CSE 167: Introduction to Computer Graphics
Lecture #13: Visibility Culling

Jürgen P. Schulze, Ph.D.
University of California, San Diego
Fall Quarter 2020
Announcements

- **Sunday, November 15th at 11:59pm:**
  - Late deadline for Project 2
- **Wednesday, November 18th at 1pm:**
  - Discussion Project 3
- **Sunday, November 22nd at 11:59pm:**
  - Homework Project 3 due
- **Hollow Ocean Internship**
  - With UCSD Prof. Pinar Yoldas
  - [https://docs.google.com/document/d/106Fh0AnHd4YLtHKpNi iP9i1rZ7Z-o8jMq-pWs7Gtc/edit](https://docs.google.com/document/d/106Fh0AnHd4YLtHKpNi iP9i1rZ7Z-o8jMq-pWs7Gtc/edit)
Visibility Culling

- **Goal:** Discard geometry that does not need to be drawn to speed up rendering

- **Types of culling:**
  - View frustum culling
  - Small object culling
  - Degenerate culling
  - Backface culling
  - Occlusion culling
View Frustum Culling

- Triangles outside of view frustum are off-screen

Images: SGI OpenGL Optimizer Programmer's Guide
Videos

- Rendering Optimizations - Frustum Culling
  - http://www.youtube.com/watch?v=kvVHp9wMAO8

- View Frustum Culling Demo
  - http://www.youtube.com/watch?v=bJrYTBGpwic

- View Frustum Culling in Action
  - http://giant.gfycat.com/InexperiencedMadKiskadee.webm
Bounding Volumes

- Simple shape that completely encloses an object
- Generally a box or sphere
  - Easier to calculate culling for spheres
  - Easier to calculate tight fits for boxes
- Intersect bounding volume with view frustum instead of each primitive
Bounding Box

- How to cull objects consisting of many polygons?
- Cull bounding box
  - Rectangular box, parallel to object space coordinate planes
  - Box is smallest box containing the entire object

*Image: SGI OpenGL Optimizer Programmer's Guide*
View Frustum Culling

- Frustum defined by 6 planes
- Each plane divides space into “outside”, “inside”
- Check each object against each plane
  - Outside, inside, intersecting
- If “outside” of at least one plane
  - Outside the frustum
- If “inside” all planes
  - Inside the frustum
- Else partly inside and partly out
Distance to Plane

- A plane is described by a point $p$ on the plane and a unit normal $n$
- Find the (perpendicular) distance from point $x$ to the plane
The distance is the length of the projection of $\mathbf{x} - \mathbf{p}$ onto $\mathbf{n}$.

$$\text{dist} = (\mathbf{x} - \mathbf{p}) \cdot \mathbf{n}$$
Distance to Plane

- The distance has a sign
  - positive on the side of the plane the normal points to
  - negative on the opposite side
  - zero exactly on the plane
- Divides 3D space into two infinite half-spaces

\[
dist(x) = (x - p) \cdot \hat{n}
\]
Distance to Plane

- **Simplification**
  \[
  \text{dist}(\mathbf{x}) = (\mathbf{x} - \mathbf{p}) \cdot \mathbf{n} \\
  = \mathbf{x} \cdot \mathbf{n} - \mathbf{p} \cdot \mathbf{n} \\
  \text{dist}(\mathbf{x}) = \mathbf{x} \cdot \mathbf{n} - d, \quad d = \mathbf{p} \cdot \mathbf{n}
  \]

- \(d\) is independent of \(\mathbf{x}\)
- \(d\) is distance from the origin to the plane
- We can represent a plane with just \(d\) and \(\mathbf{n}\)
Frustum With Signed Planes

- Normal of each plane points outside
  - “outside” means positive distance
  - “inside” means negative distance
For sphere with radius $r$ and origin $x$, test the distance to the origin, and see if it is beyond the radius

Three cases:

- $\text{dist}(x) > r$
  - completely above
- $\text{dist}(x) < -r$
  - completely below
- $-r < \text{dist}(x) < r$
  - intersects
Culling Summary

- Transform view frustum plane equations in camera space.
- Pre-compute the normal $\mathbf{n}$ and value $d$ for each of the six planes.
- Given a sphere with center $\mathbf{x}$ and radius $r$ in camera space.
- For each plane:
  - if $\text{dist}(\mathbf{x}) > r$: sphere is outside! (no need to continue loop)
  - add 1 to count if $\text{dist}(\mathbf{x}) < -r$
- If we made it through the loop, check the count:
  - if the count is 6, the sphere is completely inside
  - otherwise the sphere intersects the frustum
    - (can use a flag instead of a count)
Culling Groups of Objects

- Want to be able to cull the whole group quickly
- But if the group is partly in and partly out, want to be able to cull individual objects
Hierarchical Bounding Volumes

- Given hierarchy of objects
- Bounding volume of each node encloses the bounding volumes of all its children
- Start by testing the outermost bounding volume
  - If it is entirely outside, don’t draw the group at all
  - If it is entirely inside, draw the whole group
Hierarchical Culling

- If the bounding volume is partly inside and partly outside
  - Test each child’s bounding volume individually
  - If the child is in, draw it; if it’s out cull it; if it’s partly in and partly out, recurse.
  - If recursion reaches a leaf node, draw it normally
Video

- Math for Game Developers - Frustum Culling
  - [http://www.youtube.com/watch?v=4p-E_3lXOPM](http://www.youtube.com/watch?v=4p-E_3lXOPM)
Find the frustum planes

- \( p \) – the camera position
- \( d \) – a vector with the direction of the camera’s view ray. In here it is assumed that this vector has been normalized
- \( W_{\text{near}} \) – the “width” of the near plane
- \( \text{nearDist} \) – the distance from the camera to the near plane
- \( \text{farDist} \) – the distance from the camera to the far plane
- \( \text{up} \) – the up vector obtained by normalizing \((ux, uy, uz)\) from the last parameters of \text{gluLookAt}
- \( \text{right} \) – the right vector obtained by cross product between \text{up} and \text{d}.

\[ nc = p + d \times \text{nearDist} \]
\[ fc = p + d \times \text{farDist} \]
Find the frustum planes

- **near plane**: d as normal, nc as a point on the plane.
- **far plane**: –d as normal, fc as a point on the plane.
- **right plane**: p as a point on the plane. normal can be found in this tutorial, the pseudocode is copied here.

\[
\begin{align*}
nc &= p + d \times \text{nearDist} \\
\mathbf{a} &= (\mathbf{nc} + \text{right} \times \frac{W\text{near}}{2}) - p \\
\mathbf{a}.\text{normalize()} \\
\text{normalRight} &= \mathbf{up} \times \mathbf{a}
\end{align*}
\]
Visibility Culling

Goal:
Discard geometry that does not need to be drawn to speed up rendering

Types of culling:
- View frustum culling
- Small object culling
- Degenerate culling
- Backface culling
- Occlusion culling
Small Object Culling

- Object projects to less than a specified size
  - Cull objects whose screen-space bounding box is less than a threshold number of pixels
Degenerate Culling

- Degenerate triangle has no area
  - Normal \( \mathbf{n} = 0 \)
  - All vertices in a straight line
  - All vertices in the same place

Backface Culling

- Consider triangles as “one-sided”, i.e., only visible from the “front”
- Closed objects
  - If the “back” of the triangle is facing away from the camera, it is not visible
  - Gain efficiency by not drawing it (culling)
  - Roughly 50% of triangles in a scene are back facing
**Backface Culling**

- **Convention:**
  Triangle is front facing if vertices are ordered counterclockwise.

*Front-facing* and *Back-facing* triangles are illustrated with arrows showing the vertex ordering.
Backface Culling

- Compute triangle normal after projection (homogeneous division)

\[ \mathbf{n} = (\mathbf{p}_1 - \mathbf{p}_0) \times (\mathbf{p}_2 - \mathbf{p}_0) \]

- Third component of \( \mathbf{n} \) negative: front-facing, otherwise back-facing
  - Remember: projection matrix is such that homogeneous division flips sign of third component
OpenGL

- OpenGL allows one- or two-sided triangles
  - One-sided triangles:
    ```
    glEnable(GL_CULL_FACE); glCullFace(GL_BACK)
    ```
  - Two-sided triangles (no backface culling):
    ```
    glDisable(GL_CULL_FACE)
    ```

```glDisable(GL_CULL_FACE); glEnable(GL_CULL_FACE);```
Occlusion Culling

- Geometry hidden behind occluder cannot be seen
  - Many complex algorithms exist to identify occluded geometry

Images: SGI OpenGL Optimizer Programmer's Guide
Video

- Umbra 3 Occlusion Culling explained
  - http://www.youtube.com/watch?v=5h4QgDBwQhc
Level-of-Detail Techniques

- Don’t draw objects smaller than a threshold
  - Small feature culling
  - Popping artifacts
- Replace 3D objects by 2D impostors
  - Textured planes representing the objects
- Adapt triangle count to projected size

(Data: Stanford Armadillo)