Rendering

- Rasterizing Lines and Polygons
- Hidden Surface Remove
- Multi-pass Rendering with Accumulation Buffers

9/23/02

Basic Math Review

Slope-Intercept Formula For Lines

Given a third point on the line:

P = (X,Y)

Slope = (Y - Y1)/(X - X1)

= (Y2 - Y1)/(X2 - X1)

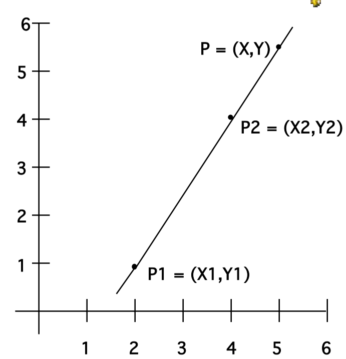
Solve for Y

Y = [(Y2-Y1)/(X2-X1)]X +

[-(Y2-Y1)/(X2-X1)]X1 + Y1

or

Y = mx + b



$$\text{SLOPE} = \frac{\text{RISE}}{\text{RUN}} = \frac{\text{Y2-Y1}}{\text{X2-X1}}$$

9/23/02

Basic Math Review: Other Helpful Formulas

Length of line segment between P1 and P2:

$L = \sqrt{[\ (X2\text{-}X1)2 + (Y2\text{-}Y1)2\ ]}$

Midpoint of a line segment between P1 and P3:

P2 = ((X1+X3)/2 , (Y1+Y3)/2)

Two lines are perpendicular iff:

M1 = -1/M2

9/23/02

Basic Math Review: Parametric Form of a 2D Line

Given points P1 = (X1, Y1) and P2 = (X2, Y2)

X = X1 + t(X2-X1)

Y = Y1 + t(Y2-Y1)

When

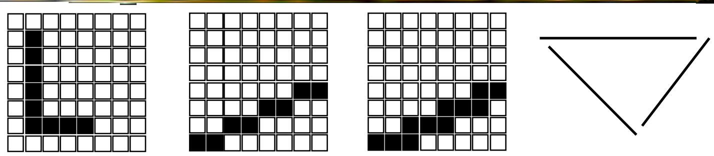
t = 0 we get (X1,Y1)

t = 1 we get (X2,Y2)

As 0 < t < 1, we get all points between (X1,Y1) and (X2,Y2)

9/23/02

Basic Line Algorithm



Must:

1. Compute integer coordinates of pixels which lie on or near a line.
2. Be efficient.
3. Create visually satisfactory images.
   - Lines should appear straight
   - Lines should terminate accurately
   - Lines should have constant density
   - Line density should be independent of line length and angle
4. Always be defined.

9/23/02

Simple DDA Line Algorithm

{Based on the parametric equation of a line}

```
Procedure DDA(X1,Y1,X2,Y2 :Integer);
Var  Length, I              :Integer;
     X,Y,Xinc,Yinc          :Real;

Begin
     Length := ABS(X2 - X1);
     If ABS(Y2 - Y1)  > Length Then
          Length := ABS(Y2-Y1);
     Xinc := (X2 - X1)/Length;
     Yinc := (Y2 - Y1)/Length;
     X := X1;
     Y := Y1;
     For I := 0 To Length Do
     Begin
          Plot(Round(X), Round(Y));
          X := X + Xinc;
          Y := Y + Yinc
     End  {For}
End;  {DDA}
```

- Creates good lines, but problems …

9/23/02

DDA Example

Render the line from (6,9) to (11,12):

Length := Max of (ABS(11-6), ABS(12-9)) = 5

Xinc := 1

Yinc := 0.6

Values computed are:

(6,9),

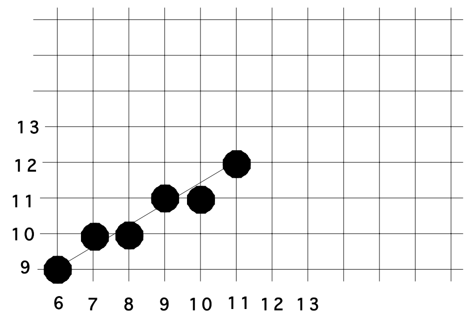
(7,9.6),

(8,10.2),

(9,10.8),

(10,11.4),

(11,12)

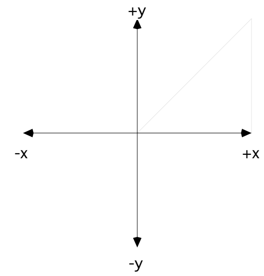


9/23/02

Fast Lines Using The Midpoint Method

Assumptions: line between points (0,0) and (a,b) with slope $0 \le m \le 1$

i.e. lies in first octant:



Recall: $y = mx + B$ (m is the slope, B is the y-intercept)

$\Rightarrow m = b/a$ and $B = 0$

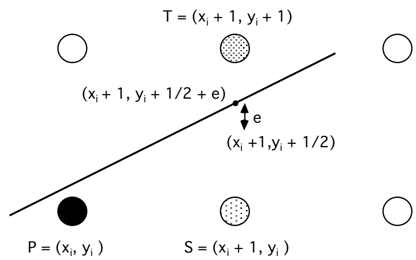
$\Rightarrow y = (b/a)x + 0$

$\Rightarrow f(x,y) = bx - ay = 0$

9/23/02

Fast Lines (cont.)

Two choices for next pixel (T or S), want the pixel closer to line!



Assume distance between pixel centers is 1

Midpoint is (xi + 1,yi + 1/2)

e is difference between midpoint and where line crosses between S and T

If e is positive, line crosses above the midpoint and is closer to T

If e is negative, line crosses below the midpoint and is closer to S

⇒ don't need exact value of e

9/23/02

Fast Lines: The Decision Variable

f(xi+1,yi+ 1/2 + e) = b(xi+1) - a(yi+ 1/2 + e) = b(xi + 1) - a(yi + 1/2) -ae
= f(xi + 1, yi + 1/2) - ae = 0

Let di = f(xi + 1, yi + 1/2) = ae; di is known as the decision variable.

Since a ≥ 0, di has the same sign as e.

Algorithm:

If di ≥ 0 then
- Choose T = (xi + 1, yi + 1) as next point
- di+1 = f(xi+1 + 1, yi+1 + 1/2) = f(xi +1+1, yi +1+1/2)
  = b(xi +1+1) - a(yi +1+1/2) = f(xi + 1, yi + 1/2) + b - a
  = di + b - a

else
- Choose S = (xi + 1, yi) as next point
- di+1 = f(xi+1 + 1, yi+1 + 1/2) = f(xi +1+1,yi +1/2)
  = b(xi +1+1) - a(yi +1/2) = f(xi + 1, yi + 1/2) + b
  = di + b

9/23/02

Fast Line Algorithm

Calculate initial value for d_0 directly from f(x,y) at (0,0):

$d_0 = f(0 + 1, 0 + 1/2) = b(1) - a(1/2) = b - a/2$

Algorithm for a line from (0,0) to (a,b) in the first octant is:

```
x := 0;
y := 0;
d := b - a/2;
For i := 0 to a do Begin
        Plot(x,y);
        If d ≥ 0 Then Begin
                x := x + 1;
                y := y + 1;
                d := d + b - a
        End
        Else Begin
                x := x + 1;
                d := d + b
        End
End
```

The only non-integer value is a/2. How can we get rid of it?

9/23/02

Bresenham's Line Algorithm

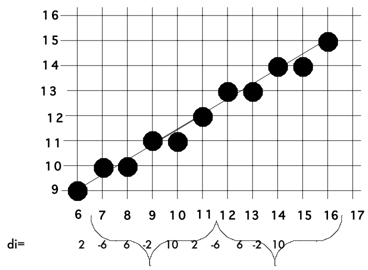
Generalize for lines beginning at points other than (0,0)

```
Begin  {Bresenham for lines with slope between 0 and 1}
    a := ABS(xend - xstart);
    b := ABS(yend - ystart);
    d := 2*b - a;
    Incr1 := 2*(b-a);
    Incr2 := 2*b;
    If xstart > xend Then Begin
        x := xend;
        y := yend
    End
    Else Begin
        x := xstart;
        y := ystart
    End;
    For I := 0 to a Do Begin
        Plot(x,y);
        x := x + 1;
        If d ≥ 0 Then Begin
            y := y + 1;
            d := d + incr1
        End
    Else
        d := d + incr2
    End  {For Loop}
End;  {Bresenham}
```

9/23/02

Optimizations

- Detect cycles in the decision variable
  - correspond to a repeated pattern of pixel choices
- Save pattern, reuse if a cycle is detected

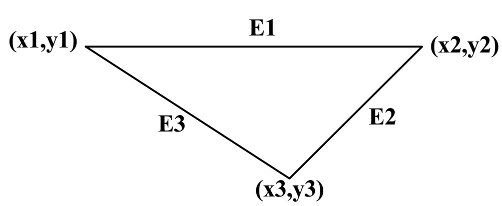


9/23/02

Polygons

- Polygon: many-sided planar figure of vertices and edges
- Vertices: represented by points (x,y)
- Edges: represented as line segments between two points, (x_i,y_i) and (x_{i+1},y_{i+1})

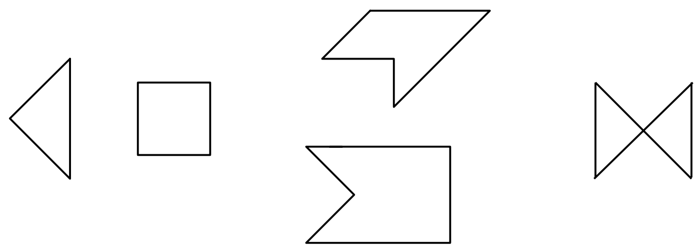
$P = \{ (x_i , y_i) \}$ i=1,n



9/23/02

Convex and Concave Polygons

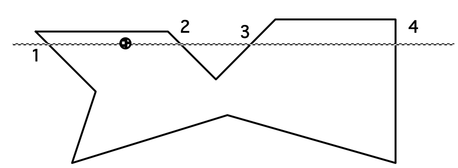
- Convex Polygon:
  - Given P1, P2 inside polygon
  - All P = uP1 + (1-u)P2, u in [0,1] is inside polygon
- Concave = not convex!



9/23/02

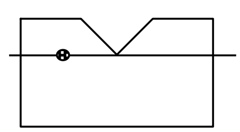
How do we know a point is "inside" a polygon?

- P is inside a polygon iff a scanline intersects the polygon edges an odd number of times moving from P in either direction

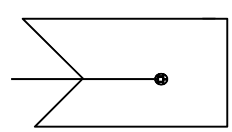


- Problem when scan line crosses a vertex:

Does the vertex count as two points?



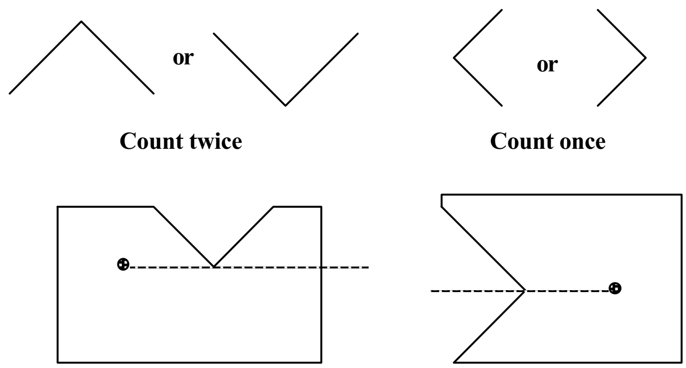
Or should it count as one point?



9/23/02

Max-Min Test

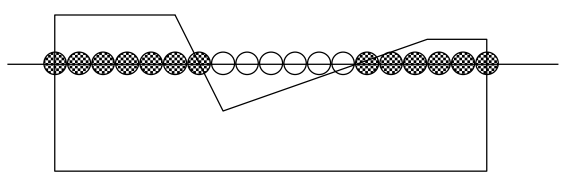
- Vertex = local max or min
  - Count it twice, else count it once.



9/23/02

Filling Polygons

- Fill polygon 1 scanline at a time

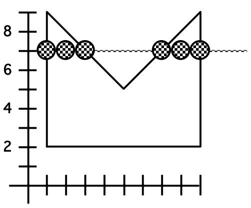


- Set pixels inside polygon on each scanline to the appropriate value
- Look only for those pixels at which changes occur

9/23/02

Scan-Line Algorithm

- For each scan-line:
  1. Find intersections of scan line with all edges
  2. Sort intersections by increasing x-coordinate
  3. Fill all pixels between pairs of intersections



For scan-line number 7 the sorted list of x-coordinates is (1,3,7,9)

How do we know a point is "inside" a polygon?

9/23/02

Possible Problems

- Horizontal edges
  - Ignore
- Vertices
  - If local max or min, count twice, else count once (Implemented by shortening one edge by one pixel)
- Calculating intersections is slow

9/23/02

Edge Coherence

- Observations:
  - Not all edges intersect each scanline
  - If edge intersected in scanline i, will probably be intersected by scanline i+l
- Consider scanline s, the line y = s

$$s = mx^s + b$$

$$\Rightarrow x^s = (s\text{-}b)/m$$

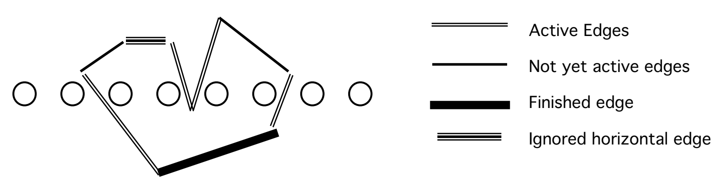
- For scanline s + 1,

$$x^s+1 = (s+1 - b)/m = x^s + 1/m$$

9/23/02

Processing Polygons

- Polygon edges sorted according to minimum Y
- Scan lines processed in increasing (decreasing) Y order
- When current scan line reaches edge, it becomes active
- When current scan line passes edge, it becomes inactive

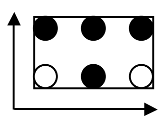


- Active edges sorted according to increasing X
- Fill the scan line between alternating edge intersections

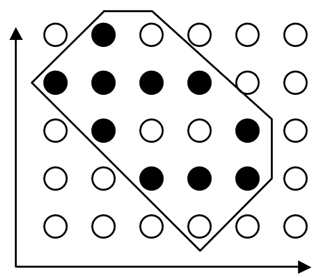
9/23/02

Fill Patterns: Simple "textures"

- Defined as a 0-based, m x n array
- Pixel (x,y) is assigned the value found in:
  - pattern((x mod m), (y mod n))



Pattern

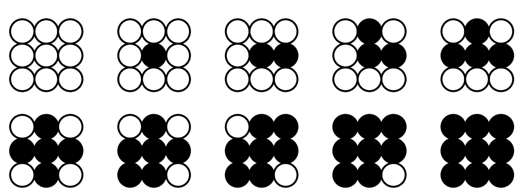


Pattern filled polygon

9/23/02

Halftoning

- Mimic greyscale on bitmapped displays
- Tradeoff resolution (addressability) for range of intensities
- Patterns should be designed to avoid being noticed



9/23/02

Dithering

- Another way to mimic grey on bit-mapped displays
- Ordered dither: turn pixel on or off at (x,y) based on
  - desired intensity I(x,y) at that point
  - an (n by n) dither matrix Dn
- Each integer 0 to n2 -1 appears once in the matrix Dn

e.g. D4

0	8	2	10
12	4	14	6
3	11	1	9
15	7	13	5

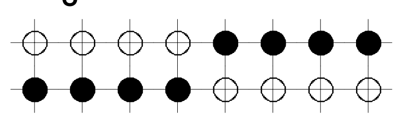
- let i = x MOD 4, j = y MOD 4
- if I(x,y) > D4(i, j) then (x,y) is turned on; otherwise it is not

9/23/02

Antialiasing

- Aliasing caused by finite addressability of CRT
- Approximation of lines with discrete points can result in a staircase appearance or "Jaggies"
- Desired line

- Aliased rendering of the line

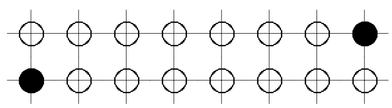


9/23/02

Antialiasing - Solutions

- Aliasing can be smoothed out by using higher addressability.
- Problem: addressability usually fixed

- Solution: intensity is variable, so use it
  - ⇒ two adjacent pixels can give impression of point part way between
  - ⇒ perceived location of point dependent upon ratio of intensities

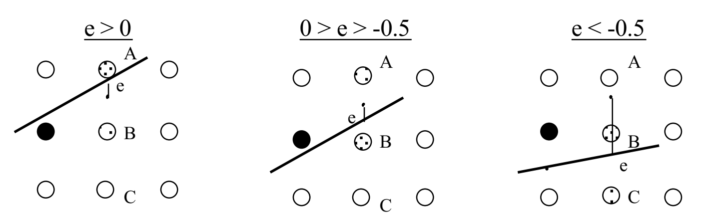


- An antialiased line has virtual pixels "located" at the proper addresses

9/23/02

Antialiased Bresenham Lines

- Use the distance (e = di/a) value to determine pixel intensities.
- Three possible cases for the Bresenham algorithm:



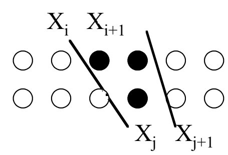
$e > 0$	$0 > e > -0.5$	$e < -0.5$
A = 0.5 + e	A = 0.5 + e	A = 0
B = 1 - abs(e+0.5)	B = 1 - abs(e+0.5)	B = 1 - abs(e+0.5)
C = 0	C = 0	C = -0.5 - e

- What about color?

9/23/02

Antialiasing Polygons

- Polygon edges suffer from aliasing as lines
- Similar method can be used on the scan line fill

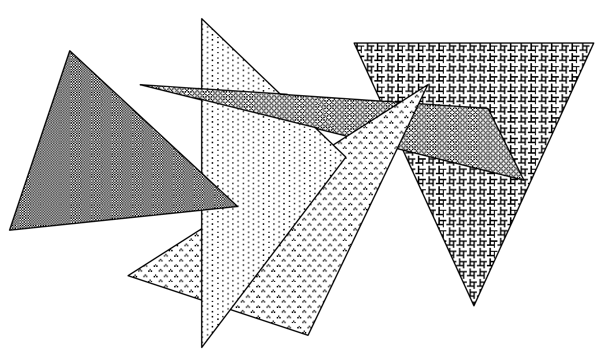


- If odd intersection between two pixels Xi < X < Xi+1
  - pixel Xi is assigned the intensity (Xi+1 -X)
  - pixel Xi+1 is assigned intensity 1.0
- At even intersection, reverse is true

9/23/02

Hidden Surface Elimination

(Visible Surface Determination)



9/23/02

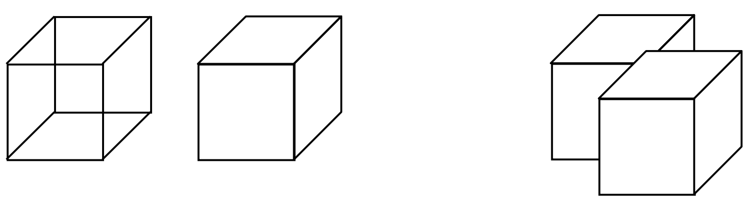
Approaches

1. Back-Face Removal
2. z-Buffer (Depth-Buffer)
3. Depth-Sort
4. BSP-Tree
5. Scanline Algorithm (see book)

9/23/02

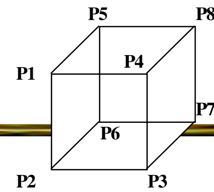
1) Back-Face Removal (Culling)

- Remove unseen polygons from convex, closed polyhedron (Cube, Sphere)
- Does not completely solve problem
  - One polyhedron may obscure another



9/23/02

Back Face Algorithm



- Idea. For each polygon:
  - If surface normal faces toward eye, keep
  - If surface normal faces away from eye, toss
- Adopt convention for vertex order
  - ie. assume counter-clockwise w.r.t. front
    - (P1, P2, P3, P4)
  - Compute N

9/23/02

Back Face Algorithm

- Look at surface normal
  - In World Coordinates
    - If $A\,xe + B\,ye + C\,ze + D < 0$
      The polygon is a backface.
  - After Normalizing/Perspective Projection?
    - What is (xe, ye, ze)?

9/23/02

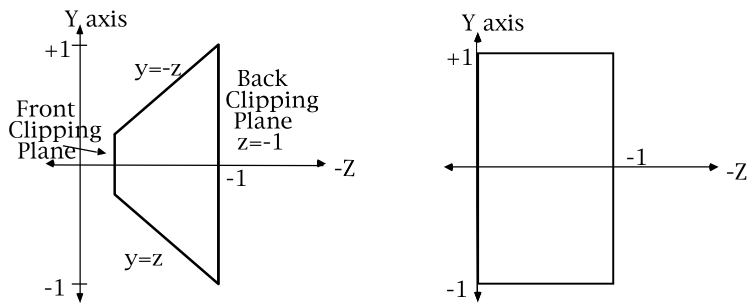
2) z-Buffer (Depth-Buffer)

- Look at each pixel
  - Pixel shows closest object in world
- We have all info to compute z(x,y)

9/23/02

Which Z?

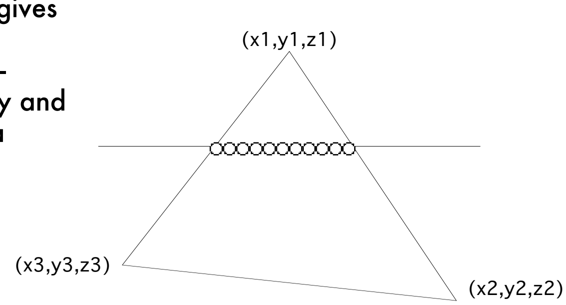
- Recall canonical view volumes:



9/23/02

Computing Pixel z-values

- Perspective projection gives z-values for vertices of polygons. To find the z-values for the boundary and interior pixels you do a linear interpolation



- Vertically: $z_{i+1} = z_i + \Delta zv$, $\Delta zv = (z1-z3)/(y1-y3)$
- Horizontally: $z_{i+1} = z_i + \Delta zh$, $\Delta zh = (z1-z3)/(x1-x3)$

9/23/02

z-Buffer Algorithm

- Initialize:
  - Each z-buffer location ⇐ Max z value
  - Each frame buffer loc. ⇐ background color
- For each polygon:
  - Compute z(x,y), depth at the pixel (x,y)
  - If z(x,y) < z buffer value at pixel (x,y) then
    - z buffer(x,y) ⇐ z(x,y)
    - pixel(x,y) ⇐ color of polygon at (x,y)

9/23/02

z-Buffer

- Advantages
  - Linear performance
  - Polygons may be processed in any order
  - Hardware implementation ⇒ very fast
- Disadvantages
  - Lots of memory (nowadays … so what?)
  - Problems with linear interpolation under perspective
  - Modifications needed for antialiasing, transparency, translucency effects

9/23/02

3) Depth Sort

- Sort polygons by distance
- Paint in back-to-front order
- Problems?

9/23/02

4) Binary Space Paritition (BSP) Trees

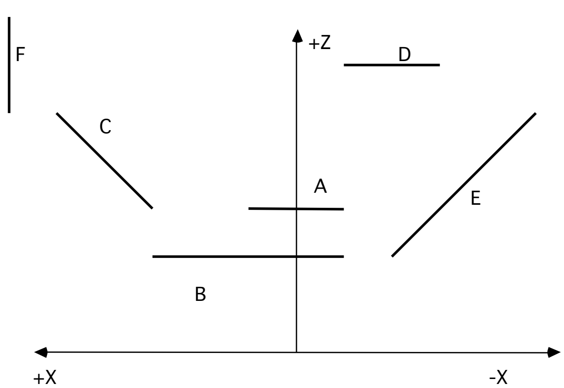
Easy way to sort the polygons relative to eye-point

To Build a BSP Tree

1. Choose a polygon, T, and compute the equation of the plane it defines.
2. Test all the vertices of all the other polygons to determine if they are in front of, behind, or in the same plane as T. If the plane intersects a polygon, divide the polygon at the plane.
3. Polygons are placed into a binary search tree with T as the root.
4. Call the procedure recursively on the left and right subtree.

9/23/02

BSP Tree Example



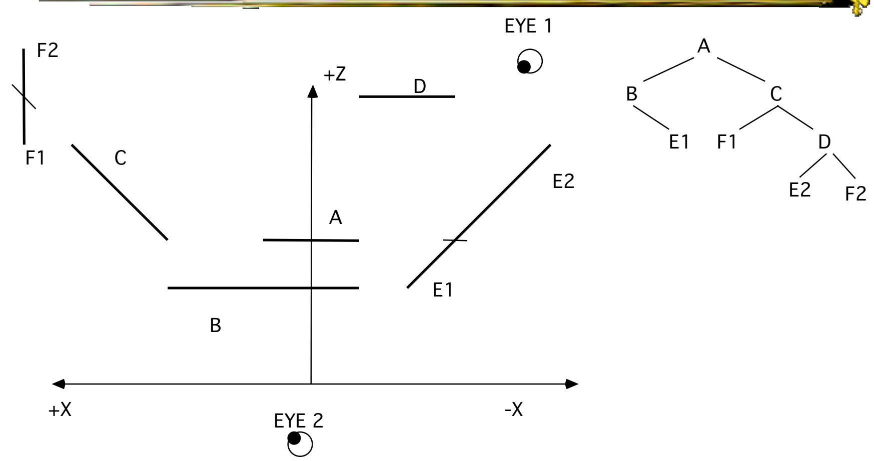
9/23/02

Traversing the BSP-Tree

- Traverse the BSP tree such that the branch decended first is the side that is away from the eyepoint. This can be determined by substituting the eye point into the plane equation for the polygon at the root.
- When there is no first branch to descend, or that branch has been completed then render the polygon at this node.
- After the current node's polygon has been rendered, descend the branch that is closer to the eyepoint.

9/23/02

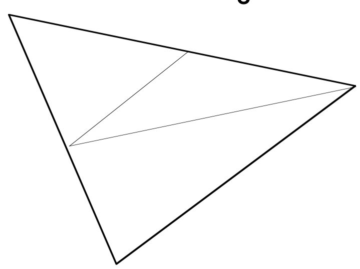
Traversing the BSP Tree Example



9/23/02

Splitting Triangles

- If all our polygons are triangles when we always divide a triangle into more triangles when it is intersected by the plane.
- Possible for the number of triangles to increase exponentially but in practice it is found that the increase may be as small as two fold
- A heuristic to help minimize the number of fractures is to enter the triangles into the tree in order from largest to smallest.



9/23/02

Accumulation Buffers

- Multi-pass rendering
  - For anitaliasing, depth of field, soft shadows, motion blur
- Render scene multiple times, different params
  - Viewpoint (antialiasing, depth of field)
  - Time (motion blur)
  - Light position (soft shadows)
- Accumulate rendered images into accum buffer
  - Using ops such as + and * to "add with weight"

9/23/02