Outline

1. Clipping triangles
2. Clipping arbitrary polygons
3. Culling
4. Lighting and Shading
5. Triangle strips and fans
Camera transformation
- Move camera viewpoint to origin

Orthographic projection
- Transform view frustum to axis-parallel box

Canonical view volume
- Windowing transformation
If we combine all steps, we get:

\[
M = M_o M_p M_v
\]

for each line segment \((a_i, b_i)\) do

\[
p = M a_i \\
q = M b_i \\
drawline(x_p/h_p, y_p/h_p, x_q/h_q, y_q/h_q)
\]

homogeneous divide
Stages in the graphics pipeline (recap lect. 7)

We distinguish several stages in the graphics pipeline:

- Triangles that lie (partly) outside the view frustum need not be projected, and are clipped.
- The remaining triangles are projected if they are front facing.
- Projected triangles have to be shaded and/or textured.

Graphics, 2nd period 2007/08  Lecture 11: A full graphics pipeline
In general, we cannot expect all triangles to lie within the view frustum. Triangles that lie partly outside the view frustum must be clipped.

First question: where in the graphics pipeline should we do clipping?
Homogeneous coordinates (recap lect. 6)

We have seen homogeneous coordinates before, but so far, the fourth coordinate of a 3D point has always been 1.

In general, however, the homogeneous representation of a point \((x, y, z)\) in 3D is \((hx, hy, hz, h)\). Choosing \(h = 1\) has just been a convenience.

So we have:

\[
M_p \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \frac{n+f}{n} - f \\ 1 \end{pmatrix} \quad \text{homogenize} \quad \begin{pmatrix} \frac{nx}{z} \\ \frac{ny}{z} \\ z \frac{n+f}{n} - f \\ 1 \end{pmatrix} \]

Advantage: equations for these hyperplanes are quite simple:

\[-x + l = 0\]
\[x - r = 0\]
\[-y + b = 0\]
\[y - t = 0\]
\[-z + n = 0\]
\[z - f = 0\]

But: there’s a problem at the XY-plane . . .
Clipping after the perspective divide may lead to incorrect results if line segments cross the $XY$-plane, since a division by $z$ is involved:

$$z' = n + f - \frac{fn}{z}$$
Clipping after the perspective divide

\[ z' = n + \frac{fn}{z} \]

Fig. 12.3: \( n \)
It is possible to clip against the six clipping planes right before the perspective divide.

The eight corners of the view frustum are easily found by an inverse transform. (Which one?)

From these, we can derive the plane equations for the view frustum.
Clipping in homogeneous coordinates

Surprisingly, it turns out to be easiest to clip in homogeneous coordinates, which means that we clip in four dimensions against three-dimensional clipping hyperplanes.

The equations for these hyperplanes are quite simple:

\[-x' + lw' = 0\]
\[x - rw = 0\]
\[-y + bw = 0\]
\[y - tw = 0\]
\[-z + nw = 0\]
\[z - fw = 0\]
Clipping against a hyperplane

The implicit equation for a hyperplane through a point \( q \) and with normal \( n \) is
\[
f(p) = n \cdot (p - q) = 0.
\]

This is often written as
\[
f(p) - n \cdot p + D = 0.
\]

Convention: normals of the clipping planes point outward, so if \( f(p) < 0 \) then \( p \) is inside the plane, and if \( f(p) > 0 \) then \( p \) is outside the plane.
If two points $a$ and $b$ are on different sides of a hyperplane, we first determine the **parametric equation** of the line through the points:

$$p = a + t(b - a)$$

Substituting this into the hyperplane equation yields

$$t = \frac{n \cdot a + D}{n \cdot (a - b)}$$
Clipping triangles against a hyperplane is done as follows:

- If two vertices are on the positive side, we get one new triangle.
- If one vertex is on the positive side of the hyperplane, we get two new triangles.
But what do we do if a triangle is intersected by two clipping planes?
We have to deal with such situations one clipping plane at a time.

We first clip the initial triangle against one of the clipping planes...
...and clip the remaining triangle(s) against the other clipping plane.
Some rendering engines deal with **arbitrary polygons** rather than with **triangles**.

Also, in **drawing programs** we often need to clip arbitrary polygons.
The Sutherland-Hodgman clipping algorithm is straightforward:

Clip the polygon subsequently against every clipping hyperplane.
The Sutherland-Hodgman algorithm

Informal description:

- Extend upper side of rectangle across 2D space
- Start at one vertex of the polygon and follow its path
- Create new vertex where path crosses clipping line
- Repeat till we are back at starting vertex
- Create new polygon from sets over vertices on or beneath clipping line
- Repeat for every clipping line

The Sutherland-Hodgman algorithm

However, sometimes the Sutherland-Hodgman algorithm results in degenerate polygons.

The resulting polygon on the right has vertices $p_0, i_0, r_2, i_3, p_4, p_5, i_2, r_2$, and $i_1$, consecutively.
The Weiler-Atherton algorithm
The Weiler-Atherton algorithm

Building the graph:

1. Make a **graph** with three groups of vertices: polygon vertices, clipping region vertices, and intersection vertices.

2. Insert edges by walking along the boundary of the polygon, including the intersection vertices. Distinguish **outgoing** intersections (colored red in the image) and **incoming** intersections (colored pink).

3. Insert edges by walking along the boundary of the clipping region, including the intersection vertices.
The Weiler-Atherton algorithm
The Weiler-Atherton algorithm

Using the graph:

1. Start at an **outgoing intersection vertex**, and walk along the **boundary of the clipping region**, reporting every vertex along the way.

2. Upon encountering an **incoming intersection vertex**, continue along the **boundary of the polygon**.

3. Continue, changing from polygon boundary to clipping region boundary and the other way around at outgoing and incoming intersection vertices, respectively, until we reach the starting vertex.

4. Start at the next unvisited outgoing intersection vertex to output the next clipped polygon, etc, until no unvisited outgoing intersection vertices are left.
When a triangle lies entirely outside the view frustum, it can be **culled**, i.e., removed from the graphics pipeline.

Testing **individual triangles** is expensive, however.

Culling **bounding volumes** of complicated objects (consisting of many triangles) will generally **improve the performance** of the graphics pipeline.

cf. Ray Tracing
When 3D objects are modeled by triangles, the convention is that the triangle normals point to the outside of the object.

This means that we can ignore triangles of which the normals point away from the view point.

This is called backface culling.
The place of **lighting and shading calculations** in the graphics pipeline depends on the shading model.

For **Gouraud shading**, these calculations can be done early in the pipeline, i.e., in **world coordinates**.

However, for more complicated shading models, i.e., **Phong shading**, where normals have to be interpolated, shading is best deferred until the rasterization stage. This is the trend in modern graphics hardware.
Triangle strips and fans

Transferring data from main memory to the graphics hardware is expensive.

Therefore, instead of transferring individual triangles, we would like to exploit the structure of a triangle mesh by sending triangle fans and triangle strips.