The Visible Surface Problem

- Important for determining realistic images.
- The fundamental concept of visible surface determination is simple - find those surfaces which we can see and draw them.
- Two main implementations:
  - the pixel approach is:

```
for (each pixel in the image) {
   determine the object closest to the viewer,
      pierced by the projector through the pixel.
   draw the pixel in the appropriate colour.
}
```

The Visible Surface Problem

- Second implementation:
  - the object approach is:

```
for (each object in the image) {
   determine the parts of the object which are not
      obstructed by itself or other objects.
   draw those parts in the appropriate colour.
}
```

- Which is best to use depends upon the image being draw.

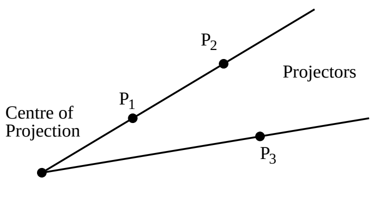
Image or Object precision?

- Assume we have an image with p pixels and n objects.
- The cost of the image-precision algorithm would be of order np.
- The cost of the object-precision algorithm would be of order n^2.
- The individual steps in the object-precision algorithm are more complex.
- Object-precision calculations have an advantage if we need to change the resolution.
- The optimal efficiency can be obtained by combining the benefits of both methods.

Coherence – some useful examples

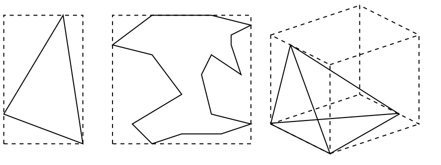
- Object coherence
- Face coherence / Area coherence
- Edge coherence
- Scan-line coherence
- Depth coherence
- Frame coherence – time

How to check visibility?



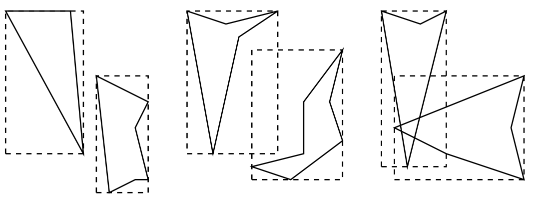
- For parallel projections we can simply test for occlusion of two points $p_1 = [x_1, y_1, z_1]'$ and $p_2 = [x_2, y_2, z_2]'$ by checking whether $x_1 = x_2$ and $y_1 = y_2$.
- For perspective projections we must first apply $N^*_{per} = MN_{per}$. This ensures that projectors are parallel to the z axis.

Extents and bounding volumes



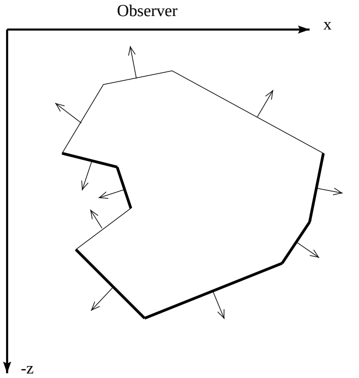
- Can simplify the problem (in object precision) using many methods.
- Bounding elements or volumes are commonly used.

Extents and bounding volumes

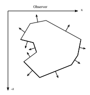


Three possible cases when using bounding elements.

Back Face Culling



Back Face Culling



- For solid objects, in both approaches we can roughly half the complexity.
- In the canonical view volume the DOP will be parallel to the z axis:

$$\begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} \; .$$

will be positive if the face is a back-face: in practice just test the sign of the z component.

Spatial partitioning and Hierarchical models

- By dividing the volume considered into a number of disjoint regions (such as used in quadtree and octree schemes) we can readily reduce the number of object comparisons.
- Speeds up both methods.
- It may often be the case that the bounding volume of the top level in the hierarchy will define the bounding volume of all the components in the hierarchy.
- This is an example of object coherence.

The z-buffer algorithm

- The most widely used algorithm, easily implemented in hardware.
- In addition to a frame buffer we also have a z-buffer which stores 16 to 32 bits of depth information.
- Simple to implement but increases memory requirements.
- The z-buffer is initialised to zero (back clipping plane).
- The largest z value (which depends on the number of bits used in the z-buffer) is allocated to the front clipping plane.
- Polygons are scan converted in arbitrary order.

```
 void zBuffer ()
{
    int pz; /* Polygons z at pixel (x,y) */
    for (y = ymin; y <= YMAX; y++) {
       for (x = xmin; x <= XMAX; x++) {
         WritePixel(x,y,BACKGROUND_VALUE);
         WriteZ(x,y,0);
       }
    }
    for (each polygon) {
       for (each pixel in the polygons prjn.) {
         pz = polygons z value at (x,y);
          if (pz >= ReadZ(x,y)) {
            WritePixel(x,y,polygon colour);
            WriteZ(x,y,pz);
         }
       }
    }
}
```

The z-buffer algorithm

- If the computation of the polygon colour (lighting model) is expensive, then some pre-sorting of the polygons will produce a speed up.
- The z-buffer algorithm combines scan conversion and visible surface determination.
- A-buffer is very much like the z-buffer algorithm but includes anti-aliasing.

The z-buffer algorithm

- We can use depth coherence to speed up the implementation – as we use scan line coherence in the mid-point line algorithm.
- If the polygon is planar we can write its equation as $ax + by + cz + d = 0$. We can solve this equation for z:

$$z = -\frac{ax + by + d}{c} \; ,$$

- Can use similar trick to scan conversion e.g.:

$$z_2 = z_1 - \frac{a}{c}\Delta x \; ,$$

when we only change the x direction.

Other algorithms

- Scan-line algorithms - active edge tables.
- The depth sort algorithm:
  - sort all polygons by their z coordinate;
  - resolve any ambiguities by splitting polygons that inter-penetrate;
  - scan convert the polygons in order, from the back to the front.
- Painter's algorithm, assigns a unique z value to each polygon.
- No inter-penetration allowed, thus works best in 2.5D.

Alternative methods

- Binary space partition trees (object precision).
  - Can be reused for any view angle - thus quick to recompute if only camera position changes.
- Visible surface ray tracing (image precision).
  - Has more powerful cousin, used in illumination modelling.
- Area subdivision algorithms (like quadtree) - divide image until it is easy to decide on occlusion.
  - Mix of both object and image precision methods.

Visible Surface Methods

- Can use either object or image precision methods.
- Both have advantages.
- z-buffer algorithm is the most simple and easily implemented.
- Some pre-sorting might help speed up algorithm.
- Also back-face culling and spatial partitioning are simple and fast.

Summary

- Having finished this lecture you should:
  - understand what visible surface determination is;
  - be able to contrast object and image precision approaches;
  - be able analyse the z-buffer algorithm;
  - know the various speed ups which can be used and understand why they work.
- Of course OpenGL implements visible surface determination for us in practice!.