Culling Techniques

Sung-Eui Yoon
(윤성의)

Course URL:
http://jupiter.kaist.ac.kr/~sungeui/SGA/
At the Previous Class

- The overview of the course
  - Two main rendering techniques: rasterization and ray tracing
  - Their issues with different configurations
Rasterization: Rendering Pipeline

Host Commands → Vertex Transforms → Cull, Clip & Project → Process And Rasterize Primitive → Fragment Processing → Per-Fragment Operations → Frame Buffer Operations → Frame Buffer → Display
Depth Buffer

Algorithm:

Maintain current closest surface at each pixel

Rendering Loop:

set depth of all pixels to $Z_{MAX}$

foreach primitive in scene

    foreach pixel in primitive

        compute $z_{prim}$ at pixel

        if ($z_{prim} < depth_{pixel}$) then

            pixel = object color

            $depth_{pixel} = z_{prim}$

        endif

    endfor

endfor
Depth Buffer: Advantage

- Simple
- Can process one primitive at a time in any order
- Can easily composite one image/depth with another image/depth
  - Useful for parallel rendering especially for sort-last based method
- Spatial coherence
  - Incremental evaluation in loops
  - Good memory coherence
Depth Buffer: Disadvantage

- Transparency is hard to handle
  - Has to be done in strict back-to-front order
- Lots of overdraw
- Read/Modify/Write is hard to make fast
- Requires a lot of storage
- Quantization artifacts
Limitations of Rasterization

- The performance ~ linear to # of triangles
- Massive models with high-depth or low-depth complexity
  - Require output sensitive rendering methods
  - Culling techniques for high-depth complexity
  - Multi-resolution techniques for low-depth complexity
What are Culling and Clipping?

- Culling
  - Throws away entire objects and primitives that cannot possibly be visible
- Clipping
  - “Clips off” the visible portion of a primitive
  - Simplifies rasterization
  - Used to create “cut-away” views of a model
Visibility Culling Methods

- Back-face culling
- View frustum culling
- Occlusion culling
- Hierarchical culling
Culling Example

- Power plant model
  - 13 M triangles
  - 1.7 M triangles - gutted version show here with no internal pipes
Culling Example

Full model
1.7 Mtris

View frustum culling
1.4 Mtris

Occulsion culling
89 Ktris
Back-Face Culling

- Special case of occlusion - **convex self-occlusion**
  - for closed objects (has well-defined inside and outside) some parts of the surface must be blocked by other parts of the surface
- Specifically, the backside of the object is not visible
Face Plane Test

- Compute the plane for the face:
  \[ n = (\mathbf{v}_1 - \mathbf{v}_0) \times (\mathbf{v}_2 - \mathbf{v}_0) \]
  \[ d = n \cdot \mathbf{v}_0 \]

- Cull if eye point in the negative half-space
Lines and Planes

- Implicit equation for line (plane):

\[ n_x x + n_y y - d = 0 \]

\[
\begin{bmatrix}
  n_x & n_y & -d
\end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  1
\end{bmatrix} = 0 \implies n \cdot p = 0
\]

- If \( n \) is normalized then \( d \) gives the distance of the line (plane) from the origin along \( \hat{n} \).
View Frustum Culling

- Test objects against planes defining view frustum

- Uses BVs of objects to improve the performance of view-frustum culling
Depth Complexity

- Number of triangles per each pixel
  - Likely to grow as the model complexity increases
Occlusion Culling

- Detects visibility of primitives
- If invisible, do not need to process such primitives

What are ingredients for the success of the method?
- Fast visibility checking
- Conservative primitive enclosing with BVs, etc.
Occlusion Query

- An occlusion query asynchronously returns the number of fragments that pass z-test
- Typical use: In multi-pass rendering, skip subsequent passes if the first one rendered too few pixels

Usage:
- Create the query
- Issue a begin event to start counting
- Draw something
- Issue an end event to stop counting
- Get the result
Occlusion Query: OpenGL

Extension: GL_ARB_occlusion_query

```c
// Generate ID
GLuint queryID;
glGenQueriesARB(1, &queryID);
...

// Count
glBeginQueryARB(GL_SAMPLES_PASSED_ARB, queryID);
Draw(...);
glEndQueryARB(GL_SAMPLES_PASSED_ARB);
...

// Get result
int fragmentsDrawn;
glGetQueryObjectuivARB(queryID, GL_QUERY_RESULT_ARB, &fragmentsDrawn);
```
Occlusion Culling with Occlusion Queries

- Render objects visible in previous frame (occlusion representation)
Occlusion Culling with Occlusion Queries

- Turn off color and depth writes
- Render object bounding boxes with occlusion queries

newly visible
Occlusion Culling with Occlusion Queries

- Re-enable color writes
- Render newly visible objects
Assumptions & Limitations

- Assume temporal coherence
  - How about the initial frame?

- Can we take advantage of spatial coherence between objects?
Hierarchical Culling

- Culling needs to be cheap!
- Bounding volume hierarchies accelerate culling by trivially rejecting/accepting entire sub-trees at a time

Example of hierarchical view-frustum culling
Visibility Computations

- **Fundamental question:**
  - Between which parts of a scene does there exist an unobstructed path?

- **Types of visibility computations**
  - Hidden surface removal
  - Visibility culling

- **Some other related applications**
  - Line-of-sight
  - Sound propagation
  - Path planning and robotics
Classes of Visibility Algorithms

- **Point vs. Region visibility**
  - Compute parts of the scene visible at a point or any point in a region

- **Object vs. Image precision**
  - Compute parts of an object (or which pixel) that are visible
  - Operates directly on or discretized representation of the geometry
Ray Tracing: Visibility Issue

- For each pixel, find closest object along the ray and shade pixel accordingly

- Advantages
  - Conceptually simple
  - Can support CSG
  - Can be extended to handle global illumination effects (ex: shadows and reflectance)

- Disadvantages
  - Renderer must have access to entire retained model
  - Hard to map to special-purpose hardware
Next Time..

- Study culling techniques
  - E.g., Multi-resolution methods