Introduction to Graphics

Lecture 7:

Hidden Line Removal

Lecture Overview

- Wireframe Representation
- The Painter's Algorithm
- The Z-Buffer
- Hidden Line Removal in Convex Objects
- General Purpose Hidden Line Algorithm
  - Line and Face Edge Intersections
  - Back Projection

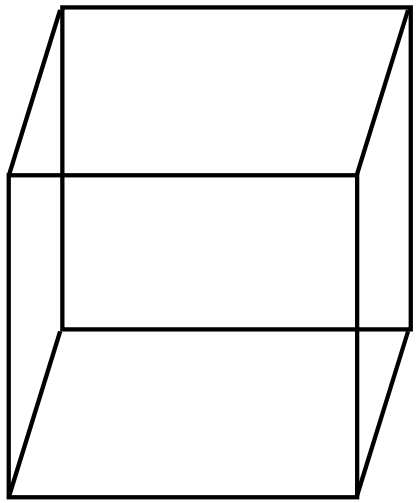
Wireframe Representation

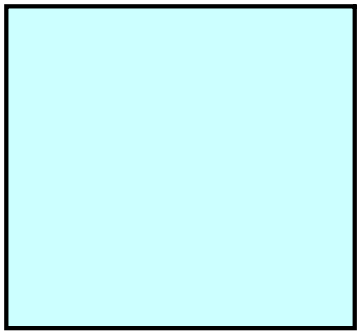
- So far our treatment has concerned only points and lines.
- This has utility in some applications, but for anything of complexity wire frame representations become confusing.
- We need to eliminate the hidden parts of the picture.

The Painter's Algorithm

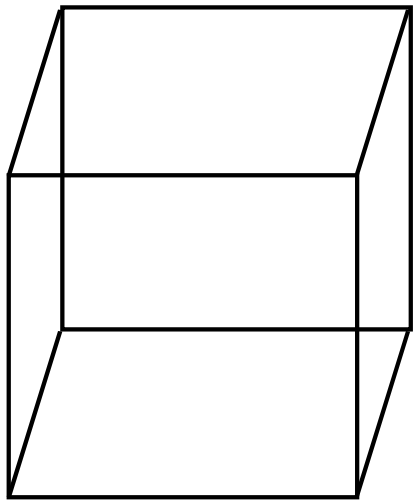
- Suppose we take each face of our wire frame and draw it in two dimensions as a filled polygon.
- By filling it we hide anything behind it.
- So, if we draw the furthest parts first then the hidden parts of the scene are automatically eliminated

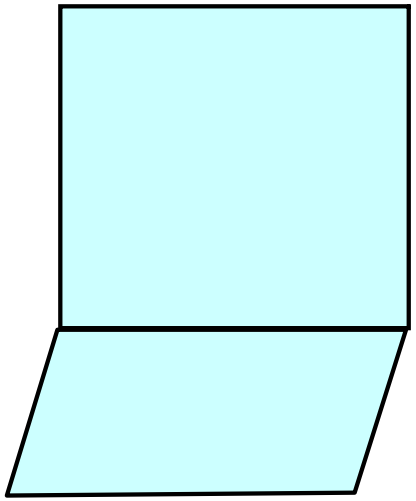
The Painter's Algorithm



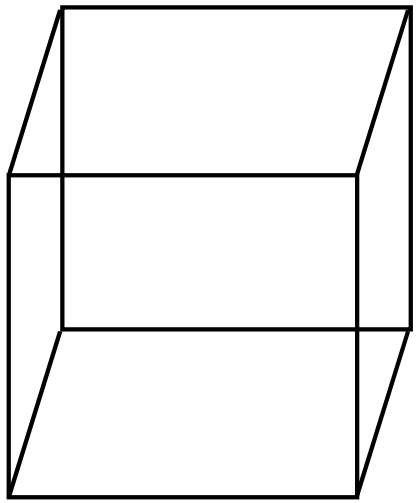


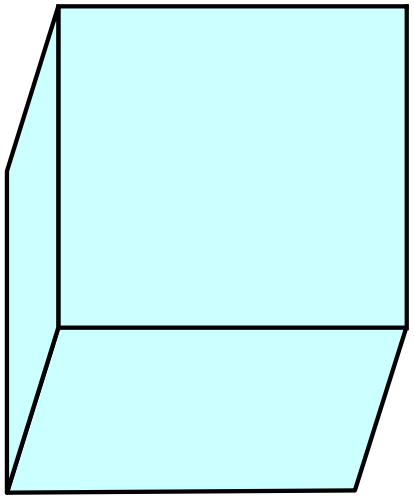
The Painter's Algorithm



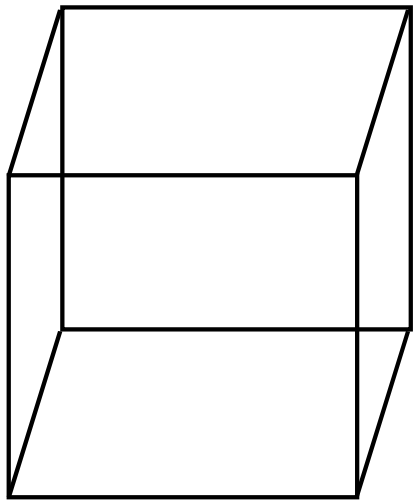


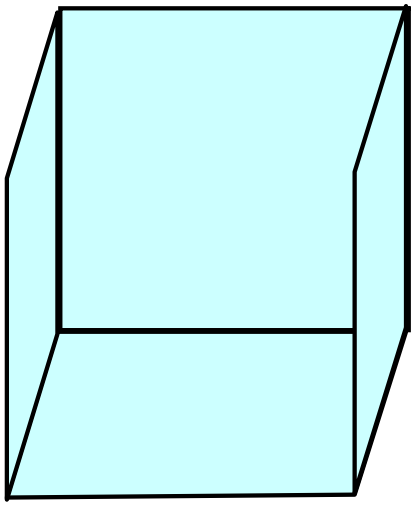
The Painter's Algorithm



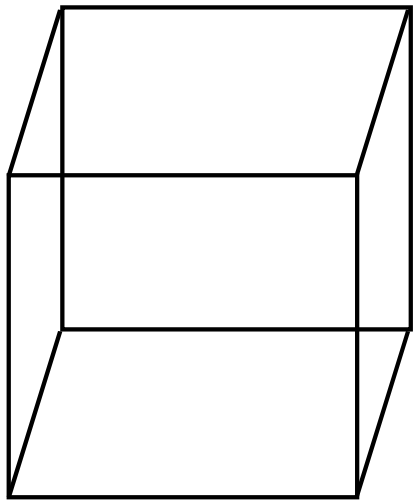


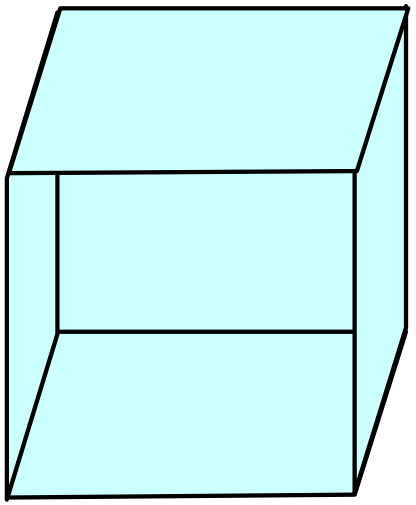
The Painter's Algorithm



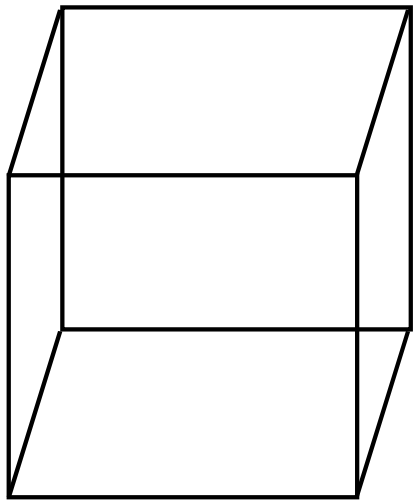


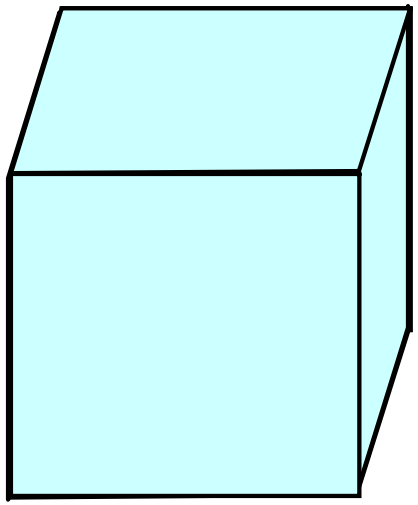
The Painter's Algorithm



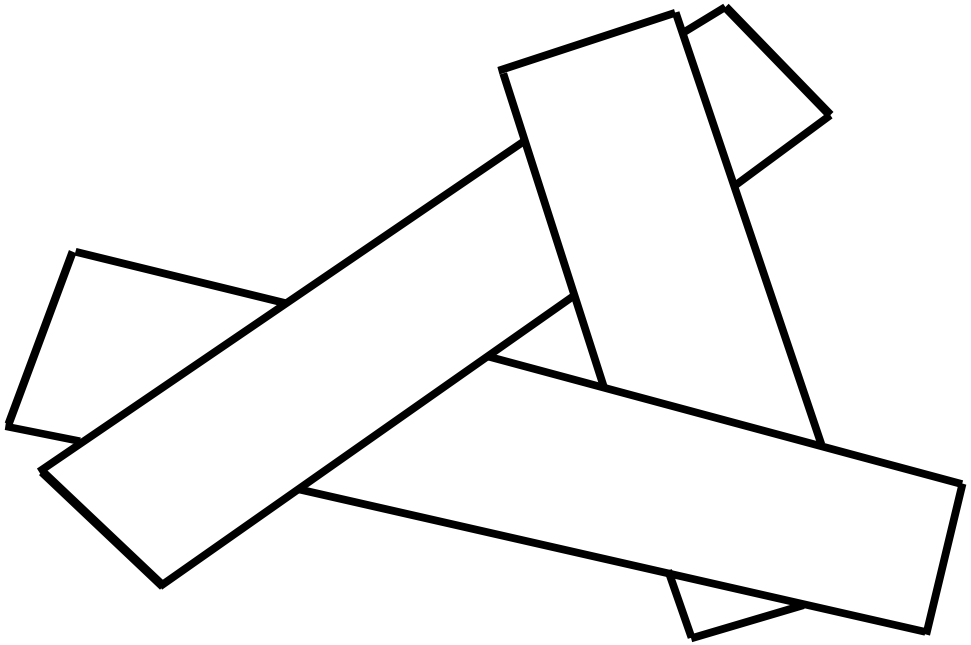


The Painter's Algorithm





The Painter's algorithm doesn't always work!



The Z-Buffer

- The painter's algorithm requires the objects to be sorted into depth order.
- A Z-Buffer algorithm is similar, but avoids sorting.
- Every time we set a pixel in our image we record in a separate array (the z-buffer) the 3D z co-ordinate at that pixel.

The Z-Buffer again

- When we render a new polygon we check at each pixel to see whether it is nearer or further than the object currently drawn at that pixel.
- We only set the pixel if the polygon is closer.
- A further advantage of the z-buffer is that it can be simply implemented in hardware (e.g. graphics card)

Line Drawings

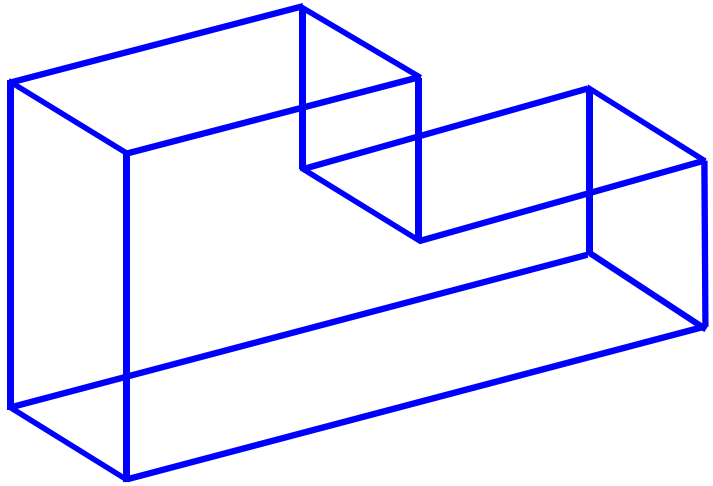
Faster methods than the painter's algorithm (or Z buffer) can be used when making line drawings.

For example:

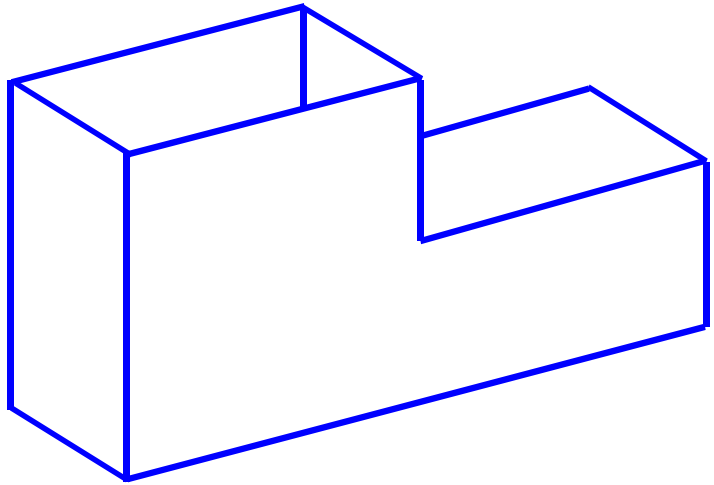
1. Architectural Visualisation

2. Engineering Drawings

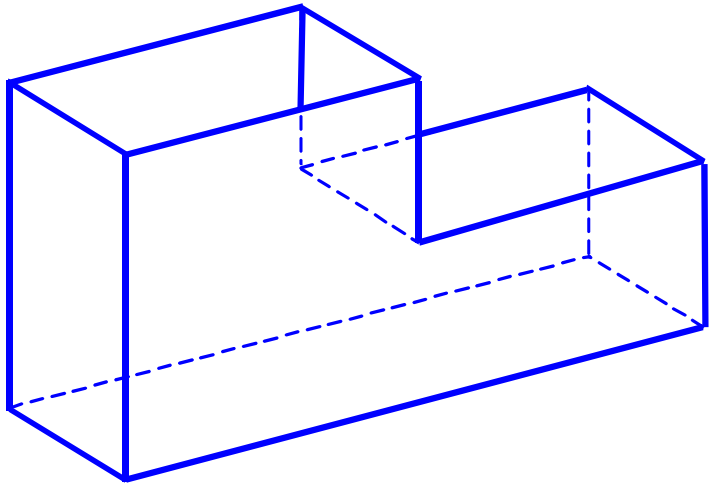
Wireframe representation



Wireframe with hidden lines removed



Wireframe engineering drawing representation



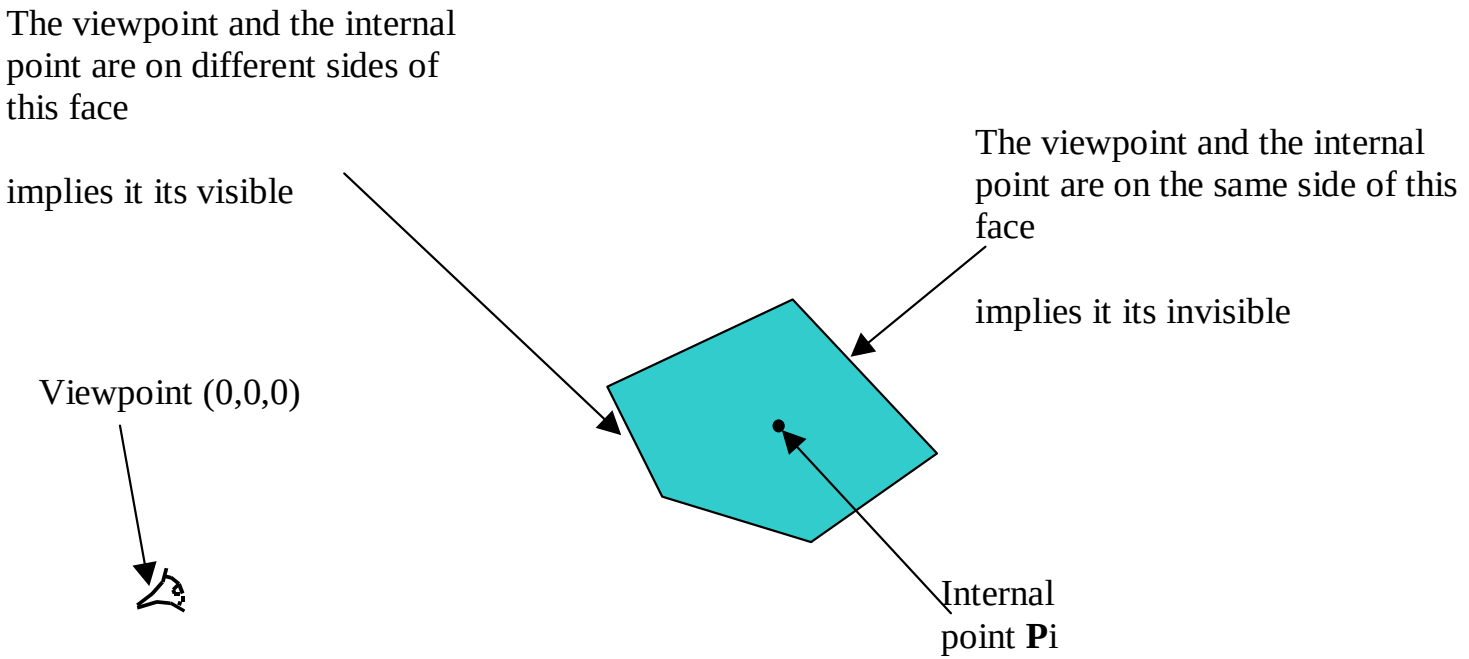
Convex Objects

The following rule applies to single convex objects

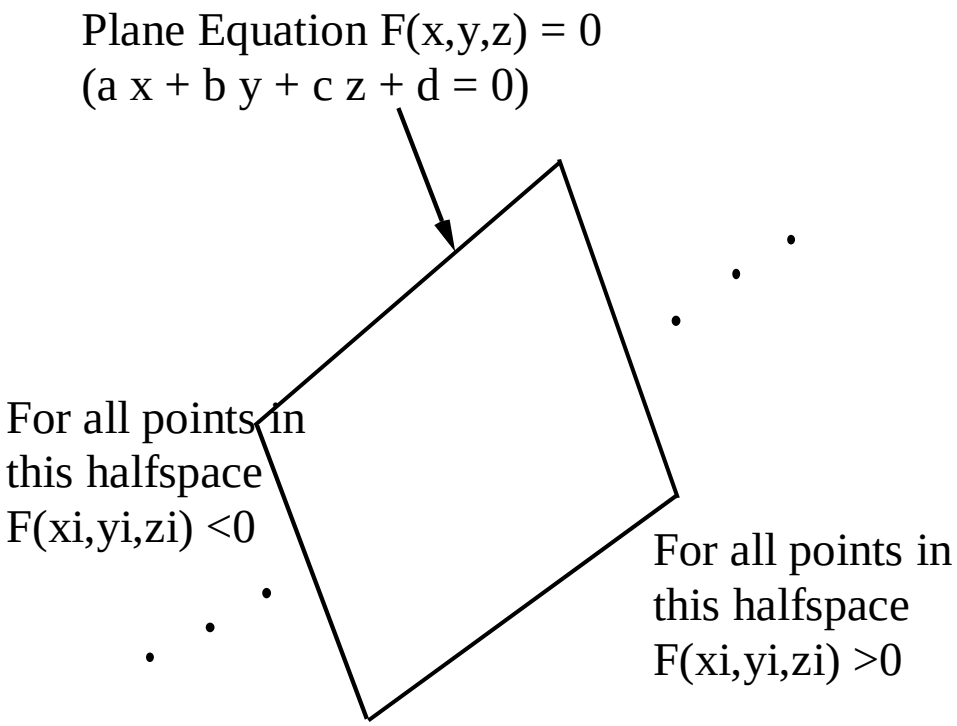
If any part of a face is invisible the whole face is invisible

Thus a hidden line algorithm could also be considered a visible surface algorithm.

The principle is illustrated in 2D



We test in 3D using the halfspace idea



Using the halfspace property

<* find an internal point xi,yi,zi by averaging the vertices *>

for <*each face of the object *>

<* find the plane equation ax+by+cz+d=0, ie f(x,y,z)=0 *>

<* for the viewpoint find *sign(f(0,0,0))* *>

<* for a point inside find *sign(f(xi,yi,zi))* *>

if *sign(f(0,0,0))* not equal to *sign(f(xi,yi,zi))*

then <*draw all the edges of the face *>

end if

end for

Finding the equation of an object face

Take three vertices $\mathbf{V_1}, \mathbf{V_2}, \mathbf{V_3}$, which are not co-linear

construct two vectors on the plane:

$$\mathbf{p_1} = \mathbf{V_1} - \mathbf{V_2}$$
$$\mathbf{p_2} = \mathbf{V_1} - \mathbf{V_3}$$

The normal vector

The normal vector to the plane is given by

$$\mathbf{n} = \mathbf{p_1} \times \mathbf{p_2} = \begin{aligned} &(v_1y*v_2z - v_1z*v_2y) - \\ &(v_1x*v_2z - v_2x*v_1z) + \\ &(v_1x*v_2y - v_2x*v_1y) \end{aligned}$$

Now, the normal vector is also the coefficients of the plane equation:

$$\mathbf{n} = \{a,b,c\}$$

Finally the plane equation

So to find the equation we just need to find d by using any point on the plane, say $\mathbf{V_1} = (V_{1x}, V_{1y}, V_{1z})$

$$a*V_{1x} + b*V_{1y} + c*V_{1z} + d = 0$$
$$d = -(a*V_{1x} + b*V_{1y} + c*V_{1z})$$

Hence we have found the plane equation:

$$a*x + b*y + c*z + d = 0$$

Example 1

A convex object has a face with vertices:

(1,0,0) (0,1,0) and (0,0,1)

a. What is the Cartesian equation of the plane in which the face lies?

b. If an internal point of the object is (3,1,3) determine if the face is visible with the viewpoint at the origin.

Solution

a. two vectors on the plane are

(1,0,0) - (0,1,0) = (1,-1,0)

(1,0,0) - (0,0,1) = (1,0,-1)

the normal vector is found from their cross product

$$\begin{vmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} \\ 1 & -1 & 0 \\ 1 & 0 & -1 \end{vmatrix} = \quad (1,1,1)$$

Solution (continued)

Thus the equation is:

$$x + y + z + d = 0$$

substitute any vertex, say (1,0,0) gives d = -1

$$f(x,y,z) = x + y + z - 1 = 0$$

b. at the viewpoint: Sign(f(0,0,0)) = negative

at the internal point: Sign(f(3,1,3)) = positive

Hence the face is between the origin and a internal point and is therefore visible.

A general purpose hidden line algorithm:

(Multiple object, concave or convex)

- We need to test every line against every face to see whether the face obscures any part of the line.
- For simplicity we will consider first the case where the object facets are convex polygons.

(NB The object can still be concave)

Line face intersections

- For a given line and a given face we need to compute all line face intersections.

- Since the face is convex there are three results:

  No intersections,

  One intersection.

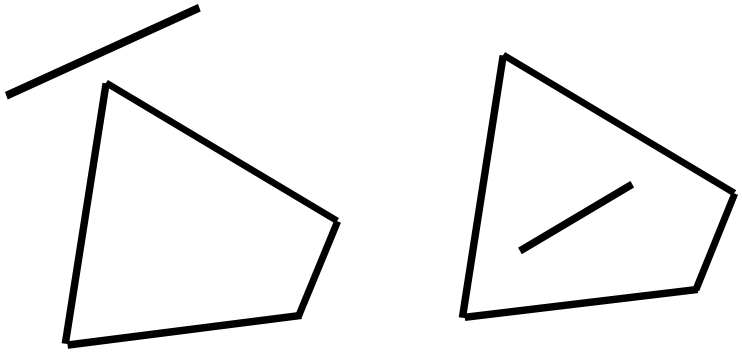
  Two intersections.

Outline Algorithm

```
for <*each line of the scene*>
    <*set up a list of line segments *>
    for <*each face of each object*>
    for <*each line segment on the list*>
        <* find all intersections between the line and the face*>
        <*process the intersections to remove hidden parts *>
    end for
    end for
    <* draw all line segments that remain on the list (if any) *>
end for
```

No intersections

```
if <* no intersections *>
then if <*both points inside the polygon *>
        then    if  <* the face obscures the line segment *>
                then <* delete the segment from the list *>
                end if
        end if
end if
```



One intersection

if <*one intersection*>

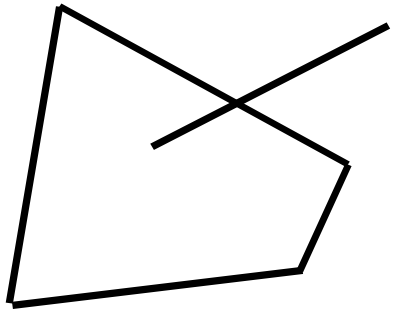
then <* find which part of the line is inside*>

if <*the face obscures the contained segment*>

then <* replace the line segment being tested with the visible part *>

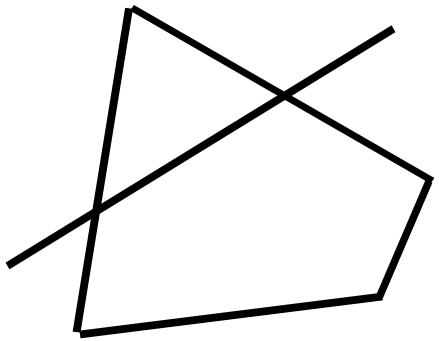
end if

end if



Two intersections

if <*two intersections*>
thenif <* the face obscures the inner part of the line*>
then <* replace the line segment with the outer two segments *>
end if
end if



Intersection between a line and a face edge

line segment: $\mathbf{p} = \mu\, \mathbf{p_2} + (1-\mu)\, \mathbf{p_1}$

face edge: $\mathbf{p} = \nu\, \mathbf{p_4} + (1-\nu)\, \mathbf{p_3}$

at intersection: $\mu\, \mathbf{p_2} + (1-\mu)\, \mathbf{p_1} = \nu\, \mathbf{p_4} + (1-\nu)\, \mathbf{p_3}$

reduce this to Cartesian form:

$$\mu\, p_{2x} + (1-\mu)\, p_{1x} = \nu\, p_{4x} + (1-\nu)\, p_{3x}$$
$$\mu\, p_{2y} + (1-\mu)\, p_{1y} = \nu\, p_{4y} + (1-\nu)\, p_{3y}$$

Intersection between a line and a face edge

re-arrange and eliminate μ

$$(p_{2x}-p_{1x})*(\nu\,(p_{4y}-p_{3y}) + p_{3y} - p_{1y}) =$$
$$(p_{2y}-p_{1y}) * (\nu\,(p_{4x}-p_{3x}) + p_{3x} - p_{1x})$$

Solve for ν

Substitute back to find μ

There is a valid intersection if:

$0<\mu<1$

$0<\nu<=1$

Back Projection

- If a part of a line is covered by a face in the 2D image plane, it is necessary to determine whether it is in front of or behind the face in 3D.

- This is done by finding the midpoint of the line segment in 3D and *back projecting* that point.

Back Projection – How to

Let the midpoint of the line (on the screen) be **m** (this is a 3D coordinate (x,y,f))

The projector equation, for perspective projection is given by

$$\mathbf{P} = \nu\mathbf{m}$$

Let the line in 3D space (ie the object edge) be:

$$\mathbf{P} = \mu\,\mathbf{P}_2 + (1-\mu)\,\mathbf{P}_1$$

Back Projection – How to

The required point in 3D space is given by the intersection of the projector and the edge, ie where:

$$\nu\mathbf{m} = \mu\,\mathbf{p}_2 + (1-\mu)\,\mathbf{p}_1$$

We solve for ν and hence deduce the corresponding 3D point

And Finally

We know the coordinate of a point on the line

We know the plane equation of the face

To find out whether the point is visible or not we simply do a test to see if the origin is in the same halfspace as the edge.

Another Example

Two points of a 3D scene are (10,10,5) and (40,20,20)

The scene is viewed from the origin with plane of projection z=2

What 3D point corresponds to the mid point of the projected line?

Solution

First we find the 2D projections of the points:

(10,10,5) projects to (4,4)
(40,20,20) projects to (4,2)

so the mid point of the projected line segment is:

(4,3)

Solution (continued)

Now we find the corresponding 3D point by back projecting:

The projector in 3D goes through the mid point and the origin, it is:

$$P = \nu(4,3,2)$$

The line in 3D space is given by:

$$P = \mu\,(10,10,5) + (1-\mu)(40,20,20)$$

$$P = \mu\,(-30,-10,-15) + (40,20,20)$$

Solution (concluded)

To find the intersection we equate the two expressions for P:

$\nu\ (4,3,2) = \mu\ (-30,-10,-15) + (40,20,20)$

thus

$4\ \nu + 30\ \mu = 40$

$3\ \nu + 10\ \mu = 20$

$\nu = 4,\ \mu = 4/5$

and the intersection is (16,12,8)