Lecture 3

Transformations and animation

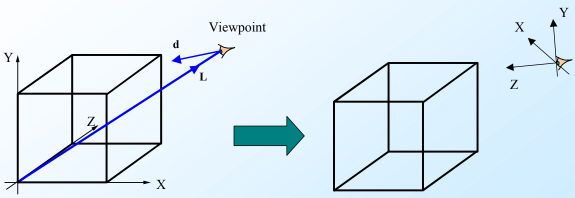
Flying Sequences

We now return to the question of transforming the origin of a graphics scene

This would be used in generating animated flying sequences where the viewpoint moves round the scene.

Let the required viewpoint be **L** = [Lx,Ly,Lz]
and the required view direction be **d** = [dx,dy,dz]
Let |**d**| = 1

Transformation of viewpoint



Coordinate System for definition

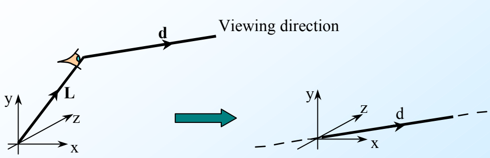
Coordinate System for viewing

Flying Sequences

The required transformation is in three parts:

1. Translation of the Origin
2. Rotate about Y
3. Rotate about X

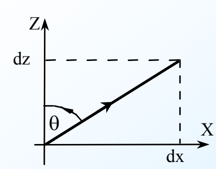
Translation of the Origin



Step 1: Move origin to the required viewpoint

$$\boldsymbol{A} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -L_x & -L_y & -L_z & 1 \end{pmatrix}$$

Rotate about Y until dx = 0

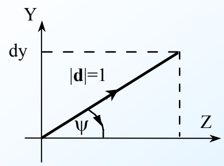


Step 2: Rotate about Y

Cos θ = dz/√(dx*dx + dz*dz)
Sin θ = dx/√(dx*dx + dz*dz)

$$\boldsymbol{B} = \begin{pmatrix} d_z/v & 0 & d_x/v & 0 \\ 0 & 1 & 0 & 0 \\ -d_x/v & 0 & d_z/v & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotate about X until dy = 0



$$\boldsymbol{C} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & v & d_y & 0 \\ 0 & -d_y & v & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Step 3: Rotate about X

$\cos\psi = \sqrt{(dx*dx+dz*dz)}/\lvert d\rvert$ $\sin\psi = dy/\lvert\mathbf{d}\rvert = dy$

Combining the matrices

The matrix that transforms the origin is:

$\boldsymbol{T} = \boldsymbol{A} * \boldsymbol{B} * \boldsymbol{C}$

and for every point in the graphics scene we calculate

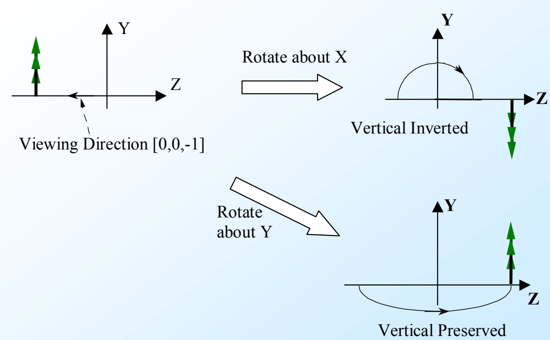
P' = P * $\boldsymbol{T}$

Verticals

Notice we have not introduced verticals in the above analysis.

Usually, the y direction is treated as vertical, and by doing the Ry transformation first things work out correctly

However it is possible to invert the vertical



Rotation about a general line

Special effects, such as rotating a scene about a general line can be achieved by transformations,

The transformation is formed by:

Making the line of rotation one of the Cartesian axes
Doing the rotation
Restoring the line to its original place

Rotation about a general line

The first part is achieved by the same matrix that we derived for the flying sequences

$\boldsymbol{T} = \boldsymbol{A} * \boldsymbol{B} * \boldsymbol{C}$

and the rest is achieved by a rotation followed by the inversion of T

$\boldsymbol{T} = \boldsymbol{A} * \boldsymbol{B} * \boldsymbol{C} * \boldsymbol{Rz} * \boldsymbol{C}^{-1} * \boldsymbol{B}^{-1} * \boldsymbol{A}^{-1}$

Other Effects

Similar effects can be created using this approach

eg Making objects shrink

1. Move the object to the origin
2. Apply a scaling matrix
3. Move the object back to where it was

Projection by Matrix multiply

Usually projection and drawing of a scene comes after transformation.

It is therefore convenient to combine the projection with the other parts of the transformation

Orthographic Projection Matrix

For orthographic projection we simply drop the z coordinate

$$\boldsymbol{M_o} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Perspective Projection Matrix

$$[x,y,z,1] \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1/f \\ 0 & 0 & 0 & 0 \end{pmatrix} = [x,y,z,z/f]$$

Normalisation

Remember that homogenous coordinates need to be normalised, so we need to divide by the last ordinate as a final step:

[x,y,z,z/f] is normalised to [x*f/z, y*f/z, f, 1]

as required

Projection matrices are singular

Notice that projection matrices are singular (they cannot be inverted)

This is because a projection cannot be inverted, ie

Given a 2D image, we cannot in general reconstruct the 3D original.

Affine transformations

Affine transformations:

translation

scaling

rotation

orthographic projection

preserve parallelism and linearity.

Non-affine transformations:

perspective projection

Homogenous Coordinates as Vectors

We now take a second look at homogeneous coordinates, and their relation to vectors.

In the previous lecture we described the fourth ordinate as a scale factor.

[X, Y, Z, h] is equivalent to [X/h, Y/h, Z/h]

Homogenous Cartesian

Homogenous co-ordinates and vectors

Homogenous co-ordinates fall into two types:

1. Those with the final ordinate non-zero, which can be normalised into position vectors.

2. Those with zero in the final ordinate which are direction vectors, and have direction magnitude.

Vector Addition

If we add two direction vectors, we obtain a direction vector. ie:

[xi,yi,zi,0] + [xj,yj,zj,0] = [xi+xj, yi+yj, zi+zj,0]

This is the normal vector addition rule.

Adding position and direction vectors

If we add a direction vector to a position vector we obtain a position vector:

[Xi,Yi,Zi,1]+[xj,yj,zj,0] = [Xi+xj,Yi+yj,Zi+zj,1]

This is a nice result, because it ties in with our definition of a straight line in Cartesian space being defined by a one point and a direction:

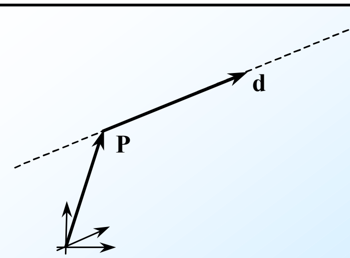


Diagram 4.2 Adding a direction vector to a position vector

Adding two position vectors

If we add two position vectors we obtain their mid-point:

$$[Xi,Yi,Zi,1] + [Xj,Yj,Zj,1] = [Xi{+}Xj,Yi{+}Yj,Zi{+}Zj,2]$$

$$= [(Xi{+}Xj)/2,(Yi{+}Yj)/2,(Zi{+}Zj)/2,1]$$

Note that this is a reasonable result since adding two position vectors has no meaning in vector algebra.

$$\begin{array}{cl} \left(\begin{array}{cccc} qx & qy & qz & 0 \\ \hline rx & ry & rz & 0 \\ \hline sx & sy & sz & 0 \\ \hline Tx & Ty & Tz & 1 \end{array}\right) & \begin{array}{l} \leftarrow \textbf{Direction Vector} \\ \leftarrow \textbf{Direction Vector} \\ \leftarrow \textbf{Direction Vector} \\ \leftarrow \textbf{Position Vector} \end{array} \end{array}$$

Diagram 4.3 The composition of an affine transformation matrix

Characteristics of Transformation matrices

In a direction vector the zero in the last ordinate ensures vectors will not be affected by the translation.

In a position vector the 1 in the last ordinate means all vectors will have the same displacement.

If we do not shear the object the three vectors **q r** and **s** will remain orthogonal, ie:

$$\mathbf{q} \bullet \mathbf{r} = \mathbf{r} \bullet \mathbf{s} = \mathbf{q} \bullet \mathbf{s} = 0.$$

What the individual rows mean?

To see this we consider the effect of the transformation in simple cases.

For example take the unit vectors along the Cartesian axes eg along the x axis, i = [1,0,0,0]

$$[1,0,0,0] \begin{pmatrix} qx & qy & qz & 0 \\ rx & ry & rz & 0 \\ sx & sy & sz & 0 \\ Tx & Ty & Tz & 1 \end{pmatrix} = [qx, qy, qz, 0]$$

Axis Transformation

Similarly we find that direction

j = [0,1,0,0]

will be transformed to direction

[rx,ry,rz,0]

and k = [0,0,1,0]

will be transformed to [sx,sy,sz,0].

Transforming the Origin

$$[0,0,0,1] \begin{pmatrix} qx & qy & qz & 0 \\ rx & ry & rz & 0 \\ sx & sy & sz & 0 \\ Tx & Ty & Tz & 1 \end{pmatrix} = [Tx, Ty, Tz, 1]$$

Meaning of a transformation matrix

$$\begin{pmatrix} qx & qy & qz & 0 \\ rx & ry & rz & 0 \\ sx & sy & sz & 0 \\ Tx & Ty & Tz & 1 \end{pmatrix}$$

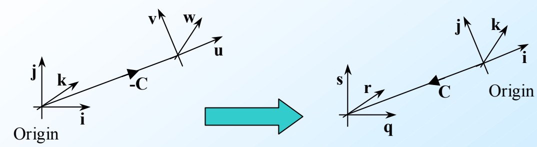
The old X axis after transformation

The old Y axis after transformation

The old Z axis after transformation

The old origin after transformation

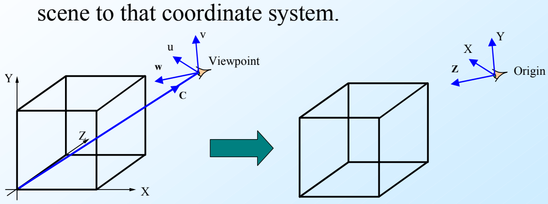
Effect of a transformation matrix



$$\begin{pmatrix} qx & qy & qz & 0 \\ rx & ry & rz & 0 \\ sx & sy & sz & 0 \\ Cx & Cy & Cz & 1 \end{pmatrix}$$

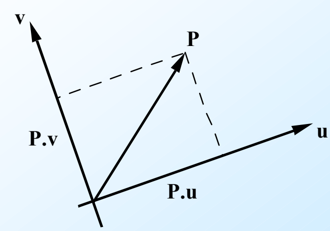
What we want is the other way round

We are given the values of [u,v,w] and C and would like to know the transformation matrix that moves the scene to that coordinate system.

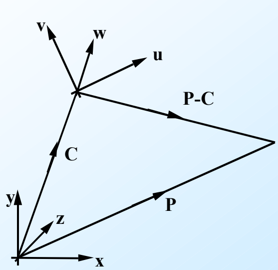


To see how to do this we introduce the notion of the dot product as a projection.

The dot product as projection



Changing axes by projection



Transforming point P

Given point P in the [x,y,z] axis system, we can calculate the corresponding point in the [u,v,w] space as:

P'x = (**P**-**C**).***u*** = **P.*u*** - **C.*u***

P'y = (**P**-**C**).***v*** = **P.*v*** - **C.*v***

P'z = (**P**-**C**).***w*** = **P.*w*** - **C.*w***

Or in Matrix form:

$$[P'x,P'y,P'z,1] = [Px,Py,Pz,1] \begin{pmatrix} ux & vx & wx & 0 \\ uy & vy & wy & 0 \\ uz & vz & wz & 0 \\ -C\cdot u & -C\cdot v & -C\cdot w & 1 \end{pmatrix}$$

Verticals

Unlike the previous analysis we now can control the vertical,

ie the **v** direction is taken as the vertical and constrained by the software to be upwards

Back to flying sequences

Given a viewpoint point **C** and a view direction **d** we need to find the transformation matrix.

We know that **d** is the direction of the new z axis, so we can write immediately:

w = **d**/|**d**|

Now the horizontal direction

Let the horizontal direction be **p**

Thus ***u*** = **p**/|**p**|

To keep the horizontal direction we need

py = 0

(**p** has no vertical component)

And the vertical direction

Let **q** be the vertical direction, thus

v = **q**/|**q**|

q must have a positive y component, so we can say:

qy = 1

So we have four unknowns

p = [px,0,pz]
q = [qx,1,qz]

To solve for these we use the cross product and dot product. Since the axis system is left handed:

d = **p** x **q**

(we can do this because **p**'s magnitude is not set)

Evaluating the cross product

$$[dx,dy,dz] = \begin{vmatrix} i & j & k \\ px & 0 & pz \\ qx & 1 & qz \end{vmatrix}$$

dx = -pz

dy = pz qx - px qz

dz = px

so we have now completely specified vector **p**

Using the dot product

Lastly we can use the fact that the vectors **p** and **q** are orthogonal, thus

p.q = 0

px qx + pz qz = 0

and from the cross product (last slide)

dy = pz qx - px qz

So we have a simple linear equation to solve for **q**

The final matrix

As defined we have

u = **p**/|**p**| ***v*** = **q**/|**q**| ***w***= **d**/|**d**|

so we can write down the matrix.