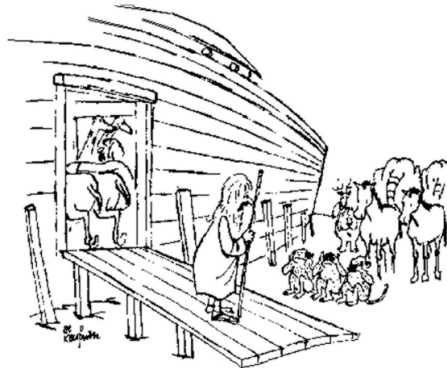


Visibility - Part 2



"I hate to break up the set, but one of you has to go."

[Lecture 15](#)

Slide 1

6.837 Fall '00



<http://graphics.lcs.mit.edu/classes/6.837/F00/Lecture15/Slide01.html> [11/9/2000 12:02:59 PM]

Visibility - Last Time

Back-Face Culling

Removes faces of closed (oriented manifold) objects that are facing away from the eye point.
Simple test based on the normal of each face.

Could be done in three ways:

- View-direction culling
(done after projection)
- Oriented-face culling
(done at triangle set-up)
- Viewpoint culling
(can be done anywhere)

$O(n)$

Painters Algorithm

Sort triangles by their depth
(min, max, centroid)
Subdivide cycle overlaps or intersecting triangles
 $O(n \log n)$



[Lecture 15](#)

Slide 2

6.837 Fall '00



<http://graphics.lcs.mit.edu/classes/6.837/F00/Lecture15/Slide02.html> [11/9/2000 12:03:03 PM]

Power of Plane Equations

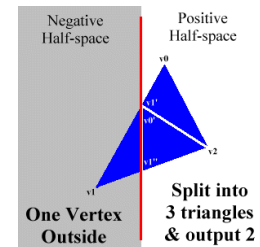
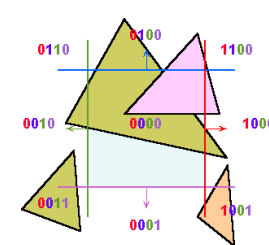
We've gotten a lot of mileage out of one simple equation.

Basis for 3D outcode-clipping

Basis for plane-at-a-time clipping

Basis for viewpoint back-face culling

$$\begin{bmatrix} n_x & n_y & n_z & -d \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = 0$$



[Lecture 15](#)

Slide 3

6.837 Fall '00



<http://graphics.lcs.mit.edu/classes/6.837/F00/Lecture15/Slide03.html> [11/9/2000 12:03:04 PM]

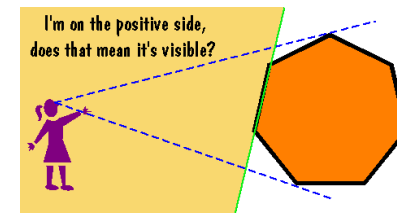
One More Trick with Planes

We can develop a visibility algorithm based on evaluating plane equations.

Consider the complement argument of the viewpoint culling algorithm.

- Any facet that contains the eye point within its negative half-space is invisible.
- Any facet that contains the eye point within its positive half-space is visible.

Well almost... it would work if there were no overlapping facets.
However, notice how the overlapping facets partition each other.
Suppose we build a tree of these partitions.



[Lecture 15](#)

Slide 4

6.837 Fall '00



<http://graphics.lcs.mit.edu/classes/6.837/F00/Lecture15/Slide04.html> [11/9/2000 12:03:06 PM]

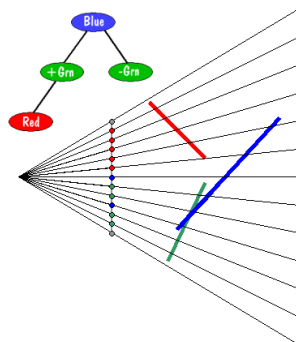
Constructing a BSP Tree

The algorithm to build a BSP tree is very simple:

1. Select a partitioning plane/facet.
2. Partition the remaining planes/facets according to the side of the partitioning plane that they fall on (+ or -).
3. Repeat with each of the two new sets.

Partitioning facets:

Partitioning requires testing all facets in the active set to find if they lie entirely on the positive side of the partition plane, entirely on the negative side, or if they cross it. In the case of a crossing facet we clip it into two halves (using our plane-at-a-time clipping algorithm).


[Lecture 15](#)

Slide 5

6.837 Fall '00


<http://graphics.lcs.mit.edu/classes/6.837/F00/Lecture15/Slide05.html> [11/9/2000 12:03:07 PM]

BSP Tree Example

Computing visibility or depth-sorting with BSP trees is both simple and fast.

It resolves visibility at the primitive level.

Visibility computation is independent of screen size

Requires considerable preprocessing of the scene primitives

Primitives must be easy to subdivide along planes

Supports CSG


[Lecture 15](#)

Slide 7

6.837 Fall '00

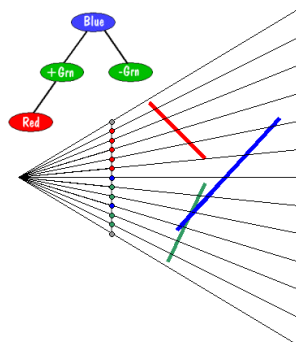

<http://graphics.lcs.mit.edu/classes/6.837/F00/Lecture15/Slide07.html> [11/9/2000 12:03:09 PM]

Computing Visibility with BSP trees

Starting from the root of the tree.

1. Classify viewpoint as being in the positive or negative halfspace of our plane
2. Call this routine with the negative child (if it exists)
3. Draw the current partitioning plane
4. Call this routine with the positive child (if it exists)

Intuitively, at each partition, we first draw the stuff further away than the current plane, then we draw the current plane, and then we draw the closer stuff. BSP traversal is called a "hidden surface elimination" algorithm, but it doesn't really "eliminate" anything; it simply orders the drawing of primitive in a back-to-front order like the Painter's algorithm.


[Lecture 15](#)

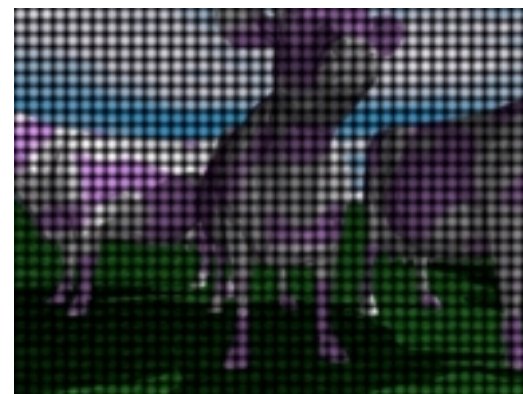
Slide 6

6.837 Fall '00


<http://graphics.lcs.mit.edu/classes/6.837/F00/Lecture15/Slide06.html> [11/9/2000 12:03:08 PM]

Pixel-level Visibility

Thus far, we've considered visibility at the level of primitives. Now we will turn our attention to a class of algorithms that consider visibility at the level of each pixel.


[Lecture 15](#)

Slide 8

6.837 Fall '00


<http://graphics.lcs.mit.edu/classes/6.837/F00/Lecture15/Slide08.html> [11/9/2000 12:03:10 PM]

Ray-Casting

Algorithm:

Cast a ray from the viewpoint through each pixel to find the closest surface

Rendering Loop:

```
foreach pixel in image
  compute ray for pixel
  set depth = ZMAX;
  foreach primitive in scene
    if (ray intersects primitive) then
      if (distance < depth) then
        pixel = object color
        depth = distance to object
      endif
    endif
  endfor
endfor
```


[Lecture 15](#)

Slide 9

6.837 Fall '00


<http://graphics.lcs.mit.edu/classes/6.837/F00/Lecture15/Slide09.html> [11/9/2000 12:03:12 PM]

Ray Casting Advantages

Conceptually simple

Can support CSG

Can take advantage of spatial coherence in scene

Can be extended to handle global illumination effects
(ex: shadows and reflectance)


[Lecture 15](#)

Slide 10

6.837 Fall '00


<http://graphics.lcs.mit.edu/classes/6.837/F00/Lecture15/Slide10.html> [11/9/2000 12:03:13 PM]

Ray Casting Disadvantages

Renderer must have access to entire retained model

Hard to map to special-purpose hardware

Visibility determination is coupled to sampling

Subject to aliasing

Visibility computation is a function of resolution


[Lecture 15](#)

Slide 11

6.837 Fall '00


<http://graphics.lcs.mit.edu/classes/6.837/F00/Lecture15/Slide11.html> [11/9/2000 12:03:14 PM]

Depth Buffering

Algorithm:

Cast a ray from the viewpoint through each pixel to find the closest surface

Rendering Loop:

```
set depth of all pixels to ZMAX
foreach primitive in scene
  determine pixels touched
  foreach pixel in primitive
    compute z at pixel
    if (z < depth) then
      pixel = object color
      depth = z
    endif
  endfor
endfor
```


[Lecture 15](#)

Slide 12

6.837 Fall '00


<http://graphics.lcs.mit.edu/classes/6.837/F00/Lecture15/Slide12.html> [11/9/2000 12:03:15 PM]

Depth-Buffering Advantages

Primitives can be processed immediately
Hence: Immediate mode graphics API's

Primitives can be processed in any order
Exception: primitives at same depth

Well suited to H/W implementation
simple control of low-level (per pixel) operations

Spatial coherence
Incremental evaluation of loops
Good memory access pattern



Lecture 15

Slide 13

6.837 Fall '00


<http://graphics.lcs.mit.edu/classes/6.837/F00/Lecture15/Slide13.html> [11/9/2000 12:03:16 PM]

Depth-Buffering Disadvantages

Visibility determination is coupled to sampling
Subject to aliasing

Requires a Raster-sized array to store depth

Read-Modify-Write
Hard to make fast

Excessive over-drawing



Lecture 15

Slide 14

6.837 Fall '00


<http://graphics.lcs.mit.edu/classes/6.837/F00/Lecture15/Slide14.html> [11/9/2000 12:03:17 PM]

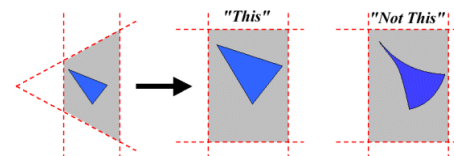
What Exactly Gets Stored in a Depth-Buffer?

Recall that we augmented our projection matrix to include a mapping for z values.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{width \times near}{right - left} & 0 & \frac{-left \times width}{right - left} & 0 \\ 0 & \frac{height \times near}{bottom - top} & \frac{-height \times top}{bottom - top} & 0 \\ 0 & 0 & \frac{z_{max} \times far}{far - near} & \frac{-z_{max} \times far \times near}{far - near} \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

There are two important considerations.

1. The mapping is from 3D (Affine) to 3D (Projective)
2. The mapping is linear (in general, planes map to planes)



Lecture 15

Slide 15

6.837 Fall '00

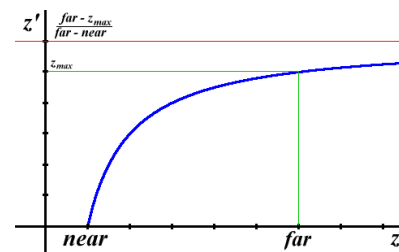

<http://graphics.lcs.mit.edu/classes/6.837/F00/Lecture15/Slide15.html> [11/9/2000 12:03:19 PM]

Computing the "Z" Term

We get the following expression for z' from our projection matrix:

$$z' = \frac{far \cdot z_{max} \cdot (z - near)}{z \cdot (far - near)} = \frac{far \cdot z_{max}}{far - near} \left(1 - \frac{near}{z} \right)$$

This mapping of z values is non-linear:



Lecture 15

Slide 16

6.837 Fall '00


<http://graphics.lcs.mit.edu/classes/6.837/F00/Lecture15/Slide16.html> [11/9/2000 12:03:21 PM]

Linear yet Non-Linear?

What does it mean for the projection mapping to preserve planes and lines, yet, have a non-linear mapping of z values. How can this be?

Consider these examples:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + t \begin{bmatrix} dx \\ dy \end{bmatrix} \quad \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + t^2 \begin{bmatrix} dx \\ dy \end{bmatrix} \quad \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + \frac{2t}{t+1} \begin{bmatrix} dx \\ dy \end{bmatrix}$$

As the parameter t is varied from 0 to 1, what geometric object is described?

Linearity of the space is preserved, but *linearity of the parameterization* is not.

So, suppose we have function of that maps one of these forms to one of the others. Allow this function to *even* change the actual values of x_0 , y_0 , dx , and dy , as long as this **linear combination** form is preserved.

Does this mapping preserve lines?
What is the advantage of doing this?
Why do we care?


[Lecture 15](#)

Slide 17

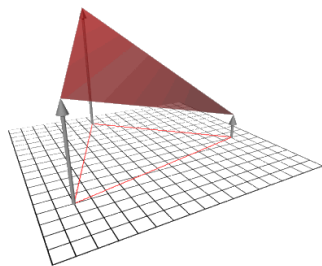
6.837 Fall '00



<http://graphics.lcs.mit.edu/classes/6.837/F00/Lecture15/Slide17.html> [11/9/2000 12:03:23 PM]

Interpolating Z

Preserving the linearity of the space allows us to use our plane equation method for interpolating



values of z in the interior of the triangle.

Thus, we can used

$$\frac{1}{2\text{area}} \begin{bmatrix} A_2 & A_3 & A_1 \\ B_2 & B_3 & B_1 \\ C_2 & C_3 & C_1 \end{bmatrix} \begin{bmatrix} z_0 \\ z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} A_r \\ B_r \\ C_r \end{bmatrix}$$

our handy formula for interpolating parameters within a triangle.


[Lecture 15](#)

Slide 18

6.837 Fall '00

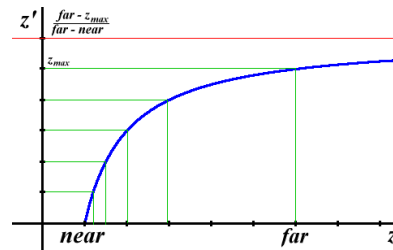


<http://graphics.lcs.mit.edu/classes/6.837/F00/Lecture15/Slide18.html> [11/9/2000 12:03:24 PM]

Monotonic Z values

We need to be careful when reading the values out of a z-buffer and interpolating them. Eventhough, our interpolated values of z lie on a plane, uniform differences in depth-buffer values do not correspond to a uniform differences in space.

However, our z-comparisons will still work because this parameter mapping, while not linear, is monotonic.



Note that when the z values are uniformly quantized the number of discrete discernable depths is greater closer to the near plane than near the far plane. Is this good?


[Lecture 15](#)

Slide 19

6.837 Fall '00



<http://graphics.lcs.mit.edu/classes/6.837/F00/Lecture15/Slide19.html> [11/9/2000 12:03:25 PM]

Next Time


[Lecture 15](#)

Slide 20

6.837 Fall '00

<http://graphics.lcs.mit.edu/classes/6.837/F00/Lecture15/Slide20.html> [11/9/2000 12:03:27 PM]