Table of Contents

Three-Dimensional Affine Transformations

Affine transformations in three dimensions allow us to manipulate 3D objects by altering their position, orientation, and shape. A 3D point is expressed as:

$$P = X\vec{\boldsymbol{i}} + Y\vec{\boldsymbol{j}} + Z\vec{\boldsymbol{k}}$$

where

$$\vec{\boldsymbol{i}} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad \vec{\boldsymbol{j}} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad \vec{\boldsymbol{k}} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

We use homogeneous coordinates and column vectors such that points are written as follows:

$$P = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Generally, a 3D affine transformation is written in matrix form as:

$$M = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

such that transforming point P into point Q with matrix M is mathematically expressed as $Q = MP$.

Elementary Transformations

- Translation:

$$\begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Scaling:

$$\begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Rotation around the x -axis:

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

- Rotation around the y -axis:

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

- Rotation around the z -axis:

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

These are the most used 3D transformation matrices (there exist others). Such transformations can be combined to form a single matrix encompassing many transformations. As an example, consider rotating a 3D point P around an arbitrary axis expressed with unit vector $\vec{v}=(v_x, v_y, v_z)^T$:

- First, project $\vec{v}$ onto the xy plane, by setting its 3rd coordinate to 0
- Normalize projected $\vec{v}$ and compute the angle ϕ between $\vec{v}$ and the x -axis in the following way: $\phi = \arccos(\vec{i} \cdot \vec{v})$, where $\vec{i} = (1,0,0)$
- Apply a ϕ degree z -axis rotation to $\vec{v}$ as $R_z(\phi)\vec{v}$
- Now $R_z(\phi)\vec{v}$ is located in the xz plane and we need to find the angle between it and the z axis.
- To find this angle, we compute $\psi = \arccos\left(\frac{\vec{k} \cdot R_z(\phi)\vec{v}}{\|R_z(\phi)\vec{v}\|}\right)$ where $\vec{k} = (0,0,1)$ and apply a ψ degree y axis rotation to $\vec{v}$ and obtain $R_y(\psi)R_z(\phi)\vec{v}$
- Now $R_y(\psi)R_z(\phi)\vec{v}$ is aligned with the z -axis and we can apply the desired rotation to point P around that axis.

Hence, to rotate a point P by an angle α around an arbitrary vector $\vec{v}$, combine the following rotations:

$$Q = R_z(-\phi)R_y(-\psi)R_z(\alpha)R_y(\psi)R_z(\phi)P$$

Alternatively, we can construct a rotation matrix about any axis represented with unit vector $\vec{v}$ directly. First we express the rotation axis in matrix form as:

$$J_{\vec{v}} = \begin{bmatrix} 0 & -v_z & v_y \\ v_z & 0 & -v_x \\ -v_y & v_x & 0 \end{bmatrix}$$

and compute the rotation matrix as

$$R = I + \sin(\theta)J_{\vec{v}} + (1-\cos(\theta))J_{\vec{v}}^2$$

It is also possible to derive both the angle of rotation and the axis vector when given matrix R :

$$\theta = \cos^{-1}\left(\frac{tr(R)-1}{2}\right)$$

and

$$\frac{(R-R^T)}{2\sin\theta} = J_{\vec{v}}$$

Direction of Positive Rotations per Axis

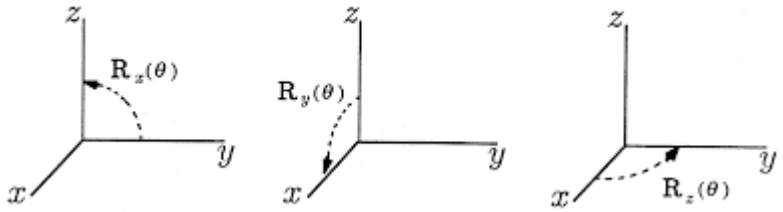

Illustration 1: Positive rotations per axis

3D Transform Hierarchy

3D Transform	Degrees of Freedom	Preserves
Translation	3	Orientation
Rigid Body	6	Lengths
Similarity	7	Angles
Affine	12	Parallelism
Projective	15	Straight Lines

Perspective Projections

Assuming a viewing coordinate system defined by $\vec{u}, \vec{v}, \vec{n}$ in which the line of sight is given by $-\vec{n}$, projections are performed onto the near plane of the viewing volume, and N is the *distance* from the origin of the camera coordinate system to the near plane (such that the point $(0,0,-N)$ is contained in the near plane). Hence a perspective projection of a point $P=(X,Y,Z)^T$ is:

$$p = (x,y)^T = \left(-N\frac{X}{Z}, -N\frac{Y}{Z}\right)$$

Under this type of transformation, lines project onto lines but parallel lines do not project to parallel lines in general.

Vanishing Points

Under perspective projection, parallel lines in 3D do not map to parallel lines in 2D, except in one case when the lines are parallel to the viewing plane. Otherwise, parallel lines, once perspective projected, will meet at a point called the vanishing point. Suppose a 3D line passes through a point $A=(X,Y,Z)^T$ with unit direction vector $\vec{n}=(n_x,n_y,n_z)^T$. In parametric form, this line is written as $P(t)=A+\vec{n}t$. If we project this line onto the viewing plane, we obtain:

$$p(t) = \left(N\frac{X+n_x t}{-(Z+n_z t)}, N\frac{Y+n_y t}{-(Z+n_z t)} \right)^T$$

Also suppose $P(t)$ is not parallel to the viewing plane and $n_z>0$, which make the line go away from the camera position as t increases. If we take the following limit:

$$\lim_{t\to\infty} p(t)$$

then we find it is equal to:

$$\left(-N\frac{n_x}{n_z}, -N\frac{n_y}{n_z} \right)^T$$

which is the vanishing point onto the viewing plane.

Pseudo-Depth

When a perspective projection is performed, the depth information is lost. This prevents from removing hidden surfaces while rendering objects. We choose to keep a quantity we call pseudo-depth, which is easy to obtain and is sufficient to sort surfaces with respect to their depth in the viewing volume. To accomplish this, we make any point P project to $(X',Y',Z')^T$, where Z' is pseudo-depth:

$$\begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} = \begin{bmatrix} -N\frac{X}{Z} \\ -N\frac{Y}{Z} \\ \frac{-(aZ+b)}{Z} \end{bmatrix}$$

where a and b some constants, chosen such that:

$$-1 \le \frac{-(aZ+b)}{Z} \le 1$$

In other words, we keep the pseudo-depth values comprised between -1 and 1 for Z between $-N$ and $-F$. In order to do this, we set the constants as:

$$a = -\frac{F+N}{F-N} \quad b = \frac{-2FN}{F-N}$$

These constants are obtained as follows: When $Z=-N$, we require that

$$\frac{-(aZ+b)}{Z} = -1$$

Therefore,

$$\frac{-(a(-N)+b)}{-N} = -1$$

which in turn implies

$$a = \frac{N+b}{N}$$

When $Z=-F$, we require that

$$\frac{-(aZ+b)}{Z} = 1$$

and therefore

$$\frac{-(a(-F)+b)}{-F} = 1$$

which in turn also implies

$$a = \frac{b-F}{F}$$

We now have two expressions for a that we equate in order to find b :

$$\frac{N+b}{N} = \frac{b-F}{F}$$

Isolating b , we obtain

$$b = \frac{-2FN}{F-N}$$

Now using

$$a = \frac{N+b}{N}$$

and substituting for b results in

$$a = \frac{-(F+N)}{F-N}$$

Now this perspective transformation of a point p_v in homogeneous coordinates can be written as:

$$M_p p_v = \begin{bmatrix} N & 0 & 0 & 0 \\ 0 & N & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} NX \\ NY \\ aZ+b \\ -Z \end{bmatrix}$$

Dividing the result by the last element, we obtain:

$$\begin{bmatrix} -N\frac{X}{Z} \\ -N\frac{Y}{Z} \\ \frac{-(aZ+b)}{Z} \\ 1 \end{bmatrix}$$

In general, with perspective projections we have as many finite vanishing points as there are intersections between the coordinate axes and the extended viewing plane.

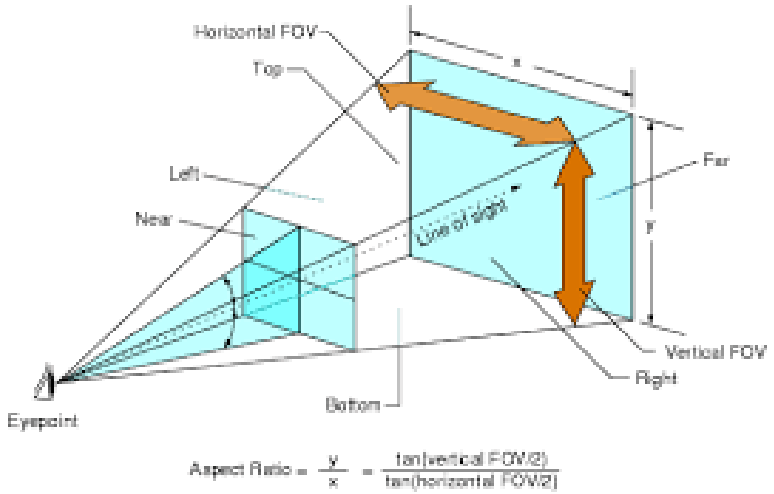

Illustration 2: The viewing volume with the camera as the eyepoint

Other Projections

Perspective projections are most common because this is how humans see. However, for purposes other than realism, other forms of projections may be useful, such as these that

preserve proportions, or display objects from particular angles, etc. Here is a brief description of some of these projections:

- **Orthographic Projections:** In this type of projection, the direction of the projection is perpendicular to the viewing plane. In other words, this projection is realized by simply removing the third coordinate of the point to be projected.

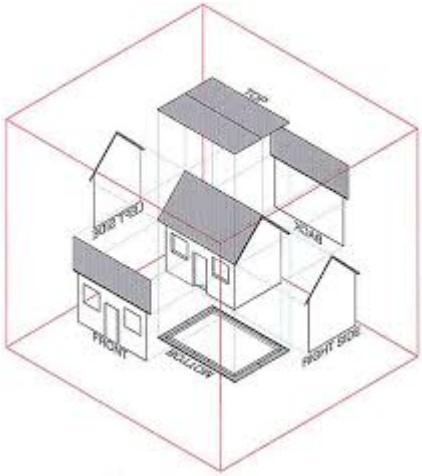

Illustration 3: Orthographic projection of a house

- **Isometric Projections:** This projection is obtained by aligning the viewing plane in a way such that it intersects each coordinate axis at the same distance from the origin.

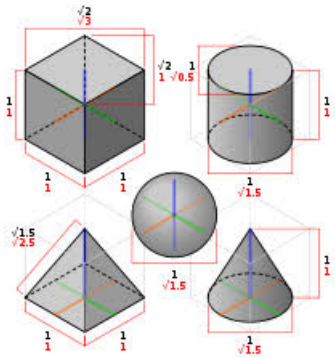

Illustration 4: Isometric projections of some common shapes

- **Oblique Projections:** Obtained by projecting points along parallel lines that are not perpendicular to the viewing plane.

Synthetic Camera

The synthetic camera allows the user to choose the viewing position in 3D. To create a

camera, we define a coordinate system within the world coordinate system. This is accomplished by specifying a 3D position, a gaze vector, and an indication of where up is. With these three elements, it is possible to obtain the unit vectors of the coordinate system of the camera: $\vec{u}$, $\vec{v}$, $\vec{n}$.

Pose e , the position of the origin of the camera coordinate system, g a point through which the gaze direction unit vector $\vec{n}$ points to, and $\vec{p}$, a unit vector in the direction in which up is. We can then obtain unit vector $\vec{n}$ of the coordinate system as:

$$\vec{n} = \frac{e-g}{\|e-g\|}$$

It is natural to consider the vector $\vec{u}$ as a unit vector perpendicular to $\vec{n}$ and $\vec{p}$ and thus $\vec{u}=\vec{p}\times\vec{n}$. Further, $\vec{v}=\vec{n}\times\vec{u}$. One must normalize these vectors to unit length.

Camera Transformation Matrix

The transformation matrix for the camera operates by transforming $\vec{\boldsymbol{i}}$, $\vec{\boldsymbol{j}}$, and $\vec{\boldsymbol{k}}$ into $\vec{u}$, $\vec{v}$, and $\vec{n}$. Hence we can write:

$$M = \begin{bmatrix} \vec{u} & \vec{v} & \vec{n} & \vec{e} \\ \vec{0}^T & & & 1 \end{bmatrix}$$

where $\vec{e}$ is the location of the origin of the camera coordinate system. Observe that $M\vec{\boldsymbol{i}}=\vec{u}+\vec{e}$, $M\vec{\boldsymbol{j}}=\vec{v}+\vec{e}$, and $M\vec{\boldsymbol{k}}=\vec{n}+\vec{e}$, where

$$\vec{\boldsymbol{i}}=\begin{bmatrix}1\\0\\0\\1\end{bmatrix} \quad \vec{\boldsymbol{j}}=\begin{bmatrix}0\\1\\0\\1\end{bmatrix} \quad \vec{\boldsymbol{k}}=\begin{bmatrix}0\\0\\1\\1\end{bmatrix}$$

Hence a point p_v expressed in the camera (or viewing) coordinates can be transformed into a point in the world coordinate system p_w by computing $M\,p_v=p_w$. The inverse transformation also exists and is simply $p_v=M^{-1}p_w$. Note that M is a composite transform created with a rotation matrix and a translation matrix:

$$M = TR = \begin{bmatrix} [I] & \vec{e} \\ \vec{0}^T & 1 \end{bmatrix} \begin{bmatrix} \vec{u} & \vec{v} & \vec{n} & \vec{0} \\ \vec{0}^T & & & 1 \end{bmatrix} = \begin{bmatrix} u_x & v_x & n_x & e_x \\ u_y & v_y & n_y & e_y \\ u_z & v_z & n_z & e_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The inverse matrices for the rotation and the translation are easy to find:

$$R^{-1} = \begin{bmatrix} \vec{u}^T & \\ \vec{v}^T & \vec{0} \\ \vec{n}^T & \\ \vec{0}^T & 1 \end{bmatrix}$$

and

$$T^{-1} = \begin{bmatrix} [I] & -\vec{e} \\ \vec{0}^T & 1 \end{bmatrix}$$

Hence,

$$M_v = M^{-1} = (TR)^{-1} \quad R^{-1}T^{-1} = \begin{bmatrix} \vec{u}^T & -\vec{e}\cdot\vec{u} \\ \vec{v}^T & -\vec{e}\cdot\vec{v} \\ \vec{n}^T & -\vec{e}\cdot\vec{n} \\ \vec{0}^T & 1 \end{bmatrix}$$

Given a point in world coordinates, we now can transform it into the camera (viewing) coordinate system, and apply the perspective transformation (including pseudo-depth). If p_w is a point in world coordinates, then its representation in the camera coordinates including its perspective transformation is given by

$$p_v = M_p M_v p_w$$

The near plane extends infinitely and so does the viewing volume. We need to specify four points on the near plane that will limit the extent of it. We define the points as

$$p_1 = (l, t, -N) \quad p_2 = (r, t, -N) \quad p_3 = (r, b, -N) \quad p_4 = (l, b, -N)$$

Generally, we let $l = -r$ and $b = -t$. The points form a rectangle on the near plane that, along with the far plane, define the extent of the viewing volume.

We already have the pseudo depth comprised between -1 and 1. We now need to find the transformations that will do the same for the rest of the coordinates within the viewing volume. A scaling and a translation are needed to accomplish this:

$$S_1 = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad T_1 = \begin{bmatrix} 1 & 0 & 0 & \frac{-(r+l)}{2} \\ 0 & 1 & 0 & \frac{-(t+b)}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

At this point, we can transform a point in world coordinates into the canonical view volume by applying the following transformations:

$$p_v = S_1 T_1 M_p M_v p_w$$

where

$$S_1 T_1 M_p = \begin{bmatrix} \frac{2N}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2N}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(F+N)}{F-N} & \frac{-2FN}{F-N} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

We can also define the viewing volume by specifying a viewing angle θ and an aspect ratio A. In this case, we write:

$$t = N \tan\left(\frac{\pi}{180}\frac{\theta}{2}\right)$$

and

$$b = -t \qquad r = A\,t \qquad l = -r$$

We now need to transform objects from the warped viewing volume into screen coordinates. A scaling and translation are needed to perform this operation. Suppose the screen window has its upper left corner located at $(0,0)$ with width w and height $h.$. First, we need to translate the (x, y) coordinates to the positive quadrant:

$$T_2 = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Next we need to scale in such a way that these translated coordinates now fit in a space defined by $(0 \ldots w, 0 \ldots h)$:

$$S_2 = \begin{bmatrix} \frac{w}{2} & 0 & 0 & 0 \\ 0 & \frac{h}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

These two transforms would be sufficient if the origin of the window was at its bottom left corner. However, in screen coordinates, the origin is at the top left corner and we need to compose a last transform which takes care of this:

$$W = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & h \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Composing these transformations together yields

$$W S_2 T_2 = \begin{bmatrix} \frac{w}{2} & 0 & 0 & \frac{w}{2} \\ 0 & \frac{-h}{2} & 0 & \frac{-h}{2} + h \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We are now in a position to transform a 3D point in world coordinates into the image coordinates of the window by performing the following suite of transformations:

$$p_s = W S_2 T_2 S_1 T_1 M_p M_v p_w$$

The last thing is to perform the perspective projection in order to obtain the point in pixel coordinates.