Lecture 8: Clipping

- Removing what is not seen on the screen
The Rendering Pipeline

The Graphics pipeline includes one stage for clipping
The view frustum is defined by six clipping planes (Frustum = clipped pyramid)
The view frustum is defined by six clipping planes

(Frustum = clipped pyramid)
The view frustum is defined by six clipping planes

(Frustum = clipped pyramid)

Object completely outside frustum. Should not be rendered!
The view frustum is defined by six clipping planes

(Frustum = clipped pyramid)

Object completely outside frustum. Should not be rendered!
The view frustum is defined by six clipping planes

(Frustum = clipped pyramid)

Part of object outside the view frustum -> clipping is needed!
The clipped 3D object seen through the viewport.
Remember that the objects are represented by a set of polygons…
Examples

Types of operations

– Accept
– Reject
– Clip

Polygon

Viewport
Normalization

We map the view frustum into a cube (Normalized Device Coordinates)
Clipping in a cube is easier!
Observe the x-coordinate of the points!

Viewport

COP

Range
\[-w/2 \leq x \leq w/2\]
\[-w/2 \leq y \leq w/2\]
\[-w/2 \leq z \leq w/2\]

\(w = \text{side of the cube}\)
How it is done

Transfer all vertices into normalized device coordinates by perspective division
(Remember viewing lecture)
Scale the coordinates in the range

Viewport

COP
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 viewport plane
– Clip polygons in 2D (on the projection plane)

Most approaches for 2D clipping can be extended to 3D
The Rendering Pipeline

Modeling Transformation → Viewing Transformation → Clipping → Projection Transformation → Rasterization

3D Clipping

Modeling Transformation → Viewing Transformation → Projection Transformation → Clipping → Rasterization

2D Clipping

Modeling Transformation → Viewing Transformation → Projection Transformation → Rasterization → Clipping

Scissoring
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 (4 bits)
Each side corresponds to one bit in the codes (outcode)
Example

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

<table>
<thead>
<tr>
<th></th>
<th>1001</th>
<th>1000</th>
<th>1010</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0101</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0110</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The outcode $o_1 = \text{outcode}(x_1, y_1) = (b_0 b_1 b_2 b_3)$ is easily assigned:

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

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

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

\[
b_3 = \begin{cases} 1 & \text{if } x < x_{\text{min}}, \\ 0 & \text{otherwise}. \end{cases}
\]
Decision based on the outcode

$o_1 = o_2 = 0000$

Both endpoints are inside the clipping window (Accept, no clipping needed)
Decision based on the outcode

\[ o_1 \neq 0000, \; o_2 = 0000; \text{ or vice versa} \]

One endpoint is inside and the other is outside

- The line segment must be shortened (clipped)
Decision based on the outcode

\[ o_1 \& o_2 \neq 0000 \ (Bitwise \textbf{and} \ operator) \]

Both endpoints are on the same side of the clipping window

- Trivial Reject
Decision based on the outcode

$O_1 = 1001, O_2 = 0101$ bitwise and operator gives: $1001 \& 0101 = 0001 \neq 0000$, both ends are on the same side of the clipping window $\rightarrow$ reject!
Decision based on the outcode

\[ O_1 = 0101, \ O_2 = 0110 \] bitwise **and** operator gives:
\[ 0101 \& 0110 = 0100 \neq 0000, \text{ both ends are on the same side of the clipping window} \implies \text{ reject!} \]
Decision based on the outcode

$o_1 \& o_2 = 0000 \ (Given \ o_1 \neq 0000 \ and \ o_2 \neq 0000)$

Both endpoint are outside but outside different edges

- The line segment must be investigated further
Decision based on the outcode

<table>
<thead>
<tr>
<th>1001</th>
<th>1000</th>
<th>1010</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>0000</td>
<td>0010</td>
</tr>
<tr>
<td>0101</td>
<td>0100</td>
<td>0110</td>
</tr>
</tbody>
</table>
Decision based on the outcode

$O_1 = 0101, O_2 = 1000$ bitwise \textbf{and} operator gives:

$0101 \& 1000 = 0000$, part of the line could be inside the viewport --- further investigation needed!

<table>
<thead>
<tr>
<th></th>
<th>1001</th>
<th>1000</th>
<th>1010</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>0000</td>
<td>0010</td>
<td></td>
</tr>
<tr>
<td>0101</td>
<td>0100</td>
<td>0110</td>
<td></td>
</tr>
</tbody>
</table>
Intersections with border can easily be computed by regarding the line as a parametric line. This is a linear interpolation with:

\[ p(\alpha) = (1-\alpha)p_1 + \alpha p_2 \]

\[ 0 \geq \alpha \geq 1 \]
Intersection computation

Example: Intersection with right border $x_{\text{max}}$

The parameter $\alpha$ can easily be computed.

\[ x_{\text{max}} = (1 - \alpha)x_1 + \alpha x_2 \]
\[ x_{\text{max}} = x_1 + \alpha(x_2 - x_1) \]
\[ x_{\text{max}} - x_1 = \alpha(x_2 - x_1) \]
\[ \alpha = \frac{x_{\text{max}} - x_1}{x_2 - x_1} \]

\[ p_1 = (x_1, y_1) \]
\[ p_2 = (x_2, y_2) \]
Finally we compute the *y-coordinate* by putting $\alpha$ into the line equation.

$$\alpha = \frac{x_{\text{max}} - x_1}{x_2 - x_1}$$

The two point formula!

$$y = y_1 + \alpha(y_2 - y_1)$$

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

$$y = y_1 + \frac{y_2 - y_1}{x_2 - x_1}(x_{\text{max}} - x_1)$$
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 intersection with the border?

- The parameter \( \alpha \) is out of range \([0, 1]\).
- \( \alpha = \infty \) if the line is parallel with the border we check for intersections with.
Parameter $\alpha$ out of $[0, 1]$ range:

$$\alpha = \frac{x_{\text{max}} - x_1}{x_2 - x_1}$$

Viewport

$x_2 > x_1 > x_{\text{max}}$ \quad $\alpha < 0$
Parameter $\alpha$ out of $[0, 1]$ range:

$$\alpha = \frac{x_{\text{max}} - x_1}{x_2 - x_1}$$

$\alpha = \infty$ when $x_2 = x_1$
Cohen-Sutherland in 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

\[
p(\alpha) = (1 - \alpha)p_1 + \alpha p_2
\]

\[
n \cdot (p(\alpha) - p_0) = 0
\]

Demo time!
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!
Liang Barsky

Uses the parametric line!
Compute $\alpha$ for each (extended) border in a clockwise order

$1 > \alpha_4 > \alpha_3 > \alpha_2 > \alpha_1 > 0$
Note the changed order!

$1 > \alpha_4 > \alpha_2 > \alpha_3 > \alpha_1 > 0$
Similar equations can be derived for all possible cases
Clip using the computed $\alpha$’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...

\[(x_1, y_1)\]
\[(x_2, y_2)\]
\[(x_3, y_3)\]
\[(x_4, y_4)\]
\[(x_5, y_5)\]
Sutherland-Hodgeman

Is a 'pipeline' clipper...

Original polygon
Sutherland-Hodgeman

Is a ‘pipeline’ clipper...

Original polygon

Clip top...
Sutherland-Hodgeman

Is a ’pipeline’ clipper...

Original polygon  Clip top…  Clip bottom…
Sutherland-Hodgeman

Is a 'pipeline' clipper...

Original polygon

Clip top...

Clip bottom...

Clip left...
Is a 'pipeline' clipper…

Original polygon

Clip top…

Clip bottom…

Clip left…

Clip right
Sutherland-Hodgeman

Use the two-point formula for intersection computations

\[ y_3 - y_1 = \frac{y_2 - y_1}{x_2 - x_1} (x_3 - x_1), \]

\[ y_3 = y_{\text{max}} \Rightarrow \]

\[ x_3 = (y_{\text{max}} - y_1) \frac{x_2 - x_1}{y_2 - y_1} + x_1 \]

(Top clipping)
Sutherland-Hodgeman

Non-rectangular clipping
Polygon clipping

The previous explained approaches can be used for clipping polygons with some modifications.

Note that a triangle/polygon can have more vertices after clipping!
Convex polygon…
4 vertices.
Polygon clipping

Convex polygon…
6 vertices after clipping.
Polygon clipping

Concave polygon...
4 vertices.
Polygon clipping

Concave polygon…
7 vertices after clipping.

- Vertices inside the clip window are retained.
- Vertices outside are removed.
- New vertices are added at the clip window boundary.
Polygon clipping

Concave polygon example 2…
Polygon clipping

Concave polygons can split into multiple polygons after clipping!
Acceleration techniques

Imagine an object with many polygons

It is not efficient to check thousands of polygons for intersections!

We need some acceleration technique
Bounding Volumes

Create the smallest box that contains the Bunny

Check eight sides for intersection instead!

We can constrain the bounding box to be aligned with the axes (simpler calculations), or we can allow arbitrary orientation of the box. (possible to achieve better fit)

Axis Aligned Bounding Box
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!
gluPerspective or glFrustum easily sets up the clipping frustum in OpenGL.

gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble zNear, GLdouble zFar);

glFrustum(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble zNear, GLdouble zFar);

http://www.codesampler.com/oglsrc/oglsrc_2.htm
glm::perspective is similar:

```cpp
glm::perspective(field_of_view, aspect, near_clip, far_clip);
```
Some final words...

Even though clipping is implemented in graphics API/ hardware, it is essential to understand the basics of it!
Project

- Volume rendering of CT data.
- Global Illumination
- Image processing
- Polygonal techniques
- Physics Simulation
- Your own suggestion

Deadline May 30
Submit to Studentportalen