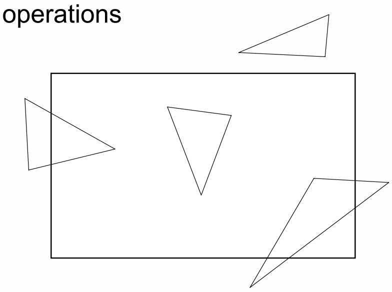
Clipping

Removing what is not seen on the screen

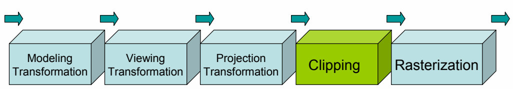
Examples

- Types of operations
  - Accept
  - Reject
  - Clip



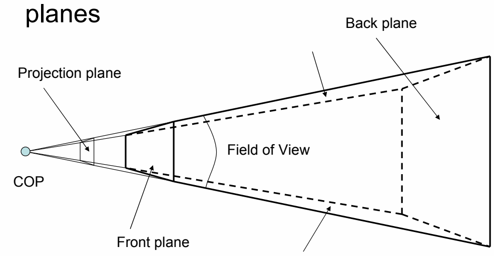
The Rendering Pipeline

- The Graphics pipeline includes one stage for clipping
- Clipping is most often performed in Normalized device coordinates



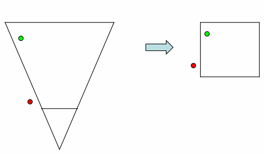
View Frustum

- The view frustum is divided by six clipping planes



Normalization

- Clipping in a cube is easier!
- Observe the x-coordinate of the points!

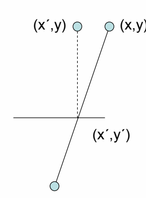


Range

$$-w \le x \le w$$
$$-w \le y \le w$$
$$-w \le z \le w$$

How it is done

- Use perspective division
- We can now use parallel projection
- Scale the coordinates in the range



Clipping in 2D

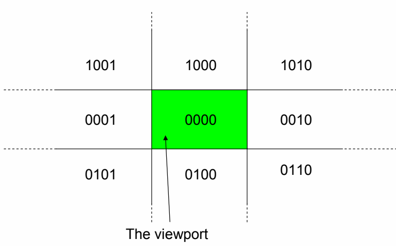
- Clipping can also be performed in 2D
- But it is usually less effective
  - Discard all polygons behind the camera
  - Project on the clipping plane
  - Clip polygons in 2D (on the projection plane)
- We can also clip in the Frame buffer (scissoring)
- Most approaches for 2D clipping can be extended to 3D

Algorithms

- Some well known clipping algorithms
  - Cohen-Sutherland
  - Liang-Barsky
  - Sutherland-Hodgeman
  - Weiler-Atherton
  - Cyrus-Beck
- We will look at the 2D version for Line clipping and discuss extensions to polygon clipping in 3D

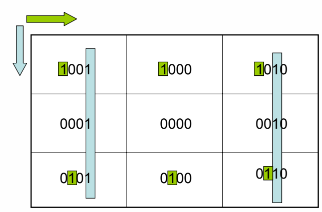
Cohen-Sutherland

- Divide space in 9 regions
- And assign codes to them



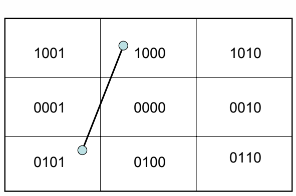
Pattern

- Each side corresponds to one bit in the codes (outcode)



Example

- The endpoints are assigned an outcode
  - 1000 and 0101 in this case



Assignment

- The outcode $o_1 = outcode(x_1, y_1) = (b_0\, b_1\, b_2\, b_3)$ is easily assigned:

$$b_0 = \begin{cases} 1 & \text{if } y > y_{\max}, \\ 0 & \text{otherwise.} \end{cases}$$

$$b_1 = \begin{cases} 1 & \text{if } y < y_{\min}, \\ 0 & \text{otherwise.} \end{cases}$$

$$b_2 = \begin{cases} 1 & \text{if } x > x_{\max}, \\ 0 & \text{otherwise.} \end{cases}$$

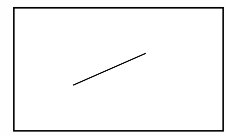
$$b_3 = \begin{cases} 1 & \text{if } x < x_{\min}, \\ 0 & \text{otherwise.} \end{cases}$$

Decision based on the outcode

- *$o_1=o_2=0$*
  Both endpoints are inside the clipping window

Decision based on the outcode

- *$o_1=o_2=0$*
  Both endpoints are inside the clipping window

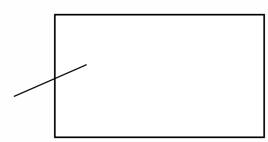


Decision based on the outcode

- *$o_1!=0$, $o_2=0$; or vice versa*
  One endpoint is inside and the other is outside
  - The line segment must be shortened

Decision based on the outcode

- *$o_1!=0$, $o_2=0$; or vice versa*
  One endpoint is inside and the other is outside
  - The line segment must be shortened



Decision based on the outcode

- *$o_1\&o_2!=0$*
  Both endpoints are on the same side of the clipping window
  - Trivial Reject

Decision based on the outcode

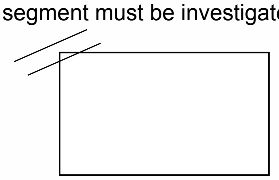
- *$o_1\&o_2!=0$*
  Both endpoints are on the same side of the clipping window
  - Trivial Reject

Decision based on the outcode

- $o_1 \& o_2 = 0$
  Both endpoint are outside but outside different edges
  - The line segment must be investigated further

Decision based on the outcode

- $o_1 \& o_2 = 0$
  Both endpoint are outside but outside different edges
  - The line segment must be investigated further



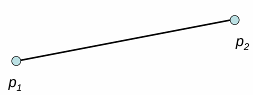
Parametric Lines

- Intersections with can easily be computed by regarding the line as a parametric line

$$p(\alpha) = (1-\alpha)\,p_1 + \alpha p_2$$

- This is a linear interpolation with

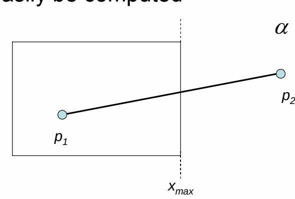
$$0 \geq \alpha \geq 1$$



Intersection computation

- Example: Intersection with right border x_{max}
- The parameter can easily be computed

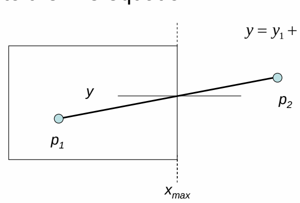
$$x_{\max} = (1-\alpha)x_1 + \alpha x_2$$
$$x_{\max} = x_1 + \alpha(x_2 - x_1)$$
$$x_{\max} - x_1 = \alpha(x_2 - x_1)$$
$$\alpha = \frac{x_{\max} - x_1}{x_2 - x_1}$$



Intersection computation

- Finally we compute the *y-coordinate* by putting the parameter into the line equation

$$y = y_1 + \alpha(y_2 - y_1), \quad \alpha = \frac{x_{\max} - x_1}{x_2 - x_1}$$
$$y = y_1 + \frac{x_{\max} - x_1}{x_2 - x_1}(y_2 - y_1)$$
$$y = y_1 + \frac{y_2 - y_1}{x_2 - x_1}(x_{\max} - x_1)$$



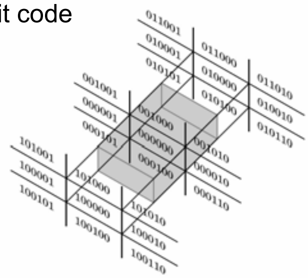
The *two point formula!*

Intersections

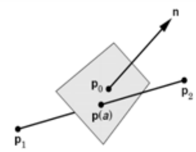
- We can therefore compute intersections with the border using the *two point formula*
- We will obtain similar equations for the other borders
- What happens if we have no intersections with the view port?
  - The parameter is out of range!

3D

- A little bit more complicated…. 27 regions with a 6 bit code



Intersections in 3D



If we write the line and plane equations in matrix form (where n is normal to the plane and p_0 is a point on the plane), we must solve the equations

$$p(\alpha) = (1 - \alpha)p_1 + \alpha p_2$$

$$n \cdot (p(\alpha) - p_0) = 0$$

Intersections in 3D

- The first equation into the second equation

$$n \cdot ((1 - \alpha)p_1 + \alpha p_2 - p_0) = 0 \Rightarrow$$

$$n \cdot (p_1 - \alpha p_1 + \alpha p_2 - p_0) = 0 \Rightarrow$$

$$n \cdot (p_1 - p_0 + \alpha(p_2 - p_1)) = 0 \Rightarrow$$

$$n \cdot (p_1 - p_0) + n \cdot (\alpha(p_2 - p_1)) = 0 \Rightarrow$$

$$\alpha(n \cdot (p_2 - p_1)) = n \cdot (p_0 - p_1) \Rightarrow$$

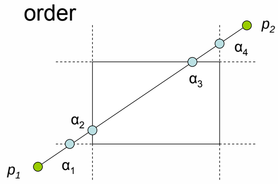
$$\alpha = \frac{n \cdot (p_0 - p_1)}{n \cdot (p_2 - p_1)}.$$

A hybrid approach

- Use 3D Cohen Sutherland for trivial Reject and trivial Accept
- Then project onto viewport
- And finally do final clipping in 2D
  - Trivial cases need not to be handled!
- Or perhaps even use scissoring instead?

Liang Barsky

- Uses the parametric line!
- Compute α for each border in a clockwise order

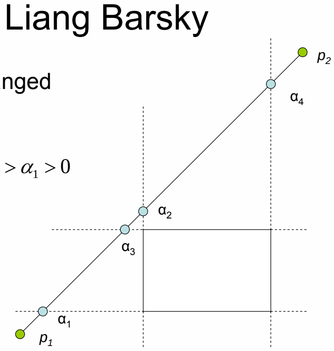


$$1 > \alpha_4 > \alpha_3 > \alpha_2 > \alpha_1 > 0$$

Liang Barsky

- Note the changed order!

$$1 > \alpha_4 > \alpha_2 > \alpha_3 > \alpha_1 > 0$$



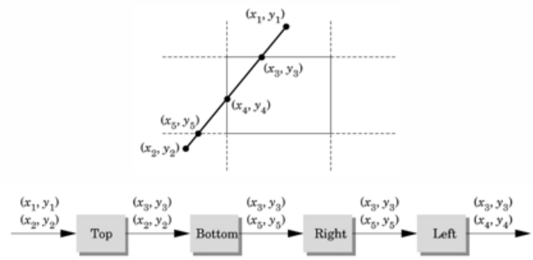
Liang Barsky

- Similar equations can be derived for all possible cases
- Clip using the computed α's
- 3D: just add one dimension in the parametric line

$$z(\alpha) = (1-\alpha)z_1 + \alpha z_2$$

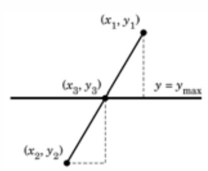
Sutherland-Hodgeman

- Is a 'pipeline' clipper…



Computing intersections

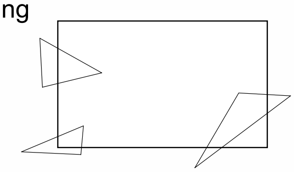
- Use the two-point formula for intersection computations



$$y_3 - y_1 = \frac{y_2 - y_1}{x_2 - x_1}(x_3 - x_1),$$
$$y_3 = y_{max} \Rightarrow$$
$$x_3 = (y_{max} - y_1)\frac{x_2 - x_1}{y_2 - y_1} + x_1$$

Polygon clipping

- The previous explained approaches can be used for clipping polygons with some modifications
- Note that a triangle can have more vertices after clipping



Acceleration techniques

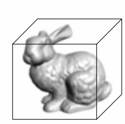
- Imagine an object with many polygons

The Stanford Bunny

- It is not so funny to check thousands of polygons for intersections! (69451 to be exact)
- We need some acceleration technique

Bounding Volumes

- Create the smallest box that contains the bunny

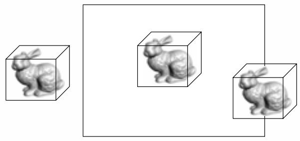


A boxed Bunny

- Check eight sides for intersections instead!

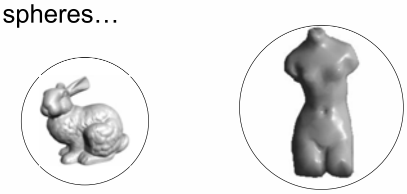
Intersection test

- Trivial cases are easy!
- Non trivial cases still need extensive computations
  - Unless you use a hierarchy of cubes



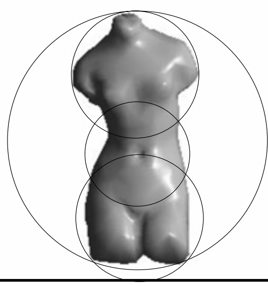
Bounding Spheres

- Only one center and a radius have to be checked!
- But not all objects are suitable for spheres…



Bounding Spheres

- Make a hierarchy of spheres for elongated objects!



Some final words…

- Not only 3D Objects need to be clipped
  - Also splines, letters etc…
- A mirror or a portal in a game can have non rectangular shape
  - Clipping is needed
- Even though clipping is implemented in hardware, it is essential to understand the basics of it!