *Z-Cull*.
- ATI: “Top of the Pipe Z Reject”

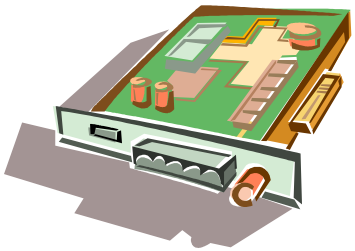
See NVIDIA GPU Programming Guide for exact details

Front-to-Back Sorting

- Utilize *Early-Z* for opaque objects
- Old hardware still has less z-buffer writes
- CPU overhead. Need efficient sorting
  - Bucket Sort
  - Octtree
- Conflicts with state sorting



Double Speed Z-Only

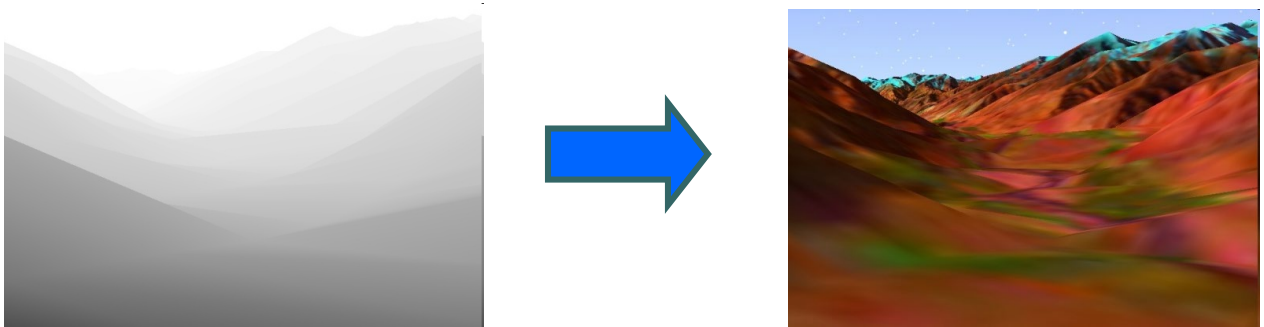


- GeForce FX and later render at double speed when writing only depth or stencil
- Enabled when
  - Color writes are disabled
  - Fragment shader discards or write depth
  - Alpha-test is disabled

See NVIDIA GPU Programming Guide for exact details

Early-Z Pass

- Software technique to utilize *Early-Z* and *Double Speed Z-Only*
- Two passes
  - Render depth only. “Lay down depth” – *Double Speed Z-Only*
  - Render with full shaders – *Early-Z* (and *Z-Cull*)



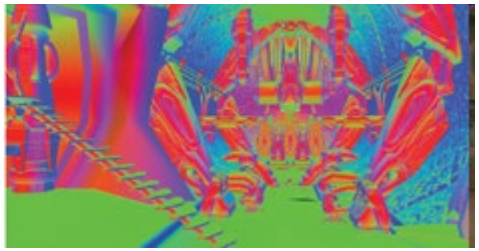
Deferred Shading

- Similar to *Early-Z Pass*
  - 1st Pass: Visibility tests
  - 2nd Pass: Shading
- Different than *Early-Z Pass*
  - Geometry is only transformed once

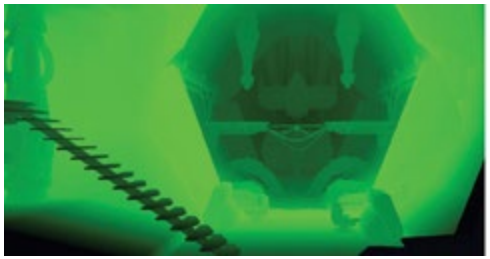
Deferred Shading

- 1st Pass
  - Render geometry into *G-Buffers:*

Fragment Colors



Normals



Depth

Edge Weight

Images from Tabula Rasa. See Resources.

Deferred Shading

- 2nd Pass
  - Shading == post processing effects
  - Render full screen quads that read from *G-Buffers*
  - Objects are no longer needed

Deferred Shading

- Light Accumulation Result

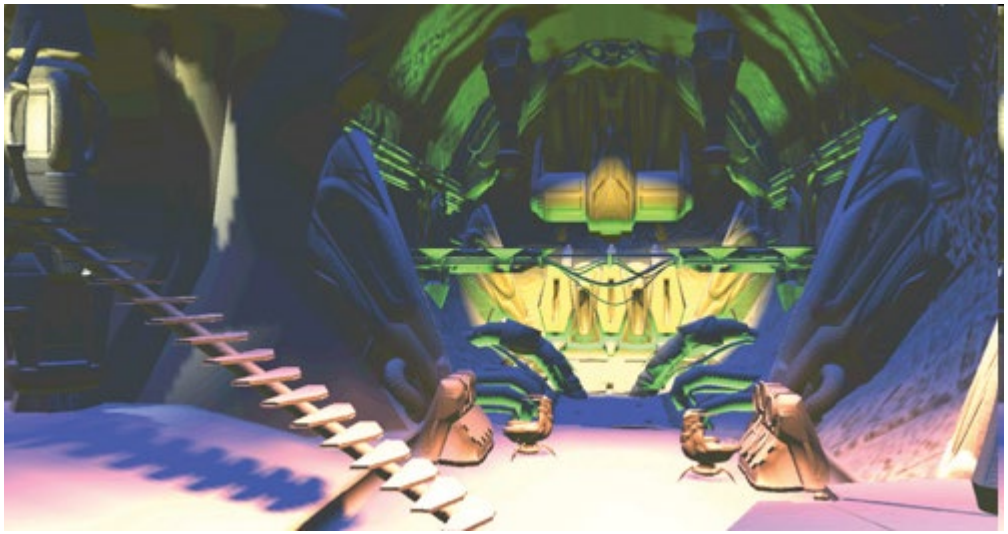
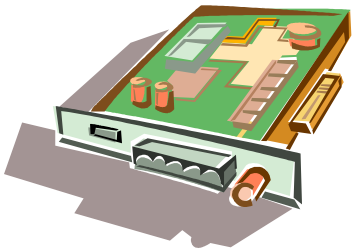


Image from Tabula Rasa. See Resources.

Deferred Shading

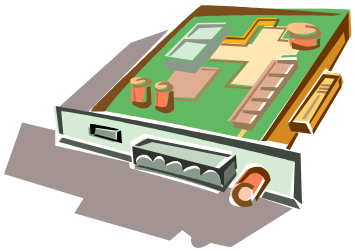
- Eliminates shading fragments that fail Z-Test
- Increases video memory requirement
- How does it affect bandwidth?

Buffer Compression



- Reduce depth buffer bandwidth
- Generally does not reduce memory usage of actual depth buffer
- Same architecture applies to other buffers, e.g. color and stencil

Buffer Compression

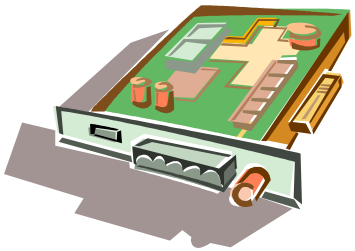


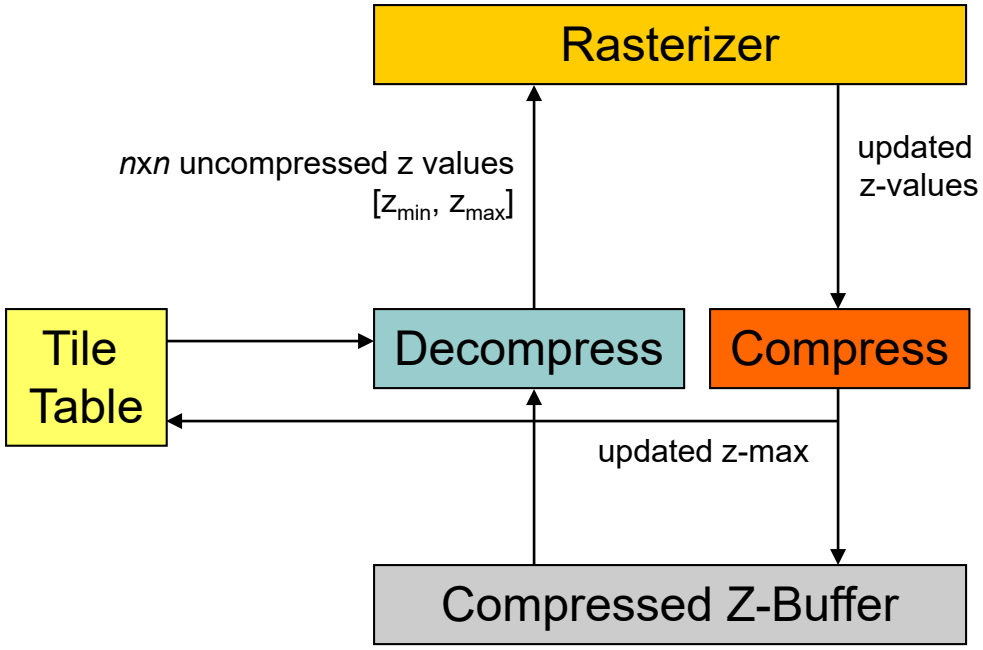
- Tile Table: Status for *n*x*n* tile of depths, e.g. *n*=8
  - [state, z_{min}, z_{max}]
  - state is either *compressed*, *uncompressed*, or *cleared*

[*uncompressed*, 0.1, 0.8]

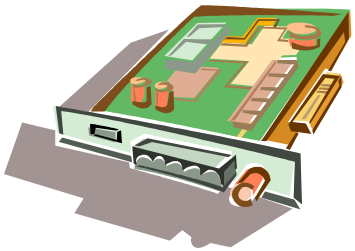
0.1	0.5	0.5	0.1
0.5	0.8	0.8	0.5
0.5	0.8	0.8	0.5
0.1	0.5	0.5	0.1

Buffer Compression



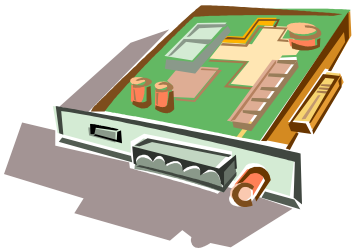


Buffer Compression



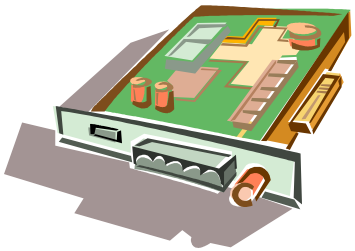
- Depth Buffer Write
  - Rasterizer modifies copy of uncompressed tile
  - Tile is lossless compressed (if possible) and sent to actual depth buffer
  - Update Tile Table
    - z_{min} and z_{max}
    - status: *compressed* or *decompressed*

Buffer Compression



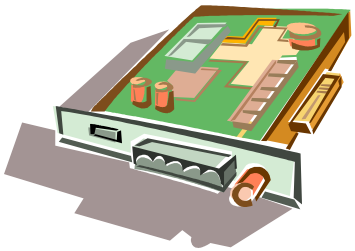
- Depth Buffer Read
  - Tile Status
    - *Uncompressed*: Send tile
    - *Decompress*: Decompress and send tile
    - *Cleared*: See Fast Clear

Fast Clear



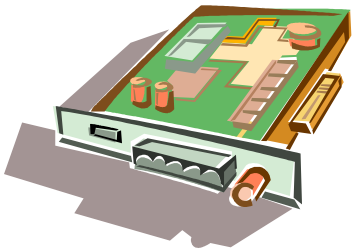
- Don’t touch depth buffer
- glClear sets state of each tile to *cleared*
- When the rasterizer reads a cleared buffer
  - A tile filled with GL_DEPTH_CLEAR_VALUE is sent
  - Depth buffer is not accessed

Fast Clear

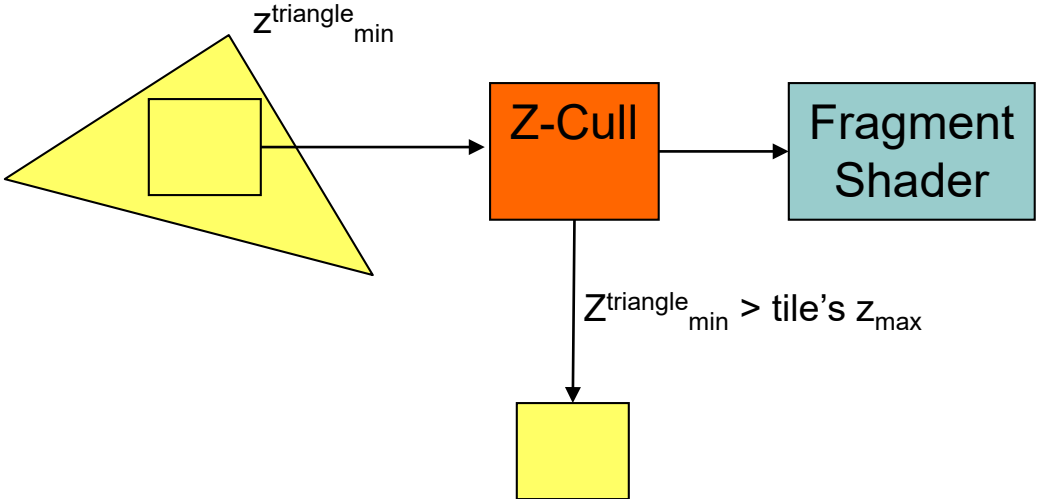


- Use glClear
  - Not full screen quads
  - No "one frame positive, one frame negative“ trick
- Clear stencil together with depth

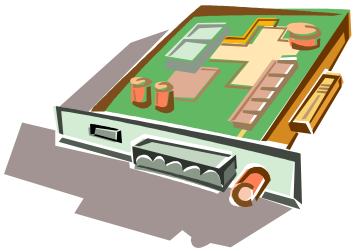
Z-Cull



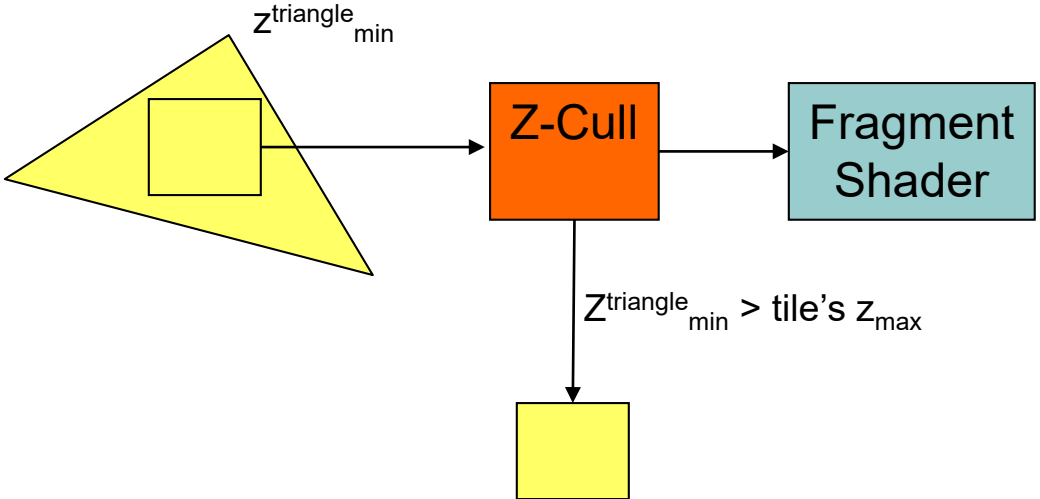
- Cull blocks of fragments before shading
- Coarse-grained as opposed to *Early-Z*



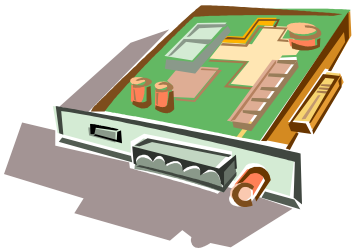
Z-Cull



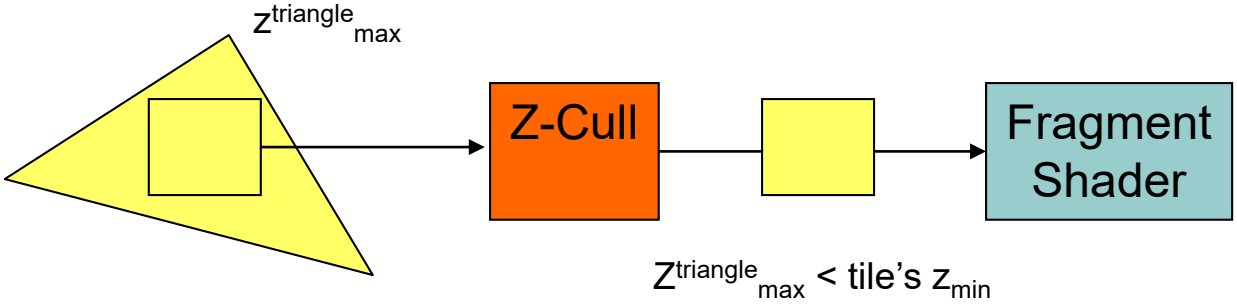
- Z_{max}-Culling
  - Rasterizer fetches z_{max} for each tile it processes
  - Compute $z^{triangle}_{min}$ for a triangle
  - Culled if $z^{triangle}_{min} > z_{max}$



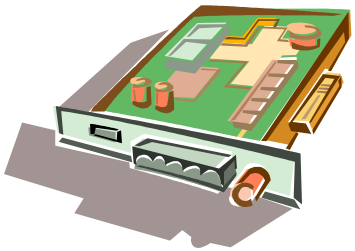
Z-Cull



- Z_{min}-Culling
  - Support different depth tests
  - Avoid depth buffer reads
  - If triangle is in front of tile, depth tests for each pixel is unnecessary



Z-Cull



- *A*utomatically enabled on GeForce (6?) cards unless
  - glClear isn't used
  - Fragment shader writes depth (or discards?)
  - Direction of depth test is changed
- ATI recommends avoiding = and != depth compares and stencil fail and stencil depth fail operations
- Less efficient when depth varies a lot within a few pixels

See NVIDIA GPU Programming Guide for exact details

Programmable Culling Unit

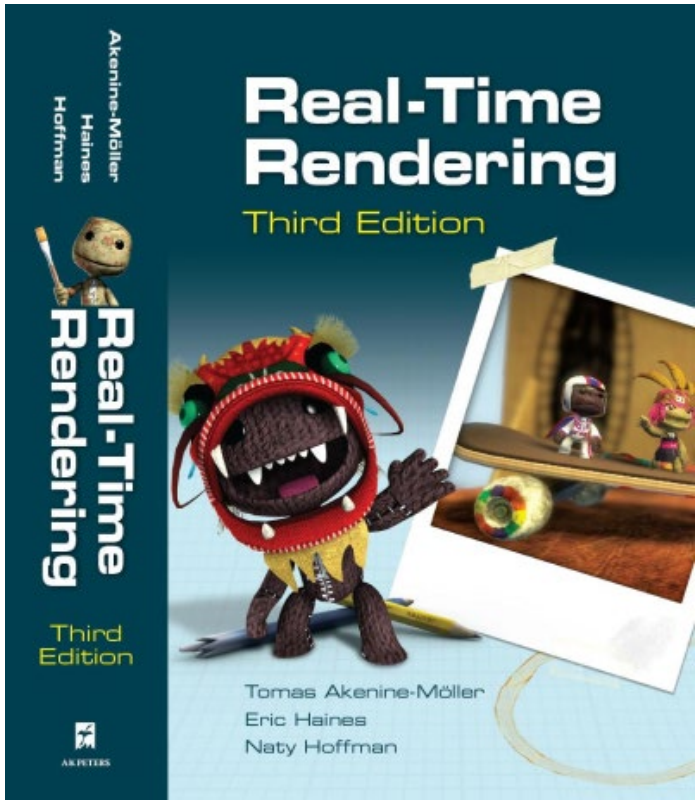
- Cull before fragment shader even if the shader writes depth or discards
- Run part of shader over an entire tile to determine lower bound z value

- Hasselgren and Akenine-Möller, “PCU: The Programmable Culling Unit,” 2007

Summary

- What was once “ridiculously expensive” is now the primary visible surface algorithm for rasterization

Resources



Sections 7.9.2 and 18.3

www.realtimerendering.com

Resources

GeForce 8 Guide: sections 3.4.9, 3.6, and 4.8

GeForce 7 Guide: section 3.6

developer.nvidia.com/object/gpu_programming_guide.html

Resources

ATI Radeon HyperZ Technology
Steve Morein

http://www.graphicshardware.org/previous/www_2000/presentations/ATIHot3D.pdf

Resources

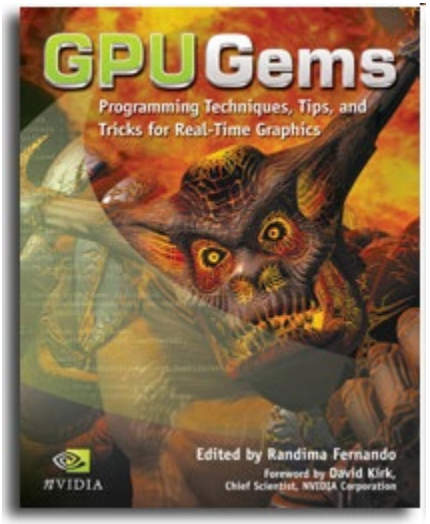
Performance Optimization Techniques for ATI Graphics Hardware with DirectX® 9.0

Guennadi Riguer

Sections 6.5 and 8

http://ati.amd.com/developer/dx9/ATI-DX9_Optimization.pdf

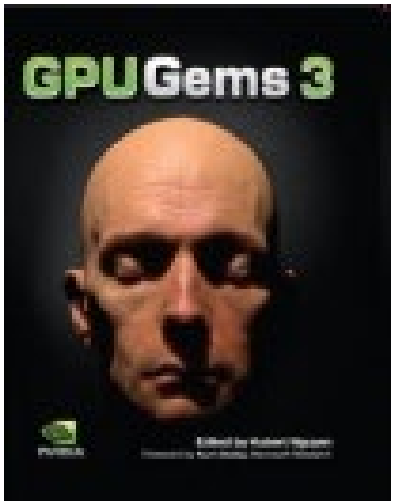
Resources



Chapter 28: Graphics Pipeline Performance

developer.nvidia.com/object/gpu_gems_home.html

Resources



Chapter 19: Deferred Shading in Tabula Rasa

developer.nvidia.com/object/gpu-gems-3.html