Chapter 6

3-D Geometry Engine

6.1 Developing a Basic 3D Graphics Pipeline

The world we are representing is a 3-D world. However, we are displaying it on a 2-D surface. We need to implement a set of transformations which will take an object from our 3-D world to a 2-D surface. We will lose one coordinate, the z-coordinate. However, we need to try to keep some of the information the z-coordinate gives us. This will be accomplished using perspective. In addition, the transformations we will implement will more than likely involve resizing the various objects as our 3-D world does not have the same dimensions as the 2-D surface on which it will be projected. Furthermore, since our 3-D world is approximated with flat polygons, and flat polygons are made of lines, the transformations we will implement will simply have to take line segments in a 3-D space to a 2-D surface. Therefore, we need to develop a basic 3D engine which will transform line segment from a 3-D space to the 2-D screen. We will call this a basic 3-D graphics pipeline. We call this a pipeline because the geometric primitives making up our 3-D object will be sent through it. As they go through it, they will be transformed. When they come out of it, they will be ready to be displayed on the screen. This pipeline can be as complex or as simple as we wish. Because of time limitations in this class, we will only develop a simple 3-D graphics pipeline. It will be the starting point for a more complex one, should you decide to develop one. In this chapter, we begin with a very simple one, no hidden line removal will be implemented yet.

We will do this in several steps. The steps to be taken are outlined below:

\[
\text{projection} \quad \Rightarrow \quad \text{resizing} \quad \Rightarrow \quad \text{Screen} \quad (6.1)
\]

We will proceed backward, that is we will start with the transformations performed last in our pipeline because they are easier to explain. As we get into the detail of each transformation, we will break them into several transformations.
6.1.1 Screen Display

Let us begin with a brief review of how a computer screen functions. It is not our intention here to explain the hardware behind a computer screen. Neither is it our intention to explain all the details involved in writing to a computer screen. The programming language we will use, Java, has built-in constructs to draw the geometric primitives used to describe our objects. We will make use of these constructs. We only need to understand about screen resolution, and the coordinate system a computer screen uses.

Each displayable element in a raster display is called a pixel. The position of pixels are given in real screen coordinates where the position is indicated by a pair \((x, y)\). This is similar to representing points in a Cartesian coordinates system with a few differences:

1. Screen coordinates are integers. We can only represent points with integer coordinates. Non-integer values are therefore rounded off.
2. In many API’s, the origin at not at the lower left corner, but at the top left corner. It is then understood that the positive \(y\) direction is downward, not upward. The java API we will use does it this way. An extra transformation will have to be added in order to implement this.

If the display has a resolution of \(n_x\) by \(n_y\) pixels, then the coordinates run from 0 to \(n_x - 1\) in the \(x\) direction and from 0 to \(n_y - 1\) in the \(y\) direction.

From figure 6.1 (which appears in [SP1]), we can see that the pixel coordinates stand at the center of a grid made of rectangles. The pixel coordinates are, of course, integers. However, between any two integers, there are infinitely many points. Each point who falls in a rectangle will be transformed to the coordinate at the center of that rectangle. So, we can see that the each pixel actually extends 0.5 units in each direction around its center. The pixel centered at \((0, 0)\) is actually a rectangle \([-0.5, 0.5] \times [-0.5, 0.5]\). This means that if a screen has a resolution of \(n_x \times n_y\), its actual extent is \([-0.5, n_x - 0.5] \times [-0.5, n_y - 0.5]\).

This is important. It means that no matter the dimensions of the world we are representing, this world will have to be transformed into the rectangle \([-0.5, n_x - 0.5] \times [-0.5, n_y - 0.5]\). It is only natural that we look next at the mathematics involved when transforming a rectangle into another one.

6.1.2 Transformation a rectangle into another rectangle

Let us assume that we wish to transform all the points in the red rectangle \((R1)\) into points in the green rectangle \((R2)\) as shown on figure 6.2.

One approach is to proceed as follows:

1. Translate the first rectangle so that its lower left corner is at the origin. Let \(T_1\) be this transformation.
2. Rescale this new rectangle so that it has the same size as \(R2\). Let \(T_2\) be this transformation.
6.1. DEVELOPING A BASIC 3D GRAPHICS PIPELINE

Figure 6.1: Coordinates on a screen whose pixel resolution is $4 \times 3$

Figure 6.2: Mapping a rectangle into another rectangle
3. Translate the new resized rectangle so that its lower left corner is at \((c, d)\). Let \(T_3\) be this transformation.

If we label the matrices of each transformation the same way the transformations are labeled, then the matrix of the resulting transformation will be

\[
T_3T_2T_1
\]

where

\[
T_1 = \begin{bmatrix}
1 & 0 & -a \\
0 & 1 & -b \\
0 & 0 & 1
\end{bmatrix}
\]

\[
T_2 = \begin{bmatrix}
\frac{C-c}{A-a} & 0 & 0 \\
0 & \frac{D-d}{B-b} & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

and

\[
T_3 = \begin{bmatrix}
1 & 0 & c \\
0 & 1 & d \\
0 & 0 & 1
\end{bmatrix}
\]

Therefore,

\[
T_3T_2T_1 = \begin{bmatrix}
\frac{C-c}{A-a} & 0 & \frac{Ac-aC}{B} \\
0 & \frac{D-d}{B-b} & \frac{BD-dD}{B} \\
0 & 0 & 1
\end{bmatrix}
\] (6.2)

### 6.1.3 Transformation a 3D volume into another 3D volume

Let us assume we want to transform the volume \(V_1\) into the volume \(V_2\) as shown on figure 6.3. One approach is to proceed as follows:

1. Translate \(V_1\) so that its bottom left near corner is at the origin. Let \(T_1\) be this transformation.

2. Rescale this new volume so that it has the same size as \(V_2\). Let \(T_2\) be this transformation.

3. Translate the new resized volume so that its bottom left near corner is at \((d, e, f)\). Let \(T_3\) be this transformation.

If we label the matrices of each transformation the same way the transformations are labeled, then the matrix of the resulting transformation will be

\[
T_3T_2T_1
\]

where

\[
T_1 = \begin{bmatrix}
1 & 0 & 0 & -a \\
0 & 1 & 0 & -b \\
0 & 0 & 1 & -c \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
6.1. DEVELOPING A BASIC 3D GRAPHICS PIPELINE

Figure 6.3: Mapping a 3D volume into another 3D volume

\[
T_2 = \begin{bmatrix}
    \frac{D-d}{A-a} & 0 & 0 & 0 \\
    0 & \frac{E-e}{B-b} & 0 & 0 \\
    0 & 0 & \frac{F-f}{C-c} & 0 \\
    0 & 0 & 0 & 1
\end{bmatrix}
\]

and

\[
T_3 = \begin{bmatrix}
    1 & 0 & 0 & d \\
    0 & 1 & 0 & e \\
    0 & 0 & 1 & f \\
    0 & 0 & 0 & 1
\end{bmatrix}
\]

Therefore

\[
T_3T_2T_1 = \begin{bmatrix}
    \frac{D-d}{A-a} & 0 & 0 & \frac{Ad-aD}{A} \\
    0 & \frac{E-e}{B-b} & 0 & \frac{Be-bE}{B} \\
    0 & 0 & \frac{F-f}{C-c} & \frac{Cc-cF}{C} \\
    0 & 0 & 0 & 1
\end{bmatrix}
\]  \hspace{1cm} (6.3)

6.1.4 Transforming our 3-D World Onto the Screen: First Approach - Outline

We are now ready to discuss the representation of a 3D object on a 2D surface, in screen (or pixel) coordinates. For what follows, unless stated otherwise, we will assume that the screen resolution will be \(n_x \times n_y\) pixels. Let us assume that our world is as shown on figure 6.4 (which appears in [SP1]). That is, our
world is a cube and we are looking at it from the origin, along the $z$ axis. Our world is along the negative $z$ axis, so $n > f$. To represent our world in 2D, we will project it on a plane perpendicular to the direction we are looking at it (the gaze direction), that is on the $xy$ plane. Later we will generalize this to looking at our world from any point, in any direction. To represent our world in screen coordinates, we proceed as follows:

1. Project our world on the $xy$ plane. This is done by simply dropping the $z$ coordinate. Our world has now become a rectangle.

2. Transform the rectangle obtained in the previous step into $[-.5, n_x - .5] \times [-.5, n_y - .5]$.

3. Draw our world on the screen.

These transformations were outlined in 6.1. We now look at the details involved in carrying this out.

### 6.1.5 The Canonical View Volume

To develop the necessary theory, we first look at a simplified version of our world, called the canonical view volume. It is a cube with side of length 2, centered at the origin. In other words, $x$, $y$ and $z$ are between $-1$ and 1 (see figure 6.5). The goal is to map this cube and everything in it to a 2D screen. To achieve this, we follow the steps outlined above:

1. **Projection.** Projecting on the $xy$ plane is simply a matter of dropping the $z$ coordinate. A point of coordinates $(x_c, y_c, z_c)$ will simply become $(x_c, y_c)$. The subscript $c$ indicates it is a point in the canonical view volume. Because we started in the canonical view volume, we have

   $\begin{align*}
   -1 &\leq x_c \leq 1 \\
   -1 &\leq y_c \leq 1
   \end{align*}$
2. **Mapping.** We need to map $[-1, 1] \times [-1, 1]$ to $[-.5, n_x - .5] \times [-.5, n_y - .5]$. Using equation 6.2 with

$$a = -1$$
$$b = -1$$
$$A = 1$$
$$B = 1$$

and

$$c = -.5$$
$$d = -.5$$
$$C = n_x - .5$$
$$D = n_y - .5$$

we obtain the pixel coordinates of the resulting point. If we call this resulting point $(x_p, y_p)$, then we have:

$$
\begin{bmatrix}
  x_p \\
  y_p \\
  1
\end{bmatrix}
= \begin{bmatrix}
  \frac{n_x}{2} & 0 & \frac{n_x - 1}{2} \\
  0 & \frac{n_y}{2} & 0 \\
  0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
  x_c \\
  y_c \\
  1
\end{bmatrix}
$$

(6.4)

If in addition the $y$ axis points downward, then we must reflect across the
x-axis first. In this case, we have

\[
\begin{bmatrix}
    x_p \\
    y_p \\
    1
\end{bmatrix}
= \begin{bmatrix}
    \frac{nx}{2} & 0 & \frac{nx-1}{2} \\
    0 & \frac{nx}{2} & \frac{nx-1}{2} \\
    0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    1 & 0 & 0 \\
    0 & -1 & 0 \\
    0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    x_c \\
    y_c \\
    1
\end{bmatrix}
\]

\[
\begin{bmatrix}
    x_p \\
    y_p \\
    1
\end{bmatrix}
= \begin{bmatrix}
    \frac{nx}{2} & 0 & \frac{nx-1}{2} \\
    0 & -\frac{nx}{2} & \frac{nx-1}{2} \\
    0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    x_c \\
    y_c \\
    1
\end{bmatrix}
\tag{6.5}
\]

3. Drawing. We will use a built-in primitive to draw a line between two points in screen coordinates.

### 6.1.6 Orthographic Projection

**View from the origin, in the negative direction of the** \( z \) **axis.**

In general, our world is not limited to the canonical view volume. Let us assume our world is as in figure 6.4. For now, let us further assume that we are looking at our world from the origin, in the negative direction of the \( z \) axis. To obtain the 2D screen coordinates, we first transform our world into the canonical view volume. We then use the technique described above. In other words, we follow the following steps:

1. Transform \([l, r] \times [b, t] \times [n, f]\) to \([-1, 1] \times [-1, 1] \times [-1, 1]\)

2. Project onto the \( xy \) plane.

3. Transform \([-1, 1] \times [-1, 1]\) into \([-0.5, n_{x} - 0.5] \times [-0.5, n_{y} - 0.5]\).

4. Draw our world on the screen.

We will use the notation described above.

The first step can be done by using equation 6.3. We can actually make the operation a little simpler if we translate our world so that its center is at the origin. We then resize it so that it has the same size as the canonical view volume. The matrices describing this are:

\[
\begin{bmatrix}
    x_c \\
    y_c \\
    z_c \\
    1
\end{bmatrix}
= \begin{bmatrix}
    \frac{2}{r-l} & 0 & 0 & 0 \\
    0 & \frac{2}{t-b} & 0 & 0 \\
    0 & 0 & \frac{2}{n-f} & 0 \\
    0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    1 & 0 & 0 & -\frac{r+l}{2} \\
    0 & 1 & 0 & -\frac{t+b}{2} \\
    0 & 0 & 1 & -\frac{n+f}{2} \\
    0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    x \\
    y \\
    z \\
    1
\end{bmatrix}
\]

The next two steps will only change \( x_c \) and \( y_c \). By using equation 6.5, we see that

\[
\begin{bmatrix}
    x_p \\
    y_p \\
    z_c \\
    1
\end{bmatrix}
= \begin{bmatrix}
    \frac{nx}{2} & 0 & \frac{nx-1}{2} \\
    0 & -\frac{nx}{2} & \frac{nx-1}{2} \\
    0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    \frac{2}{r-l} & 0 & 0 & 0 \\
    0 & \frac{2}{t-b} & 0 & 0 \\
    0 & 0 & \frac{2}{n-f} & 0 \\
    0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    1 & 0 & 0 & -\frac{r+l}{2} \\
    0 & 1 & 0 & -\frac{t+b}{2} \\
    0 & 0 & 1 & -\frac{n+f}{2} \\
    0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    x \\
    y \\
    z \\
    1
\end{bmatrix}
\]
6.1. DEVELOPING A BASIC 3D GRAPHICS PIPELINE

Note that since we did not touch \( z \), it is still called \( z_c \).

Let us call

\[
M_0 = \begin{bmatrix}
\frac{n_x}{2} & 0 & 0 & \frac{n_x-1}{2} \\
0 & -\frac{n_y}{2} & 0 & \frac{n_y-1}{2} \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

We can now write an algorithm to draw 3D lines on a 2D screen. The steps are as follows:

**Algorithm 76** To draw 3D lines with end points \( a_i \) and \( b_i \), do the following:

1. Compute \( M_0 \)
2. For each \( i \), do
   - \( p = M_0 a_i \)
   - \( q = M_0 b_i \)
   - \( \text{drawline}(x_p, y_p, x_q, y_q) \)

**Linear Algebra review: Change of basis**

In order to generalize what we did above to looking at our world from any point, in any direction, we need to review some linear algebra. We need to review how to write the coordinates of a point with respect to various bases. We will simply state the results which are needed. For further detail, consult a linear algebra book.

You will recall from Linear Algebra that given a vector space, a basis for the space is a set of vectors which spans the space and is linearly independent. If \( (\vec{u}_1, \vec{u}_2, \ldots, \vec{u}_n) \) is a basis for a vector space, then every vector in the space can be expressed as a unique linear combination of the vectors in the basis. The coefficients which appear in the linear combination are the coordinates of the vector in the given basis. Though in theory any linearly independent spanning set of vectors can be used as a basis, in practice, it is more convenient to use vectors which are of length 1 (unit vectors) and pairwise orthogonal. Such a set of vectors has a special name.

**Definition 77 (Orthonormal Bases)** A basis \( (\vec{u}, \vec{v}, \vec{w}) \) is said to be an orthonormal basis if its vectors are pairwise orthogonal (perpendicular) and each vector is a unit vector (length 1).

In general, a set of vectors \( (\vec{u}_1, \vec{u}_2, \ldots, \vec{u}_n) \) form an orthonormal basis if \( \|\vec{u}_i\| = 1 \) and \( \vec{u}_i \cdot \vec{u}_j = \delta_{ij} \) for \( i = 1..n \), where \( \delta_{ij} \) is the Kronecker delta symbol (\( \delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \)).
Definition 78 (Orthogonal Matrix) An invertible matrix $M$ is said to be orthogonal if $M^{-1} = M^T$.

Theorem 79 If a matrix $M$ is formed by using for its columns the vectors from an orthogonal basis, then the matrix is orthogonal in other words $M^{-1} = M^T$.

Proof. Suppose that $(\vec{u}_1, \vec{u}_2, ..., \vec{u}_n)$ form an orthonormal basis and the columns of $M$ are made with these vectors, in other words $M = [\vec{u}_1, \vec{u}_2, ..., \vec{u}_n]$. Then the rows of $M^T$ will consist of the vectors $\vec{u}_1^T, \vec{u}_2^T, ..., \vec{u}_n^T$. Therefore, the entries of $M^T M$ will be $\vec{u}_i^T \vec{u}_j = \vec{u}_i \cdot \vec{u}_j = \delta_{ij}$. But these are the entries of the identity matrix.

Definition 80 We say that a matrix $M$ preserve lengths if for any vector $\vec{u}$ we have $\|M\vec{u}\| = \|\vec{u}\|$.

Orthogonal matrices have the important property that they preserve lengths and angles. Before we prove this important result, we prove a technical lemma.

Lemma 81 A matrix that preserves lengths also preserves angles if for any vectors $u$ and $v$ we have

$$M\vec{u} \cdot M\vec{v} = \vec{u} \cdot \vec{v}$$

Proof. Let $\theta$ be the smallest angle between $\vec{u}$ and $\vec{v}$ and $\phi$ be the smallest angle between $\vec{u}$ and $\vec{v}$. We show that $\theta = \phi$ by showing that $\cos \theta = \cos \phi$. From the properties of the dot product, we see that $\cos \theta = \frac{M\vec{u} \cdot M\vec{v}}{\|M\vec{u}\| \|M\vec{v}\|}$ By assumption, $M\vec{u} \cdot M\vec{v} = \vec{u} \cdot \vec{v}$ and $M$ preserves lengths, so we have

$$\cos \theta = \frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\| \|\vec{v}\|} = \cos \phi$$

Theorem 82 If an $n \times n$ matrix $M$ is orthogonal, then it preserves lengths and angles.

Proof. Let $\vec{u}$ and $\vec{v}$ be two arbitrary vectors. First, we prove that

$$M\vec{u} \cdot M\vec{v} = \vec{u} \cdot \vec{v} \quad (6.6)$$

To prove it, we recall that $\vec{u} \cdot \vec{v} = \vec{u}^T \vec{v}$. Therefore,

$$M\vec{u} \cdot M\vec{v} = (M\vec{u})^T (M\vec{v})$$

$$= \vec{u}^T M^T M\vec{v}$$

$$= \vec{u}^T \vec{v} \text{ since } M \text{ is orthogonal}$$

$$= \vec{u} \cdot \vec{v}$$

This implies that $M$ preserves lengths since

$$\|M\vec{u}\|^2 = M\vec{u} \cdot M\vec{u}$$

$$= \vec{u} \cdot \vec{u} \text{ by equation 6.6}$$

$$= \|\vec{u}\|^2$$
6.1. DEVELOPING A BASIC 3D GRAPHICS PIPELINE

By the lemma, it also implies that $M$ preserves lengths. □

We are now ready to work on the change of basis formula. When we say that a vector $\vec{p}$ has coordinates $(x, y, z)$ with respect to a basis $(\vec{i}, \vec{j}, \vec{k})$, we mean that the component of $\vec{p}$ in the $\vec{i}$ direction is $x$, it is $y$ in the $\vec{j}$ direction and $z$ in the $\vec{k}$ direction. In this case, we write

$$\vec{p} = x \vec{i} + y \vec{j} + z \vec{k}$$

Now, suppose that we have a point we will denote $\vec{e}$ and another orthonormal basis we will call $(\vec{u}, \vec{v}, \vec{w})$. Suppose further that $\vec{e} = (x_e, y_e, z_e)$ and $\vec{u} = (x_u, y_u, z_u)$, $\vec{v} = (x_v, y_v, z_v)$, $\vec{w} = (x_w, y_w, z_w)$.

We adopt the convention to use the basis vectors for the coordinate name and the name of the vector we are writing the coordinates of as a subscript (we use $x, y$ and $z$ for the standard basis). So, $(x_u, y_u, z_u)$ means the coordinates of $\vec{u}$ written in the basis $(\vec{i}, \vec{j}, \vec{k})$. With this convention, the coordinates of $\vec{p}$ in the $(\vec{u}, \vec{v}, \vec{w})$ basis with origin $\vec{e}$ are $(u_p, v_p, w_p)$. The goal is to find the relationship between $(u_p, v_p, w_p)$ and $(x_p, y_p, z_p)$.

On one hand, we have

$$\vec{p} = x \vec{i} + y \vec{j} + z \vec{k} \quad (6.7)$$

(we assume $\vec{o}$ is the traditional origin $(0, 0, 0)$). On the other hand

$$\vec{p} = \vec{e} + u_p \vec{u} + v_p \vec{v} + w_p \vec{w} \quad (6.8)$$

$$= \vec{e} + u_p \left( x_u \vec{i} + y_u \vec{j} + z_u \vec{k} \right)$$

$$+ v_p \left( x_v \vec{i} + y_v \vec{j} + z_v \vec{k} \right)$$

$$+ w_p \left( x_w \vec{i} + y_w \vec{j} + z_w \vec{k} \right)$$

Setting equations 6.7 and 6.8 equal, and using homogeneous coordinates, we obtain

$$\begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & x_e \\ 0 & 1 & 0 & y_e \\ 0 & 0 & 1 & z_e \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_u & x_v & x_w & 0 \\ y_u & y_v & y_w & 0 \\ z_u & z_v & z_w & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_p \\ v_p \\ w_p \\ 1 \end{bmatrix}$$
Therefore,

\[
\begin{bmatrix}
    u_p \\
v_p \\
w_p \\
1
\end{bmatrix} = \begin{bmatrix}
    1 & 0 & 0 & x_e \\
    0 & 1 & 0 & y_e \\
    0 & 0 & 1 & z_e \\
0 & 0 & 0 & 1
\end{bmatrix}^{-1} \begin{bmatrix}
    x_u & x_v & x_w & 0 \\
y_u & y_v & y_w & 0 \\
z_u & z_v & z_w & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}^{-1} \begin{bmatrix}
x_p \\
y_p \\
z_p \\
1
\end{bmatrix}
\]

\[
\begin{bmatrix}
    u_p \\
v_p \\
w_p \\
1
\end{bmatrix} = \begin{bmatrix}
x_u & x_v & x_w & 0 \\
y_u & y_v & y_w & 0 \\
z_u & z_v & z_w & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}^{-1} \begin{bmatrix}
1 & 0 & 0 & x_e \\
0 & 1 & 0 & y_e \\
0 & 0 & 1 & z_e \\
0 & 0 & 0 & 1
\end{bmatrix}^{-1} \begin{bmatrix}
x_p \\
y_p \\
z_p \\
1
\end{bmatrix}
\]

The first matrix is an orthonormal matrix (why?). Therefore, its inverse is its transpose. The second matrix is a translation matrix, its inverse is obtained by simply negating \(x_e, y_e\) and \(z_e\). Therefore,

\[
\begin{bmatrix}
    u_p \\
v_p \\
w_p \\
1
\end{bmatrix} = \begin{bmatrix}
x_u & y_u & z_u & 0 \\
x_v & y_v & z_v & 0 \\
x_w & y_w & z_w & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
1 & 0 & 0 & -x_e \\
0 & 1 & 0 & -y_e \\
0 & 0 & 1 & -z_e \\
0 & 0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
x_p \\
y_p \\
z_p \\
1
\end{bmatrix}
\]

View from any point, in any direction.

We are now ready to write the transformation which will give the 2D representation of our world, looked at from any point, in any direction. There are many possible conventions to specify the viewer’s position and orientation. We will use the following one. We will assume we are given:

1. The eye position, denoted \(e\).
2. The gaze direction, denoted \(\vec{g}\).
3. The view-up vector, denoted \(\vec{t}\).

From these, we set up a coordinate system with origin \(e\) and right-handed orthonormal basis \((\vec{u}, \vec{v}, \vec{w})\). The coordinate system we are trying to obtain is shown in figure 6.6. In this new coordinate system, we will be looking at our world as shown in figure The eye position is the location the eye sees from. If you think of a picture being taken, it is where the center of the lens is. The gaze is the direction the viewer is looking. We obtain \(\vec{u}, \vec{v}\) and \(\vec{w}\) as follows:

\[
\vec{w} = \frac{-\vec{g}}{\|\vec{g}\|}
\]

(we want \(\vec{w}\) to be of length 1. Think of \(\vec{w}\) as playing the role of \(\vec{z}\)).

\[
\vec{u} = \frac{\vec{t} \times \vec{w}}{\|\vec{t} \times \vec{w}\|}
\]
Figure 6.6: Coordinate system created from the eye position, the gaze direction and the view-up vector.

Figure 6.7: Our world in the new coordinate system
\[ \vec{v} = \vec{w} \times \vec{u} \]

No need to divide by the norm of \( \vec{w} \times \vec{u} \). It will be 1, since both vectors are unit vectors which are orthogonal. Figure 6.6 shows this new basis.

If the points we wish to transform were stored in terms of the new coordinates, we would be done. What is left to do is to project onto the \( vw \) plane, then transform the resulting points so they fit in the rectangle \([-0.5,n_x - 0.5] \times [-0.5,n_y - 0.5]\). However, our points are expressed in terms of the original basis. So, we must first perform a coordinate change. If we use the same notation as in the previous section, then the matrix which will perform this change is

\[
M_v = \begin{bmatrix}
x_u & y_u & z_u & 0 \\
x_v & y_v & z_v & 0 \\
x_w & y_w & z_w & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
1 & 0 & 0 & -x_e \\
0 & 1 & 0 & -y_e \\
0 & 0 & 1 & -z_e \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

We have the following new algorithm:

**Algorithm 83** To draw 3D lines with end points \( a_i \) and \( b_i \), do the following:

1. Compute \( M_v \)
2. Compute \( M_0 \)
3. Set \( M = M_0M_v \)
4. For each \( i \), do
   - \( p = M a_i \)
   - \( q = M b_i \)
   - \( \text{drawline}(x_p,y_p,x_q,y_q) \)

### 6.1.7 Assignment

1. Review and study this document. Be ready to discuss any questions you may have about it.
2. Discuss the approach we have taken, i.e. comment on its merits as well as drawbacks. How would you improve it?
3. Why is \( \vec{w} \times \vec{u} \) a unit vector?