Lab Report

Week 7

Ritobroto Maitra (1301CS50)

5/03/16

Title

- Implement :
  1.Cohen-Sutherland Algorithm
  2.Sutherland-Hodgman Algorithm

Cohen Sutherland

Steps : The algorithm includes, excludes or partially includes the line based on where :

1. Both endpoints are in the viewport region (bitwise OR of endpoints == 0) : trivial accept.
2. Both endpoints share at least one non-visible region which implies that the line does not cross the visible region. (bitwise AND of endpoints != 0) : trivial reject.
3. Both endpoints are in different regions : In case of this nontrivial situation the algorithm finds one of the two points that is outside the viewport region (there will be at least one point outside). The intersection of the outpoint and extended viewport border is then calculated (i.e. with the parametric equation for the line) and this new point replaces the outpoint. The algorithm repeats until a trivial accept or reject occurs.

The numbers in the figure below are called outcodes. An outcode is computed for each of the two points in the line. The outcode will have four bits for two-dimensional clipping, or six bits in the three-dimensional case. The first bit is set to 1 if the point is above the viewport. The bits in the 2D outcode represent : Top, Bottom, Right, Left. For example the outcode 1010 represents a point that is top-right of the viewport. Note that the outcodes for endpoints must be recalculated on each iteration after the clipping occurs.

1001	1000	1010
0001	0000	0010
0101	0100	0110

The Cohen–Sutherland algorithm can be used only on a rectangular clipping area.

Code Snippet

```
1
2  OutCode ComputeOutCode(double x, double y) {
3    OutCode code;
4    code = INSIDE;          // initialised as being inside of clip window
5    if (x < xmin)           // to the left of clip window
6      code |= LEFT;
7    else if (x > xmax)      // to the right of clip window
8      code |= RIGHT;
9    if (y < ymin)           // below the clip window
10     code |= BOTTOM;
11   else if (y > ymax)      // above the clip window
12     code |= TOP;
```

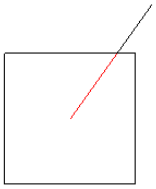

FIGURE 1 – Cohen-Sutherland

```
  return code;
}

// Cohen-Sutherland clipping algorithm clips a line from
// P0 = (x0, y0) to P1 = (x1, y1) against a rectangle with
// diagonal from (xmin, ymin) to (xmax, ymax).
bool CohenSutherlandLineClipAndDraw(double x0, double y0, double x1, double y1) {
  OutCode outcode0 = ComputeOutCode(x0, y0);
  OutCode outcode1 = ComputeOutCode(x1, y1);
  bool accept = false;

  while (true) {
    if (!(outcode0 | outcode1)) {
      accept = true;
      break;
    } else if (outcode0 & outcode1) {
      break;
    } else {
      // failed both tests, so calculate the line segment to clip
      // from an outside point to an intersection with clip edge
      double x, y;

      // At least one endpoint is outside the clip rectangle; pick it.
      OutCode outcodeOut = outcode0 ? outcode0 : outcode1;

      // Now find the intersection point;
      // use formulas y = y0 + slope * (x - x0), x = x0 + (1 / slope) * (y - y0)
      if (outcodeOut & TOP) {           // point is above the clip rectangle
        x = x0 + (x1 - x0) * (ymax - y0) / (y1 - y0);
        y = ymax;
      } else if (outcodeOut & BOTTOM) { // point is below the clip rectangle
        x = x0 + (x1 - x0) * (ymin - y0) / (y1 - y0);
        y = ymin;
      } else if (outcodeOut & RIGHT) {  // point is to the right of clip rectangle
        y = y0 + (y1 - y0) * (xmax - x0) / (x1 - x0);
        x = xmax;
      } else if (outcodeOut & LEFT) {   // point is to the left of clip rectangle
        y = y0 + (y1 - y0) * (xmin - x0) / (x1 - x0);
        x = xmin;
      }

      // Now we move outside point to intersection point to clip
      // and get ready for next pass.
      if (outcodeOut == outcode0) {
        x0 = x;
        y0 = y;
        outcode0 = ComputeOutCode(x0, y0);
      } else {
        x1 = x;
        y1 = y;
        outcode1 = ComputeOutCode(x1, y1);
      }
    }
  }
  if(accept){
    glBegin(GL_LINES);
    glVertex2f(x0,y0);
    glVertex2f(x1,y1);
    glEnd();
  }
  return accept;
```

```
}
```

Sutherland-Hodgeman

The algorithm begins with an input list of all vertices in the subject polygon. Next, one side of the clip polygon is extended infinitely in both directions, and the path of the subject polygon is traversed. Vertices from the input list are inserted into an output list if they lie on the visible side of the extended clip polygon line, and new vertices are added to the output list where the subject polygon path crosses the extended clip polygon line.

This process is repeated iteratively for each clip polygon side, using the output list from one stage as the input list for the next. Once all sides of the clip polygon have been processed, the final generated list of vertices defines a new single polygon that is entirely visible. Note that if the subject polygon was concave at vertices outside the clipping polygon, the new polygon may have coincident (i.e. overlapping) edges – this is acceptable for rendering, but not for other applications such as computing shadows.

Code Snippet

```

struct Point{
    float x,y;
} w[4],oVer[4];
int Nout;

void drawPoly(Point p[],int n){
    glBegin(GL_POLYGON);
    for(int i=0;i<n;i++)
        glVertex2f(p[i].x,p[i].y);
    glEnd();
}

bool insideVer(Point p){
        if((p.x>=w[0].x)&&(p.x<=w[2].x))
            if((p.y>=w[0].y)&&(p.y<=w[2].y))
                return true;
        return false;
}

void addVer(Point p){
    oVer[Nout]=p;
    Nout=Nout+1;
}

Point getInterSect(Point s,Point p,int edge){
    Point in;
    float m;
    if(w[edge].x==w[(edge+1)%4].x){ //Vertical Line
        m=(p.y-s.y)/(p.x-s.x);
        in.x=w[edge].x;
        in.y=in.x*m+s.y;
    }
    else{//Horizontal Line
        m=(p.y-s.y)/(p.x-s.x);
        in.y=w[edge].y;
        in.x=(in.y-s.y)/m;
    }
    return in;
}

void clipAndDraw(Point inVer[],int Nin){
    Point s,p,interSec;
    for(int i=0;i<4;i++)
    {
        Nout=0;
        s=inVer[Nin-1];
```

```
48          for(int j=0;j<Nin;j++)
49          {
50              p=inVer[j];
51              if(insideVer(p)==true){
52                  if(insideVer(s)==true){
53                      addVer(p);
54                  }
55                  else{
56                      interSec=getInterSect(s,p,i);
57                      addVer(interSec);
58                      addVer(p);
59                  }
60              }
61              else{
62                  if(insideVer(s)==true){
63                      interSec=getInterSect(s,p,i);
64                      addVer(interSec);
65                  }
66              }
67              s=p;
68          }
69          inVer=oVer;
70          Nin=Nout;
71      }
72      drawPoly(oVer,4);
73  }
```

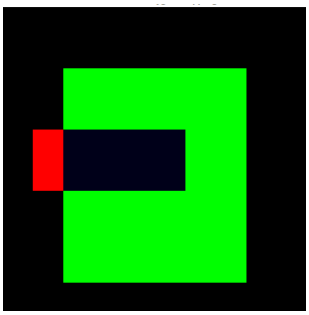

FIGURE 2 – Sutherland-Hodgemaa