Computer Graphics

Lecture 5:

Towards Solid objects:

Hidden Surfaces and Texture mapping

Hidden Lines

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

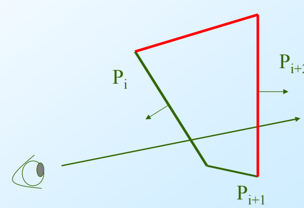
Types of Visibility Methods

- Back-face culling
- Ordering in object space
  - Depth sort
  - Painter's algorithm
- Ordering in projection space
- Ordering in image space
  - Z-buffer
  - Ray casting

Back Face Culling

Polygons should be ordered in a consistent manner (clockwise or counter clockwise)

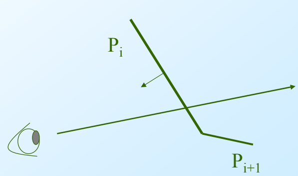
Polygons whose normal does not face the viewpoint, are not rendered



Back Face Culling

Polygons should be ordered in a consistent manner (clockwise or counter clockwise)

Polygons whose normal does not face the viewpoint, are not rendered



Back Face Culling

Polygons whose normal is pointing away from the viewer can be culled. If α is the angle between the normal and the viewer the polygon is front facing if

$$-90 < \alpha < 90 \text{ or } \cos \alpha > 90$$

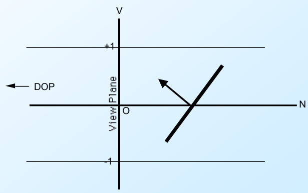
Alternatively we can test for

$$\mathbf{n} \cdot \mathbf{v} > 0$$

Back Face Culling

Alternatively, think about back projection space

The transformed polygon must have a normal with a negative z component to be visible



Types of Visibility Methods

- Back-face culling
- Ordering in object space
  - Depth sort
  - Painter's algorithm
- Ordering in projection space
- Ordering in image space
  - Z-buffer
  - Ray casting

The Painter's Algorithm

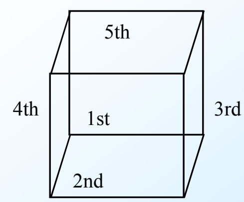
Suppose we take each face of our wire frame and draw it in two dimensions as a filled polygon.

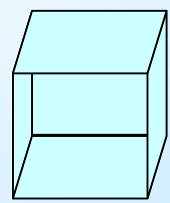
By filling it we hide anything behind it.

So, if we draw the furthest parts first then the hidden parts of the scene are automatically eliminated

The Painter's Algorithm

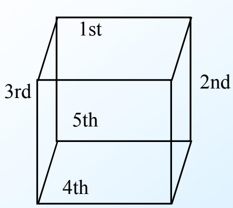
Polygons are rendered in depth order

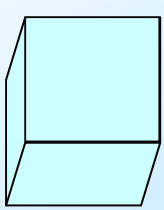




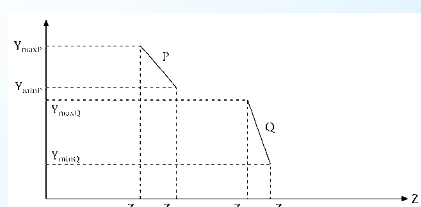
The Painter's Algorithm

Changing the order changes the result





Depth-sort



Note that two polygons P and Q can be rendered in any order, if they do not overlap in X-Y

P can be rendered before Q if its P's z range is completely behind Q's

Depth-sort

P can be rendered before Q if

- Z-extent of Q is wholly in front of P ***or***
- Y-extent of Q does not overlap P ***or***
- X-extent of Q does not overlap P ***or***
- All points on P lie on the opposite side of Q than the centre of projection (COP) ***or***
- All points on Q lie on the same side of P as the COP ***or***
- The projections of P and Q on the XY plane do not overlap

Types of Visibility Methods

- Back-face culling
- Ordering in object space
  - Depth sort
  - Painter’s algorithm
- Ordering in projection space
- Ordering in image space
  - Z-buffer
  - Ray casting

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.

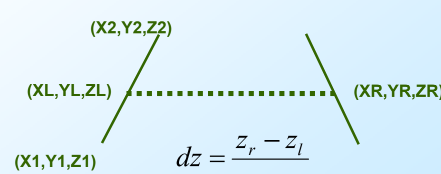
A further advantage of the z-buffer is that it can be simply implemented in hardware

The Z-Buffer

- Most common usage is a a full window sized array of size m x n of 16, 24 or 32 bit “depth” values.
- Basic idea:
  - Initialise buffer to Z_MAX
  - For each pixel in each polygon
    - If z < ZBUF[x,y] set CBUF[x,y] = col
- Now we have to write a z-value for each point
  - directly from plane equation (re-calculate for each point)
  - incremental across the scan-line (store z_start and dz)
  - interpolate …

Interpolating Depth

Interpolate z along edges AND interpolate between edges on each scan-line (bi-linear interpolation)



$$dz = \frac{z_r - z_l}{x_r - x_l}$$

Texture

The visual appearance of a graphics scene can be greatly enhanced by the use of texture.

Consider a brick building, using a polygon for every brick require a huge effort in scene design.

So why not use one polygon and draw a repeating brick pattern onto it?

Texture Mapping

Texture Mapping

Texture Mapping

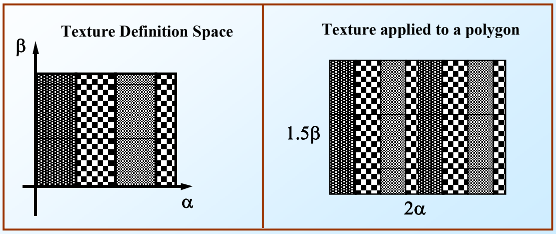
Texture Definition

Textures may be defined as:

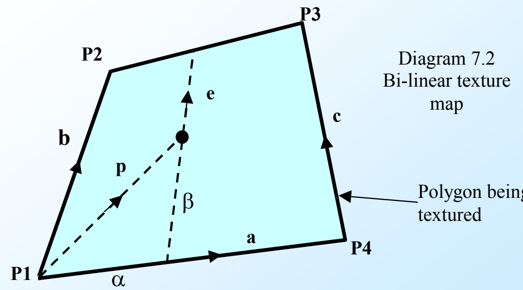
Bitmaps - Arrays containing the actual pixel values to be mapped to the polygon. The data can be derived from photographs for example.

Procedures - Suitable for repeating patterns.

Textures need not map exactly to polygons. They can be smaller or larger than the polygon



Mapping texture to individual pixels



Bi-linear Map - Solving for α and β

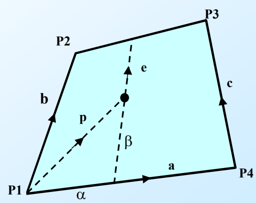
$\mathbf{p} = \alpha\mathbf{a} + \beta\mathbf{e}$

$\mathbf{e} = \mathbf{b} + \alpha(\mathbf{c} - \mathbf{b})$

so

$\mathbf{p} = \alpha\mathbf{a} + \beta\mathbf{b} + \alpha\beta(\mathbf{c}\text{-}\mathbf{b})$

Its Quadratic !



Non Linearities in texture mapping

The second order term means that straight lines in the texture may become curved when the texture is mapped.

However, if the mapping is to a parallelogram:

$\mathbf{p} = \alpha\mathbf{a} + \beta\mathbf{b} + \alpha\beta(\mathbf{c}\text{-}\mathbf{b})$

and

b=c

so $\mathbf{p} = \alpha\mathbf{a} + \beta\mathbf{b}$

Photographs as Textures

Photographs can be used to enhance reality with virtually no design effort.

For a flight simulator landing at an airport the distant landscape can be presented as a photograph which forms the back clipping plane

Alias Effects

One major problem with texture mapping is called alias effects.

These are caused by undersampling, and can cause unreal visual artefacts.

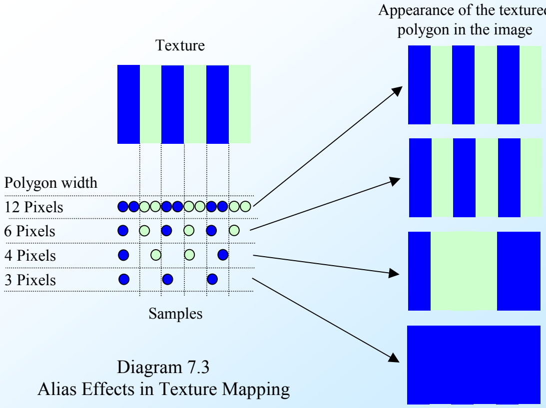


Diagram 7.3
Alias Effects in Texture Mapping

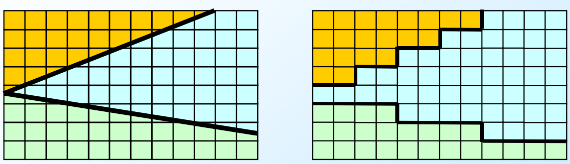
Alias effects in static images

Aliases appear in untextured static images as well.

They have the characteristic of making straight line boundaries look jagged.

This is also due to undersampling.

Alias Effects at straight boundaries in raster images.



Desired Boundaries

Pixels Set

Anti-Aliasing

The solution to aliasing problems is to apply a degree of blurring to the boundary such that the effect is reduced.

The most successful technique is called

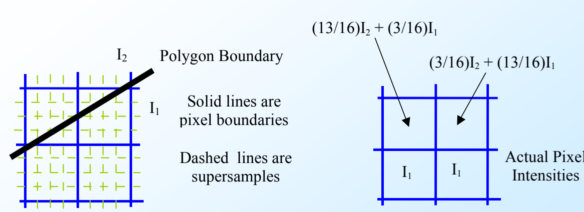
Supersampling

Supersampling

The basic idea is to compute the picture at a higher resolution to that of the display area.

Then the supersamples are averaged to find the pixel value.

This has the effect of blurring boundaries, but leaving coherent areas of colour unchanged

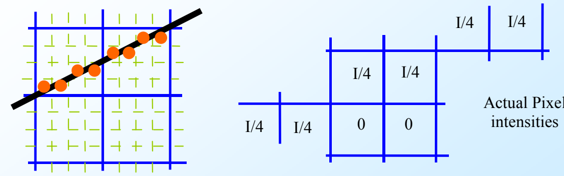


Limitations of Supersampling

Supersampling works well for scenes made up of filled polygons.

However, it does require a lot of extra computation.

It does not work for line drawings.

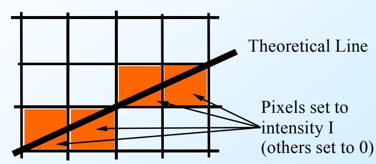


Convolution filtering

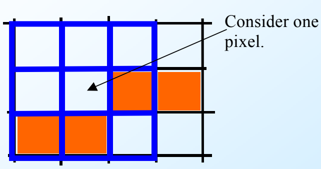
The more common (and much faster) way of dealing with alias effects is to use a ‘filter’ to blur the image.

This essentially takes an average over a small region around each pixel

For example consider the image of a line



Treat each pixel of the image



We replace the pixel by a local average, one possibility would be 3*I/9

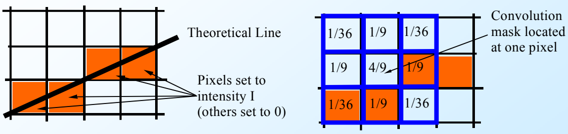
Weighted averages

Taking a straight local average has undesirable effects.

Thus we normally use a weighted average.

1/36 *

1	4	1
4	16	4
1	4	1



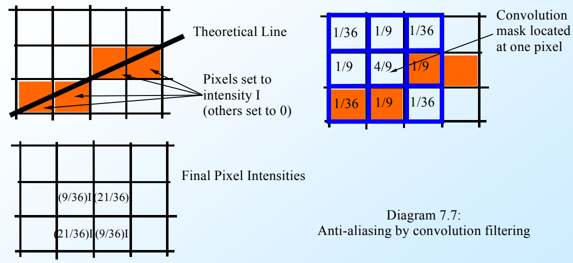


Diagram 7.7:
Anti-aliasing by convolution filtering

Advantages of Convolution filtering

It is very fast and can be done in hardware,

It is of general application

however

It does degrade the image while enhancing its visual appearance.

Anti-Aliasing textures

This is essentially the same technique.

When we identify a point in the texture map we return an average of texture map around the point.

Scaling needs to be applied so that the less the samples taken the bigger the local area where averaging is done.