8. Hidden Surface Elimination

Hongxin Zhang
State Key Lab of CAD&CG, Zhejiang University

2014-10-20
Visual Realism

- Achieved by correct rendering of:
  - View (perspective)
  - Field of view (Clip outside the window)
  - Omit hidden parts
  - Surface details like texture
  - Light effects on surfaces like continuous shading, shadows, and caustics.
  - Volumetric effects like transparency and translucency through participating media like water, steam, smoke, …
  - Dynamic effects like movement, elasticity, …
OpenGL functions

- `glEnable / glDisable (GL_CULL_FACE);`
- `glCullFace(mode)`
- `glutInitDisplayMode( ... | GLUT_DEPTH )`
- `glEnable(GL_DEPTH_TEST)`
- `glEnable(GL_DEPTH_TEST)`
- `glEnable(GL_FOG) glFog*()`
Viewing Pipeline Review

View Orientation → Projection → Mapping
Projection

Orthographic

Perspective
Visible Line Drawing
Visible Line Drawing
Visible Line Drawing
Visible Line Drawing
Visible Line Drawing
Visible Surface Determination

- Goal
  - Given: a set of 3D objects and Viewing specification,
  - Determine: those parts of the objects that are visible when viewed along the direction of projection
  - Or, equivalently, elimination of hidden parts (hidden lines and surfaces)
  - Visible parts will be drawn/shown with proper colors and shades
HLHSR Algorithms

• Two Fundamental Approach
  - Object space algorithm
    - a.k.a. **Object Precision**
    - hidden line remove
  - Image space algorithm
    - a.k.a. **Image Precision**
    - z-buffer
Object Precision Algorithm

foreach (object in the world) {
    determine those parts of the object whose view is unobstructed by other parts of it or any other object;
    draw those parts in the appropriate color;
}

Image Precision Algorithms

```plaintext
foreach (pixel in the image) {
    determine the object closest to the viewer that is pierced by the projector through the pixel;
    draw the pixel in the appropriate color;
}
```

![Diagram](projector ViewWindow)
**Back-face Culling**

- In a closed polygonal surface
  - i.e. the surface of a polyhedral volume or a solid polyhedron
  - The faces whose outward normals point away from the viewer are not visible
  - Such back-facing faces can be eliminated from further processing

- Elimination of back-faces is called back-face culling
Back-Face Culling

- **Back Face:**
  - Part of the object surface facing away from the eye.
  - i.e. surface whose normal points away from the eye position.
Back-Face Culling
Back-Face Culling

Algorithm:
1. Find angle between the eye-vector & normal to face.
2. If between 0 to 90°, discard the face.
Back-face Culling

- **Determination of back-faces**

  A polygonal face with outward surface normal $N_f$ is a back-face if $N_f \cdot D_p > 0$

  where $D_p$ is the direction of projection

  What happens when the projectors are along Z axis, i.e., (0,0,1) is the view direction.

  Let $N_f = (n_x, n_y, n_z)$, the dot product now equals $n_z$. If this is +ve, then this is a back-face!
Back-Face Culling

Back-face culling does not solve all visibility problems
Back-Face Culling

Back-face culling does not solve all visibility problems
Back-Face Culling

Back-face culling does not solve all visibility problems
Back-Face Culling

Back-face culling does not solve all visibility problems
Back-Face Culling

Back-face culling does not solve all visibility problems
Back-face Culling

If the scene consists of a single convex closed polygonal surface then back-face culling is equivalent to HLHSR.
Hidden Surface Removal
Hidden Surface Removal
Hidden Surface Removal
Hidden Surface Removal
Hidden Surface Removal

Painter's Algorithm
From back to Front
Hidden Surface Removal

Painter's Algorithm
From back to Front

??
Hidden Surface Removal

Painter's Algorithm
From back to Front
Hidden Surface Removal

Painter's Algorithm
From back to Front

Clipping
Hidden Surface Removal

Area Sorting

Clipping

Painter's Algorithm
From back to Front

Hongxin Zhang, 2014
Z-Buffer Algorithm

• Image precision algorithm
  - Apart from a frame buffer $F$ in which color values are stored,
  - it also needs a z-buffer, of the same size as the frame buffer, to store depth ($z$) values

F-Buffer  Z-Buffer

A.K.A. depth-buffer method
Z-Buffer

Screen
F-Buffer
Z-Buffer
Polygon Scan Conversion

Scan Line

$Y_1$

$Y$

$Y_2$

$Y_3$

$X_a$

$X_1$

$X_b$

$X_2$

$X_3$
Z-Buffer Pseudo-code

- for ( j=0; j<SCREEN_HEIGHT; j++ )
  - for ( i=0; i<SCREEN_WIDTH; i++ ) {
    - WriteToFrameBuffer(i, j, BackgroundColor);
    - WriteToZBuffer(i, j, MAX);
  - }

- for ( each polygon )
  - for ( each pixel in polygon's projection ) {
    - z = polygon's z value at (i, j);
    - if ( z < ReadFromZBuffer(i, j) ) {
      - WriteToFrameBuffer(i, j, polygon's color at (i, j));
      - WriteToZBuffer(i, j, z);
    - }
  - }
Z-buffer:
Z-buffer:
Z-buffer:
Z-buffer:
Z-buffer:
Z-buffer:

\[ (x_p, y_p, d) \]
Z-buffer

Project:

Orthographic

Perspective

Calculate the z of the point

\[Ax + By + Cz + D = 0\]

\[z = \frac{-Ax - By - D}{C}\]

Question: how?
Z-buffer

Project:

Orthographic

Perspective

Calculate the z of the point

\[ Ax + By + Cz + D = 0 \]

\[ z = \frac{-Ax - By - D}{C} \]

Question: how?

DDA

\[ x++, y++, z++ \]
Z-buffer

Project:

Orthographic

Perspective

Calculate the z of the point

\[ Ax + By + Cz + D = 0 \]

\[ z = \frac{-Ax - By - D}{C} \]

Question: how?
\[ Ax + By + Cz + D = 0 \]

\[(x, y, z) \rightarrow (x, y, d)\]

\[
\begin{cases}
  x_p = \frac{d}{z} \\
  x = \frac{z}{z} \\
  y_p = \frac{d}{z} \\
  y = \frac{z}{z}
\end{cases}
\]
$Ax + By + Cz + D = 0$

$(x, y, z) \rightarrow (x, y, d)$

$(x, y, z) \rightarrow (x_p, y_p, d)$

$$
\begin{align*}
\frac{x_p}{x} &= \frac{d}{z} \\
\frac{y_p}{y} &= \frac{d}{z}
\end{align*}
$$
Ax + By + Cz + D = 0

\[(x, y, z) \rightarrow (x, y, d)\]

\[(x, y, z) \rightarrow (x_p, y_p, z)\]

\[
\begin{align*}
\frac{x_p}{x} &= \frac{d}{z} \\
y_p &= \frac{d}{z}
\end{align*}
\]
\[ Ax + By + Cz + D = 0 \]

\[(x, y, z) \rightarrow (x, y, d)\]

\[(x, y, z) \rightarrow (x_p, y_p, z)\]

Orthographic project

\[ (x_p, y_p, d) \]

\[
\begin{align*}
\frac{x_p}{d} &= \frac{x}{z} \\
\frac{y_p}{d} &= \frac{y}{z}
\end{align*}
\]
perspective project
\[(x, y, z) \rightarrow (x_p, y_p, Z)\]
perspective project

\[(x, y, z) \rightarrow (x_p, y_p, Z)\]
$$\begin{align*}
(x, y, z) &\rightarrow (x_p, y_p, z) \\
\downarrow & \\
(x_p, y_p, d)
\end{align*}$$
perspective project

(\(x, y, Z\)) \rightarrow (x_p, y_p, Z)

Orthographic project

(\(x, y, Z\)) \rightarrow (x_p, y_p, d)
Perspective project

Orthographic project

$$(x, y, z) \rightarrow (x_p, y_p, Z)$$

Perspective Transformation

$$\rightarrow (x_p, y_p, d)$$
Perspective Transformation...

• We need to apply a perspective transformation to the view volume and transform it into a rectangular parallel-piped one.

• This makes the final 3D view volume of a perspective view the same as that of a parallel view, just before projection.
A perspective transformation preserves relative depth, straight lines and planes.
Perspective Transformation

\[ \mathbf{M}_P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & (f + n)/(f - n) & -2f \times n/(f - n) \\ 0 & 0 & 1 & 0 \end{bmatrix} \]
A-buffer

- Accumulation buffer
  - used in Lucasfilm REYES
  - not only store depth but also other data
  - support transparent surfaces
Depth-sorting

- space-image space hybrid method

- space or image space:
  - sort surface by depth

- image space:
  - do scan conversion from deepest surfaces
Binary Space Partitioning Trees

- BSP Tree

- Very efficient for a static group of 3D polygons as seen from an arbitrary viewpoint

- Correct order for Painter’s algorithm is determined by a suitable traversal of the binary tree of polygons
BSP Tree
BSP Tree
BSP Tree
BSP Tree

**Draw BSP Tree**

```python
function draw(bsptree tree, point eye)
if tree.empty then
    return
if f_{tree.root}(eye) < 0
    draw(tree.right)
rasterize(tree.root)
draw(tree.left)
else
    draw(tree.left)
rasterize(tree.root)
draw(tree.right)
```

![BSP Tree Diagram](image-url)
BSP Tree

rasterize(C)
rasterize(A)
rasterize(B)

rasterize(B)
rasterize(A)
rasterize(C)
BSP Tree

rasterize(C)
rasterize(A)
rasterize(B)

rasterize(B)
rasterize(A)
rasterize(C)
BSP Tree

• Code works for any view
• Tree can be pre-computed
• Requires evaluation of

\[ f_{\text{plane of the triangle}}(\text{eye}) \]
BSP Tree Construction

- The binary tree is constructed using the following principle:
  - For each polygon, we can divide the set of other polygons into two groups
  - One group contains those lying in front of the plane of the given polygon
  - The other group contains those in the back
  - The polygons intersecting the plane of the given polygon are split by that plane
BSP Tree

• Split Triangle:
  How to?
BSP Tree

• Split Triangle:
  How to?
BSP Tree

• Split Triangle:
  How to?
Summary: BSP Trees

• Pros:
  
  Simple, elegant scheme

  Only writes to frame-buffer (i.e., painters algorithm)

  Thus very popular for video games (but getting less so)

• Cons:

  Computationally intense preprocess stage restricts algorithm to static scenes

  Worst-case time to construct tree: $O(n^3)$

  Splitting increases polygon count

  Again, $O(n^3)$ worst case
Z-buffer

Scan-line

Warnock:

A divide and conquer
Computational expensive of clipping

Warnock:
A divide and conquer

Z-buffer
Scan-line
Warnock’s Area Subdivision (Image Precision)

- Start with whole image
- If one of the easy cases is satisfied, draw what’s in front
  - front polygon covers the whole window or
  - there is at most one polygon in the window.
- Otherwise, subdivide region into 4 windows and recurse
- If region is single pixel, choose surface with smallest depth

- Advantages:
  - No over-rendering
  - Anti-aliases well - just recurse deeper to get sub-pixel information
- Disadvantage:
  - Tests are quite complex and slow
Warnock’s Algorithm

- Regions labeled with case used to classify them:
  - One polygon in front
  - Empty
  - One polygon inside, surrounding or intersecting

- Small regions not labeled
Octree

http://en.wikipedia.org/wiki/View_frustum_culling
ray casting

Rays through view plane

View plane

Eye position
Ray Casting

- For each sample ...
  - Construct ray from eye position through view plane
  - Find first surface intersected by ray through pixel
  - Compute color sample based on surface radiance
Thank You