Chapter 6

Line Clipping

revised March 20, 2013
... program code revisions & assignment to follow ...

6.1 Writing Visible Lines

The Cohen-Sutherland clipping algorithm takes among its inputs the left, right, top and bottom limits of a world coordinate window; let these be denoted W_L, W_R, W_T and W_B, respectively. These values may be positive or negative, astronomically large or microscopically small. The clipping method to be developed here will modify the line so that what remains of it lies within the window; further, clipping returns a boolean value of false if no part of the line is in the window. This boolean value can be used to direct the subsequent rendering of a device coordinate line whose endpoints correspond to those of the clipped world coordinate line.

```
struct BoxDouble
{
   double left;
   double right;
   double bottom;
   double top;
};
struct BoxInt
{
   int left;
   int right;
   int bottom;
   int top;
};
struct Point
{
   double x;
   double y;
};

// WC  = world coordinates
// NDC = normalized device coordinates
```

```
// DC  = device coordinates (pixels)

void writeLine (BoxDouble window, BoxDouble viewport, BoxInt screen, Point pA_WC, Point pB_WC)
{
   bool visible = clip (window, pA_WC, pB_WC);

   if (visible)
   {
      map_WCtoNDCtoDC (window, viewport, screen, pA, pB, pA_DC, pB_DC);

      writeLine (pA_DC, pB_DC);
   }
   return;
}
%void LineWC2d::write (EnvMap& map) const
%{
%   LineWC2d lineCopy = *this;  // copy prevents modification of original
%   bool visible = lineCopy.clip (map);
%   if (visible)
%   {
%      LineDC line (map.wcTOdc (lineCopy.p1), map.wcTOdc (lineCopy.p2), getColor());
%      line.write (map);
%   }
%   return;
%}
```

6.2 The Cohen-Sutherland Clipping Algorithm

The Cohen-Sutherland clipping algorithm categorizes a line as falling into one of three action cases with respect to a given window:

1. trivial accept: Both of the line's endpoints fall within the window.
2. trivial reject: Both of the line's endpoints fall outside of the window on the same side.
3. further refine: One endpoint lies outside, but both do not fall outside on the same side.

Further refinement moves the one endpoint known to be outside of the window to a window boundary. The same endpoint cannot be left of and right of the window; neither can it be above and below the window. Consequently, the iteration of the Cohen-Sutherland clipping algorithm can take place at most four times: twice per endpoint to move them to window boundaries.

```
bool clip (const EnvMap& map)
{
   bool done    = false;  // initialize loop control
   bool visible = false;  // assume nothing resides within window; prove otherwise

   do // while ! done
   {
      if // both points are inside the window
      {
```

Figure 6.1: Unclipped Vertical and Horizontal Lines

```
            visible = true;  // everything resides within window; trivial accept
            done    = true;
         }
         else if // both points are left of, right of, above or below the window

            done    = true;  // nothing resides within window; trivial reject

         else // at least one point is outside window & line may cross window
         {
            if // according to the map, p1 is inside the window
               swapEndpoints ();  // make p1 outside

            nonTrivialClip (map);  // focus can be on p1 because of swap
         }
      }
      while (! done);

      return visible;
   }
```

6.3 The Non-Trivial Clip

At this point, it is known that of the two endpoints of the line, at least p_1 lies outside of the window; also, both endpoints do not lie outside and on the same side of the window. The objective is to improve the situation by moving p_1 to a window boundary. Note in the following how p_1 may be moved to a boundary but still lie outside of the window.

Vertical and horizontal lines occur frequently in computer graphics, and examples of each can certainly require clipping; see lines L_{V_T} , L_{V_B}, L_{H_L} and L_{H_R} in Figure 6.1. Because they occur frequently, streamlining their solution may increase the overall speed of the application. Clipping such lines is quite simple. For vertical lines, move in y from the outside point to the window boundary; the x value is unchanged. In the case of L_{V_T}, p_1's y is changed to W_T while p_1's x is unchanged. Likewise, in the case of L_{V_B}, p_1's y is changed to W_B. Horizontal lines are dealt with in complementary fashion: move in x from the outside point to the window boundary; the y value is unchanged. In the case of L_{H_L}, p_1's x is changed to W_L while p_1's y is unchanged. Likewise, in the case of L_{H_R}, p_1's x is changed to W_R.

The remaining cases are more complicated as the line may cross *more than one* of the window boundary values; see Figure 6.2. As before, the y-value of a line exceeding the window's top or bottom boundary is moved to the window's corresponding boundary; the difference is that the x-value now moves as well. The

Figure 6.2: Unclipped Lines Neither Vertical Nor Horizontal

Figure 6.3: Worst and Best Cases for Clipping Order *left*, *right*, *top*, *bottom*

x-value where the line crosses the window's top boundary may be obtained from the formula for slope.

$$\frac{dy}{dx} = \frac{y_2 - y_1}{x_2 - x_1} \tag{6.1}$$

$$= \frac{W_T - y_1}{x_T - x_1} \tag{6.2}$$

$$x_T - x_1 = (W_T - y_1)/\frac{dy}{dx} \tag{6.3}$$

$$x_T = x_1 + (W_T - y_1)\frac{dx}{dy} \tag{6.4}$$

The situation is analogous for x_B.

Similarly, when the x-value of a line is moved to the left or right boundary, the corresponding y-value may be obtained from the formula for slope. Consider the case of crossing the left boundary.

$$\frac{dy}{dx} = \frac{y_2 - y_1}{x_2 - x_1} \tag{6.5}$$

$$= \frac{y_L - y_1}{W_L - x_1} \tag{6.6}$$

$$y_L - y_1 = (W_L - x_1)\frac{dy}{dx} \tag{6.7}$$

$$y_L = y_1 + (W_L - x_1)\frac{dy}{dx} \tag{6.8}$$

The analogous situation here is for y_R.

Only one of the above clips will be performed by the Cohen-Sutherland algorithm per iteration. The order in which the clipping tests are performed affects how many iterations are required for a particular line. Figure 6.3 illustrates the worst case for line L_1 when the tests are performed in the following order for the window sides: left, right, top, bottom. Of course, for every worst case there is an opposing best case for the same order; see line L_2.