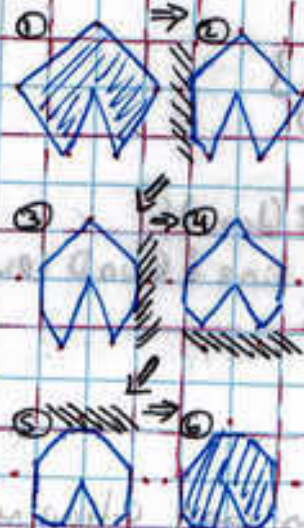


20 Sutherland - Hodgeman Polygon Clipping

- 1) Clip correctly a polygon by processing the whole polygon boundary as a whole against each window edge.
- 2) This is accomplished by processing all polygon vertices against each clip rectangle boundary in turn.
- 3) Clip first side should be: clip left, clip right, clip bottom and then clip top.
- 4) For each step, a new sequence output vertices is generated and passed to the next window boundary clipper.



- 5) Four cases when processing vertices in sequence around a polygon:

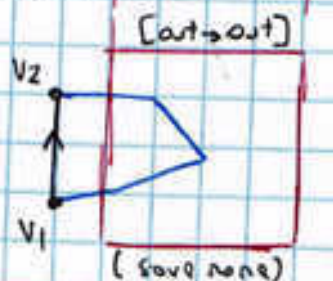
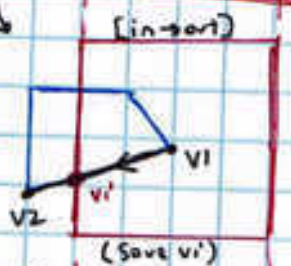
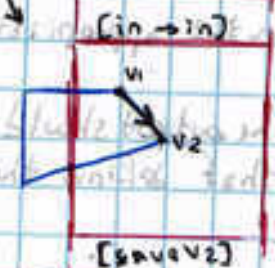
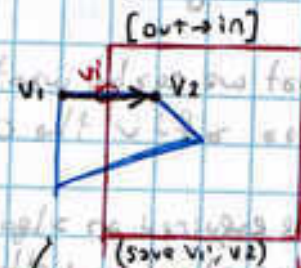
a) As each pair of adjacent polygon vertices is passed to a window boundary clipper test:

i) If the first vertex is outside the window boundary and the second vertex is inside
Then: both the intersection point of the polygon edge w/ the window boundary and the second vertex are added to the output vertex list;

ii) If both input vertices are inside the window boundary, only the second vertex is added to the output vertex list

iii) If the first vertex is inside the window boundary and the second vertex is outside, only the edge intersection w/ the window boundary v_2 is added to the output vertex list.

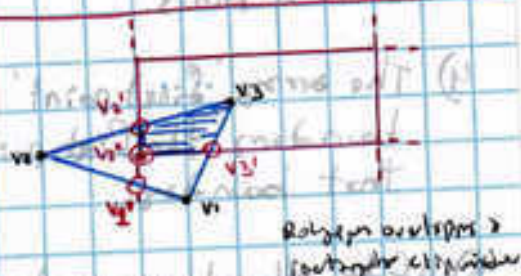
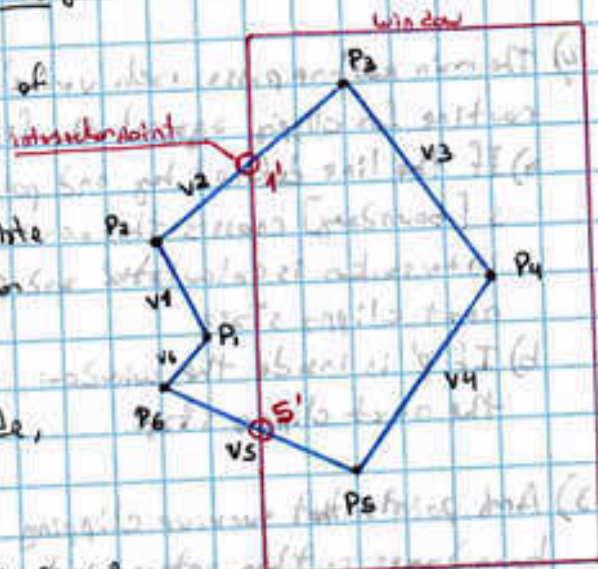
iv) If both input vertices are outside the window boundary, nothing is added to the output list.



Sutherland-Hodgman Polygon Clipping (Continued)

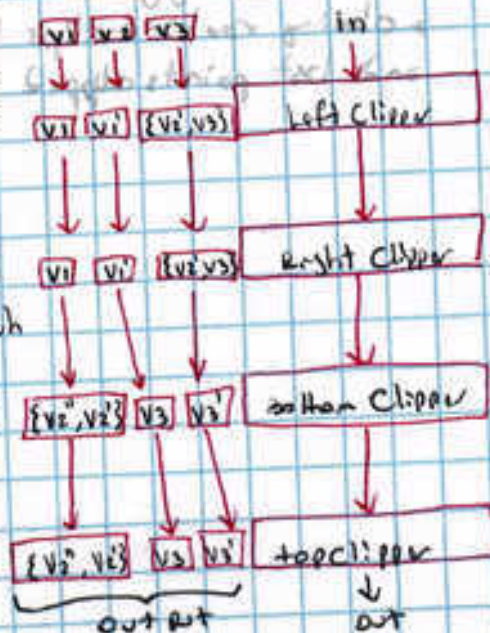
Clipping area against Left window Boundary:

- 1) Vertices 1 and 2 are found to be outside of the boundary.
- 2) Moving along to vertex 3 (which is inside), we calculate the intersection and save both the intersection point (I') and vertex 3.
- 3) Vertices 4 and 5 are determined to be inside, and they also are saved.
- 4) Sixth and seventh (final) vertex are outside, so we find and save the intersection point (S').
- 5) Using the five saved points: I' , P_3 , P_4 , P_5 , S' we should repeat the algorithm for the next window boundary.



Implementing the algorithm

- 1) We are required to set up storage for an output list of vertices as a polygon is clipped against each window boundary.
- 2) We can eliminate the intermediate output vertex list by simply clipping individual vertices at each step and passing the clipped vertices on to the next boundary clipper. (with protection scale bars)
- 3) A point is added to the output vertex list only after it has been determined to be inside or on a window boundary by all four boundary clippers.



22 Piper line Clipping Approach

- 1) An array 'S' record next point that was clipped for each clip-window boundary.
- 2) The main routine passes each vertex 'p' to the clip point routine for clipping against the first window boundary.
 - a) If the line defined by end points 'p' and 's [boundary]' crosses this window boundary, the intersection is calculated and passed to the next clipping stage.
 - b) If 'p' is inside the window, it is passed to the next clipping stage.
- 3) Any points that survive clipping against all window boundaries is then entered into the array output array of points.
- 4) The array 'first point' stores for each window boundary the first point clipped against that boundary.
- 5) After all polygon vertices have been processed, a closing routine clips lines defined by the first and last points clipped against each boundary.

Sutherland - Hodgeman Polygon Clipping Code

```
typedef enum {Left, Right, Bottom, Top} Edge;
```

```
#define N_EDGE 4
```

```
int inside(wcPt2 p, Edge b, dcPt wMin, dcPt wMax){  
    switch(b){  
        case Left:  
            if (p.x < wMin.x) return False; break;  
        case Right: if (p.x > wMax.x) return False; break;  
        case Bottom: if (p.y < wMin.y) return False; break;  
        case Top: if (p.y > wMax.y) return False; break;  
    }  
    return True;  
}
```

```
int class(wcPt2 p1, wcPt2 p2, Edge b, dcPt wMin, dcPt wMax){  
    if (inside(p1, b, wMin, wMax) == inside(p2, b, wMin, wMax))  
        return False;  
    else  
        return True;  
}
```

```
wcPt2 intersect(wcPt2 p1, wcPt2 p2, Edge b, dcPt wMin, dcPt wMax){  
    wcPt2 ipt;  
    float m;  
    if (p1.x != p2.x)  
        m = (p1.y - p2.y) / (p2.y - p1.y) / (p1.x - p2.x);  
    switch(b){  
        case Left:  
            ipt.x = wMin.x;  
            ipt.y = p2.y + (wMin.x - p2.x) * m;  
            break;  
        case Right:  
            ipt.x = wMax.x;  
            ipt.y = p2.y + (wMax.x - p2.x) * m;  
            break;  
        case Bottom:  
            ipt.y = wMin.y;  
            if (p1.x != p2.x) ipt.x = p2.x + (wMin.y - p2.y) / m;  
            else ipt.x = p2.x;  
            break;  
    }
```


case top:

$ipt.y = wMax.y;$

if ($p1.x \neq p2.x$) $ipt.x = p2.x + (wMax.y - p2.y) / m;$

else $ipt.x = p2.x$

break;

}

return(ipt);

}

void clipPoint (wClip2 p, Edge b, dClip wMin, dClip wMax, wClip2 *pOut, int *cnt, wClip2 *first, wClip2 *s)

wClip2 ipt;

/* If no prev. points exist on this edge, save this point. */

if (!first[b]) first[b] = ipt;

else

/* Previous point exists. If 'p' and previous point cross edge, find intersection.

Clip against next boundary, if any.

If no more edges, add intersection to output list. */

if (cross(p, s[b], b, wMin, wMax))

ipt = intersect(p, s[b], b, wMin, wMax);

if (b < Top) clipPoint(ipt, b+1, wMin, wMax, pOut, cnt, first, s);

else {post[*cnt] = ipt; (*cnt)++;}

}

/* save 'p' as most recent point for this edge */

s[b] = p;

/* For all, if point is 'inside' proceed to next clip edge, if any */

if (b < Top)

clipPoint(ipt, b+1, wMin, wMax, pOut, cnt, first, s);

else {post[*cnt] = ipt; (*cnt)++;}

}

}

Geometric Transformations in 3D Space

25

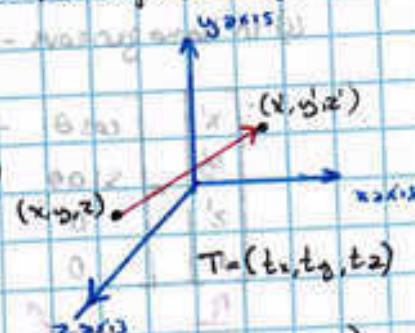
1. At difference of 2D rotations in the xy plane where it is not 2d to consider only rotations about axes perpendicular to the xy plane, in the 3D space, we can select any spatial orientation as the composite of 3 rotations, one for each of the 3 cartesian axes.

2. We can set up the orientation of a rotation axis and the required angle.

3. 3D Translation

a. $P \rightarrow P'$ $P = (x, y, z) \rightarrow P' = (x', y', z')$

$$x' = x + tx \quad y' = y + ty \quad z' = z + tz$$



b. Matrix representation (homogeneous coordinate w/ four column matrix)

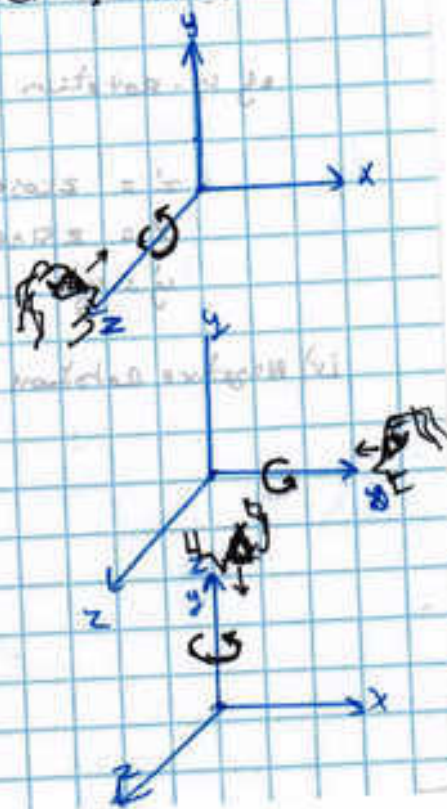
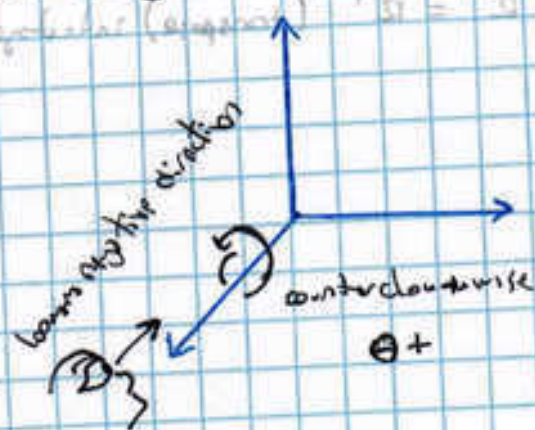
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$P' = T \cdot P$$

4. 3D Rotation

a. Even do we can rotate in any axis, we are going to rotate axes to handle those that are parallel to the Cartesian coordinates. Since it is easier

b. By convention, positive rotation angles produce positive counter clockwise rotations about a coordinate axis (assuming we are looking in the negative direction along the coordinate axis)



26 c. 3D Coordinate-Axis Rotations:

i) 2D z-axis rotation are extended to 3D:

$$\begin{cases} x' = x \cos \theta - y \sin \theta \\ y' = x \sin \theta + y \cos \theta \\ z' = z \end{cases} \quad \left. \begin{array}{l} \theta \text{ specifies rotation angle} \\ \text{about the } z\text{-axis} \end{array} \right\}$$

ii) In homogeneous-coordinate form

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$P' = R_z(\theta) \cdot P$$

iii) Translations are obtained if a cyclic permutation of the coordinates parameters x, y, z

eg: x-rotation

$$x \rightarrow y \rightarrow z \rightarrow x$$

$$\begin{cases} y' = y \cos \theta - z \sin \theta \\ z' = y \sin \theta + z \cos \theta \\ x' = x \end{cases}$$

eg: y-rotation

$$\begin{cases} z' = z \cos \theta - x \sin \theta \\ x' = z \sin \theta + x \cos \theta \\ y' = y \end{cases}$$

iv) Negative rotations is $R^{-1} = R^T$ (transpose) interchange row and columns

Geometric Transformations in 3D space (Continued)

27

4. (continued)

d. General 3D Rotations

- i. A rotation matrix for any axis that does not coincide w/ a coordinate axis can be setup as a composite transformation: (by combining translations and coordinate axis rotation)

- ii. In the case an object is to be rotated about an axis that is parallel to one of the coordinate axis:

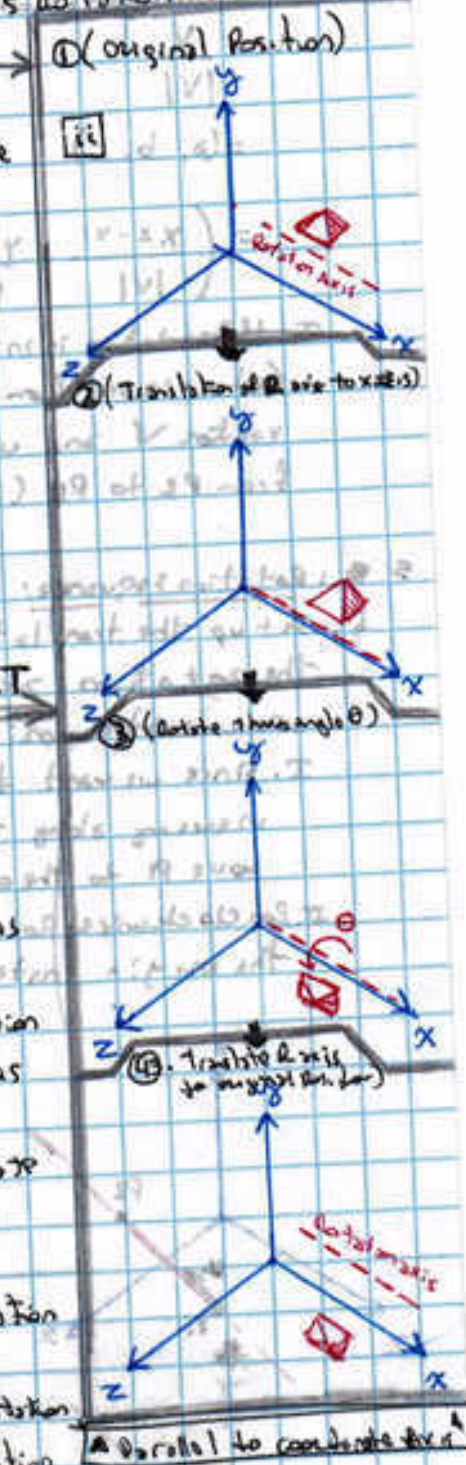
- I. Translate object in such way that the rotation axis coincides with the parallel coordinate axis ① → ②
- II. Perform the rotation on axis ② → ③
- III. Translate the object so the rotation axis is moved back to its original position ③ → ④

IV. Sequence: $P' = T^{-1} \cdot R_x(\theta) \cdot T \cdot P$

- iii. where the composite rotation matrix for the transformation is $R(\theta) = T^{-1} R_x(\theta) T$

- iii. In the case an object is to be rotated about an axis that is not parallel to one of the coordinate axis:

- I. Translate the object so that the axis passes through the coordinate origin. ① → ②
- II. Rotate object so that the axis of rotation coincides with one of the coordinate axes ② → ③
- III. Perform rotation about selected coordinate axis ③ → ④
- IV. Apply Inverse rotations to bring the rotation axis back to its original orientation ④ → ⑤
- V. Apply the inverse translation to bring the rotation axis back to its original spatial position ⑤ → ⑥



iv. The z axis is often a convenient choice.

v. The components of the rotation - axis vector are:

$$V = P_2 - P_1$$

$$= (x_2 - x_1, y_2 - y_1, z_2 - z_1)$$

vi. The unit rotation - axis vector u is:

$$u = \frac{V}{|V|}$$

$$= (a, b, c)$$

$$= \left(\frac{x_2 - x_1}{|V|}, \frac{y_2 - y_1}{|V|}, \frac{z_2 - z_1}{|V|} \right)$$

I. If the rotation is in the opposite direction (counterwise)

(when viewed from P_2 to P_1), then reverse axis

vector V and unit vector u so its point from P_2 to P_1 (instead of P_1 to P_2 as before)

5. i. Rotation sequence:

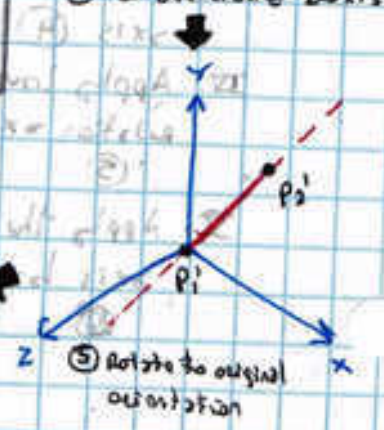
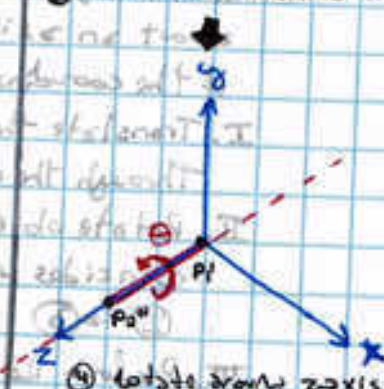
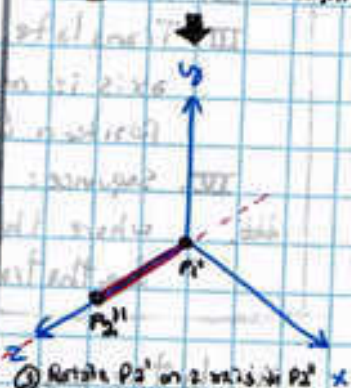
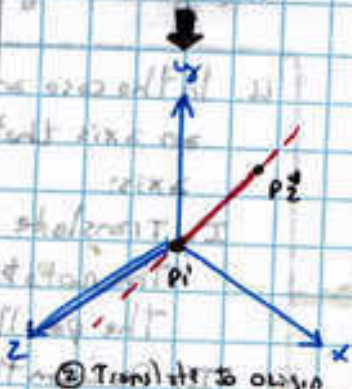
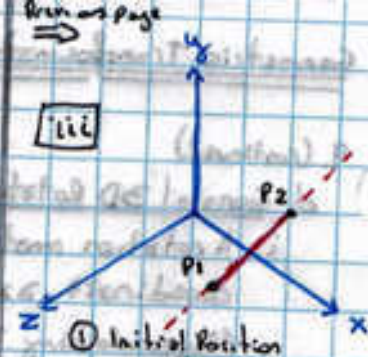
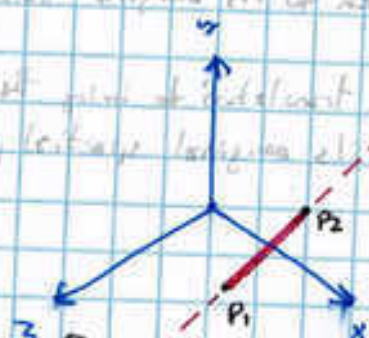
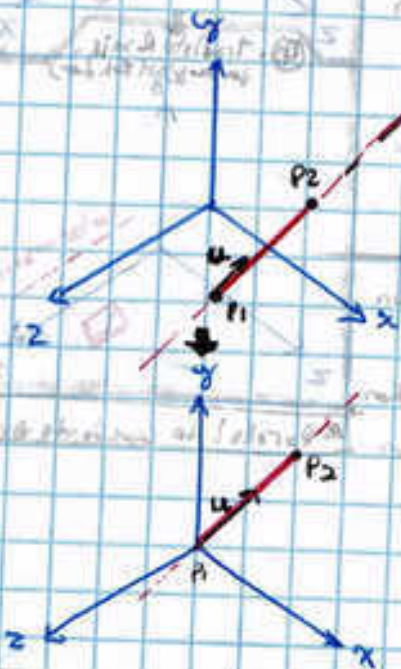
i. set up the translation matrix that repositions the rotation axis so it passes through the coordinate origin

I. since we want to counter clockwise when viewing along the axis from $P_2 \rightarrow P_1$ move P_1 to the origin

II. for clockwise rotation we would move P_2 to the origin instead

$$T = \begin{bmatrix} 1 & 0 & 0 & -x_1 \\ 0 & 1 & 0 & -y_1 \\ 0 & 0 & 1 & -z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(repositions the rotation axis and the object)



⑥ Translate rotation axis to original position

Geometric Transformation in 3D space (Continued)

5. (Continue 2)

ii. Formulate the transformations that will position

Example:

I. Rotate about x-axis

II. Then rotate about y-axis.

iii. The x-axis rotation gets vector u into the xy -plane and the y-axis rotation swings u around the z-axis

iv. Since sine and cosine functions are used for the rotation calculations, we can obtain elements of two rotation matrices

I. A vector dot product can be used to determine

The cosine term.

II. A vector cross product can be used to calculate the sine term.

v. To get a vector into the yz -plane

v. To get a vector into the xz -plane, we need to establish the transformation matrix for rotation around the x-axis.

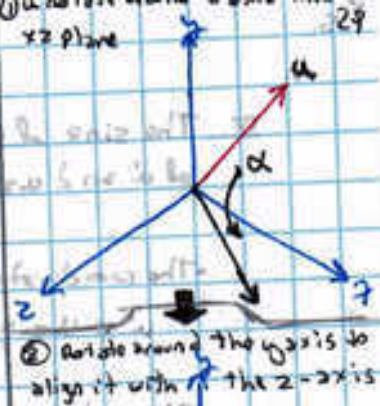
I This rotation angle is the angle between the projection of u in the yz plane and the positive z-axis.

II. we represent u in the yz plane as the vector $u' = (0, b, c)$

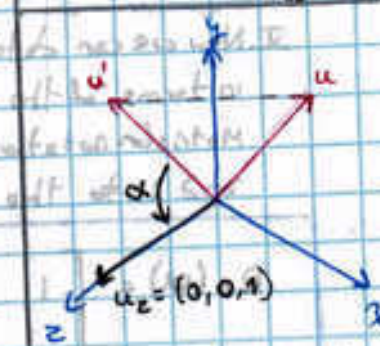
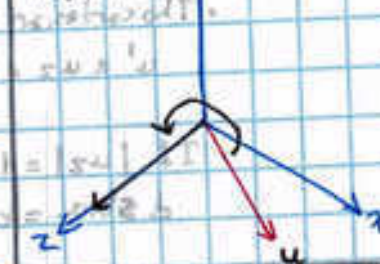
III. The rotation angle α can be determined from the dot product of u' and the unit vector u_z along the z-axis.

$$\cos \alpha = \frac{u' \cdot u_z}{|u'| |u_z|} = \frac{c}{d}$$

$$\text{magnitude of } u' = d = \sqrt{b^2 + c^2}$$



② Rotate around the y-axis to align it with the z-axis



① Rotation of u around x-axis into xz plane is done by:

① Rotating u' (projection of u into the yz plane) through the angle α onto the z-axis

IV. The sine of α can be obtained from the cross product of u' and u_z .

The coordinate-independent form: (cross product)

$$u' \times u_z = u_s |u'| |u_z| \sin \alpha$$

The Cartesian Form: (cross product)

$$u' \times u_z = u_s b$$

If $|u_z| = 1$ and $|u'| = d$

$$d \sin \alpha = b$$

V. Now we can determine the values of $\cos \alpha$ and $\sin \alpha$

in terms of the components of vector u' .

Matrix notation of this vector around the x-axis and into the xz plane

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{c}{d} & -\frac{b}{d} & 0 \\ 0 & \frac{b}{d} & \frac{c}{d} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha & 0 \\ 0 & \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

vi. Next, determine the matrix that will swing the unit vector in the xz plane counterclockwise around the y-axis onto the positive z axis.

I. The rotation of the unit vector in the xz plane

(from the rotation around x-axis) results in u'' vector

II. u'' has the value of α limits x-component

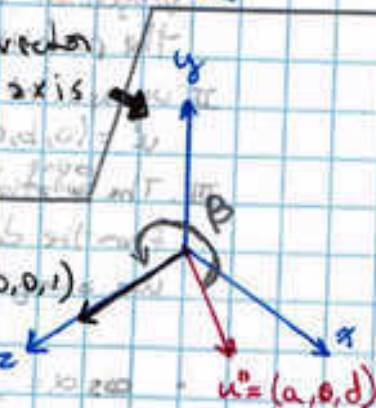
(since rotation around x-axis leave x-component unchanged)

$$a = -\sin \beta$$

III. u'' 's z component is d (magnitude of u') since u' has been rotated around onto z axis.

$$\cos \alpha = \frac{u'' \cdot u_z}{|u''| |u_z|} = d$$

Rotation of u'' (u' after rotation into the xz plane) around the y-axis. Positive rotation angle β align u'' w/ vector u_z



Geometric transformation in 3D spaces.

5. (continued)

vi. (continued)

IV. we can determine the cosine of rotation angle β from the dot product of unit vectors u'' and u_z

$$\cos \beta = \frac{u'' \cdot u_z}{|u''| \cdot |u_z|} = d$$

V. since $|u_z| = |u''| = 1$, by comparing the coordinates in the product form of the cross product

$$u'' \times u_z = u_y |u''| |u_z| \sin \beta$$

$$\text{the cross term: } u'' \times u_z = u_y \cdot (-a)$$

$$\text{we find: } \sin \beta = -a$$

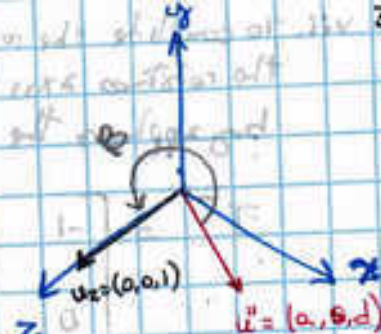
VI. The matrix transformation of u'' about the y-axis is:

$$R_y(\beta) = \begin{bmatrix} d & 0 & 0 & -a & 0 \\ 0 & 1 & 0 & 0 & 0 \\ a & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos \beta & 0 & \sin \beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \beta & 0 & \cos \beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

vi. with T , $R_x(\alpha)$, and $R_y(\beta)$, we have aligned the rotation axis w/ the positive z-axis

II) Rotation angle θ can now be applied as a rotation about the z-axis.

$$R_z(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Rotation of u'' (u'' after rotation into the xz plane) around the y-axis. Positive rotation angle β aligns u'' w/ vector u_z .

32 vii. To complete the rotation given axis, we have to transform the rotation axis back to its original position by applying the inverse transformation of T , $R_x(\alpha)$, $R_y(\beta)$.

$$T^{-1} = \begin{bmatrix} -1 & 0 & 0 & +x_1 \\ 0 & 1 & 0 & +y_1 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_x^{-1}(\alpha) = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -\frac{c}{a} & +\frac{b}{a} & 0 \\ 0 & +\frac{b}{a} & +\frac{c}{a} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -\cos \alpha & +\sin \alpha & 0 \\ 0 & +\sin \alpha & +\cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_y^{-1}(\beta) = \begin{bmatrix} -d & 0 & 0 & +a & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ a & 0 & 0 & d & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -\cos \beta & 0 & 0 & -\sin \beta & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ +\sin \beta & 0 & 0 & -\cos \beta & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

viii. The expressed composition of these, even in 2 individual transformations

$$R(\theta) = T^{-1} \cdot R_x^{-1}(\alpha) \cdot R_y^{-1}(\beta) \cdot R_z(\theta) \cdot R_y(\beta) \cdot R_x(\alpha) \cdot T$$

$$= \begin{bmatrix} \cos \theta & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Camera

1. Transformations base on global coordinate system (world)
2. " " " on camera local coordinate system.
3. Travel along spline
4. Look at & follow images

• Camera transformations are base on axes defined as vectors

• Default orientation for axes:

$$x = (1.0, 0, 0)$$

$$y = (0, 1.0, 0)$$

$$z = (0, 0, 1)$$

$$1 \text{ radian} = 57.2957795 \text{ degrees}$$

- i) default x axes: point in direction (1, 0, 0)
 one unit left (or right)
 zero unit up
 zero unit forward

ii) two modes:

Mode 1: Transformations based on default axes

Mode 2: Rotate axes w/ the camera while taking in account the previous orientation axis

(i) $m = \cos \theta$ (beta)

$n = \cos \gamma$ (gamma)

$I = \cos \alpha$ (Alpha)

Y axis
(Y constant)

$$\begin{cases} x' = m_1 x + I_1 z = \cos \theta_1 x + \cos \alpha_1 z \\ y' = m_2 x + I_2 z = \cos \theta_2 x + \cos \alpha_2 z \end{cases}$$

rotate x axis
(beta sweep)

$$\begin{cases} y' = n_2 y + I_2 z = \cos \gamma_2 y + \cos \alpha_2 z \\ z' = n_3 y + I_3 z = \cos \gamma_3 y + \cos \alpha_3 z \end{cases}$$

z axis
(gamma sweep)

$$\begin{cases} x' = m_1 x + n_1 y = \cos \theta_1 x + \cos \gamma_1 y \\ y' = m_2 x + n_2 y = \cos \theta_2 x + \cos \gamma_2 y \end{cases}$$

origins of x, y, z and x', y', z are the same:

$$\begin{aligned} x' &= m_1 x + n_1 y + I_1 z \\ &= \cos \theta_1 x + \cos \gamma_1 y + \cos \alpha_1 z \end{aligned}$$

$$\begin{aligned} y' &= m_2 x + n_2 y + I_2 z \\ &= \cos \theta_2 x + \cos \gamma_2 y + \cos \alpha_2 z \end{aligned}$$

$$\begin{aligned} z' &= m_3 x + n_3 y + I_3 z \\ &= \cos \theta_3 x + \cos \gamma_3 y + \cos \alpha_3 z \end{aligned}$$

Coordinate system

1. A coordinate system is composed of three vectors:

a. vector \vec{x}

b. vector \vec{y}

c. vector \vec{z}

(fig 1)

(Right-handed coordinate system)

2. To position an object in this coordinate system, we are required to know this object position and orientation in this coordinate system.

3. Let's assume the object in question is the camera. This object would define a second coordinate system with three vectors:

a. vector \vec{x}'

b. vector \vec{y}'

c. vector \vec{z}'

This object is considered fixed in this axis system

4. The ^{camera's} orientation (object) position is the eye position while The camera's orientation is defined by the view direction: \vec{z}' vector.

We will represent the fixed axis system $(\vec{x}, \vec{y}, \vec{z})$ and the moving camera axis system $(\vec{x}', \vec{y}', \vec{z}')$ (fig 2)

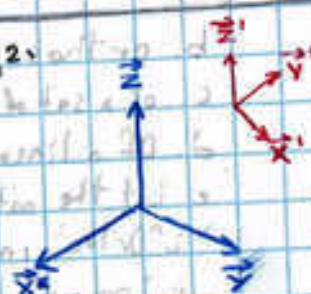
5. In order to translate the fixed axis system of the camera, we will use a temporary coordinate system $(\vec{x}_t, \vec{y}_t, \vec{z}_t)$ (instead of translating the object to origin, work on it, and move it back).

6. By rotating the temporary coordinate system $(\vec{x}_t, \vec{y}_t, \vec{z}_t)$ we obtain the camera coordinate system.

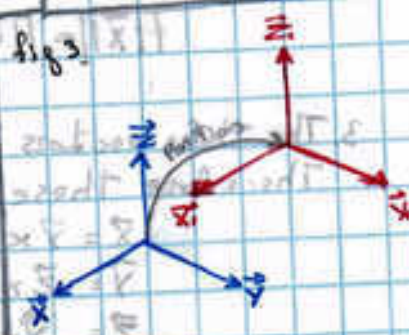
7. In this case the \vec{z}' vector would be the view direction (forward)

The \vec{y}' vector would be the upward (upview) vector

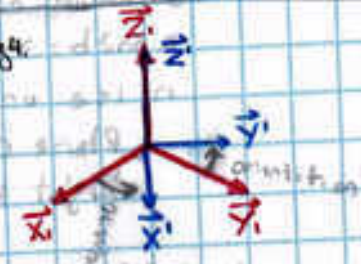
The \vec{x}' vector would be the side vector left. In this case if we use the Right handed coordinate system in fig 1



(fixed axis system $\vec{x}, \vec{y}, \vec{z}$, and camera's axis system $\vec{x}', \vec{y}', \vec{z}'$)



(translating the fixed axis system of the camera (origin) obtaining temporary coordinate system $\vec{x}_t, \vec{y}_t, \vec{z}_t$)



(rotation of temporary axis system $\vec{x}_t, \vec{y}_t, \vec{z}_t$ to obtain the camera coordinate)

Coordinate System Properties

1. The coordinate system is orthonormal

In linear algebra, we say that two vectors

a. Orthogonality is when two things can vary independently.
(They are uncorrelated or perpendicular \perp)

For example: when two vectors are perpendicular (they form a right angle) they are said to be orthogonal.

b. orthonormal (straight) gonial (angle)

c. of a set of vectors both orthogonal and normalized are called orthonormal

d. Of a linear transformation that preserve both angles and length

e. Let the orthonormal basis vectors be $(1, 0)$ and $(0, 1)$

i. "Ortho" in orthonormal means that $(1, 0) \cdot (0, 1) = 0$

ii. "normal" means that $(1, 0) \cdot (1, 0) = 1$ and $(0, 1) \cdot (0, 1) = 1$.

2. The norm (magnitude) of the three vectors $(\vec{x}, \vec{y}, \vec{z})$ that define the coordinate system is always 1:

$$\|\vec{x}\| = \|\vec{y}\| = \|\vec{z}\| = 1$$

3. The three vectors $(\vec{x}, \vec{y}, \vec{z})$ are orthogonal (perpendicular \perp)

Therefore these equalities are always true:

$$\vec{x} = \vec{y} \times \vec{z}$$

$$\vec{y} = \vec{z} \times \vec{x}$$

$$\vec{z} = \vec{x} \times \vec{y}$$

4. The cross product $(\vec{A} \times \vec{B})$ is defined as a vector perpendicular to both \vec{A} and \vec{B}

a. The formula is:

$$\vec{A} \times \vec{B} = ab \cdot \sin \theta \cdot \vec{n}$$

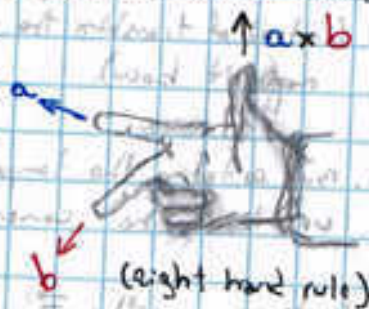
θ : smaller angle between \vec{A} and \vec{B} ($0 < \theta < 180^\circ$)

a and b : are the magnitudes of vector \vec{A} and \vec{B}

\vec{n} : is a unit vector perpendicular to the

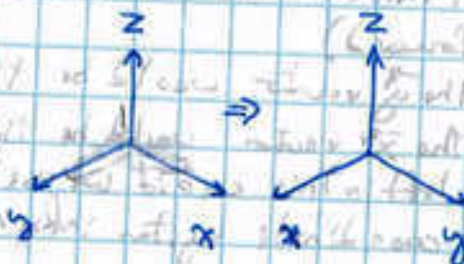
plane containing \vec{a} and \vec{b} in the

Right hand direction



b. Left hand rule:

Right-handed to Left handed rule:



c. Note: $(\vec{x}, \vec{y}, \vec{z})$ do not represent scalars

Coordinate System Representation

1. Let's assume we wish to store the position and rotation of the camera's position and orientation:

a. The camera position (the center of its coordinate system) is stored using vectors (eyeX, eyeY, eyeZ)

b. For the camera orientation, we can store:

i) the rotation angles

-or-

ii) the x', y', z' vectors (if we use the orthogonal property, we may only need two of them)

2. Let's:

a. The fixed coordinate system be represented by $\vec{x}, \vec{y}, \vec{z}$

b. The camera coordinate system " " " $\vec{x}', \vec{y}', \vec{z}'$

c. The new camera coordinate system after transformation be represented by $\vec{x}'', \vec{y}'', \vec{z}''$ or $\vec{x}_i, \vec{y}_i, \vec{z}_i$.

3. To represent a vector $V = (v_1, v_2, v_3)$ in the fixed coordinate system $\vec{x}, \vec{y}, \vec{z}$, we write:

$$a. V = v_1 \cdot \vec{x} + v_2 \cdot \vec{y} + v_3 \cdot \vec{z} \quad \left. \begin{array}{l} \vec{x}, \vec{y}, \vec{z} \text{ are vectors} \\ v_1, v_2, v_3 \text{ are scalars (numbers)} \end{array} \right\}$$

$$= \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

4. Rotation transformation

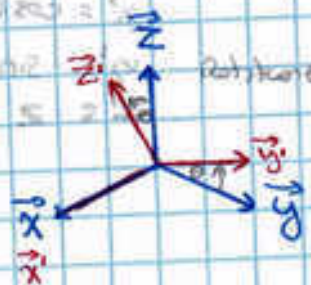
a. By using matrices, we can represent rotation

b. Let's assume we wish to rotate around the x-axis

(Rx) so we rotate from $(\vec{x}, \vec{y}, \vec{z})$ to $(\vec{x}', \vec{y}', \vec{z}')$

$$i) R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & \sin \theta & 0 \\ 0 & -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix}$$

$$[(x', y', z')] = R_x(\theta) \cdot (x, y, z)$$



$$ii) x' = 1 \cdot x + 0 \cdot y + 0 \cdot z = x$$

$$y' = 0 \cdot x + \cos \theta \cdot y + \sin \theta \cdot z = \cos \theta \cdot y + \sin \theta \cdot z$$

$$z' = 0 \cdot x - \sin \theta \cdot y + \cos \theta \cdot z = -\sin \theta \cdot y + \cos \theta \cdot z$$

\therefore

$$x' = x$$

$$y' = \cos \theta \cdot y + \sin \theta \cdot z$$

ii) For y rotation:

$$R_y = \begin{bmatrix} \cos \theta & 0 & -\sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix}$$

$$x' = \cos \theta \cdot x + 0 \cdot y + (-\sin \theta) \cdot z = \cos \theta \cdot x + (-\sin \theta) \cdot z$$

$$y' = 0 \cdot x + 1 \cdot y + 0 \cdot z = y$$

$$z' = \sin \theta \cdot x + 0 \cdot y + \cos \theta \cdot z = \sin \theta \cdot x + \cos \theta \cdot z$$

\therefore

$$x' = \cos \theta \cdot x + (-\sin \theta) \cdot z$$

$$y' = y$$

$$z' = \sin \theta \cdot x + \cos \theta \cdot z$$

iv) For z rotation:

$$R_z = \begin{bmatrix} \cos \theta & \sin \theta & 0 & 0 \\ -\sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix}$$

$$x' = \cos \theta \cdot x + \sin \theta \cdot y + 0 \cdot z = \cos \theta \cdot x + \sin \theta \cdot y$$

$$y' = -\sin \theta \cdot x + \cos \theta \cdot y + 0 \cdot z = -\sin \theta \cdot x + \cos \theta \cdot y$$

$$z' = 0 \cdot x + 0 \cdot y + 1 \cdot z = z$$

$$x' = \cos \theta \cdot x + \sin \theta \cdot y$$

$$y' = -\sin \theta \cdot x + \cos \theta \cdot y$$

$$z' = z$$



$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$x' = \cos \theta \cdot x + \sin \theta \cdot y + 0 \cdot z = \cos \theta \cdot x + \sin \theta \cdot y$$

$$y' = -\sin \theta \cdot x + \cos \theta \cdot y + 0 \cdot z = -\sin \theta \cdot x + \cos \theta \cdot y$$

$$z' = 0 \cdot x + 0 \cdot y + 1 \cdot z = z$$

$$\begin{aligned} x &= \cos \theta \cdot x' + \sin \theta \cdot y' \\ y &= -\sin \theta \cdot x' + \cos \theta \cdot y' \\ z &= z' \end{aligned}$$