Johns Hopkins
Engineering for Professionals

605.767 Applied Computer Graphics

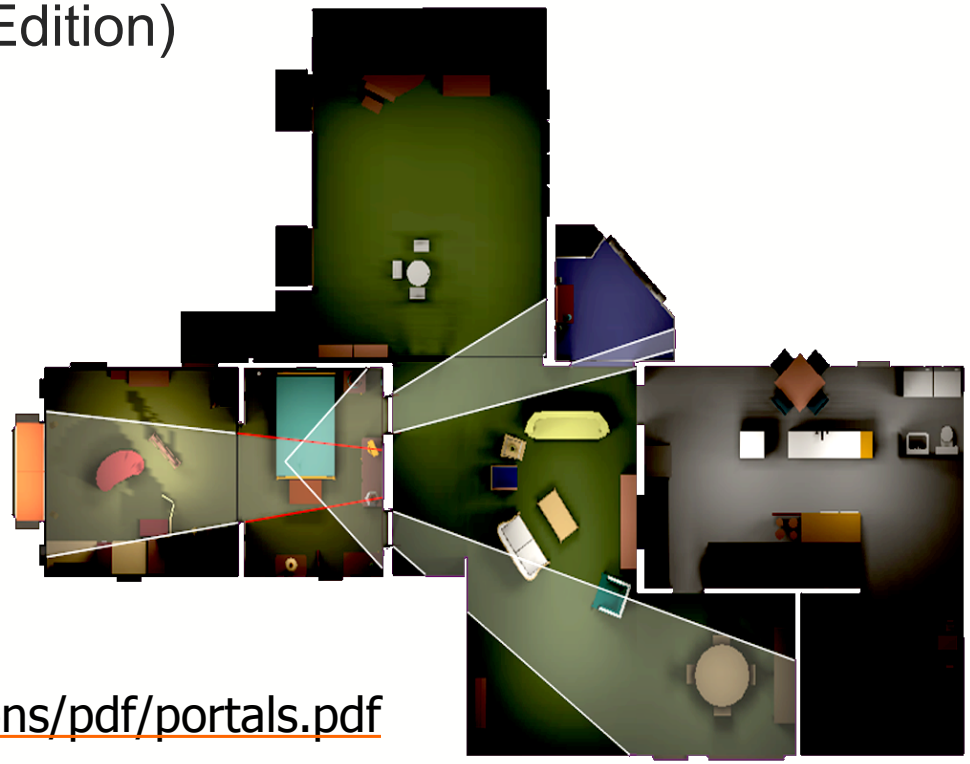
Brian Russin

Module 5E
Other Culling Techniques

Portal Culling

- Architectural models often use a set of algorithms called **Portal Culling**
  - Walls often act as large occluders
  - Perform view frustum culling against portals through windows and doors
  - See section 19.5 (14.4 in 3rd Edition)

"Portals and Mirrors: Simple, Fast Evaluation of Potentially Visible Sets", David P. Luebke and Chris Georges, Proceedings of the 1995 Symposium on Interactive 3D Graphics, ACM Press, NY (April, 1995).



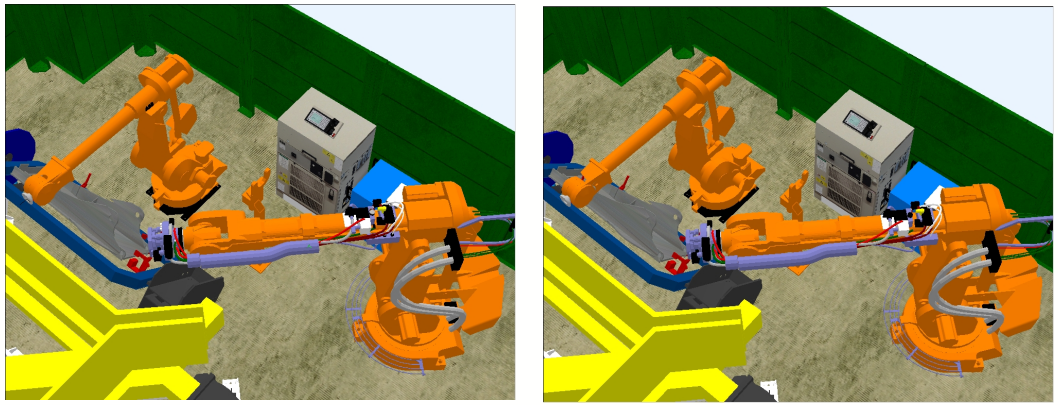
https://luebke.us/publications/pdf/portals.pdf

Portal Culling

- Small amount of preprocessing (often manually created)
  - Divide scene into cells
    - Usually correspond to rooms or hallways
    - List of objects and walls of the cell are stored
    - Store information on adjacent cells and the portals that connect them
- To render
  - Locate cell V where the viewer is positioned
  - Create a 2D AABB (P) equal to the screen rectangle
  - Render current cell with VF cull with respect to AABB (whole screen)
  - Recursively process the portals to cells adjacent to current cell
    - Project the portal onto the screen and find 2D AABB (BB)
    - Compute intersection of P and BB
      - If empty the cell is invisible from the current point of view
      - If not empty then contents of the neighbor cell can be culled by the frustum from V through the intersection of P and BB
        - Need to recurse into neighboring cells that are visible
        - They may have portals to other visible geometry

Detail Culling

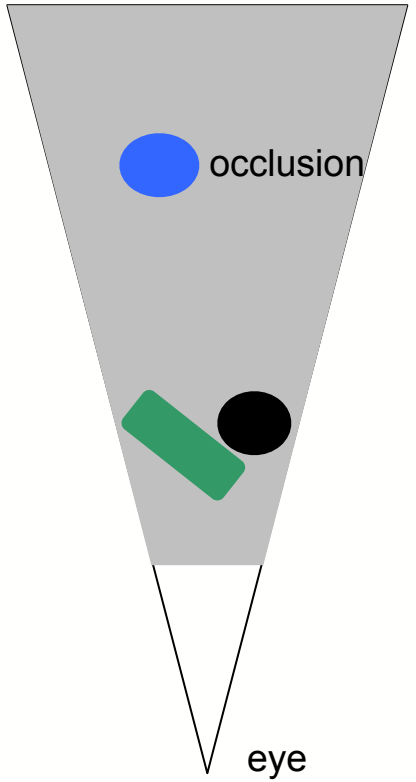
- Cull objects whose projected BV occupy less than N pixels
  - Objects you cull away may actually contribute to the final image
  - Trade off quality vs. speed
  - Often applied when view is in motion
    - Culling disabled when view motion stops
- Project BV onto the projection plane
  - Estimate area of projection
  - If below a threshold – do not render the objects within the BV
  - Often called **screen size culling**



Example from
graphics.stanford.edu/courses/cs248-00/real-time-programming/moller-cs248-00-lecture.ppt
(Lecture from Moller – dead link…)
Left image – no culling, right image with detail level culling
Over 80% speedup

Occlusion Culling

- Objects that lie completely “behind” other objects can be culled
  - Hard problem to solve efficiently
- Depth buffer solves surface visibility for an application
  - Polygons will be scan converted and depth tested
    - Often numerous polygons touch a single pixel
      - Known as depth complexity
    - Figure 19.18 (14.15 in 3rd Edition) – 10 spheres drawn in a row
      - Center pixel is “touched” 10 times
- There are many methods
  - Hierarchical Z-buffering
  - Hierarchical Occlusion Maps
  - Hardware occlusion queries



Occlusion Culling Pseudo-code

```
OcclusionCullingAlgorithm(Scene G)
{
   Or = empty  // occlusion representation
   for each object g in G
   {
      if (isOccluded(g, Or))
         skip(g)
      else
      {
         Render(g)
         Update(Or, g)
       }
   }
}
```

- isOccluded is the visibility test
- If the object is not included it has to be rendered
- Object is used to update the occlusion representation
- Note some similarities to Z-buffer
  - Visibility occurs at pixel level

Occlusion Culling

- Performance of occlusion algorithms depends on the order that objects are traversed
  - Best performance if occluding objects are processed first
  - Performance generally improved if rough front to back sorting occurs
    - Occluders more likely to be drawn first
- Even small objects can be excellent occluders
  - Haines and Moller give the example of a matchbox occluding the Golden Gate bridge
    - If placed properly in front of the viewer and close!
- https://www.gamedeveloper.com/programming/occlusion-culling-algorithms
  - Simplified excerpts from Haines and Moller Real Time Rendering, Edition 2

Hardware Occlusion Culling

- GPUs support occlusion culling via a special rendering mode
  - Query whether a set of polygons is visible when compared to the current buffer contents
    - Bounding volume polygons generally used in the query
    - Query returns a count of pixels visible
- Performance implications
  - If complex object is occluded we gain performance
    - Only scan convert one BB instead of the entire object
  - If not occluded we lose a little performance
    - Cost of rendering the BB and performing the query
  - Drawing order matters
    - Drawing front to back gives more occlusion
  - Note: latency of the query is high
    - Often hundreds or thousands of polygons can be rendered during the query time
- Text discuses several hardware implementations
  - HP Visualize fx (2000)

Occlusion Culling with OpenGL

- Generate a query ID
  - **glGenQueries(GLsizei n, GLUint* ids)**
- Specify the start of the occlusion query
  - **glBeginQuery(GLenum target, GLuint id)**
  - Target is GL_SAMPLES_PASSED
- Render the geometry for the occlusion test
- Specify that you have completed the query
  - **glEndQuery(GLenum target)**
- Retrieve the number of samples that passed the occlusion / depth buffer test
  - **glGetQueryObjectuiv(GLuint id, GLenum pname, GLuint* params)**
  - pname = GL_QUERY_RESULT
- NOTE: disable writing to color and depth buffers
  - **glColorMask** and **glDepthMask**
  - Re-enable when done with query

Occlusion Culling with OpenGL

```
GLuint query;
glGenQueries(1, &query); // generate query object

glColorMask(GL_FALSE, GL_FALSE, GL_FALSE, GL_FALSE);
glDepthMask(GL_FALSE);
GLuint sampleCount=1;
glBeginQuery(GL_SAMPLES_PASSED, query);
model.drawModel();
glEndQuery(GL_SAMPLES_PASSED);

glColorMask(GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE);
glDepthMask(GL_TRUE);
glGetQueryObjectuiv(query, GL_QUERY_RESULT, &sampleCount);
```

- https://developer.nvidia.com/gpugems/gpugems2/part-i-geometric-complexity/chapter-6-hardware-occlusion-queries-made-useful
- Note: occlusion culling is not part of OpenGL ES 2.0

Occlusion Horizons

- Method useful for rendering urban scenes
  - Buildings are occluders – connected to ground
  - Treat the buildings as a height field
- Render objects from front to back
  - Maintain an occlusion horizon
  - If projected bounding box is within horizon it does not get rendered
  - Otherwise, render the object and add its occluding power to the horizon