Binghamton University

EngiNet™

State University of New York

Thomas J. Watson

School of Engineering
and Applied Science

EngiNet™

WARNING

CS 460/560

Computer Graphics

Professor Richard Eckert

Lecture # 16

March 28, 2001

Lecture 16

- **Polygon Clipping**
  - **Sutherland-Hodgeman Polygon Clipper**
  - **Weiler-Atherton Polygon Clipper**
- **Clipping other Curves**
- **Text Clipping**
- **Modeling Complex Shapes**
  - **Parametric Polynomial Curves**

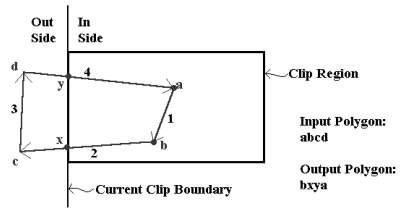
Sutherland-Hodgeman Polygon Clipper

- Approach:
  - Clip all polygon edges with respect to each clipping boundary
  - Do four passes; on each pass:
    - Traverse current polygon and clip with respect to one of the four boundaries
    - Assemble output polygon edges as you go
    - vin[] --> Clip Left --> vtemp1[] --> Clip Right --> vtemp2[] --> Clip Bottom --> vtemp3[] --> Clip Top --> vout[]

- On any polygon traversal the clip boundary divides plane into "in" side and "out" side
- For any given edge (vertices i and i+1),
  - during traversal, there are four possibilities:
  - (Assume vertex i has already been processed)

```
VERTEX i VERTEX i+1  ACTION
 in       in       Add Vertex i+1 to output list
 out      out      Add no vertex to output list
 in       out      Add intersection point with
                    edge to output list
 out      in       Add intersection point with edge
                    and vertex i+1 to output list
```

Sample Traversal



Traversal		Type	Action
1	a → b	in-in	Add point b
2	b → c	in-out	Add intersection point x
3	c → d	out-out	Add nothing
4	d → a	out-in	Add intersection point y and point a

Implementation

- Function sh_clip()
  - Will clip an input polygon (ni, vi[])
  - With respect to a given boundary (bndry)
  - Generating an output polygon (no, vo[])
- Enumerate the boundaries as:
  - LEFT, RIGHT, BOTTOM, and TOP

sh_clip(ni, vi[], no, vo[], xmin, ymin, xmax, ymax, bndry);

vi[] and vo[]: could be arrays of POINTs

ni, no: number of points in each array

xmin, ymin, xmax, ymax: clip region boundaries

Using sh_clip() to clip a polygon

- Make four calls to sh_clip():

```
sh_clip(nin, vin[], ntemp1, vtemp1[], xmin, ymin,
        xmax, ymax, LEFT);
sh_clip(ntemp1, vtemp1[], ntemp2, vtemp2[], xmin,
        ymin, xmax, ymax, RIGHT);
sh_clip(ntemp2, vtemp2[], ntemp3, vtemp3[], xmin,
        ymin, xmax, ymax, BOTTOM);
sh_clip(ntemp3, vtemp3[], nout, vout[], xmin, ymin,
        xmax, ymax, TOP);
```

Three Helper Functions

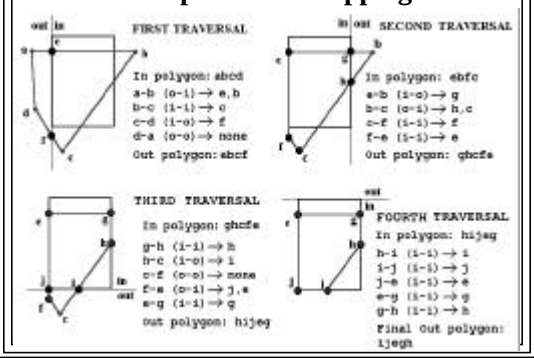
BOOL inside(V, xmin, ymin, xmax, ymax, Bndry)
- Returns TRUE if vertex point V is on the "in" side of boundary Bndry

intersect(V1, V2, xmin, ymin, xmax, ymax, Bndry, Vnew)
- Computes intersection point of edge whose endpoints are V1 and V2 with boundary Bndry
- Returns the resulting point in Vnew

output(V, n, vout[])
- Adds vertex point V to the polygon (n, v[])
  - n will be incremented by 1
  - vertex V added to end of polygon's vertex list v[]

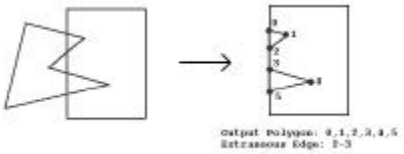
```
sh_clip (ni, vi[], no, vo[], bndry)
no = 0                          // output list begins empty
First_V = vi[0]                 // first vertex (i)
For (j=0 to ni-1)               // traverse polygon
  Second_V = v[(j+1) % ni]      // second vertex (i+1)
  If (inside(first_V, bndry)
    If (inside(Second_V, bndry)  // "in-in" case
       output(Second_V, no, vo)
    Else                         // "in-out" case
       intersect(First_V, Second_V, bndry, Vtemp)
       output (Vtemp, no, vo)
  Else
    If (inside(Second_V, bndry)  // "out-in" case
       intersect(First_V, Second_V, bndry, Vtemp)
       output(Vtemp, no, vo)
       output(Second_V, no, vo)   // no "out-out" case
  First_V = Second_V             // prepare for next edge
```

Example of S-H Clipping



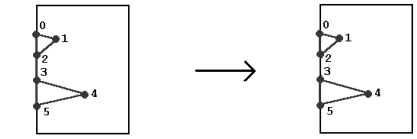
Sutherland-Hodgeman Problems

- Works fine with convex polygons
- But some concave polygons problematic
  - Extraneous edges along a clip boundary may be generated as part of the output polygon
  - Could cause problems with polygon filling



Solutions to S-H Problems

- Add a postprocessing step
  - Check output vertex list for multiple (>2) vertex points along any clip boundary
  - Correctly join pairs of vertices



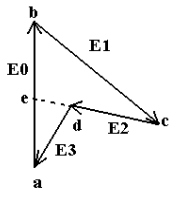
Polygon: 0,1,2,3,4,5
0,2,3,5 are vertices along the left boundary

So break into two polygons:
0,1,2 and 3,4,5

Other Solutions

- Add a preprocessing step
  - Triangularize the Concave Polygon
    - Break it up into triangles
    - Can always be done
- Better: Use a more general clipping algorithm
  - For example, the Weiler-Atherton polygon clipper

Triangularization



abcd --> aed & ebc

E0 X E1 --> -k
E1 X E2 --> -k
E2 X E3 --> +k ==> find int. pt.
E3 X E0 --> -k

$$E0 \times E1 = \begin{vmatrix} i & j & k \\ Dx0 & Dy0 & 0 \\ Dx1 & Dy1 & 0 \end{vmatrix}$$

Dx0 = xb-xa
Dy0 = yb-ya

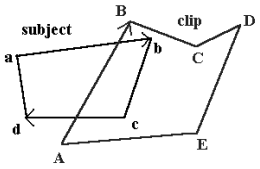
Weiler-Atherton Polygon Clipper

- Clips a "Subject Polygon" to a "Clip Polygon"
- Both polygons can be of any shape
- Result: one or more output polygons that lie entirely inside the clip polygon
- Basic idea:
  - Follow a path that may be a polygon edge or a window boundary, depending on the type of edge

The Weiler-Atherton Algorithm

1. Set up vertex list for subject and clip polygons
   Ordering: as you move down each list, inside of polygon is always on the right side (clockwise)
2. Compute all intersection points between subject polygon and clip polygon edges
   Insert them into each polygon's list
   Mark as intersection points
   Mark o-i intersection points
   (subject polygon edge moving from outside to inside of clip polygon edge)

Intersection Points & out-in marking



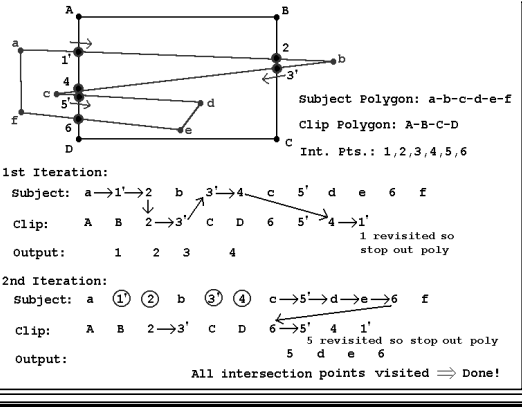
$\overline{ab}$:
x = xa + (xb-xa)t
y = ya + (yb-ya)t

$\overline{AB}$:
x = xA + (xB-xA)s
y = yA + (yB-yA)s

Solve for s and t
0<=t<=1 and 0<=s<=1 ==>
Intersection Point

Vector cross product:
$\overline{ab} \times \overline{AB}$ = +k ==> in-out
$\overline{cd} \times \overline{AB}$ = -k ==> out-in

3. Do until all intersection points have been visited:
   - Traverse subject polygon list until a non-visited out-in intersection point is found;
   - Output it to new output list
   - Make subject polygon list be the active list
   - Do until a vertex is revisited:
     - Get next vertex from active list and output
     - If vertex is an intersection point,
       - make the other list active
   - End current output polygon list



Clipping Other Curves

- Must get intersection with clip boundaries
- In general solve nonlinear equations
- Time consuming

Clipping Text

- Use successively more expensive tests
  1. Clip string
     Embed string in rectangle
     Clip rectangle (4 point tests)
     - entirely in ==> keep string
     - entirely out==>reject string
     - neither==>next test

2. Clip each Character
   Embed character in rectangle
   Clip rectangle (4 point tests)
   - entirely in ==> keep character
   - entirely out==>reject character
   - neither==>next test
3. Two possibilities for Character Clipping
   - Bitmapped: look at each pixel
   - Stroked: Apply line clipper to each stroke

Modeling Complex Shapes

- Can use line/polygon primitives to approximate
- But complex objects-->huge number of primitives
- Better to use more complex primitives
- Use curves (2-D) or surfaces (3-D)

Curves in Space

- Three forms:
  - Explicit
  - Implicit
  - Parametric

Explicit Form

- y = f(x)
- example--line:
  y = m*x + b
  But this is not a finite line segment
- Not all curves can be put into this form

Implicit Form

- f(x,y)=0
- Example--circle:
  $(x-h)^2 + (y-k)^2 - R^2 = 0$
- Indicates if a point x,y is on the curve
- Can be difficult to plot

Parametric Form

- x and y expressed as explicit functions of a parameter, t
  x = f(t)
  y = g(t)
- Range of parameter also given
  – Delimits the extent of the curve
- To plot, let t vary over its range
  – Points on curve are generated
- Easily extended to curves in 3-D
  z = h(t)

Parametric Equations for a Line Segment

- Given endpoints P1(x1,y1), P2(x2,y2)
- Assume:
  t=0: endpoint P1
  t=1: endpoint P2
- Linear equation ==>
  x = a*t + b
  y = c*t +d
- Need to get constants a,b,c,d

- Apply boundary conditions:
  t=0 ==> x=x1, y=y1
  x1 = a*0 + b, so b=x1
  y1 = c*0 + d, so d=y1
  t=1 ==> x=x2, y=y2
  x2 = a*1 + b, so a = x2 - b, or a = x2 - x1
  y2 = c*1 + d, so c = y2 - d, or c = y2 - y1
- Resulting Parametric equations:
  x = (x2-x1)*t + x1
  y = (y2-y1)*t + y1 **0<=t<=1**

Polynomials

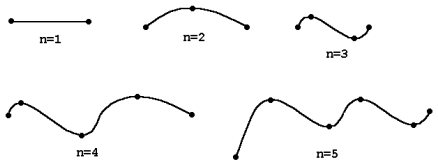
- Explicit Form of n-degree polynomial:
  $y = a_0 + a_1*x + a_2*x^2 + ... a_n*x^n$
- Assume we have a set of n+1 known control points: (xi,yi)
- Get polynomial coefficients a_i from the control points
- Two Methods:
  – Interpolation
  – Approximation

Interpolating Polynomial, degree n

- Curve passes through all n+1 control points (xi,yi)
- Given (x0,y0), (x1,y1), (x2,y2) ... (xn,yn):
  $y0 = a_0 + a_1*x0 + a_2*x0^2 \ldots a_n*x0^n$
  $y1 = a_0 + a_1*x1 + a_2*x_1^2 \ldots a_n*x1^n$
  …
  $yn = a_0 + a_1*xn + a_2*xn^2 \ldots a_n*xn^n$
- n+1 equations in n+1 unknown constants:
  $a_0, a_1, a_2, \ldots a_n$

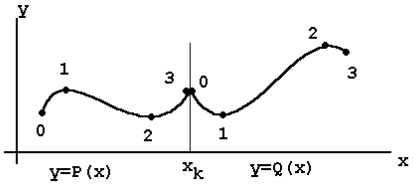
May not be good in graphics

- Many control points==>high degree polynomial
- Many calculations
- Polynomial "wiggle"



Segmented Interpolating Polynomials

- Break curve into segments
- Each with different low-degree polynomial
- Easier computations



Joining Segmented Curves

- Join points called knots
- kth knot at $x=x_k$
- Level-0 continuity: $P(x_k)=Q(x_k)$
  - Continuous, but not smooth (kinks)
- Level-1 continuity: $P'(x_k)=Q'(x_k)$
  - First derivative-->smoother curve
- Level-2 conitnuity: $P''(x_k)=Q''(x_k)$
  - Second derivative-->still smoother

Approximating Polynomials

- Curve determined by control point
- But does NOT go through all of them
- Control Points act as magnets
- Better for many graphics applications
- Most commonly used:
  - Bezier curves
  - B-spline curves

- Bezier Curves
  - See CS-460/560 Notes:
  - Bezier Polynomial Curves

  http://www.cs.binghamton.edu/~reckert/460/bezier.htm
- B-Spline Curves
  - See CS-460/560 Notes:
  - B-spline Polynomial Curves

  http://www.cs.binghamton.edu/~reckert/460/bspline.htm