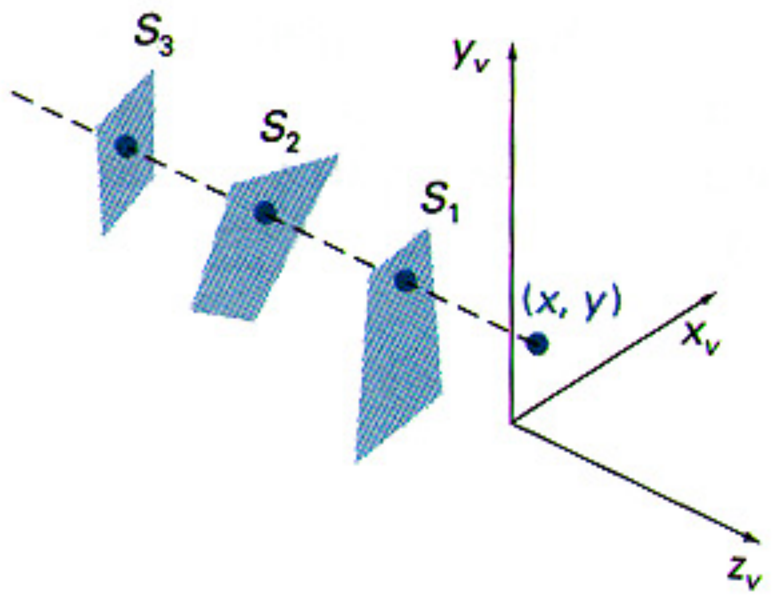


At view-plane position (x, y), surface S_1 has the smallest depth from the view plane and so is visible at that position.

Monash University • Clayton's School of Information Technology

CSE3313 Computer Graphics

Lecture 20: Hidden Surface Removal

Hidden Surface Removal

- Hidden surface and hidden line removal algorithms are often put into 2 categories:

  1. **object space** methods compare objects and parts of objects to each other to determine what is visible from a given point of view;
  2. **image space** algorithms deal with projected images of objects and determine for each pixel in the viewport what is visible.

- Hidden line removal algorithms are often in object space, while hidden surface algorithms are often in image space.

- There is no single best algorithm: which algorithm is best depends on many factors. These include:
  - complexity of the scene
  - kind of lighting model used
  - shading features to be used (e.g. transparency).
  - coherence properties of the image and the objects
  - resolution
  - type of output device

Backface Removal

Reference: Angel 8.8.3

- **Back faces** of solid objects are invisible from the point of view.
- Such backfaces can immediately be removed from the viewing pipeline since they cannot appear in the final scene.
- Suppose the face of a solid object lies in the plane:

$$Ax + By + Cz + D = 0$$

- Suppose the coordinates for the point of view (centre of projection) are (x_i, y_i, z_i). When the coordinates for the point are substituted into the plane equation we have:

$$Ax_i + By_i + Cz_i + D = r_i$$

The plane divides space into two halves.

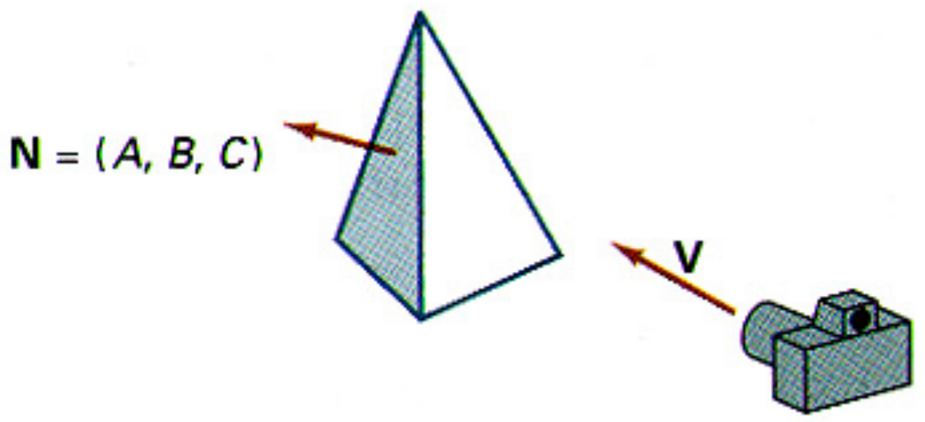
If $r_i = 0$, the point is in the plane.

If $r_i \neq 0$ then the point lies to one side of the plane or the other, depending on the sign of r_i

Backface Removal (cont.)

- If the point of view lies on the same side of the face as any other point on the object, then that face must be a **back face** and can be removed.
- If r_i < 0 then the plane is a backface with respect to the point of view and is therefore invisible.
- The normal to the plane can also be used to determine if a face is a back face.
- The coordinates for the normal to the plane, $\mathbf{N}$, are (A, B, C). If $\mathbf{V}$ is a vector in the viewing direction from the "eye" or camera position, then the polygon is a backface if:

$$\mathbf{V} \cdot \mathbf{N} > 0$$



Backface Removal (cont.)

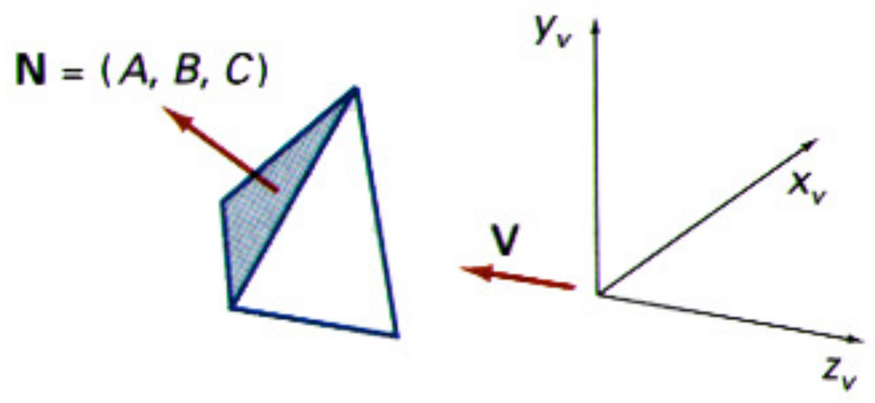
- If the polygon has been converted to projection or eye coordinates and our viewing direction is parallel to the viewing z axis, then

  $\mathbf{V} = (0,0,V_z)$ and $\mathbf{V.N} = V_z C$

  Thus we are only interested in the sign of C.

- For a right-handed viewing system, if $C < 0$ the normal points away from the viewing position and the plane must be a backface.

- In a left-handed coordinate system, $C > 0$ for backfaces. If $C = 0$ the viewing direction grazes the polygon and the polygon can thus be considered invisible.

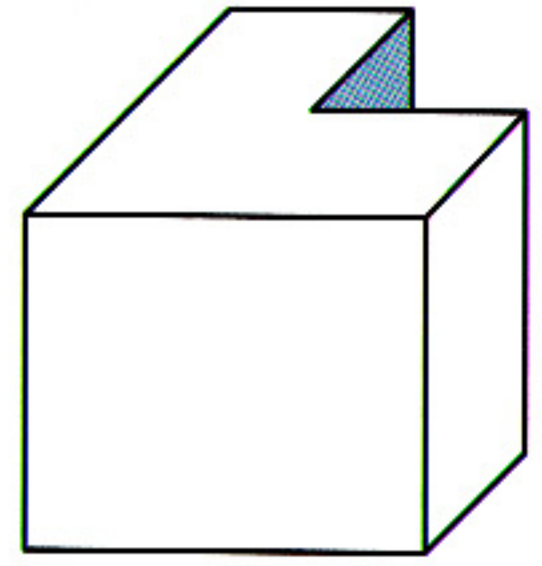


A polygon surface with plane parameter $C < 0$ in a right-handed viewing coordinate system is identified as a back face when the viewing direction is along the negative z_v axis.

Surface Normals

- For a right-handed coordinate system, given $\mathbf{v}_1, \mathbf{v}_2,\ldots\mathbf{v}_n$ ($n \geq 3$) are vertices of the polygon under consideration and that those vertices are not collinear and also that the vertices have been taken in an anti-clockwise order, then $(\mathbf{v}_3 - \mathbf{v}_2) \times (\mathbf{v}_2 - \mathbf{v}_1)$ will be the normal to the plane in which lie.
- For a left-handed system, the points are taken in clockwise order.
- Backface removal is most effective for convex objects. For concave objects faces may be partly visible.
- To enable backface removal in OpenGL:

```
glEnable(GL_CULL_FACE);
glCullFace(GL_BACK);
```



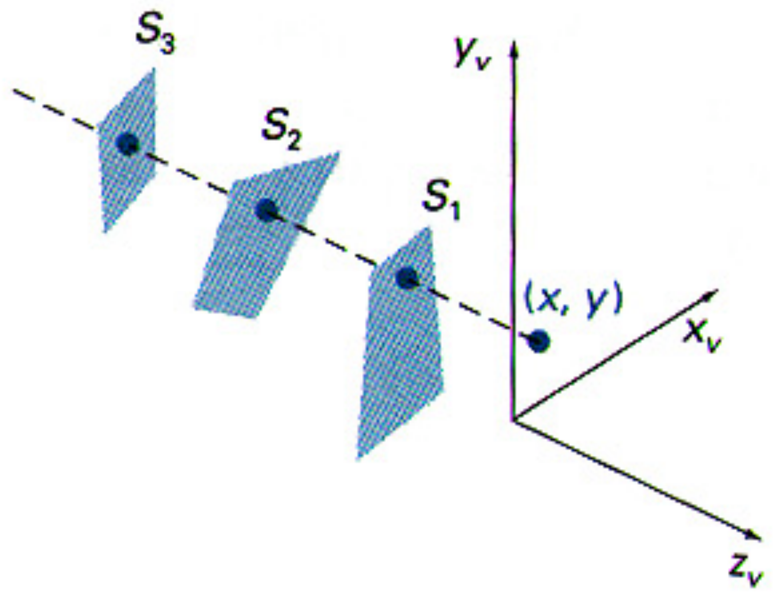
View of a concave polyhedron with one face partially hidden by other faces.

z-buffer Algorithm

Reference: Angel 8.8.4

- The *z-buffer* (or *depth buffer*) method is an image space approach to hidden surface elimination. It is one of the most widely used hidden surface removal algorithms, particularly in hardware.
- For each pixel (x, y) on the display screen we need to remember the intensity of the pixel and the z coordinate of the object voxel that was projected onto the pixel.
- As we project objects onto the depth buffer, we change the intensity value associated with the pixel at (x, y) if the point we are projecting has a z value less than the value remembered in the z buffer for (x, y), i.e. the projected pixel is closer to the viewer.

z-buffer Algorithm (cont.)



At view-plane position (x, y), surface S_1 has the smallest depth from the view plane and so is visible at that position.

Depth values for a surface position (x, y) are calculated from the plane equation for each surface:

$$z = \frac{-Ax - By - D}{C}$$

At each scan line, positions differ by 1.

$$z_{x+1,y} = \frac{-A(x+1) - By - D}{C}$$

or :

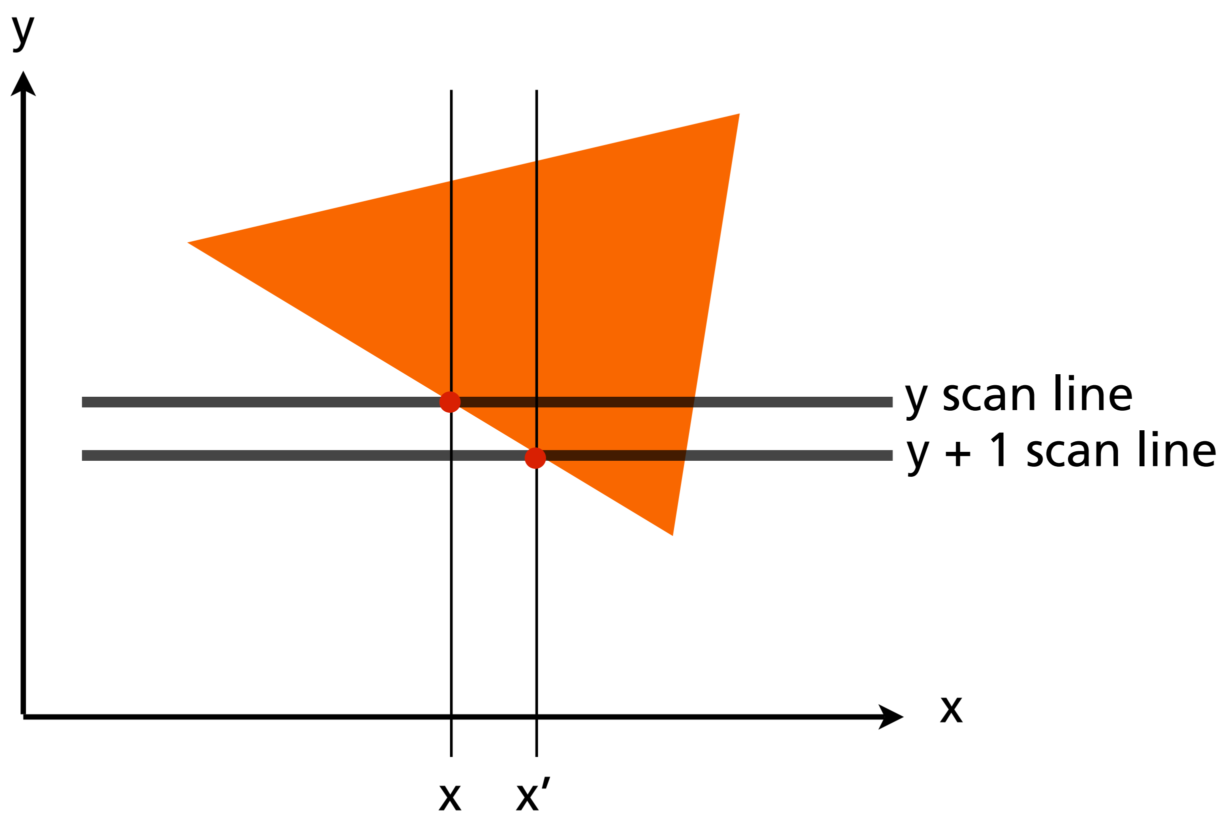
$$z_{x+1,y} = z_{x,y} - \frac{A}{C}$$

The ratio A/C is constant for each surface.

z-buffer Algorithm (cont.)

- We first determine the y-coordinate extents of each polygon and process the polygon from top to bottom. The x positions can be calculated recursively down the left edge as $x' = x - 1/m$ where m is the slope of the edge. Depth values down the edge are then calculated recursively as:

$$z' = z + \frac{A/m + B}{C}$$



OpenGL Hidden Surface Removal

- The z-buffer needs a certain number of bit planes to store both colour and depth information. In hardware implementations this is related to the size of the frame buffer.
- A 1280 x 1024 display might have 24 bits per pixel for colour and 16 or 24 bits per pixel for depth. The more bit planes allocated to the depth buffer, the more accurate it will be in correctly displaying visible surfaces.
- OpenGL uses a z-buffer for hidden surface removal. To use the z-buffer it first needs to be requested when initializing the display:
  ```
  glutInitDisplayMode(GLUT_RGB|GLUT_DOUBLE|GLUT_DEPTH)
  ```
- Depth buffering can be enabled with
  ```
  glEnable(GL_DEPTH_TEST)
  ```
- The depth buffer needs to be cleared before drawing each frame
  ```
  glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
  ```

Double Buffering

Reference: Angel 3.9.2

- It takes a finite amount of time to draw objects into the frame buffer. A complex object may take several refresh cycles of the display before it is completely drawn.
- If we change the contents of the frame buffer during a refresh cycle, we may see the object being drawn or other undesirable artifacts of how we generate the display.
- **Double buffering** is a technique that provides two identical buffers. The back buffer is never shown on the display, but is used to draw in by the graphics hardware. The front buffer is only used for image display.
- To draw an image, we draw into the back buffer. When drawing is completed we swap the buffers around (back becomes front, front becomes back). The swap operation can be done between refresh cycles.

Double Buffering in OpenGL

- Like depth buffering, double buffering needs to be set initialized

  ```
  glutInitDisplayMode(GLUT_RGB|GLUT_DOUBLE|GLUT_DEPTH)
  ```

- Drawing automatically takes place in the back buffer. The last line in the display function should call

  ```
      glutSwapBuffers( )
  ```

  This will swap the buffers and display the contents of the back buffer.

-
  ```
  void display ( void ) {
      /* clear the window */
      glClear( GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
      /* set camera and draw your objects here */
      . . .   /* swap GL buffers */
      glutSwapBuffers();
  }
  ```

The A-buffer Algorithm

- The A-Buffer is an extension to the depth buffer: an antialiased, area-averaged, accumulation-buffer method.
- The depth buffer cannot deal with transparent surfaces and has one visible surface at each pixel position.
- With the A-Buffer each position in the buffer can reference a linked list of surfaces. Each position in the buffer has 2 fields:
  - depth field – stores a positive or negative real number;
  - intensity field – stores surface intensity information or a pointer value (dependent on the sign of the depth field).

The A-buffer (cont.)

- Each field in the linked list includes:
  - RGB intensity components;
  - opacity parameter;
  - depth;
  - percent of area coverage;
  - surface id;
  - the surface rendering parameters;
  - pointer to next surface.
- This method is better suited to software rendering.
- For more information:

  Carpenter, L. (1984), The A-Buffer, an Antialiased Hidden Surface Method. *SIGGRAPH '84 Conference Proceedings* (Minneapolis, Minnesota, July 23-27, Christiansen, H., ed). In *Computer Graphics* **18**(3) ACM SIGGRAPH, New York, NY, pp. 103-108.