1 Moving on to 3D

Today, we move on to 3D coordinates. But first, let’s recap of what we did in 2D:

1. We represented a data point in 2D data space using a homogeneous coordinate vector (it has 3 entries)
2. We transformed that homogeneous coordinate vector to the screen by multiplying on the left by four transformation matrices.

In 3D space, the basic idea is the same, but the details are different:

1. It is 3D - not 2D. We represent a data point in 3D space using a homogeneous coordinate vector (it has 4 entries).
2. The transformation is more complicated. We transform the point from data space to the screen by multiplying on the left by several more transformation matrices.

As you can see, step 1 is simple. It is just a matter of putting data into a vector. All the work is in making the transformation matrices. Together, they are called the viewing pipeline. We are interested in an orthographic viewing pipeline. That means that when we project the 3D data onto the 2D screen, we use the simplest project possible – an orthographic projection.
In an orthographic projection, we simply use the (transformed) X and Y coordinates and ignore Z.

In other words, our goal is to transform a vector like this:

\[
\begin{pmatrix}
  x_{\text{norm}} \\
  y_{\text{norm}} \\
  z_{\text{norm}} \\
  1
\end{pmatrix}
\]

to a vector like this:

\[
\begin{pmatrix}
  x_{\text{screen}} \\
  y_{\text{screen}} \\
  z_{\text{ignore}} \\
  1
\end{pmatrix}
\]

where we simply pluck the first two entries out and use them as the x- and y-values for the Tk graphics object.

2 Generic Orthographic Viewing Pipeline

The 3D viewing pipeline requires more input than the screen size and the bounding box of the data. We need to know what angle to use as our view reference point (where are we standing and how are we leaning as we look at the data?)

So there are several things we need to know in order to project 3D data onto a 2D screen.

- VRP: view reference point and center of the view window; origin of the view reference coordinates
- VPN: view plane normal, direction of viewing
- VUP: view up vector, up orientation of the view volume
- \( \vec{U} \): x-axis of view reference coordinates
• extent: \((E_x, E_y, E_z)\), size of the bounding box in data space in view reference coordinates

• screen: \((s_x, s_y)\), size of the output device window in pixels

\section*{2.1 Setting up the view axes}

We choose the position in the data we would like to view and the position from which we would like to do the viewing (this is the view reference point). From this information and an educated guess about how we should orient the view axes (i.e. what is up), we can build an orthonormal set of axes.

To do this, we will need the cross product.

\subsection*{2.1.1 Tool we need: Cross Product}

The cross product, also called the outer product, or vector product, is a standard operation on two vectors. The cross product of two vectors is a third vector that is orthogonal to both original vectors. For 3D vectors, the cross product is defined as in \((\text{??})\).

\[
\vec{v} = \begin{bmatrix} y_0z_1 - y_1z_0 \\ z_0x_1 - z_1x_0 \\ x_0y_1 - x_1y_0 \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} \times \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix}
\]

(1)

We ignore the homogeneous coordinate when calculating the cross product.

\subsection*{2.1.2 Build the view reference axes}

Now we are ready to build the view reference coordinates. (Notice that this doesn’t mean we are yet building the transformation matrices we will use. The view reference axes will be used to determine what the entries of the transformation matrices are.)

Normalize the axes \(\vec{U}, \vec{V}_U, \vec{P}, \vec{V}_P\) by dividing each vector by its length.
(square root of sum of squares).

\[ \vec{U} = \vec{V} \vec{U} \times \vec{V} \vec{P} \vec{N} \]
\[ \vec{V} \vec{U} \vec{P}' = \vec{V} \vec{P} \vec{N} \times \vec{U} \]  \hfill (2)

(Note that we use normalize in two different ways: To normalize a column of data, we want each entry to have a value between 0 and 1. To normalize a vector, we want the square of the entries to sum to 1.)

**Running example setup.** Let’s look at a set of points that look like an F floating in a normalized data space. We will look at the center of the data in the Z=0 plane, (0.5,0.5,0) from the center of the data in the Z=1 plane (0.5,0.5,1). We will construct the view reference axes using just the 3 usual coordinates, because we are going to be using the cross-product, which is really meant for just the 3 coordinates.

The VPN is

\[
\begin{pmatrix}
0.5 \\
0.5 \\
0
\end{pmatrix} - \begin{pmatrix}
0.5 \\
0.5 \\
1
\end{pmatrix} = \begin{pmatrix}
0 \\
0 \\
-1
\end{pmatrix}
\]

The up vector begins as

\[
\begin{pmatrix}
0 \\
1 \\
0
\end{pmatrix}
\]

We create the u vector by crossing the up and VPN vectors

\[ \vec{u} = \vec{v} \vec{u} \vec{p} \times \vec{v} \vec{p} \vec{n} = \begin{pmatrix}
-1 \\
0 \\
0
\end{pmatrix} \]

then recompute the up vector by crossing the view normal and u vectors

\[ \vec{u} \vec{p} = \vec{v} \vec{p} \vec{n} \times \vec{u} = \begin{pmatrix}
0 \\
1 \\
0
\end{pmatrix} \]

(Why do we need to recompute the up vector? Because we simply guessed its value. There is no reason we can assume it started out orthogonal to the VPN vector. We need to do this cross product to map it orthogonal to the VPN).
2.2 Creating a transformation matrix

Let’s follow the process with our F example.

1. Set the view transformation matrix to a 4x4 identity matrix.

\[ V = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \]

Running example. Below are two views of the data. We keep the X, Y, and Z axes fixed (and color-code them red for X, green for Y, and blue for Z). We show the view reference axes (color-coded so that the view normal (vpn) is cyan, the up vector is magenta, and the u vector is brown) and the data as they have been affected by the transformation so far. One thing to keep in mind is that the origin of the view reference axes is the view reference point. The data will always look the way we want it to when we look form the view reference point. Our goal is to get the data at the appropriate X, Y positions for them to be plotted on the screen.
The view on the left is from the Z=1 plane looking back. The view on the right and the view on the bottom have been rotated so we can see the relationship between the axes. The u vector is going in the negative X direction. The vpn vector is going in the negative Z direction. And the up vector is going in the Y direction.
2. Translate VRP to the origin.

\[ V = T(-VRP_x, -VRP_y, -VRP_z)V \]

**Running example.** Below are the axes after they have been translated. The relationship between the two sets of vectors is more clear (up and Y are the same, u is opposite X, vpn is opposite Z).
3. Align the axes.

\[
V = \begin{bmatrix}
U_x & U_y & U_z & 0 \\
VUP'_x & VUP'_y & VUP'_z & 0 \\
VPN_x & VPN_y & VPN_z & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

**Running example.** Below we show the axes after they have been aligned. Up is aligned with Y, u is aligned with X, and vpn is aligned with Z. Notice that the center of the data is now the origin of the axes. That is not what we want, ultimately.
4. Translate the lower left of the view window to the origin. (option 2: skip this step)

\[ V = T\left(\frac{1}{2}E_x, \frac{1}{2}E_y, 0\right)V \]

**Running example.** Below are the axes after they have been translated so that the data are all positive again.

5. Scale the view volume to normalize the view volume.

\[ V = S\left(\frac{1}{E_x}, \frac{1}{E_y}, \frac{1}{E_z}\right)V \]

**Running example.** Since the data in the running example were already normalized, the extents were 1, and this has no effect.
6. Scale to screen coordinates and invert the x and y axes.

\[ V = S(-s_x, -s_y, 1)V \]

**Running example.** Below are the axes after the U and Up vectors have been inverted.
7. Translate by the screen size (option 2: $T(\frac{1}{2}s_x, \frac{1}{2}s_y, 0)$

$$V = T(s_x, s_y, 0)V$$

**Running example.** Below are the axes after they have been translated. Notice that it gets them back up into the $X > 0, Y > 0, Z > 0$ octant. But, in the image on the left, they still look flipped. Why? Because we still need to remember that we are viewing from the view reference point. The image on the right is oriented to make it easy for use to look from the view reference point. We see that the up vector is shooting up out of the “top” of F. The u vector is shooting out of the left of F. And the X and Y positions make sense for plotting on the screen. The left of F has small positive X value. The right of F has larger positive X values. The top of F has small positive Y values. The bottom of F has larger positive Y values.

![Diagram](image.png)

2.3 Example to do on your own:

3D Data manipulation

Consider a 3D data set with a range of [6, 10] in dimension A, [10, 20] in dimension B, and [20, 40] in dimension C. The mean value of the data is $\mu = (8, 15, 30)$. You want to set up the view such that the user is looking at the mean data location from the point $\vec{vrp} = (10, 20, 40)$ using a view volume that has an extent of [20, 20, 20]. To properly orient the view volume, we also need to know which direction is up. We can use dimension B as a default up direction, in which case $\vec{vup} = (0, 1, 0)$.
1. Convert to view volume coordinates. If we position the viewer at the center of one end of the view volume, the process has three steps.

(a) Translate the view reference point to the origin \(T(-10, -20, -40)\)

(b) Orient the axes of the data space and the view volume. The three axes of our view space are the View Plane Normal, which is the direction we’re looking, the View Up Vector, and the U vector. We can calculate them using the following process.

\[
v\hat{p}n = \text{lookat} - \text{viewer} = (8, 15, 30) - (10, 20, 40) = (-2, -5, -10)
\]

\[
\vec{u} = v\hat{u}p \times v\hat{p}n
\]

\[
v\hat{u}p' = v\hat{p}n \times \vec{u}
\]

Use normalized versions of these three vectors to orient the axes.

(c) Once the axes are oriented, we want to shift the volume so the lower left corner of the viewing face is at the origin. If the viewing face has size \((du, dv)\), then the translation is \(T(0.5du, 0.5dv) = T(10, 10)\).

2. Convert to normalized view coordinates. Scale each dimension of the view volume to 1:

\[S\left(\frac{1}{du}, \frac{1}{dv}, \frac{1}{dw}\right) = (0.05, 0.05, 0.05).\]

3. Convert to screen coordinates. Scale the two dimensions of the view volume that are perpendicular to the view direction to the screen coordinates, flip the axes, and then translate. Note that both the X and Y axes are reversed from screen coordinates because of the way we set up our view reference coordinate system.

(a) Scale and flip the coordinate systems: \(S(-s_x, -s_y)\)

(b) Translate so the window is all positive: \(T(s_x, s_y)\)

In summary, 3D viewing from arbitrary locations is a bit more complex than 2D. The important parameters are the View Reference Point, which is the center of the view volume face through which the viewer is looking, the
**View Up Vector**, which orients the view volume, and the **View Volume Extent**, which is given in View Reference coordinates (which have units similar to the data space). From these parameters we can calculate all of the necessary transformations into normalized view coordinates. Note that, if the data space has drastically different ranges, then arbitrary viewing may not produce good visualizations without scaling the data so that distances in each dimension have roughly similar meanings. We’ll be looking at how to do this a bit later.

Acknowledgement: Most of the material in this lecture is taken from Bruce Maxwell’s spring 2012 CS251 notes.