WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
Deutsch
Español
Italiano
עברית
Polski
Русский
Српски / srpski
Українська
Edit links

Cohen–Sutherland algorithm

From Wikipedia, the free encyclopedia
(Redirected from Cohen–Sutherland)

The **Cohen–Sutherland algorithm** is a computer graphics algorithm used for line clipping. The algorithm divides a two-dimensional space into 9 regions (or a three-dimensional space into 27 regions), and then efficiently determines the lines and portions of lines that are visible in the center region of interest (the viewport).

The algorithm was developed in 1967 during flight simulator work by Danny Cohen and Ivan Sutherland.[1]

Contents [hide]

1 The Algorithm
2 Example C/C++ implementation
3 Notes
4 See also
5 References
6 External links

The Algorithm [edit]

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

- Both endpoints are in the viewport region (bitwise OR of endpoints == 0): trivial accept.
- 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.
- 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. For other convex polygon clipping windows, use the Cyrus–Beck algorithm.

Example C/C++ implementation [edit]

```
typedef int OutCode;

const int INSIDE = 0; // 0000
const int LEFT = 1;   // 0001
const int RIGHT = 2;  // 0010
const int BOTTOM = 4; // 0100
const int TOP = 8;    // 1000

// Compute the bit code for a point (x, y) using the clip rectangle
// bounded diagonally by (xmin, ymin), and (xmax, ymax)

// ASSUME THAT xmax, xmin, ymax and ymin are global constants.
```

```
OutCode ComputeOutCode(double x, double y)
{
 OutCode code;

 code = INSIDE;          // initialised as being inside of clip window

 if (x < xmin)           // to the left of clip window
  code |= LEFT;
 else if (x > xmax)      // to the right of clip window
  code |= RIGHT;
 if (y < ymin)           // below the clip window
  code |= BOTTOM;
 else if (y > ymax)      // above the clip window
  code |= TOP;

 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).
void CohenSutherlandLineClipAndDraw(double x0, double y0, double x1, double y1)
{
 // compute outcodes for P0, P1, and whatever point lies outside the clip rectangle
 OutCode outcode0 = ComputeOutCode(x0, y0);
 OutCode outcode1 = ComputeOutCode(x1, y1);
 bool accept = false;

 while (true) {
  if (!(outcode0 | outcode1)) { // Bitwise OR is 0. Trivially accept and get out of
loop
   accept = true;
   break;
  } else if (outcode0 & outcode1) { // Bitwise AND is not 0. Trivially reject and
get out of loop
   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) {
               // Following functions are left for implementation by user based on
               // their platform (OpenGL/graphics.h etc.)
               DrawRectangle(xmin, ymin, xmax, ymax);
               LineSegment(x0, y0, x1, y1);
	}
}
```

Notes [edit]

1. ^ *Principles of Interactive Computer Graphics* p.124 and p.252, by Bob Sproull and William M. Newman, 1973, McGraw–Hill Education, International edition, ISBN 0-07-085535-8

See also [edit]

Algorithms used for the same purpose:

- Liang–Barsky
- Cyrus–Beck
- Nicholl–Lee–Nicholl
- Fast-clipping

References [edit]

- James D. Foley. *Computer graphics: principles and practice*. Addison-Wesley Professional, 1996. p. 113.

External links [edit]

- Animated javascript implementation of Cohen-Sutherland algorithm
- Delphi implementation
- C# implementation

Categories: Clipping (computer graphics)