Lecture 15:
Hardware Rendering

Fall 2004

Kavita Bala

Computer Science

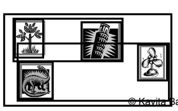
Cornell University

Announcements

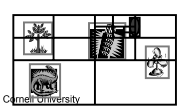
- Project discussion this week
  – Proposals: Oct 26
- Exam moved to Nov 18 (Thursday)

Bounding Volume vs. Spatial Hierarchy

- Object subdivision
  – Hierarchical object representation
  – Hierarchically cluster objects
- Siblings could overlap
- Object in single leaf
- Ray marches down
- AABB,OBB,Spheres



- Spatial subdivision
  – Hierarchical spatial representation
  – Hierarchically cluster space
- Siblings distinct
- Object in >1 leaf (higher)
- Ray marches across
- Octree,kd-tree,Grid



Culling of Complex Scenes

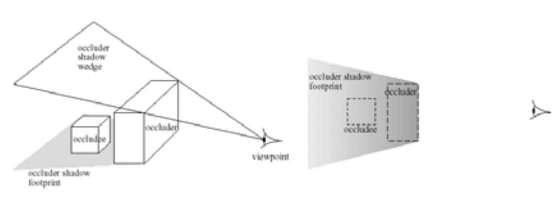
- Remove geometry that is not visible … cull it away
  – View Frustum Culling
  – Hierarchical z-buffer
  – Cell-portal visibility
  – Many others….

Hierarchical View Frustum Culling

- Use an octree/BVH
- Start at o = root of octree/BVH
- Test(Node o) {
  – Check 6 planes of frustum for intersection with bbox(o)
  – If in or out, terminate testing
  – If it intersects
    - For each child c = child[i], Test (c)

Occlusion Culling

- Occlusion Culling/Visibility Culling
- Don't send all polygons to hardware
  – Remove polygons that are not visible
  – Conservative: find visible superset

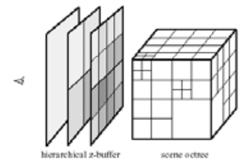


Occlusion Culling

- On-line
  - Remove geometry on-the-fly
- Off-line
  - Determine potentially visible set (PVS)
  - When rendering only display PVS

Hierarchical Z-buffer

- On-line
- Use nearby polygons to remove far polygons
- Construct an octree subdivision of scene
  - Could use other data structures as well

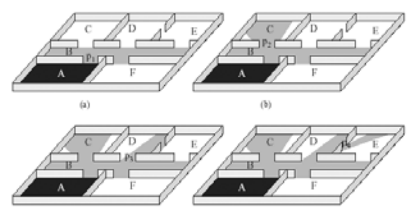


Off-line

- Interactive walkthroughs of very complex systems
  - Radiosity systems
  - Too many polygons
- Teller: Cell/Portal for indoor scenes
  - Used in games: Doom, Descent

Cell Portal Architecture

- Internal architectural scene
- Cells: Rooms
- Portals: Doors and Windows

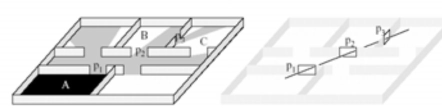


First Idea

- Find all cells visible from current cell
- Recursively propagate visibility
- Problem: Too conservative

Stab Tree

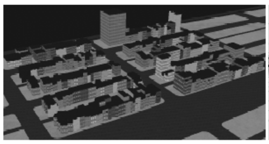
- Form a stab tree



- Check if a line exists that stabs all portals
  - Use linear programming

More Recent Work

- How to deal with non-architectural scenes
  - Cityscapes
  - Small Blockers: add up to large occlusion



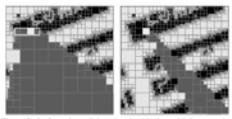


Figure 2: Left: a viewcell in red and a blocker with its extension outlined in yellow. The blocker hides the voxels in blue. Right: occlusion without blocker extension. (Opaque voxels are shown in black.)

More Recent Work

- But forests? Not really… Not yet!
- What about dynamic scenes?

Summary

- Acceleration structures for:
  - Ray Tracing
  - Collision detection
  - Point finding
  - Visibility culling
  - View frustum culling
  - …

CS 665

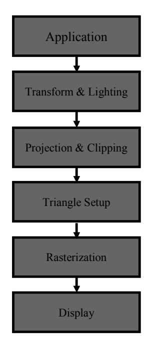
Hardware Rendering

Images from Real-Time Rendering
Courtesy Eric Haines

Strengths of Hardware

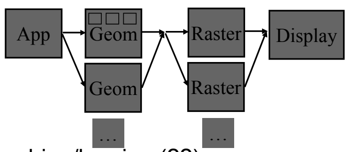
- High throughput
  - Lots of polygons/second or pixels/second
- In the past:
  - Fixed functionality
  - Shading, transformations, clipping, etc.
- Hardware has been transformed
  - Programmable

Review: Graphics Pipeline



Why is Hardware Fast

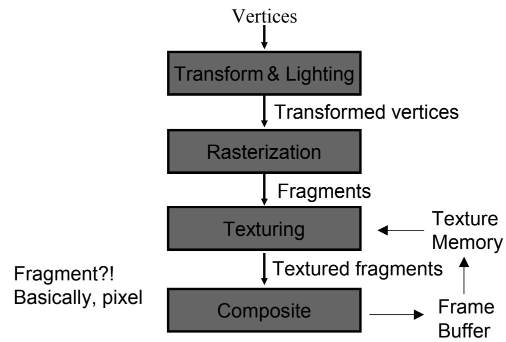
- Pipelined and parallel



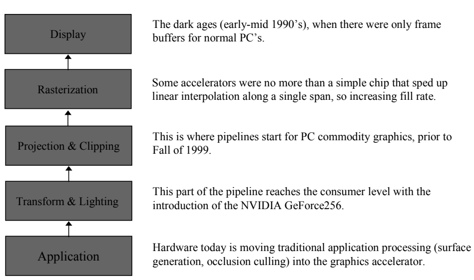
- No branching/looping (??)
- Aggressive prefetching from memory
- Pixel arithmetic is usually 8 bit fixed-point (this has changed)

This is the traditional “fixed-function pipeline”

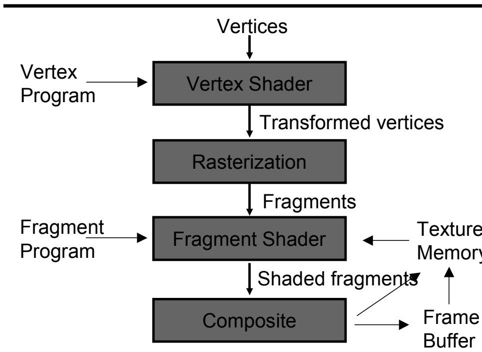
Traditional OpenGL



Brief History



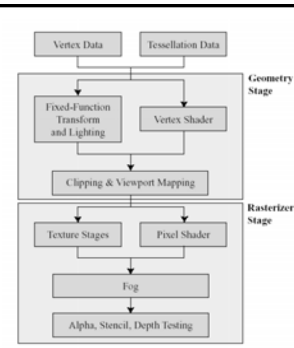
New OpenGL



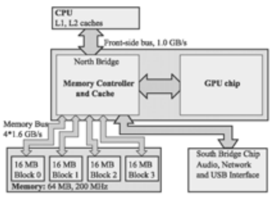
New Programmable GPUs

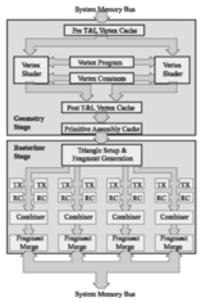
- Pipelined and parallel
  - Current pipeline 600-800 stages deep!
- Branching/looping??
- Floating point arithmetic
- Programmable Vertex and Shader programs
- Essentially writing assembly/C code

Fixed-Function vs. Programmable



Xbox Architecture





UMA (Unified Memory Architecture)

GeForce 4 Vertex Shaders

- Sets of 4 word registers
- Data passed in through v, c registers
- Data out through o registers to next stage
- Vertex attribute registers
  - v[POS], v[NML],…
  - position, normal, …
  - Programmer loads data
- Result registers
- Temporary registers
- Constant registers

Vertex Program Assembly

- Instructions operate on scalar or 4 vector
- Result is 4 vector or scalar (replicated)
- Examples
  - MOV v : Out v
  - DP4 v, v: ssss
  - Swizzle, .wxyz
- glLoadProgramNV

Shading Languages

- Assembly hard to program
- In 2002, many shading languages
  - e.g. Cg from NVIDIA, HLSL in DirectX 9
- Benefits:
  - Ease of use
  - Hardware independence
  - Reusable code

Cg Example

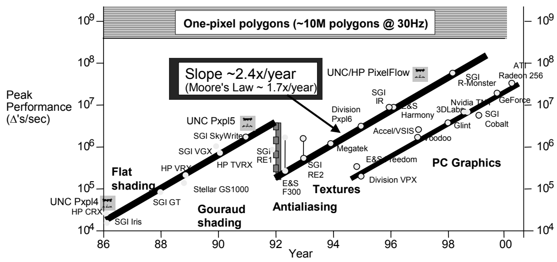
```
Vertout main (… uniform float4 LightVec)
{
vertout Out;
Out.Hposition = …

float 4 light = normalize (LightVec)
float diffuse = dot (normal, light);
…
return Out;
}
```


What’s New Since 1999

- Vertex Shader
- Pixel/Fragment Shader
- And much faster, of course
  - Peak fill rates
  - 1999 GeForce256: 0.35 Gigapixel
  - 2001 GeForce3: 0.8 Gigapixel
  - 2003 GeForceFX Ultra: 2 Gigapixel
  - ATI Radeon 9800 Pro : 3 Gigapixel

Faster than Moore's Law



Graph courtesy of Professor John Poulton

Programmable GPU Shading

- Multithreaded SIMD organization
- Multiple parallel units
- Same instructions executed on *n* vertices or fragments in parallel

Performance Issues

- Pipeline: bottleneck analysis
- Parallelism: load balancing
- Memory bandwidth limits
  - Texture reads
  - Z-Buffering
  - Host interface

GPU Parallelism

- GPUs exploit both
  - Task parallelism: pipeline
  - data (vertex, triangle, fragment) parallelism
    - Process k triangles in parallel, m fragments in parallel
    - But, some triangles generate more fragments, some parts of screen written to more than others
- Various approaches to load balancing
  - FIFO buffering
- Pipeline in GeForce3 up to 800 clocks long (compare to 10-20 on CPUs)

Bandwidth

- Bandwidth scales with perimeter
- Computation scales with area
- Memory, buses MUCH slower than internal processing
- CPUs: use lots and lots and lots of caches to match memory speeds
- GPUs: exploit streaming computation, prefetching, block transfers, coherence

Texture Cache

- Prefetch texture block
- Texture data spatially organized to maximize coherence
- May reorder texture lookups to improve temporal coherence

Current/Future

- Faster
  - More parallelism
- More generalized shading languages
  - Fewer constraints in programs

Key Hardware Capabilities

- Z-Buffering
- Accumulation Buffer
- Antialiasing
- Transparency/Compositing
- Stencil Buffer
- Filtered Texturing

Accumulation Buffer

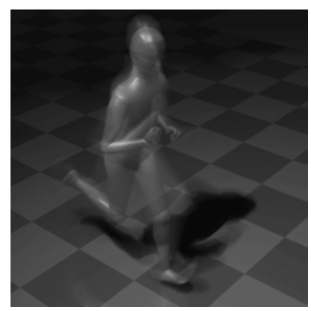
- Render a scene a number of times, making small variations
- Blend the results to make a single image.

Effects produced include:

- Antialiasing
- Depth of Field
- Motion Blur
- Soft Shadows

- Needs more precision than ordinary buffers

Accumulation Buffer



Anti-aliasing
Motion Blur
Depth of Field

Stereo Buffers

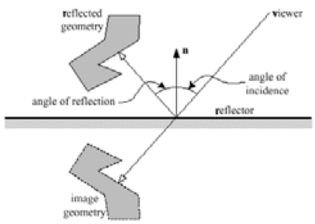
- Render left and right eye views
  - Both front and back left and right buffers
- Display hardware alternates frames
- Controls shutter glasses
- May also
  - use head or eye tracking
  - use head-mounted-display (HMD) and multiple parallel outputs instead

Stencil Buffers

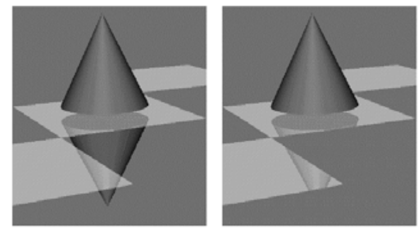
- 1 to 8 bitplanes
  - Usually “leftover” bits in depth buffer
  - May need to use 24-bit depth to use
- Stencil test used
  - Tests: ==, <, <=, >, >= ref value
- Operations can modify stencil value:
  - Ex: increment stencil if passes depth test
  - Different ops for fail, zfail, zpass
- Can mask out parts of stencil to modify
- Used for shadow volumes, reflections, etc.

Reflections

Planar: flip object through mirror



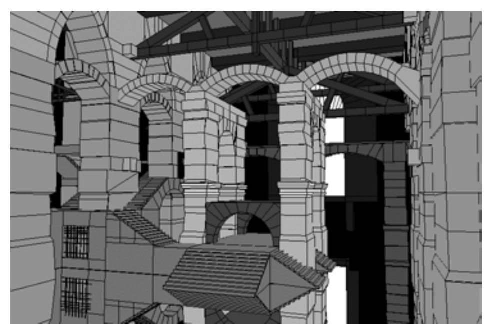
Reflection & Stencil Buffer



Reflection Example - Castle

Agata & Andrzej Wojaczek, Advanced Graphics Applications Inc.

Castle's Geometry



Agata & Andrzej Wojaczek, Advanced Graphics Applications Inc.

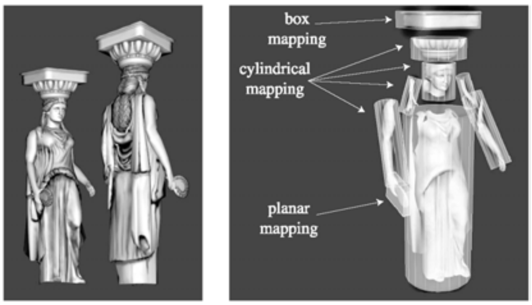
P-Buffers

- Permit rendering to off-screen target
  - Addresses dimension limitations
  - Example, used for shadow maps
- P-buffer in one context can be associated with texture in another
- ATI Radeon 9700:
  - fragment shader can write to up to four output buffers simultaneously
  - potentially useful for multipass algorithms

Key Hardware Capabilities

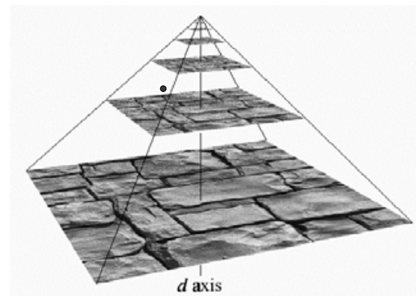
- Z-Buffering
- Accumulation Buffer
- Antialiasing
- Transparency/Compositing
- Stencil Buffer
- Filtered Texturing

Texture Mapping



Images courtesy Tito Pagan

Mipmapping Filtering



Fast Texture Map Lookup

- Very powerful feature of hardware
- Most flexible part of graphics hardware
  - Surface texturing
  - Bump mapping: normals
  - Reflection mapping
  - Shadow mapping
  - Even arbitrary BRDF approximations
- Cheap anti-aliasing & anisotropic filtering

Many types of Texture Maps

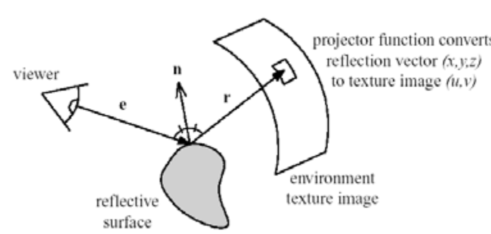
- Texture modulates diffuse coefficients in shading model
- Textures can modulate
  - Normals: bump mapping and normal mapping
  - Positions: displacement mapping
  - Lighting: environment mapping

Environment Map

- Want to compute reflections of environment on surfaces
  - Planar surfaces?
  - Curved surfaces

- Assumptions:
  - Environment Map represents objects at infinity
- Index into EM using reflection vector

Environment Mapping



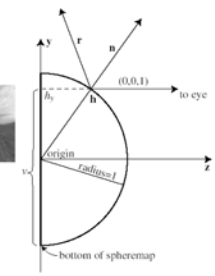
- EM gives reflections in curved surfaces
  - Not very good for flat surfaces

Env Map Algorithm

- Generate 2D environment map
  - – Spherical, cubical, paraboloid
- For each pixel on a reflective object
  - – Find N on surface of object
  - – Compute R from V and N: R = V – 2 (N.V) N
  - – Index into EM using R
  - – Modulate pixel color

Types of Mappings

Cube mapping



Sphere mapping