CSI 4341, Computer Graphics

Lecture 12: Viewing

Date: 2012-10-09
Author(s): Kathleen Cadenhead, Alex Pitzen
Lecturer: Fredrik Niemelä

This lecture is about clipping and rasterization.

1 Clipping Basics

2D clipping is performed within a 2D clipping window. Similary, 3D clipping is performed within a 3D clipping volume. Clipping is very easy for lines and polygons, but hard for curves and text. To make clipping curves and text easier, we can convert them into lines and polygons.
We let the clipping window be between y_{max} and y_{min}, and x_{max} and x_{min}.

2 Clipping 2D Line Segments

2.1 Brute Force Approach

The brute force method for clipping calculates the intersections of a line with all sides of the clipping window/volume. This is of course very inefficient, because multiple division operations are required.

2.2 Cohen-Sutherland

The Cohen-Sutherland approach starts by eliminating as many cases as possible. Starting with the 4 lines representing the sides of the clipping window, determine which of the following cases applies:

1. Both endpoints of the line segment are between all 4 lines of the clipping window. In this case, simply draw the segment.
2. Both endpoints are outside all lines and on the same side of an outside line (i.e. left of the left line, below the bottom line, etc.). Reject this segment (don't draw it at all).
3. One endpoint is outside and the other is inside. In this case, at least one intersection must be calculated to determine which part of the segment is drawn.
4. Both endpoints are outside all lines. Part of the segment may be inside, so at least one intersection must be calculated.

2.2.1 Outcodes

Outcodes are a 4-bit representation of the position of a point relative to the clipping window. It is written as $b_0b_1b_2b_3$, where:

- b_0 is set if $y > y_{max}$
- b_1 is set if $y < y_{min}$
- b_2 is set if $x > x_{max}$
- b_3 is set if $x < x_{min}$

Computing an outcode requires at most 4 subtractions. They can help us easily compute which of the above cases applies. For a line segment with endpoints A and B and outcodes out(A) and out(B), respectively, we know the following.

- If out(A) = out(B) = 0, case 1.
- If out(A) & out(B) $\neq$ 0, case 2.
- If out(A) = 0 and out(B) $\neq$ 0, case 3.
- If out(A) & out(B) = 0, case 4.

Using outcodes is very efficient when working with large amounts of data, but can be inefficient when the code has to be reexecuted for line segments that must be shortened in more than one step.

2.2.2 3D

To expand Cohen-Sutherland to 3D, simply add another 2 bits to the outcodes, for a total of 6 bits.
Note: The clipping volume doesn't have to be a rectangular prism, but calculating outcodes is much easier if it is.

2.3 Liang-Barsky

Liang-Barsky clipping uses the parametric form of a line. For points p_1 and p_2, the parametric form is

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

If we compute the intersections between $p(\alpha)$ and the 4 lines of the clipping window, we can determine whether to reject or to clip based on the relative ordering of the intersections. Label y_{min} 1, x_{min} 2, y_{max} 3, and x_{max} 4, and their intersections with $p(\alpha)$ as α_1, α_2, etc., then

- If $\alpha_4 > \alpha_3 > \alpha_2 > \alpha_1$, then we need to clip the line.
- If $\alpha_4 > \alpha_2 > \alpha_3 > \alpha_1$, then the line can be rejected (no part of it is inside the clipping window).

Liang-Barsky can accept/reject as easily as Cohen-Sutherland, and is usually faster in 3D.

3 Polygon Clipping

Clipping polygons is more complicated than clipping line segments. It can yield multiple polygons, unless the polygon and viewport are both convex.
To make clipping of concave polygons more simple, we can tessellate it. This means we split it up into triangles, which are easy to clip.

3.1 Pipeline Clipping

With pipeline clipping, the polygon is clipped against each side independently. This uses 4 independent clippers and improves throughput.

3.2 Bounding Boxes

We can create the bouding box of a polygon by calculating its x_{min}, x_{max}, y_{min}, and y_{max}. This makes it easy to see if a polygon can be completely accepted or completely rejected.

4 Overlapping polygons

4.1 Painter's algorithm

1. Do a depth sort on polygons.
2. "Paint" polygons from back to front.

The ordering requires $O(n \log n)$. It's easy when polygons don't overlap in x or y or if they don't overlap in z. It doesn't necessarily work for polygons that aren't completely in front or behind one another. It is also bad for polygons that overlap cyclically or polygons that penetrate others.