Polygon clip (against convex polygon)
Sutherland-Hodgeman polygon clip

- Recall: polygon is convex if any line joining two points inside the polygon, also lies inside the polygon; implies that a point is inside if it is on the right side of each edge.

- Clipping each edge of a given polygon doesn’t make sense - how do we reassemble the pieces? We want to arrange doing so on the fly.

- Clipping the polygon against each edge of the clip window in sequence works if the clip window is convex.

- (Note similarity to Sutherland-Cohen line clipping)
Sutherland-Hodgeman polygon clip
Sutherland-Hodgeman polygon clip

Polygon to clip

Clip entire polygon against one edge

Clip window
Sutherland-Hodgeman polygon clip

Polygon to clip

Clip window

Then clip it against the next
Sutherland-Hodgeman polygon clip

Polygon to clip

Clip window

And finally, the last one.
Clipping against current clip edge

- Polygon is a list of vertices
- Think of process as rewriting polygon, vertex by vertex
- Check start vertex
  - in - emit it
  - out - ignore it
- Walk along vertices and for each edge consider four cases and apply corresponding action.

- Four cases:
  - polygon edge crosses clip edge going from out to in
    - emit crossing, next vertex
  - polygon edge crosses clip edge going from in to out
    - emit crossing
  - polygon edge goes from out to out
    - emit nothing
  - polygon edge goes from in to in
    - emit next vertex
polygon edge crosses clip edge going from out to in  => emit crossing, next vertex
polygon edge crosses clip edge going from in to out  => emit crossing
polygon edge goes from out to out  => emit nothing
polygon edge goes from in to in  => emit next vertex
polygon edge crosses clip edge going from out to in
emits crossing, next vertex

polygon edge crosses clip edge going from in to out
emits crossing

polygon edge goes from out to out
emits nothing

polygon edge goes from in to in
emits next vertex
Now have

- Polygon edge crosses clip edge going from out to in $\Rightarrow$ emit crossing, next vertex
- Polygon edge crosses clip edge going from in to out $\Rightarrow$ emit crossing
- Polygon edge goes from out to out $\Rightarrow$ emit nothing
- Polygon edge goes from in to in $\Rightarrow$ emit next vertex
polygon edge crosses clip edge going from out to in  ==> emit crossing, next vertex
polygon edge crosses clip edge going from in to out ==> emit crossing
polygon edge goes from out to out                   ==> emit nothing
polygon edge goes from in to in                    ==> emit next vertex
Now have

crossing edge moves from out to in

crossing edge moves from in to out

goes from out to out

goes from in to in
Clipping against next edge (right) gives

- Polygon edge crosses clip edge going from out to in => emit crossing, next vertex
- Polygon edge crosses clip edge going from in to out => emit crossing
- Polygon edge goes from out to out => emit nothing
- Polygon edge goes from in to in => emit next vertex
Clipping against final(bottom) edge gives

- polygon edge crosses clip edge going from out to in  ==> emit crossing, next vertex
- polygon edge crosses clip edge going from in to out  ==> emit crossing
- polygon edge goes from out to out                ==> emit nothing
- polygon edge goes from in to in                  ==> emit next vertex
More Polygon clipping

• Notice that we can have a pipeline of clipping processes, one against each edge, each operating on the output of the previous clipper -- substantial advantage.

• Unpleasantness can result from concave polygons; in particular, polygons with empty interior.

• Can modify algorithm for concave polygons (e.g. Weiler Atherton)
For clockwise polygon (starting outside):

- For out-to-in pair, follow usual rule
- For in-to-out pair, follow clip edge (clockwise) and then jump to next vertex (which is on the outside) and start again
- Only get a second piece if polygon is convex
Additional remarks on clipping

- Although everything discussed so far has been in terms of polygons/lines clipped against lines in 2D, all - except Nicholl-Lee-Nicholl - will work in 3D against convex regions without much change.

- This is because the central issue in each algorithm is the inside outside decision as a convex region is the intersection of half spaces.

- Inside-outside decisions can be made for lines in 2D, planes in 3D. e.g testing $x \geq 0$

- Hence, all (except N-L-N) can be used to clip:
  - Lines against 3D convex regions (e.g. cubes)
  - Polygons against 3D convex regions (e.g. cubes)

- NLN could work in 3D, but the number of cases increases too much to be practical.
2D Transformations

• Represent transformations by matrices
• To transform a point, represented by a vector, multiply the vector by the appropriate matrix.
• To transform lines, transform endpoints
• To transform polygons, transform vertices
2D Transformations

- Scale (stretch) by a factor of $k$

$$M = \begin{vmatrix} k & 0 \\ 0 & k \end{vmatrix}$$

(k = 2 in the example)
2D Transformations

• Scale by a factor of \((S_x, S_y)\)

\[
M = \begin{vmatrix}
S_x & 0 \\
0 & S_y
\end{vmatrix}
\]

(Above, \(S_x = 1/2, S_y = 1\))
2D Transformations

- Rotate around origin by $\theta$ (Orthogonal)

$$M = \begin{vmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{vmatrix}$$

(Above, $\theta = 90^\circ$)
2D Transformations

• Flip over y axis

\[
M = \begin{bmatrix}
-1 & 0 \\
0 & 1 \\
\end{bmatrix}
\]

(Orthogonal)

Flip over x axis is ?
2D Transformations

- Shear along x axis

\[
M = \begin{bmatrix}
1 & a \\
0 & 1 \\
\end{bmatrix}
\]

Shear along y axis is ?
2D Transformations

• Translation \( (P_{\text{new}} = P + T) \)
Homogenous Coordinates

- Represent 2D points by 3D vectors
- $(x, y) \rightarrow (x, y, 1)$
- Now a multitude of 3D points $(x, y, W)$ represent the same 2D point, $(x/W, y/W, 1)$
- Represent 2D transforms with 3 by 3 matrices
- Can now do translations
- Homogenous coordinates have other uses/advantages (later)
2D Translation in H.C.

\[ \mathbf{P}_{\text{new}} = \mathbf{P} + \mathbf{T} \]

\[ (x', y') = (x, y) + (t_x, t_y) \]

\[ \mathbf{M} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \]
2D Scale in H.C.

\[ M = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \]
2D Rotation in H.C.

\[
M = \begin{bmatrix}
\cos \theta & -\sin \theta & 0 \\
\sin \theta & \cos \theta & 0 \\
0 & 0 & 1 \\
\end{bmatrix}
\]
Composition of Transformations (§5.4)

- If we use one matrix, $M_1$ for one transform and another matrix, $M_2$ for a second transform, then the matrix for the first transform followed by the second transform is simply $M_2 M_1$
- This generalizes to any number of transforms
- Computing the combined matrix first can save lots of computation
Composition Example

- Matrix for rotation about a point, $P$
- Problem--we only know how to rotate about the origin.
Composition Example

• Matrix for rotation about a point, P
• Problem--we only know how to rotate about the origin.
• Solution--translate to origin, rotate, and translate back
2D transformations (continued)

- The transformations discussed so far are invertable (why?). What are the inverses?
2D viewing

- Three coordinate systems are common in graphics
  - World coordinates or modeling coordinates - where the model is defined (meters, miles, etc.)
  - Normalized device coordinates; usually (0-1) in each variable.
  - Device coordinates: the actual coordinates of the pixels on the frame-buffer or the printer
- Need to construct transformations between coordinate systems
- Terminology:
  - window = region on drawing that will be displayed (rectangle)
  - viewport = region in NDC’s/DC’s where this rectangle is displayed (often simply entire screen).
Element in modelling coordinates

Transform into Normalised Device Coordinates

Transform into Device Coordinates

Clip

Draw
- View this as a sequence of transformations in homogenous coords, then determine each element in closed form.
- Compute numerically from point correspondences.