Administrative

Lecture notes should be accessible from anywhere with alternative URL.

Let the TA have a crack at assignment problem--send E-mail to him (mingde @ cs.arizona.edu) and cc me.

Next term: There will be a grad course in graphics which will focus on advance topics--open to undergrads.
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

Polygon to clip

Clip window
Sutherland-Hodgeman polygon clip

Clip window

Polygon to clip

Clip entire polygon against one edge
Sutherland-Hodgeman polygon clip

Polygon to clip

Clip window

Then clip it against the next
Sutherland-Hodgeman polygon clip

Polygon to clip

Then 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
generate edge goes from out to out     ==> emit nothing
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
Now have

polygon edge crosses clip edge going from out to in  ==> emit crossing,
ext vertex
polygon edge crosses clip edge going from in to out ==> emit crossing
polygon edge goes from out to out                 ==> emit nothing
Clip against next edge

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
Now have

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
Clipping against next edge (right) gives

- 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
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
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.

- Options:
  - modify algorithm (Weiler Atherton)
  - write polygon fill carefully; but note extra edges, etc.
for clockwise polygon:

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

- Clipping polygon, line against concave clip region is significantly harder. In both cases, efficient algorithms are known.
- Although everything is in terms of polygons/lines clipped against lines in 2D, all - except Nicholl-Lee-Nicholl - will work in 3D against convex regions.
- This is because the central issue in each algorithm is the inside outside decision.

- Inside-outside decisions can be made for lines in 2D, planes in 3D. e.g testing x>=0
- Hence, all (except N-L-N) can be used to clip:
  - lines against 3D convex regions (e.g. cubes) (CS, LB)
  - polygons against 3D convex regions (e.g. cubes) (SH, WA)
- NLN could work in 3D, but the number of cases increases too much to be practical.
2D Transformations (§5.2)

Have a look at §5.1--it reviews math that we will be using throughout the course.
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 $k$

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

(k = 2 in the example)
2D Transformations

- Scale by a factor of \((S_x, S_y)\)

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

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

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

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

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

• Flip over y axis (Orthogonal)

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

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 \( (P_{\text{new}} = P + T) \)

\[ M = ? \]
Homogenous Coordinates (§5.3)

- 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.

\[ P_{\text{new}} = P + T \]

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

\[
M = \begin{bmatrix}
1 & 0 & t_x \\
0 & 1 & t_y \\
0 & 0 & 1 \\
\end{bmatrix}
\]
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.
• Solution--translate to origin, rotate, and translate back (see §5.4)