Computer Graphics

Hidden Surfaces

Md. Biplob Hosen
Assistant Professor, IIT-JU
Email: biplob.hosen@juniv.edu

Contents

- Hidden Surface Removal
- Hidden Surface Detection
- Depth Comparison
- Hidden Surface Detection Algorithms

Hidden Surface Removal

- Opaque objects that are closer to the eye and in the line of sight of other objects will block those objects or portion of those objects from view.
- In fact, some surfaces of these opaque objects themselves are not visible because they are eclipsed by the objects' visible parts.
- The surfaces that are blocked or hidden from view must be removed in order to construct a realistic view of 3D scene.
- The identification and removal of these surfaces is called the hidden-surface problem.
- One of the most challenging problems in computer graphics is the removal of hidden parts from images of solid objects.

Continue…

- In the computer generation, no such automatic elimination takes place when objects are projected onto the screen coordinate system.
- Instead, all parts of every object, including many parts that should be invisible are displayed.
- To remove these parts to create a more realistic image, we must apply a hidden line or hidden surface algorithm to set of objects.
- First we need hidden surface detection, then hidden surface elimination.

Hidden Surface Detection

- **Object Space Methods:** In this method, various parts of objects are compared. After comparison, visible, invisible or hardly visible surface is determined. These methods generally decide visible surface. In the wireframe model, these are used to determine a visible line. So these algorithms are line based instead of surface based. Method proceeds by determination of parts of an object whose view is obstructed by other object and draws these parts in the same color.
- **Image Space Methods:** Here positions of various pixels are determined. It is used to locate the visible surface instead of a visible line. Each point is detected for its visibility. If a point is visible, then the pixel is on, otherwise off. So the object close to the viewer that is pierced by a projector through a pixel is determined. That pixel is drawn is appropriate color.

Depth Comparison

- We assume that all coordinates (x, y, z) are described in the normalized viewing coordinate system.
- Any hidden surface algorithm must determine which edges and surfaces are visible either from the center of projection for perspective projections or along the direction of the projection for parallel projections.
- The question of visibility reduce to this:
  - Given two points: P_1 (x_1, y_1, z_1), P_2 (x_2, y_2, z_2), does either point obscure the other?
- This is answered in two steps:

  1. Are P1 and P2 on the same projection line?

  2. If not, neither point obscures the other. If so, a depth comparison tells us which point is in front of the other.

Continue…

- For parallel projection onto xy-plane, P_1, P_2 are on the same projector if $x_1=x_2$ and $y_1=y_2$.
- In this case, depth comparison reduces to comparing z_1 and z_1.
- If $z_1 < z_2$, then P_1 obscures P_2.
- For a perspective projection, the calculations are more complex.
- However, this complication can be avoided by transforming all three-dimensional objects; so that parallel projection of the transformed object is equivalent to a perspective projection of the original object.
- This is done with the use of perspective to parallel transformation.

Depth Comparison-Example

- Given points P_1 (1, 2, 0), P_2 (3, 6, 20) and P_3 (2, 4, 6), and a viewpoint C (0, 0, -10). Determine which points obscure the others when viewed from C.

Solution:

- The line joining the viewpoint C (0, 0, -10) and P_1 (1, 2, 0):
  - $x=t$ $x=x_1+t\Delta x$
  - $y=2t$ $y=y_1+t\Delta y$
  - $z=-10+10t$ $z=z_1+t\Delta z$
- To determine whether P_2 (3, 6, 20) lies on this line, wee see that:
  - $t=3$ [x=t]
  - For t=3, values have to be: x=t=3; y-2t=2*3=6; z=-10+10t=-10+(10*3)=20
  - Here, values for P_2 is same as it should have been.
  - So, P_2 lies on the projection line between C and P_1.
- Next we determine which point is in front with respect to C.
  - Here, C occurs on the line at t=0.
  - P_1 occurs at t=1.
  - P_2 occurs at t=3.
  - Thus comparing t values, P_1 is in front of P_2 with respect to C.
  - So, P_1 obscures P_2.

Continue…

- Given points P_1 (1, 2, 0), P_2 (3, 6, 20) and P_3 (2, 4, 6), and a viewpoint C (0, 0, -10). Determine which points obscure the others when viewed from C.

Solution:

- Now we determine whether P_3 (2, 4, 6) lies on this line:
  - t=2
  - For t=2, values have to be: x=t=2; y-2t=2*2=4; z=-10+10t=-10+(10*2)=10.
  - Here, values for P_3 is not same as it should have been.
  - So, P_3 does not lie on the projection line, and it neither obscure nor is obscured by P_1 and P_2.

Hidden Surface Detection Algorithms

- Some Hidden Surface Detection Algorithms-
  - Back Face Removal
  - Z-Buffer Algorithm
  - Painter Algorithm
  - Scan-Line Algorithm
  - Sub-division Algorithm

Back Face Removal Algorithm

- It is used to plot only surfaces which will face the camera.
- Object surfaces that are oriented away from the viewer or the camera are called back-face.
- The objects on the back side are not visible.
- We can therefore identify and remove these back-faces without further processing (projection and scan-conversion).
- This method will remove 50% of polygons from the scene if the parallel projection is used. If the perspective projection is used then more than 50% of the invisible area will be removed.
- It applies to individual objects. It does not consider the interaction between various objects.
- That's why, although this method works quickly, it can't handle polygons that face the viewer but are hidden behind other objects.
- It can be used as a preprocessing step for other algorithms.

Z-Buffer Algorithm

- We say that a point in display space is seen from pixel (x, y) if the projection of the point is scan-converted to this pixel.
- The Z-buffer algorithm essentially keeps track of the smallest z-coordinate (also called the depth value) of those points which are seen from pixel (x, y).
- These Z-values are stored in what is called the Z-buffer.
- Let, $Z_{buf}(x, y)$ denote the current depth value that is stored in the Z-buffer at pixel (x, y).
- We work with the projected polygons P of the scene to be rendered.
- The Z-buffer algorithm consist of the following steps:

1. Initialize the screen to a background color. Initialize the Z-buffer to the depth of the back clipping plane. That is, set:
   - $Z_{buf}(x, y) = Z_{back;}$ for every pixel (x, y)

Continue…

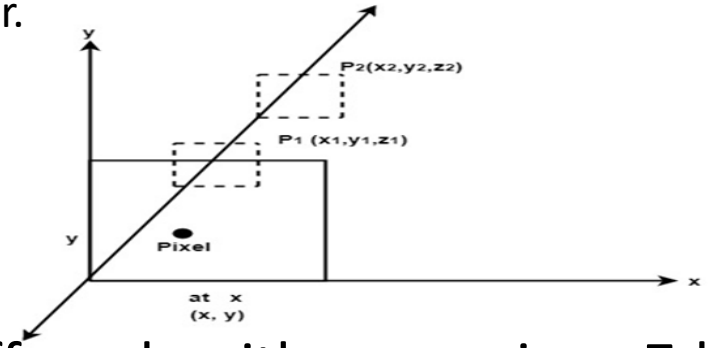
2. Scan-convert each (projected) polygon P in the scene and during this scan-conversion process, for each pixel (x, y) that lies inside the polygon:

a) Calculate Z(x, y), the depth of the polygon at pixel (x, y).

b) If $Z(x, y) < Z_{buf}(x, y)$, set $Z_{buf}(x, y) = Z(x, y)$ and set the pixel value at (x, y) to the color of the polygon P at (x, y).

c) In following figure, points P_1 and P_2 are both scan-converted to pixel (x, y);

d) However, since $z_1 < z_2$, P_1 will obscure P_2 and the z value of P_1- z_1 will be stored in the Z-buffer.



- Although the Z-buffer algorithm requires Z-buffer memory storage proportional to the number of pixels in the scene, it does not require additional memory for storing all the objects comprising the scene.
- In face, since the algorithm process polygons one at a time, the total number of objects in a scene can be arbitrarily large.

Limitations of Z-buffer algorithm

- The depth buffer Algorithm is not always practical because of the enormous size of depth and intensity arrays.
- Generating an image with a raster of 500 x 500 pixels requires 2, 50,000 storage locations for each array.
- Even though the frame buffer may provide memory for intensity array, the depth array remains large.
- To reduce the amount of storage required, the image can be divided into many smaller images, and the depth buffer algorithm is applied to each in turn.
- For example, the original 500 x 500 faster can be divided into 100 raster's each 50 x 50 pixels. Processing each small raster requires array of only 2500 elements, but execution time grows because each polygon is processed many times.
- Subdivision of the screen does not always increase execution time instead it can help reduce the work required to generate the image. This reduction arises because of coherence between small regions of the screen.

Painter Algorithm

- Depth sort algorithm or painter algorithm was developed by Newell, sancha. It is called the painter algorithm because the painting of frame buffer is done in decreasing order of distance. The distance is from view plane. The polygons at more distance are painted firstly.
- The concept has taken color from a painter or artist. When the painter makes a painting, first of all, he will paint the entire canvas with the background color. Then more distance objects like mountains, trees are added. Then rear or foreground objects are added to picture. Similar approach we will use. We will sort surfaces according to z values. The z values are stored in the refresh buffer.

Steps performed in-depth sort:

- Sort all polygons according to z coordinate.
- Find ambiguities of any, find whether z coordinate overlap, split polygon if necessary.
- Scan convert each polygon in increasing order of z coordinate.

Self-Study

- Scan-Line Algorithm
- Sub-division Algorithm

Thank You ☺