1996-11-04 Bo Nordin (bosse&cb.uu.se)

Lecture notes, computer graphics (F05) (Foley 5.7-5.9, Foley 6)

Question about bug(s) in C implementation of Cohen & Sutherland.
Also mention the use of a bounding box in clipping

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
If many objects outside window: clip early

Other efficiency aspects (reduces from 9 mults + 6 addds to 4 mults + 4 adds)

$\begin{pmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{pmatrix}$ (rather fixed structure) (calculated once and saved)

$x' = x * r_{11} + y * r_{12} + t_x$
$y' = x * r_{21} + y * r_{22} + t_y$

Transformation as a change in coordinate

Transformations can be viewed as a change in coordinate system.
A transformation that transforms a set of points in a coordinate system is the inverse of the corresponding transformation to change the coordinate system in which a point is represented
Ex: simple translation in 2D.

3D transformations (5.7, 5.8 p 180-187)

We'll talk about wireframe models (surfaces are handled slightly differently).

We often need to transform normals: if M is transformation: $N' = (M^{-1})^T N$

Homogeneous coordinates *right-handed* coordinate system
x and y as usual but z coming out of the screen towards the viewer:
Axis of rotation direction of positive rotation (counterclockwise)

x	y to z
y	z to x
z	x to y

$$T = \begin{pmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad S = \begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$ (also inverted)

$$R_z = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad R_y = \begin{pmatrix} \cos\theta & \sin\theta & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & \cos\theta & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad R_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 1 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

(inversion of R_i : $\sin\theta$ changes sign)
(warning about different conventions!!) (Mention shear matrices ??)

Order important:
Ex.: Scale: translate fix point to origin, scale, inverse translation
Ex: Rotate an object θ degrees about an axis through P_1 and P_2

a/ Translate P_1 to the origin ($d = P_1$)
b/ Rotate about the y axis to move P_2 to the yz-plane ($v^2 = d_x^2 + d_z^2$)

c/ Rotate about the x axis to move P_2 to the z axis
d/ Rotate the object the requested angle about the z axis
Inverse rotate around x (invert sign of angle)
Inverse rotation about y
Inverse translation

$$T = \begin{pmatrix} 1 & 0 & 0 & -P_{1x} \\ 0 & 1 & 0 & -P_{1y} \\ 0 & 0 & 1 & -P_{1z} \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad R_y = \begin{pmatrix} d_z/v & 0 & -d_x/v & 0 \\ 0 & 1 & 0 & 0 \\ d_x/v & 0 & d_z/v & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad R_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & v/|d| & -d_y/|d| & 0 \\ 0 & d_y/|d| & v/|d| & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad R = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Complete transformation: $T^{-1}R_y^{-1}R_x^{-1}R(\theta)R_xR_yT$

3D transformation of a point: 16 mults. 12 adds -> 12, 9 (no transl): 9, 6
In practice, we will gain by translating points outside (except if we must translate back).

Synthetic camera (6.1 p 193-195)
metaphor, can be moved to any location, oriented any way we wish:
create 3D -> 2D.
animation: moving the camera or moving objects: transformations.
I'll show how the basic 3 (5) transformations can be used to accomplish this.

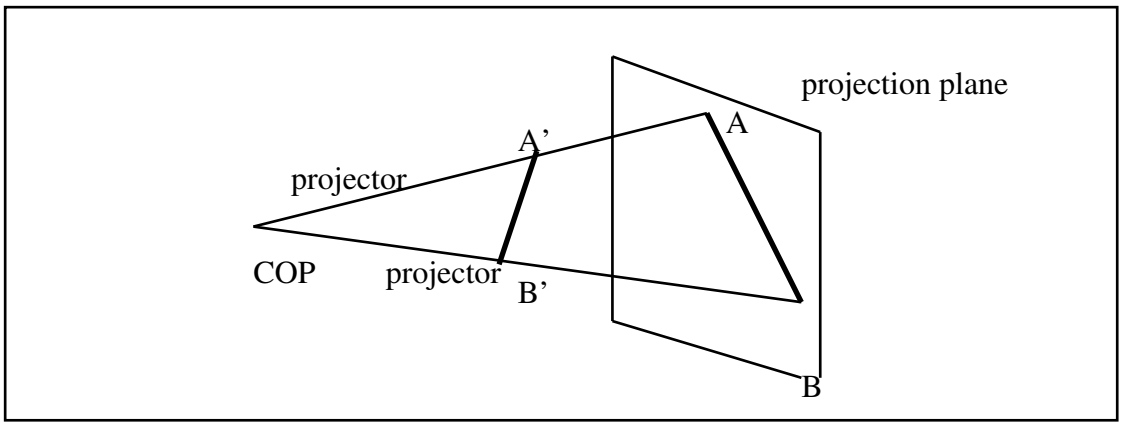
3D WC ----------> clipped WC ----------> 2D DC
clip against view volume
project
transform into viewport in 2D DC

1/ specification of projection type: parallel, perspective
2/ specification of viewing parameters: viewing / eye coordinate system: eye pos,view plane
3/ 3D clipping: view volume (ignore parts behind us or far away)
4/ projection, display: window -> viewport

Projection (6.2 195-201)
normally onto a flat surface (cartographic, eye)
Assume that the projection plane is in the xy-plane (transform otherwise)

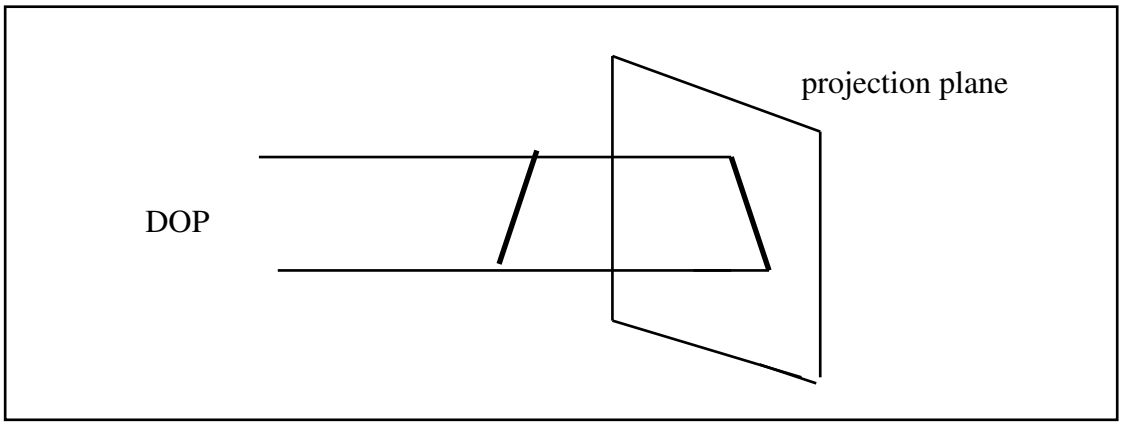
Let projection rays (*projectors*). emanting from a c*entre of projection* (COP), pass through each point of the object, and intersect a *projection plane* to form the projection (the image)

Perspective:



Perspective *foreshortening*
Perspective realistic, but distort the shape of objects
1-point, 2-point, 3-point depending in # of *vanishing points* (gränspunkter).
n-point: the projection plane intersects n coordinate axes

Parallel (centre of projection infinitely far away):

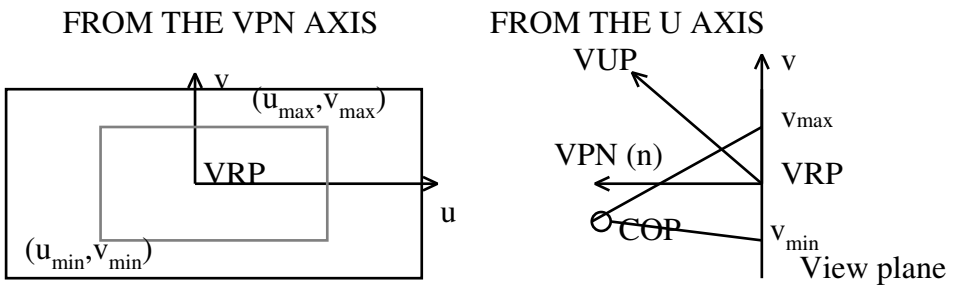


Direction of Projection (COP is (x,y,z,1), DOP is (x,y,z,1) - (x',y',z',1) = (a,b,c,0), directions and points at infinity correspond in a natural way.
Parallel less realistic, easier to draw
ortographic:projectors parallel to the normal to the projection plane
oblique: projectors ain't parallel to the ...

Commonly the projection plane is perpendicular to 1 of the axes (the axis is the proj plane): *top, front, side*.

A 3D view (6.3 p 201-206)
projection & view volume is all we really need

projection plane - *view plane* (may be in front of or behind the objects).
view reference point (VRP)
view-plane nornal (VPN)
a *window* on the view plane
we need to be able to specify the window (in what coordinate system?)
viewing-reference system (VRC)
1 axis ready (origin at VRP) VPN is one axis (n axis)
2nd axis *view-up vector* (VUP) (v axis: projection of VUP on the plane). The u-axis is then defined so that u,v, and n form a right-handed coordinate system. VRP, VPN, and VUP are defined in the world coordinate system.



The centre of the window (CW) does not have to coincide with the VRP.

The centre of the projection is also called the projection reference point (PRP) in order to be applicable to the 2 projection types (if parallel DOP is (PRP-CW))

The view volume:
perspective: pyramid with apex at PRP and edges passing through the corners of the window.
parallel: infinite parallelepiped

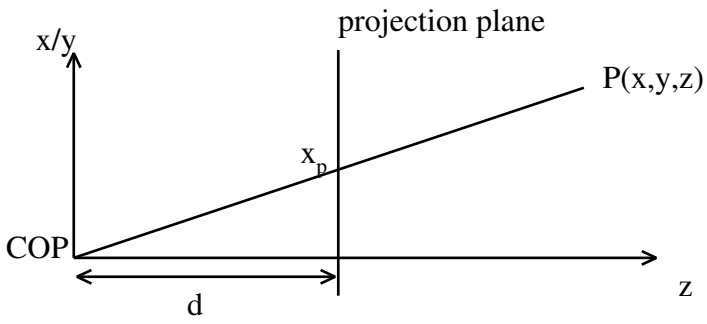
We may want the volume to be finite, i.e. limit the volume in z. *front* and *back clipping planes*. front and back distance relative to VRP along VPN.
Why clip in z: data reduction,dynamic modification, distance-object (near-obj) removal

How do we map the view volume -> display surface.
normalized projection coordinates (NPC, 0-1) -> *3D viewport* (within the unit cube 0-1)
the plane z = 1 in this cube is mapped to display

view orientation matrix (VRP, VPN, VUP) WC -> VRC i.e. takes the u, v, and n axes into the x, y, and z axes.
view mapping matrix: PRP, u_{min}, u_{max}, v_{min}, v_{max}, F, and B along & the 3D viewport

Read and understand chapter 6.4 (examples of viewing) (OH if time allows)

Planar geometric projections (6.5 p 213-215)
Perspective (NOTE: looks along the positive x axis):



the figure assumes that the projection plane normal to the z axis
plane at location z = d.
x_p /d = x / z (multiply by d) --> x_p = (x*d) / z

(x_p,y_p,z_p) = (x / (z/d), y / (z/d), d)

Matrix form:
$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{pmatrix}$$

Alternative:
$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1/d & 1 \end{pmatrix}$$
(1st row:|= view plane cuts x axis)

Multiplying $[x\ y\ z\ 1]^T$ with M above yields $[x\ y\ z\ z/d]^T$
divide by z/d and return to Cartesian coordinates: $[x/(z/d), y/(z/d), d]^T$
Alternative (projection plane at z=0 and COP at z=-d.
As above (using similarity of triangles): $x_p / d = x / (z+d)$
$x_p = x / ((x/d)+1)$, $y_p = y / ((z/d) + 1)$

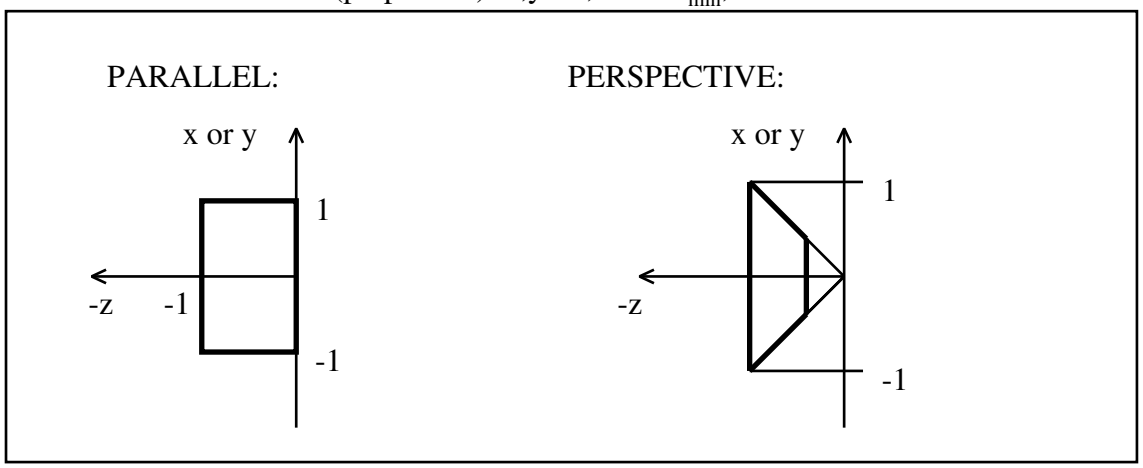
Parallel straightforward (assuming projection plane z =0): $x_p = x$, $y_p = y$, $z = 0$

$$\text{Matrix: } \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Note that as d -> infinity in alternative above -> parallel (as expected)

3D Viewing transformations (6.6.1 parallel 217-222, 6.6.2 perspective 222-227)

clip against the volume by calculating intersections ?? NO
instead we transform to: (parallel) x,y = -1 -> 1, z: 0 - -1
(perpective): x,y: -z,z z: $-z_{min}$, -1



find normalizing transformations that transform any view volume to these, then clip and project as described

perspective: 1/ translate VRP (view reference point) to origin $T(-VRP)$
2/ rotate VRC (view ref. coo): VPN (n) ->z, u -> x, and v -> y R
3/ translate COP (PRP) to the origin $T(-PRP)$
4/ shear: centre line of the view volume -> z axis SH_{per}
5/ scale to the canonical view volume S_{per}

$N_{per} = S_{per}SH_{per}T(-PRP)RT(-VRP)$

parallel: 1/ and 2/ as before
3/ shear: the DOP parallel to the z axis (for oblique) SH_{par}
4/ translate and scale into the canonical volume T_{par}, S_{par}

$N_{par} = S_{par}T_{par}SH_{par}RT(-VRP)$

Clipping in 3D (6.6.3 p 227-229)

extensions of 2D methods (just 2 more bits in Cohen-Sutherland)
parallel case: 1: y > 1; 2: y < -1; 3: x > 1; 4: x < -1; 5: z < -1; 6: z > 0
perspective: 1: y > -z; 2: y < z; 3: x > -z; 4: x < z; 5: z < -1 (behind); 6: z > $-z_{min}$ (in front)
(unit slopes make the intersection calculations simple)

Clipping in hom. coordinates: same algorithm for parallel & perspective

Window to viewport transformation (6.6.5 p 231-233)
Coordinates are in the intervals ([-1,1], [-1,1], [-1,0]).
Here we assume parallel-projection volume, If perspective use the perspective projection.

1 Translate view volume to the 3D viewport contained in ([0,1], [0,1], [0,1])
T(1,1,1) moves (-1,-1,-1) to (0,0,0)
2. Scale into the size of the 3D viewport:
S($(x_{vmax} - x_{vmin})/2$, $(y_{vmax}-y_{vmin})/2$, $z_{vmax} - z_{vmin}$))
3. Translate to the lower-left corner of the viewport (z component useful in hidden lines)
$T(x_{vmin},y_{vmin},z_{vmin})$

Overview of the 3D coordinate systems (6.7 p 234-235)

object (modelling) coordinate systems (often a hierarchy windows/doors,house, street)
<modeling transformation>
3D world coordinate system (problem coordinate, application-coordinate)
<view orientation matrix> R*T(-VRP) move to VRC
view reference coordinates (VRC) (eye, camera)
<view mapping matrix> $M * S_{per} * SH_{par} * T(-PRP) / S_{par} * T_{par} * SH_{par}$
normalized projection coordinates (3D screen coordinates).
<clip, transform into 2D coordinates>
2D device coordinates