Sutherland–Hodgman algorithm

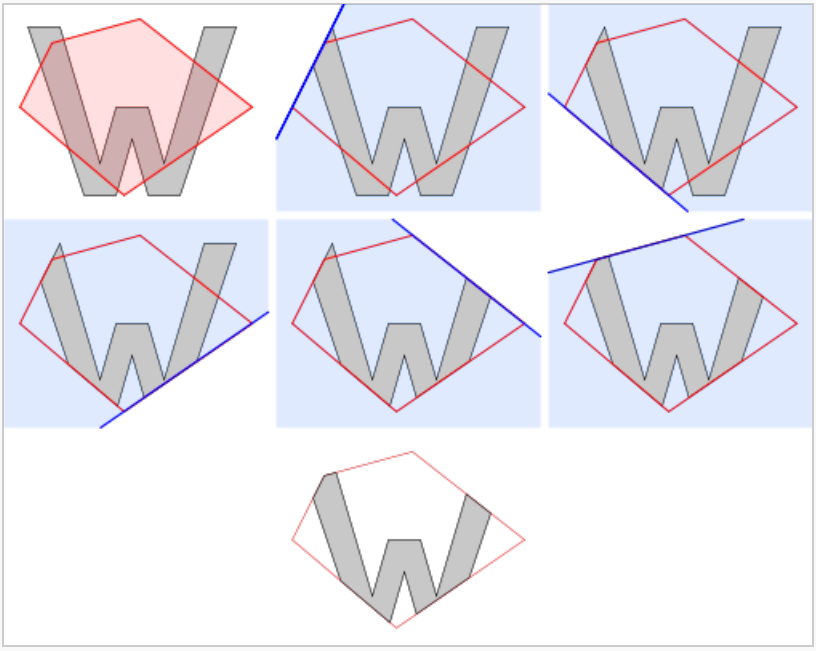
From Wikipedia, the free encyclopedia
(Redirected from Sutherland–Hodgman)

The **Sutherland–Hodgman algorithm** is used for clipping polygons. It works by extending each line of the convex *clip polygon* in turn and selecting only vertices from the *subject polygon* that are on the visible side.

Description [edit]

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.



All steps of clipping concave polygon 'W by 5-sided convex polygon

The Weiler–Atherton algorithm overcomes this by returning a set of divided polygons, but is more complex and computationally more expensive, so Sutherland–Hodgman is used for many rendering applications. Sutherland–Hodgman can also be extended into 3D space by clipping the polygon paths based on the boundaries of planes defined by the viewing space.

Pseudo code [edit]

Given a list of edges in a clip polygon, and a list of vertices in a subject polygon, the following procedure clips the subject polygon against the clip polygon.

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
فارسی
Français
Polski
Edit links

```
  List outputList = subjectPolygon;
  for (Edge clipEdge in clipPolygon) do
     List inputList = outputList;
     outputList.clear();
     Point S = inputList.last;
     for (Point E in inputList) do
        if (E inside clipEdge) then
           if (S not inside clipEdge) then
              outputList.add(ComputeIntersection(S,E,clipEdge));
           end if
           outputList.add(E);
        else if (S inside clipEdge) then
           outputList.add(ComputeIntersection(S,E,clipEdge));
        end if
        S = E;
     done
  done
```

The vertices of the clipped polygon are to be found in *outputList* when the algorithm terminates. Note that a point is defined as being *inside* an edge if it lies on the same side of the edge as the remainder of the polygon. If the vertices of the clip polygon are consistently listed in a clockwise direction, then this is equivalent to testing whether the point lies to the left of the line (left means *inside*, while right means *outside*), and can be implemented simply by using a cross product.

ComputeIntersection is a trivial function, omitted here for clarity, which returns the intersection of a line segment and an infinite edge. Note that it is only called if such an intersection is known to exist, and hence can simply treat both lines as being infinitely long.

Working Code in C++ [edit]

```
//Sutherland-Hodgman Polygon Clipping
//Copyright © 2015 by Mohammad Nazmul Saqib.
#include <iostream>
#include <vector>
#include "graphics.h"
#include "Line2d.h"
#include "Coordinates2d.h"
#include "Polygon2d.h"

class ClippingPolygon2d
{
private:
 Polygon2d clippingPolygon;
 Polygon2d candidatePolygon;

public:
 ClippingPolygon2d(){}

 void SetCandidatePolygon(Polygon2d & pol)
 {
  candidatePolygon = pol;
 }

 void SetClippingPolygon(Polygon2d & pol)
 {
  clippingPolygon = pol;
 }

public:
 std::vector<Point2d> GetClippedPoints()
    {
        std::vector<Point2d> polygonVertices = candidatePolygon.GetVertices();
  std::vector<Point2d> clipping = clippingPolygon.GetVertices();

        for (size_t i = 0; i < clipping.size(); i++)
        {
   Line2d clippingEdge(clipping[i], clipping[(i + 1) % clipping.size()]);
            std::vector<Point2d> temp;
```

```
            for (size_t j = 0; j < polygonVertices.size(); j++)
            {
                Point2d polygonEdgeStart = polygonVertices[j];
                Point2d polygonEdgeEnd = polygonVertices[(j + 1) %
polygonVertices.size()];

                if (clippingEdge.onLeft(polygonEdgeStart))
                {
                    if (clippingEdge.onLeft(polygonEdgeEnd))
                    {
                        temp.push_back(polygonEdgeEnd);
                    }
                    else //Right
                    {

temp.push_back(clippingEdge.GetIntersectionExtended(Line2d(polygonEdgeStart,
polygonEdgeEnd)));
                    }
                }
                else //Right
                {
                    if (clippingEdge.onLeft(polygonEdgeEnd))
                    {

temp.push_back(clippingEdge.GetIntersectionExtended(Line2d(polygonEdgeStart,
polygonEdgeEnd)));
                        temp.push_back(polygonEdgeEnd);
                    }
                    else //Right
                    {
                        // nothing to do: outside of the window
                    }
                }
            }
            polygonVertices = temp;
        }
        return polygonVertices;
    }
};

int main()
{
 ////////////////////// Initialize  //////////////////////
 Coordinates2d::ShowWindow("Sutherland-Hodgman Line Clipping");
 ////////////////////////////////////////////////////////////

 Polygon2d clippingPolygon;
 clippingPolygon.Add(20,20);
 clippingPolygon.Add(200,20);
 clippingPolygon.Add(200,160);
 clippingPolygon.Add(20,160);

 Polygon2d candidatePolygon;
 candidatePolygon.Add(Point2d(-20, 80));
 candidatePolygon.Add(Point2d(10, 80));
 candidatePolygon.Add(Point2d(110, 180));
 candidatePolygon.Add(Point2d(210, 80));
 candidatePolygon.Add(Point2d(240, 80));
 candidatePolygon.Add(Point2d(110, 210));

 ClippingPolygon2d clip;
 clip.SetClippingPolygon(clippingPolygon);
 clip.SetCandidatePolygon(candidatePolygon);

 std::vector<Point2d> clippedVerticesList = clip.GetClippedPoints();

 Coordinates2d::Draw(candidatePolygon, Red);
 Coordinates2d::Draw(clippingPolygon, Magenta);

 Polygon2d poly2(clippedVerticesList);
```

```
  Coordinates2d::Draw(poly2, Yellow);

  ////////////////////  Finalize  ////////////////////////
  Coordinates2d::Wait();
  return 0;
  ////////////////////////////////////////////////////////
}
```

See also [edit]

- Weiler–Atherton clipping algorithm
- Vatti clipping algorithm
- Clipping (in rasterisation)

References [edit]

- Mel Slater, Anthony Steed, Yiorgos Chrysanthou: *Computer Graphics and Virtual Environments: From Realism to Real-Time.* Addison Wesley. ISBN 0-201-62420-6.
- Ivan Sutherland, Gary W. Hodgman: *Reentrant Polygon Clipping.* Communications of the ACM, vol. 17, pp. 32–42, 1974

External links [edit]

- Polygon clipping and filling Describes the algorithm using images that are easy to understand.

This computer graphics–related article is a stub. You can help Wikipedia by expanding it.

Categories: Clipping (computer graphics) | Computer graphics stubs