Computer Graphics

(Hidden Surface and Hidden Line Removal)

Lecture 009

Hidden Surface Removal

- Hidden surface removal is the process of removing surfaces (lines) which are not visible from the chosen viewing position.
- Hidden surface/line algorithms are often classified as one of the following:
  - Object space: algorithms which work on object definitions directly
  - Image space: algorithms which work on the projected image
- There is no one place in the viewing pipeline where these algorithms should be used – the ideal position in the pipeline is often specific to the particular algorithm used.
- Some algorithms do not perform hidden surface/line removal completely, but are designed to do some of the work early on in the pipeline, thereby reducing the amount of surfaces to be processed.

Back-face culling

- Back-face culling is an object-space algorithm designed to be used early on in the pipeline just before the projection transformation is applied.

- It is useful when the objects in the scene consist of solid polyhedra i.e. polyhedra whose **edges** are all used in exactly two surfaces, so that the inside faces of the polyhedra are never visible.

- In back-face culling, the direction of the surface normal is used to determine whether the surface should be kept or eliminated. The direction of the surface normal is such that it points towards a viewer from whose perspective the listed vertices appear in (anti-) clockwise order. This depends on the API used.

Back-face culling (cont.)

- Back-face culling eliminates all those surfaces whose normals point away from the COP, in the knowledge that such surfaces would be completely hidden by other front-facing surfaces of the same polyhedron.

- This can be tested by computing the dot product of the surface normal, and a vector from the COP to any point on the polygon. If this is > 0 then the surface is eliminated.

- In case of polyhedra with missing or clipped front-faces, back face culling leads to the creation of misleading images. One solution is to have two surfaces in opposite directions defined for each polygon. This is also useful to have different colours on either of its two sides.

The depth-Buffer (or z-buffer) method

- This **image space** algorithm makes use of a *depth buffer* and a *refresh buffer.* The depth buffer stores a depth value for each pixel in the projection window, whereas the refresh buffer specifies the colour for each pixel in the projection window.

- Assuming that the view volume has been converted to a regular parallelepiped with the z-coordinate varying from 0 to 1, the algorithm proceeds as follows:

  - Set all values in the depth buffer to 1 (ie furthermost depth)
  - Set all values in the refresh buffer to the background colour
  - Loop through all the surfaces to be drawn, applying the following procedure to each projected point (x,y) (typically obtained through scan-line algorithm)
    - Calculate z-value for (x,y) (using the equation of place of polygon)
    - If this z-value is less than the current value stored for (x,y) in the z-buffer, update the buffer to this new z-value and update the (x,y) slot in the refresh buffer to reflect the current polygon's colour (possibly adjusted to reflect the depth)
  - Finally, display the refresh buffer

The depth-Buffer (or z-buffer) method (cont.)

- The z-buffer algorithm is easy to implement (and understand)
- It is reasonably fast – no sorting required.
- Requires additional depth buffer – memory hungry (however with today's modern video cards this is hardly an issue)
- Optimisations:
  - If memory is an issue, divide the projection window into horizontal segments and work on each, one at a time, to reduce the z-buffer size. This clearly complicates the algorithm because we cannot deal with each surface in one loop (possibly) !!
  - Rather than calculate the value of z for each (x,y) from scratch, we can do this incrementally. If we know the depth value at (x,y) for the polygon S is z(S,x,y), and the equation of the plane is Ax + By + Cz = D, then using some equations (notes) we can derive an incremental value for z.

The Scan-line method

- This algorithm processes all polygons simultaneously, one scan-line at a time. Essentially this is an extension of the scan-line filling for polygons. Here's how it works:

- As the scan line is processed, all polygon surfaces intersecting with that line are examined to determine which surfaces are visible. At each position, the depth-value for each intersecting polygon is worked out, and the colour of the nearest polygon is chosen.

- Of course there is no need to calculate the depth value if only one polygon is active at a particular point.

Depth Sorting Method (painters' algorithm)

- This algorithm makes use if both image-space and object-space techniques. The basic steps are:
  - Sort the surfaces in order of decreasing greatest depth (object space).
  - Scan-convert surfaces in order, starting with the surface of greatest depth (image space)
- Of course, problems occur when polygons overlap in depth. Before a polygon S is scan-converted, the next polygon in the list should be checked for depth overlap. If there is none, the polygon surface can be processed, and the process can be applied to the next polygon in the list. If there is overlap however, some extra checks are required and re-ordering of the polygons may be necessary. The following tests (figure 8.4 notes) are carried out until one is found to succeed, in which case no re-ordering is necessary, or all of them fail, in which case additional processing is required:

Depth Sorting Method (painters' algorithm cont.)

1. The bounding rectangles in the xy plane for the surfaces do not overlap.
2. Surface S is on the outside of the overlapping surface, relative to the view plane.
3. The overlapping surface is on the inside of surface S, relative to the view plane.
4. The projections of the two surfaces ontp the view plane do not overlap (check for edge intersection)

- Note that these tests are ordered so that they become progressively more computing intensive. If all four tests fail, the two surfaces are swapped and the process restarted for the new current polygon.

- This can become quite an inefficient algorithm. The z-buffer algorithms is by far the most widely used depth-buffer algorithm. It is implemented in both DirectX and OpenGL.