Lecture 2

When defining a scene that we want to draw, we will often need to perform transformations on the objects we want to draw by rotating, scaling, and translation them. To do so, we will perform linear transformations on the points that define the object. Let's begin by discussing rotation. We begin with the 2D case.

2D Rotations

If we wish to rotate by θ degrees *counterclockwise*, then we perform a matrix multiplication

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \mathbf{R} \begin{bmatrix} x \\ y \end{bmatrix}$$

where

$$\mathbf{R} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}.$$

The rotation is counterclockwise when $\theta > 0$. The way to write down this transformation in case you forget it is to note that the rotation maps (1,0) to $(\cos\theta, \sin\theta)$ and it maps (0, 1) to $(-\sin\theta, \cos\theta)$. These two observations determine the first and second columns of $\mathbf{R}$. See the slides if need be.

There are two ways to think about what's happening. First, applying the rotation by multiplying by $\mathbf{R}$ moves point (x, y) in a fixed 2D coordinate system. Second, multiplying by $\mathbf{R}$ projects the vector (x, y) onto vectors that are defined by the rows of $\mathbf{R}$. The first interpretation says that we are keeping the coordinate system but moving (rotating) the points within this fixed coordinate system. The second says we are defining a new coordinate system to represent our space $\Re^2$. Verify that this new coordinate system itself is rotated *clockwise* by θ relative to the original one. This distinction will turn out to be useful as we'll see in the next few lectures when we discuss world versus camera coordinate systems.

3D Rotations

For 3D vectors, it is less clear how to define a clockwise versus counterclockwise rotation. It depends on how we define these terms in 3D and whether the xyz coordinate system is "right handed" or "left handed". The standard coordinate *xyz* system in computer graphics is *right handed*. The axes correspond to thumb ($\hat{\mathbf{x}}$), index finger ($\hat{\mathbf{y}}$), and middle finger ($\hat{\mathbf{z}}$). We will occasionally have to switch from right to left handed coordinates, but this is something we will try to avoid.

As in the 2D case, one way to think of a 3D rotation is to move a set of points (x, y, z) in the world relative to some fixed coordinate frame. For example, you might rotate your head or rotate your chair (and your body, if its sitting on the chair). In this interpretation, we define a 3D *axis of rotation* and an *angle of rotation*. Whenever we want to express a rotation, we need a coordinate system and we rotate *about the origin* of that coordinate system, so that the origin doesn't move. For the example of rotating your head, the origin would be some point in your neck.

We can rotate about many different axes. For example, to rotate about the x axis, we perform:

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

This transformation leaves the x coordinate of each 3D point fixed, but changes the y and z coordinates, at least for points that are not on the axis i.e. $(y, z) \neq (0, 0)$. Notice that the xz rotation is essentially a 2D rotation.

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

In a right handed coordinate system, the above rotation $\mathbf{R}_x$ is clockwise if we are looking in the $\mathbf{x}$ direction, and is counterclockwise if we are looking in the $-\mathbf{x}$ direction.

If we rotate about the z axis (leaving z fixed), we use

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

This rotates counter clockwise in the xy plane, if we are looking in the $-\mathbf{z}$ direction.

If we rotate about the y axis, this leaves the values of y fixed. We write this as

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

Verify for yourself that this is counter-clockwise if we are looking in the -**y** direction for a right handed coordinate system. Curiously, the "-" sign is at the bottom left of the matrix rather than top right. Verify that this is correct, but don't stress about remembering it.

In general, by a *3D rotation matrix* we will just mean a real 3×3 invertible matrix $\mathbf{R}$ such that

$$\mathbf{R}^T = \mathbf{R}^{-1}$$

i.e.

$$\mathbf{R}^T\mathbf{R} = \mathbf{R}\mathbf{R}^T = \mathbf{I},$$

where $\mathbf{I}$ is the identity matrix. Note the rows (and columns) of $\mathbf{R}$ are orthogonal to each other and of unit length. Recall such matrices are called *orthonormal.* We also require that the determinant of $\mathbf{R}$ is 1 (see below). In linear algebra, rotation matrices are special cases of *unitary* matrices.

Rotation matrices preserve the inner product between two vectors. To see this, let $\mathbf{p}_1$ and $\mathbf{p}_2$ be two 3D vectors, possibly the same. Then

$$(\mathbf{R}\mathbf{p}_1) \cdot (\mathbf{R}\mathbf{p}_2) = \mathbf{p}_1^T\mathbf{R}^T\mathbf{R}\mathbf{p}_2 = \mathbf{p}_1^T\mathbf{p}_2 = \mathbf{p}_1 \cdot \mathbf{p}_2.$$

It follows immediately that rotations preserve the length of vectors (consider $\mathbf{p}_1 = \mathbf{p}_2$). Intuitively, rotations also preserve the angle between two vectors. A few pages from now, you should be able prove why.

Why did we require that the determinant of $\mathbf{R}$ is 1 ? An example of an orthogonal matrix that does *not* have this property is:

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

since its determinant is -1, rather than 1. This matrix performs a mirror reflection of the scene about the $x = 0$ plane. The length of vectors is preserved, as is the angle between any two vectors. But the matrix is not a rotation because of the failure of the determinant property. More intuitively, this matrix transforms objects into their mirror reflections; it turns left hands into right hands.

Another orthonormal matrix that fails the "det $\mathbf{R} = 1$" condition is

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

which swaps the x and y variables. This matrix is also a mirror reflection, namely reflects about the plane $x = y$. Swapping yz or xz also fails, of course.

Example 1

Suppose we wish to find a rotation matrix that maps $\hat{\mathbf{z}}$ to some given vector $\mathbf{p}$ which has unit length. i.e.

$$\mathbf{p} \ = \ \mathbf{R} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

We suppose that $\mathbf{p} \neq \hat{\mathbf{z}}$, since the problem would be trivial in this case.

Here is the solution. By inspection, the 3rd column of $\mathbf{R}$ has to be $\mathbf{p}$ itself. Moreover, in order for $\mathbf{R}$ to be a rotation matrix, it is necessary that the first two columns of $\mathbf{R}$ are perpendicular to the vector $\mathbf{p}$, that is, to the third column. What might these first two columns of $\mathbf{R}$ be ?

Since we have assumed $\mathbf{p} \neq \hat{\mathbf{z}}$, we consider the vector

$$\mathbf{p}' = \frac{\mathbf{p} \times \hat{\mathbf{z}}}{\parallel \mathbf{p} \times \hat{\mathbf{z}} \parallel}$$

and note that $\mathbf{p}'$ is perpendicular to $\mathbf{p}$. Hence we can use it as one of the columns of our matrix $\mathbf{R}$. (More generally, note $\mathbf{p}'$ is perpendicular to the plane spanned by $\mathbf{p}$ and $\mathbf{z}$.)

We now need a third vector, which is perpendicular to both $\mathbf{p}$ and **p'**. We chose $\mathbf{p}' \times \mathbf{p}$, noting that it is of unit length, since **p'** is already perpendicular to $\mathbf{p}$ and both $\mathbf{p}$ and **p'** are of unit length.

We also need to satisfy the determinant equals 1 condition i.e. want to build a matrix $\mathbf{R}$ which preserves handedness. For example, since $\hat{\mathbf{x}} = \hat{\mathbf{y}} \times \hat{\mathbf{z}}$, we want that $\mathbf{R}\hat{\mathbf{x}} = \mathbf{R}\hat{\mathbf{y}} \times \mathbf{R}\hat{\mathbf{z}}$. How do we do it? I didn't go over this in class, but I'll do it here so you can see. Recall $\mathbf{p} = \mathbf{R}\hat{\mathbf{z}}$, so $\mathbf{p}$ is in the third column of $\mathbf{R}$. By inspection, if were to put $\mathbf{p}'$ in the second column of $\mathbf{R}$, we would have

$$\mathbf{R}\hat{\mathbf{y}} = \mathbf{p}'$$

and similarly if we were to put $\mathbf{p}' \times \mathbf{p}$ in the first column of $\mathbf{R}$, we would have

$$\mathbf{R}\hat{\mathbf{x}} = \mathbf{p}' \times \mathbf{p}.$$

So

$$\mathbf{R}\hat{\mathbf{x}} = \mathbf{p}' \times \mathbf{p} = \mathbf{R}\hat{\mathbf{y}} \times \mathbf{R}\hat{\mathbf{z}}$$

which was what we wanted. i.e. to preserve the handedness of the axes. Thus, we could indeed let the three columns of $\mathbf{R}$ be defined:

$$\mathbf{R} = [\mathbf{p}' \times \mathbf{p}\ ,\ \mathbf{p}'\ ,\ \mathbf{p}].$$

Notice that there are many possible solutions to the problem we posed. For example, we can perform *any* rotation $\mathbf{R}_z$ about the z axis prior to applying a rotation $\mathbf{R}$ and we would still have a solution. That is, if $\mathbf{R}$ is a solution then so is $\mathbf{R}\ \mathbf{R}_z(\theta)$ for any θ since any such matrix would still take the z axis to the proscribed vector $\mathbf{p}$.

Example 2

Suppose we wish to have a rotation matrix that rotates a given unit length vector $\mathbf{p}$ to the z axis, i.e.

$$\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \mathbf{R}\ \mathbf{p}.$$

It should be obvious that this can be done by taking the transpose of the solution of Example 1, since the problem just asks for the inverse of the problem of Example 1 and the transpose of a rotation matrix is its inverse.

Axis of rotation

You may be finding it puzzling to use the term "rotation" for these matrices. Think of a spinning body such as the Earth or a baseball. When we say it is rotating, we usually mean that there is an axis about which the object spins. If you take some finite time interval, the body goes continuously from one orientation to another and the axis plays a special role in the change in orientation. It is this continous change in orientation that we usually mean by term "rotation". So, what do we mean by "rotation" for a discrete matrix $\mathbf{R}$, in particular, what do we mean by the term "axis of rotation " ?

From linear algebra, you can verify that a 3D rotation matrix (as defined earlier) must have at least one real eigenvalue. Because rotation matrices preserve vector lengths, this eigenvalue must be 1. The eigenvector $\mathbf{p}$ that corresponds to this eigenvalue satisfies

$$\mathbf{p} = \mathbf{R}\mathbf{p}\ .$$

Note that this vector doesn't change under the rotation. This vector is what we mean by the "axis of rotation" defined by the rotation matrix $\mathbf{R}$.

Given a rotation matrix $\mathbf{R}$, you could compute the axis of rotation by finding the eigenvector $\mathbf{p}$ whose eigenvalue is of length 1. This tells you the axis of rotation but it doesn't tell you the amount of rotation. For example, the family of matrices $\mathbf{R}_z(\theta)$ all have the same axis of rotation $\hat{\mathbf{z}}$ and shared eigenvalue 1. But their other eigenvalues depend on θ. [ASIDE: Their other eigenvalues are complex numbers $\cos(\theta) \pm i \sin(\theta)$. See Exercises 1.]

Example 3

Let's now come at this last problem from the other direction. Suppose we wish to define a rotation matrix that rotates by an angle θ about a given axis $\mathbf{p}$ which is a unit length vector. Obviously if $\mathbf{p}$ is one of the canonical axes ($\hat{\mathbf{x}}, \hat{\mathbf{y}}$, or $\hat{\mathbf{z}}$), then it is easy. So, let's assume that $\mathbf{p}$ is not one of the canonical axes.

One easy way to come up with such a matrix is to first rotate $\mathbf{p}$ to one of the canonical axes (Example 2), then perform the rotation by θ about this canonical axis, and then rotate the canonical axis back to $\mathbf{p}$ (Example 1). Symbolically,

$$\mathbf{R}_{p\leftarrow z}\ \mathbf{R}_z(\theta)\ \mathbf{R}_{z\leftarrow p} \ = \ \mathbf{R}_p(\theta).$$

Example 4

Let's look at a slightly trickier version of this last problem. Now we only allow for rotations about axes $\hat{\mathbf{x}}, \hat{\mathbf{y}}, \hat{\mathbf{z}}$. Here is one solution which I waved my hands through in class. First, rotate about the x axis by some angle θ_1 to bring $\mathbf{p}$ into the xy plane (z=0). Then rotate about the z axis by some angle θ_2 to bring the vector $\mathbf{R}_x(\theta_1)\mathbf{p}$ to the y axis. Now we have $\mathbf{R}_z(\theta_2)\mathbf{R}_x(\theta_1)\mathbf{p}$ and we do the desired rotation by θ around the y axis. We then undo the first two rotations to bring the y axis back to $\mathbf{p}$. This gives:

$$\mathbf{R}_x^T(\theta_1)\mathbf{R}_z^T(\theta_2)\mathbf{R}_y(\theta)\mathbf{R}_z(\theta_2)\mathbf{R}_x(\theta_1)$$

and we're done!

ASIDE: Representations of rotations in computer graphics

There are three very common methods for representing rotations in computer graphics. Here they are, from simple to complicated:

1. The rotation matrix is specified by a rotation axis $\mathbf{p}$ and by an angle of rotation θ. There is no need for $\mathbf{p}$ to be a unit vector (since it can be normalized i.e. the user doesn't need to do that). This is known as the *axis-angle* representation. OpenGL uses something like this as we'll see next week, namely `glRotate`.

2. There exists a solution for Example 4 that uses only three canonical rotations, say

$$\mathbf{R}_x(\theta_x)\ \mathbf{R}_y(\theta_y)\ \mathbf{R}_z(\theta_z).$$

   The rotations are called "*Euler angles*". For simplicity, I am omitting how you find these three angles, given say an axis-angle representation of $\mathbf{R}$.

   One problem with Euler angles arises when they are used in animation. If you want to rotate an object from one orientation to another over a sequence of image frames, you often specify an orientation in one frame and then another orientation n frames later. These are called *key frames.* You then ask your application (say *Blender* `http://www.blender.org/`) to fill in all the frames in between. A simple way for the application to do this is to calculate a sequence of rotations

$$\mathbf{R}_i = \mathbf{R}_x(\frac{\theta_x}{n}i)\ \mathbf{R}_y(\frac{\theta_x}{n}i)\ \mathbf{R}_z(\frac{\theta_x}{n}i)$$

for $i \in 1, ..n$. The problem (finally) is that this sequence of rotations can often be wonky. (In the slides, I gave a few links to videos.)

3. A more advanced technique is to use *quaternions.* This is considered by experts to be the proper way to do rotations because it performs best from a numerical analysis point of view. This technique is rather complex. If you end up doing more advanced animation then you will need to learn it, but I feel it is not worth spending our time on now.

Scaling

Scaling changes the size and shape of an object. Scaling can be done in one direction only (turning a square into a rectangle, or a cube into a rectanguloid), or in two or three directions. To scale a scene by factors (s_x, s_y, s_z) in the three canonical directions, we can multiply by a diagonal matrix as follows:

$$\begin{bmatrix} s_x x \\ s_y y \\ s_z z \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}.$$

I neglected to mention this in class but, suppose we wanted to scale the scene in other directions. How would we do it? If we wanted to scale by factors (s_1, s_2, s_3) in the direction of axes defined by the rows of some rotation matrix $\mathbf{R}$, then we can do so by the transformation:

$$\mathbf{R}^T \begin{bmatrix} s_1 & 0 & 0 \\ 0 & s_2 & 0 \\ 0 & 0 & s_3 \end{bmatrix} \mathbf{R}.$$

This transformation projects to new canonical coordinate axes (namely, the ones we want to scale, which are the rows of $\mathbf{R}$), performs the scaling, then rotates back.

Translation

Another common transformation is translation. We may need to translate the location of objects in the world, or we may need to translate the location of the camera, or we may need to transform between coordinate systems (as we will do later in the lecture). Suppose we wish to translate all points (x, y, z) by adding some constant vector (t_x, t_y, t_z) to each point. This transformation cannot be achieved by a 3×3 matrix, unfortunately. So how do we do it?

The trick is to write out scene points (x, y, z) as points in $\Re^4$, namely we append a fourth coordinate with value 1. We can then translate by performing a matrix operation as follows:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ z + t_z \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

Interestingly, we can perform rotations and scaling using a 4×4 matrix $(x, y, z, 1)$ as well. Rotation and scaling matrices go into the upper-left 3×3 corner of the 4×4 matrix.

$$\begin{bmatrix} * & * & * & 0 \\ * & * & * & 0 \\ * & * & * & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

i.e. nothing happens to the fourth coordinate. For example, we could write a scaling transformation as follows:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

We will see over the next few lectures that this "trick" is quite useful other transformations that we wish to perform.

Homogeneous coordinates

We have represented a 3D point (x, y, z) as a point $(x, y, z, 1)$ in $\Re^4$. We now generalize this by allowing ourselves to represent (x, y, z) as *any* 4D vector of the form (wx, wy, wz, w) where $w \neq 0$. Note that the set of points

$$\{ \ (wx, wy, wz, w) \ : \ w \neq 0 \ \}$$

is contained in a line in $\Re^4$ which passes through the origin (0,0,0,0) and through the point $(x, y, z, 1)$. We thus associate each point in $\Re^3$ with a line in $\Re^4$, in particular, a line that passes through the origin. All points along that line (except the origin) are considered equivalent in the sense that they all represent the same 3D scene point (x, y, z).

Note that if you add two 4D vectors together then the resulting 4D vector typically represents a different 3D point than what you might naively expect, that is,

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} + \begin{bmatrix} a' \\ b' \\ c' \\ d' \end{bmatrix} \not\equiv \begin{bmatrix} a + a' \\ b + b' \\ c + c' \\ d + d' \end{bmatrix}$$

[**Careful notation:**] where the + operator on the left means I am adding the corresponding 3D vectors, not the 4D vectors shown there. If this notation makes you uncomfortable, then what I really mean is:

$$\begin{bmatrix} a/d \\ b/d \\ c/d \end{bmatrix} + \begin{bmatrix} a'/d' \\ b'/d' \\ c'/d' \end{bmatrix} \neq \begin{bmatrix} (a + a')/(d + d') \\ (b + b')/(d + d') \\ (c + c')/(d + d') \end{bmatrix}$$

~~Only if $d = d'$ can we be sure that the equivalence holds. (Think why.)~~

Points at infinity

We have considered points (wx, wy, wz, w) under the condition that $w \neq 0$. Let's look at the remaining points $(x, y, z, 0)$, where at least one of x, y, z is non-zero i.e. we do not consider the case $(0, 0, 0, 0)$.

Consider (x, y, z, ϵ) where $\epsilon \neq 0$ and consider what happens as $\epsilon \to 0$. We can write

$$(x, y, z, \epsilon) \equiv (\frac{x}{\epsilon}, \frac{y}{\epsilon}, \frac{z}{\epsilon}, 1)$$

Thus as $\epsilon \to 0$, the corresponding 3D point goes to infinity, and stays along the line from the origin through the point $(x, y, z, 1)$. We identify the limit

$$lim_{\epsilon \to 0} \ (x, y, z, \epsilon) = (x, y, z, 0)$$

with a particular "point at infinity".

What happens to a point at infinity when we perform a rotation, translation, or scaling? Since the bottom row of each of these 4×4 matrices is (0,0,0,1), it is easy to see that these transformations map points at infinity to points at infinity. In particular, verify for yourself that:

- a translation matrix does not affect a point at infinity; i.e. it maps the point at infinity to itself.
- a rotation matrix maps a point at infinity in exactly the same way it maps a finite point, namely, $(x, y, z, 1)$ rotates to $(x', y', z', 1)$ if and only if $(x, y, z, 0)$ rotates to $(x', y', z', 0)$.
- a scale matrix maps a point at infinity in exactly the same way it maps a finite point, namely, $(x, y, z, 1)$ scales to $(s_x x, s_y y, s_z z, 1)$ if and only if $(x, y, z, 0)$ scales to $(s_x x, s_y y, s_z z, 0)$. Note that all scaling does is change the relative sizes of the x,y,z coordinates.