1996-11-02 Bo Nordin (bosse&cb.uu.se)

Lecture notes, computer graphics (F04) (Foley 5.2-5.6, 3.9, 3.11)

2D transformations (5.2)

Today we will start looking inside graphics packages:
Application -> procedural -> *functional* -> device -> hardware.
But in SRGP for instance, you have to do it yourself (SPHIGS has them...)
In the introduction about primitives I concluded by mentioning retained vs. immediate mode packages. Transformations are especially useful in retained mode.

Where do they enter i the pipeline:

immediate mode: set up transformation matrices (window-viewport) initially. Then all attrributes will be transformed.

Transformations need not be done on all points (only vertices), except for some curves

2D transformations (will assume a left-handed Cartesian coordinate system)

magnification: see detail, or minification, for overview
translation: rotation: animation, viweing scenes from different vantage points (symbols)

modelling transformations -- used in building a view
normalizing transformations
windowing
clipping

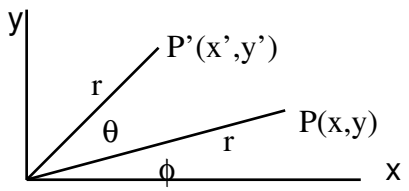
translation:

$x' = x + Tx \quad y' = y + Ty$

rotation:

$x' = x * \cos\theta - y * \sin\theta \quad y' = x * \sin\theta + y * \cos\theta$

only for rotation about the origin, angles measured counterclockwise. Derived from:



(if the object isn't at the origin its position will change as well)

$x = r * \cos\phi,\ y = r * \sin\phi$
$x' = r * \cos(\theta+\phi) = r * \cos\phi * \cos\theta - r * \sin\phi * \sin\theta$
$y' = r * \sin(\theta+\phi) = r * \cos\phi * \sin\theta + r * \sin\phi * \cos\theta$

scaling:

$x' = x * Sx \quad y' = y * Sy$

also relative the origin, i.e. changes position, too
can be used for a variety of purposes:
magnification

distorsion (simple ex, box at origin) (S_x 2, S_y = 0.5)
mirroring (simple ex, arrow) (S_x = -1, S_y = 1), (S_x=-1,S_y=-1), (S_x=1, S_y=-1)

concatenation:

reason for concatenation: fewer calculations

order is important: ex: 1/ 90 degree + 2/ translation (80,0)

1/ 2/
x' = 0 - y x' = x + 80
y' = x + 0 y' = y + 0

rotate + translate: x" = -y + 80 y" = x TRIANGLE EXAMPLE:

translate + rotate: x" = -y y" = x + 80

ex: rotation around an arbitrary point
translation, rotation, translation (DRAW)
show difference from ONLY rotation

Matrix representation (5.3-5.4)

Will assume you are familiar with matrix algebra
P - column vector (point) $T(d_x,d_y)$ --- vector (additive) : P' = T + P

R(θ) - matrix (multiplicative) $\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$ P' = R * P

$S(S_x, S_y)$ - matrix (multiplicative) $\begin{pmatrix} S_x & 0 \\ 0 & S_y \end{pmatrix}$ P' = S * P

use *homogeneous coordinates* -> consistent formulation for all 3 cases: multiplication

(x,y) - (x,y,W) (x,y,W) == (x',y',W') iff one is a multiple of the other
Triples used to represent points in 2D.

points are 3-element column vectors -> transformation matrices must be 3x3. Translation:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

concatentaion, compounding, composition -> multiplication of matrices

(Note: some books use premultiplication by row vectors! Transpose matrices to switch convention This means that the order of matrices is inverted in our convention, i.e. when we do a scaling followed by a translation we write P' = $T(d_x,d_y)$ * $S(S_x,S_y)$ * P
)

inversion of the matrices means inverted transformation

what can be accomplished with these transformations:

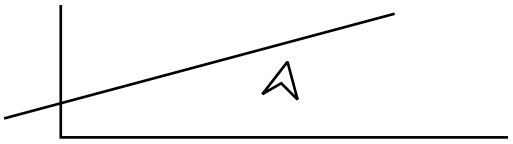
rotation, translation: rigid-body (preserves angles, length)

rotation, translation, scaling: affine transformation (preserves parallelism, not length angles)

shear transformation: $SHx = \begin{pmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ $SHy = \begin{pmatrix} 1 & 0 & 0 \\ b & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

all these are called *affine* transformations.

example: mirroring a polygon in an arbitrary line: y = Kx + L



Not easy to come up with a transformation that does it all: break down the problem into problems with well-known simple solutions.

Mirror in the Y-axis. S(-1,1)

Before we can do that we must translate and rotate

$$\text{translate: } \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & -L \\ 0 & 0 & 1 \end{pmatrix} \qquad \text{rotation } \begin{pmatrix} a & -b & 0 \\ b & a & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad a = \cos\phi = K/\sqrt{(1 + K*K)} \qquad b = \sin\phi = 1/\sqrt{(1 + K*K)}$$

$$\text{mirror: } \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad \text{rotate back: } \begin{pmatrix} a & b & 0 \\ +b & a & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad \sin(-\phi) = -\sin\phi \quad \& \quad \cos(-\phi) = \cos\phi$$

$$\text{translate back } \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & L \\ 0 & 0 & 1 \end{pmatrix}$$

Total transform $T^{-1}R^{-1}SRT$

Window-to-viewport transformation (5.5)

WHY ? Device-independence (400x400 4000*4000), user-defined scale, well-known units
We transform the result when we're ready.

world coordinates (sometimes a normalized device coordinates)
world-coordinate window (world coordinates) -> viewport (screen/device coordinates)

(this window has nothing to do with window-manager window !!)

Window: Viewport: (DRAW)

3-step transformation: translate (normalize), scale, translate:

1. Window translated to the origin in world coordinates
2. Scaled to the size of the viewport
3. Translated to the position of the viewport.

$$N_{(normalizaing.)}: \begin{pmatrix} 1 & 0 & -W_{xmin} \\ 0 & 1 & -W_{ymin} \\ 0 & 0 & 1 \end{pmatrix} \quad S_{(scaling)}: \begin{pmatrix} (VP_{xmax}-VP_{xmin})/(W_{xmax}-W_{xmin}) & 0 & 0 \\ 0 & (VP_{ymax}-VP_{ymin})/(W_{ymax}-W_{ymin}) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$T_{(translation)}: \begin{pmatrix} 1 & 0 & VP_{xmin} \\ 0 & 1 & VP_{ymin} \\ 0 & 0 & 1 \end{pmatrix}$$

Order: N -> S -> T, i.e. P = T * S * N * P (Question !)

The window/viewport transformation may distort images (change aspect ratio).

Window -> viewport transformation is often combined with clipping ->

2D clipping: (against rectangles, 3.9)

must be either first or last in the transformation pipeline (if we want to be able to compose transformations).
Clipping points is trivial: test against window limits.
Cohen-Sutherland (3.9.3) algorithm (finds the endpoints of visible segments)

step 1: for every line:: decide whether it's 1/ completely within the window, 2/ partially within the window or 3/ completely outside the window.

four-bit word; 1: $y > y_{max}$, 2: $y < y_{min}$, 3: $x > x_{max}$. 4: $x < x_{min}$

1001	1000	1010
0001	0000	0010
0101	0100	0110

when clipping the line $P_1(x_1, y_1)$ - $P_2(x_2, y_2)$ the code for both points are calculated.

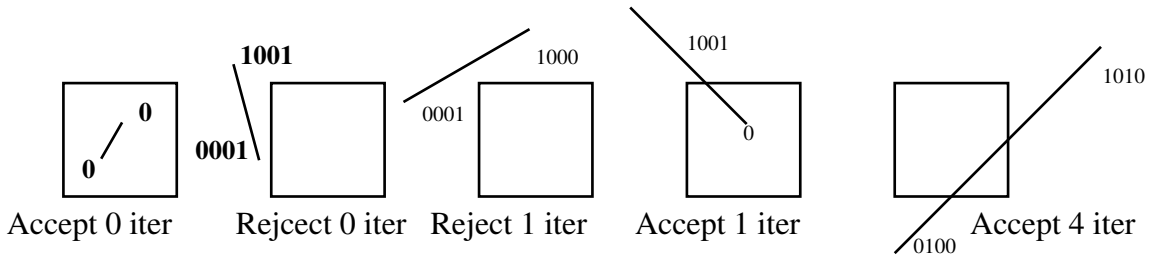
A/ if code(P_1) == code(P_2) == 0 : accept, visible line

B/ if code(P_1) AND code(P_2) |= 0: reject, invisible line

if neither A nor B is true, calculate the intersection between the line and the edge of the window

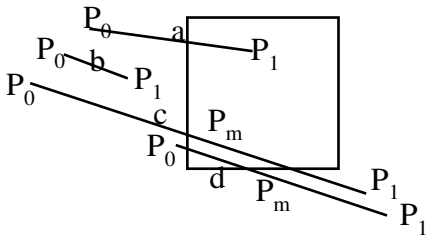
the algorithm in the book tests the edges in a fixed order -> sometimes unnecessary clips (external intersections):

ex: (trivial accept, trivial reject, simple clip, multiple clip):
testing order: top, bottom, left, right



Midpoint subdivision (uses Sutherland-Cohen, no multiplication, division):

Both endpoints are found in an identical process (can be done in parallel), the goal is to to find the visible point on the line segment that is farthest from the point. P_0 (in the example). Avoids direct computation. Application of binary search

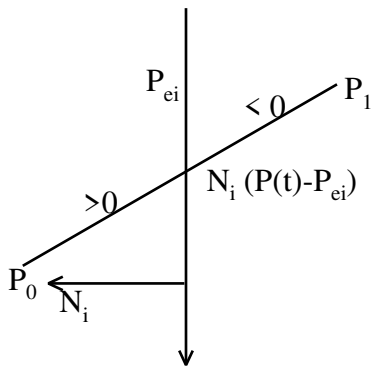


1. Test whether P_1 is visible. if so we have found the endpoint (case a)
2. else test whether $P_0 P_1$ can be trivailly rejected (case b)
3. else divide $P_0 P_1$ at its midpoint P_m. A guess!
   if $P_m P_1$ outside (case d): to 2. with $P_0 P_m$ else (case c) with $P_m P_1$.

The process may terminate in 1. or 2. or 3. (in which case there's only one point)
Division by 2 -> shift
An identical process looking at $P_1 P_0$

When clipping multi-line polygons it's a good idea to clip against the bounding box in order to reject/accept the complete polygon fast/simply.

Parametric line clipping: (Cyrus-Beck, 3.9.4)
can clip against an arbitrary polygon.
Parametric representaiton of a line: $P(t) = P_0 + t(P_1 - P_0)$ (will be used for curves, surfaces)
$t = 0$ at P_0 and $t = 1$ at P_1.
Finds the value of t for the points at which the parametric intersection of the lines segment for the point at which that segment intersects the infinite line on which the clip edge lies. Four values of t are calculated. Simple comparisons are used to determine which values of t correspond to actual intersections.



outward normal $N_i * (P(t) - P_{ei})$ (dot product <0 inside, =0 on line, > 0 outside

Solve for = 0:

substitute for P(t): $N_i * [P_0 + t(P_1 - P_0) - P_{ei})] = 0$

$$Ni * (P_0 - P_{ei}) + N_i t(P_1 - P_0)$$

solve for t:

$$t = \frac{N_i * (P_0 - P_{ei})}{-N_i * (P_1 - P_0)}$$

initially we must check that the nornal is nonzero and that $P_1 != P_0$
if the denominator is zero the lines are parallel

values outside [0,1] are discarded.

characterize intersections as PE (potentially entering) or PL (potentially leaving):
check the angle between P_0P_1 and N_i.

denominator < 0 -> PE (angle > 90)
denominator > 0 -> PL (angle < 90)

Four the 4 clip edges: get PE intersec with largest t and PL intersection with smallest t.

Discussion

C-S: +: most lines trivially rejected/accepted and cheap outcode test
-: repeated clipping
C-B +: when many segments to be clipped

Polygon clipping (3.11)

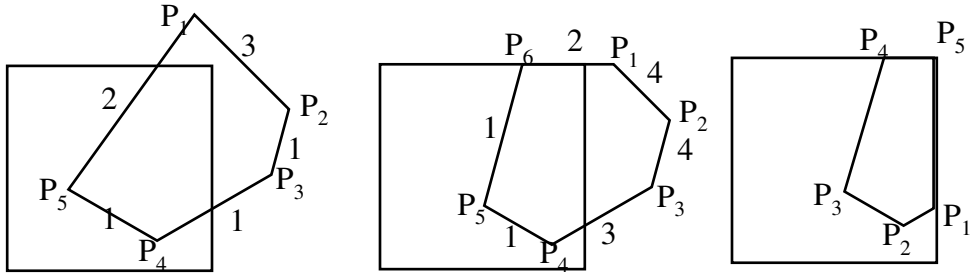
Examples, many different cases -> must often create new edges.
The Sutherland-Hodgman algorithm.
Breaks down the problem -clip a polygon against a single indefinite clip edge.

each call clips against one edge. input: vertices output: vertices

start with the line from the last vertex to the first.

four cases for each line: 1/ both inside: output both
2/ first in, 2nd out: output 1st, output intersection line/edge
3/ first out, second in: output intersection line/edge
4/ none inside: no output

ex:



Can be implemented in a pipeline:

call itself recursively. (no intermediate storage needed).

Viewing pipeline
Clipping can be placed either before or after other transformations.

LC -> [MODELLING TRANSF] ->WC [CLIP] [NORM TRANSF] -> SC

If we clip in screen coordinates all objects must be transformed, even invisible ones