Linear Algebra in Computer Graphics

Navya (2203306), Sai Devesh(2203103)

Computer Graphics

Computer Graphics involves the creation, synthesis, manipulation, and utilization of visual information using computers. Today, we will explore the mathematical foundations behind computer graphics.

Motivation:

The computer graphics used in films and video games involve three main stages:

1. A 3D model of the scene objects is created;

2. Converting the model into numerous small polygons that approximate its surfaces;

3. Applying a **linear** transformation to these polygons results in a 2D representation that can be rendered on a flat screen.

Each stage involves fascinating mathematical concepts, but today we'll focus on the linear transformations that occur in the third stage.

1 2-Dimensional Graphics

Examples of computer graphics are those which belong to 2 dimensions. Common 2D graphics include text. For example, the vertices of the letter H can be represented by the following data matrix D:

$$D = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 2 & 0 & 1 & 2 \end{bmatrix}$$

Letter Shown Here:

1.1 Scaling

There are many types of transformations that these graphics can undergo. The first one we will consider is scaling. A point in the xy-plane coordinates are given by (x,y) or

$$A = \begin{bmatrix} X \\ Y \end{bmatrix}$$

The scaling transformation is given by the matrix $S = \begin{bmatrix} C_1 & 0 \\ 0 & C_2 \end{bmatrix}$, where the C_i's are scalars. The scaling transforms the coordinates (x, y) into (C_1x, C_2y). In mathematical terms, the transformation is given by the multiplication of the matrices S and A:

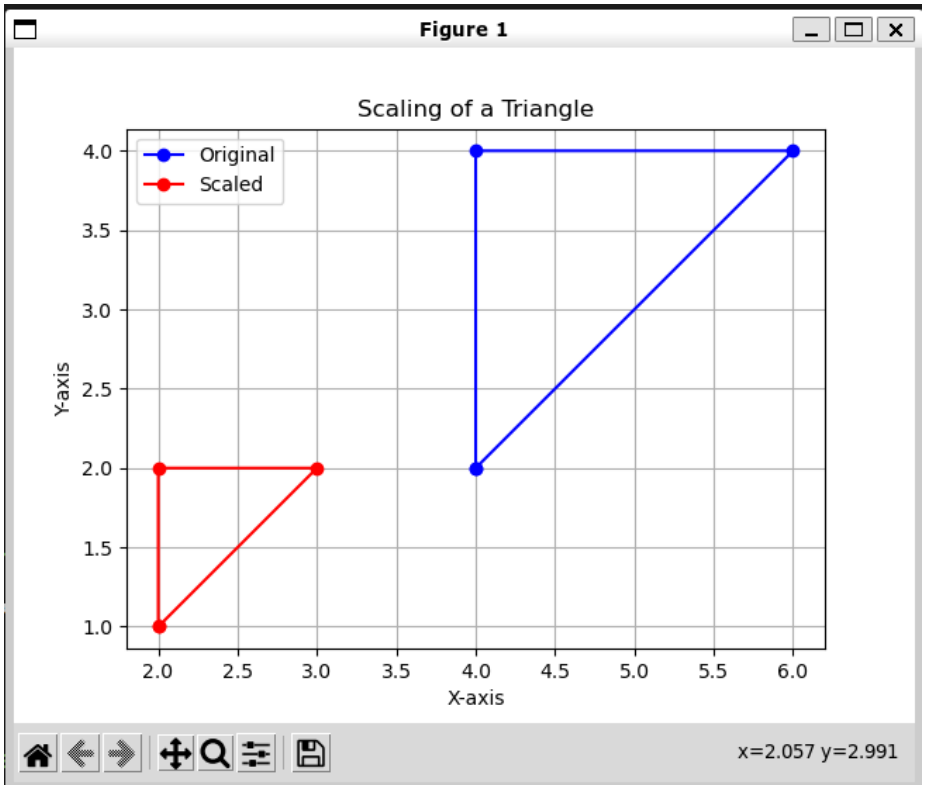
$$[S]\,[A] = \begin{bmatrix} C_1 & 0 \\ 0 & C_2 \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} C_1X \\ C_2Y \end{bmatrix}$$

Example: Now let's examine how scaling affects our example H. Let $C_1 = 2$ and $C_2 = 2$, our new coordinates for D are:

$$D = \begin{bmatrix} 0 & 0 & 0 & 2 & 2 & 2 \\ 0 & 2 & 4 & 0 & 2 & 4 \end{bmatrix}$$

The new H is now twice as long in the X-direction and twice as long in the Y-direction.

Output for a triangle from the samplecode-1. Link(samplecode-1Scale.py)



1.2 Rotation

Next, we have the transformation of rotation. A more complex transformation, rotation changes the orientation of the image about some axis, in our case either X or Y. Counter-Clockwise rotations have the rotational matrix

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

A variation of the form of the rotational matrix for counter-clockwise rotation is given by,

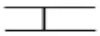
$$R(-\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

The counter-clockwise rotation of matrix A is given by:

$$R(\theta) \cdot A = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} X\cos(\theta) - Y\sin(\theta) \\ X\sin(\theta) + Y\cos(\theta) \end{bmatrix}$$

Example

Now let us visually examine a rotation of 90° of our example letter H:



Mathematically speaking, the rotation of 90° occurs when $\theta = \frac{\pi}{2}$. So now we can view the mathematics that made this transformation possible:

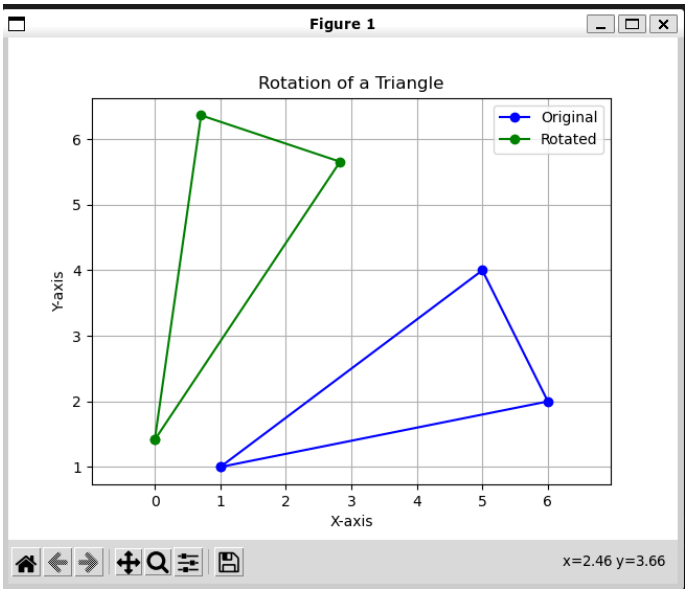
$$R(\theta) \cdot D = \begin{bmatrix} \cos(\frac{\pi}{2}) & -\sin(\frac{\pi}{2}) \\ \sin(\frac{\pi}{2}) & \cos(\frac{\pi}{2}) \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 0 & 1 & 2 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 0 & 1 & 2 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & -1 & -2 & 0 & -1 & 2 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

If we refer to both the earlier visualization and mathematical display we can that letter H has been rotated 90° about the origin.

Output for a triangle from samplecode-2. Link(samplecode-2rotate.py)



1.3 Translation

Manipulating graphics objects using matrix multiplication is very convenient. However, there is a common operation that is not a linear transformation: translation – that is, motion.

Remember that a transformation $T(x)$ is linear if:

The transformation maps the zero vector into the zero vector.

$$T(x+y) = T(x) + T(y) \quad \text{and} \quad T(cx) = cT(x)$$

If $T(x)$ is a translation, then $T(x) = x+b$ for some b. But then $T(0) = b(\neq 0)$ and $T(x+y) \neq T(x) + T(y)$.
Hence translation is not a linear transformation

1.4 Homogeneous Coordinates

There is a nice way to avoid this difficulty, while keeping things linear. We will use what are called **homogeneous coordinates**. That is, we add another coordinate. We shall explain it in terms of 3D, so as to generalize and avoid revisitng it again in 3D. We identify each point:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \in \mathbb{R}^3 \quad \text{with a corresponding point} \quad \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \in \mathbb{R}^4$$

The coordinates of the point in $\mathbb{R}^4$ are the **homogeneous coordinates** for the point in $\mathbb{R}^3$. The extra component gives us a constant that we can scale and add to the other coordinates as needed, via matrix multiplication. This means for 3D graphics, all transformation matrices are 4×4.

1.5 Constructing Matrices for Homogeneous Coordinates

For any transformation A that is linear in $\mathbb{R}^3$ (such as scaling, rotation, reflection, shearing, etc.), we can construct the corresponding matrix for homogeneous coordinates quite simply:
If

$$A = \begin{bmatrix} \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \end{bmatrix}$$

Then the corresponding transformation for homogeneous coordinates is:

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

In other words, when performing a linear transformation on x, y, and z, one simply 'carries along' the extra coordinate without modifying it.

Now we shall see scaling, translational and rotational matrices induced with homogeneous coordinates.

Scaling

$$3D - \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad 2D - \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Translation

The translation matrix is given by:

$$3D - \begin{bmatrix} 1 & 0 & 0 & h \\ 0 & 1 & 0 & k \\ 0 & 0 & 1 & m \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad 2D - \begin{bmatrix} 1 & 0 & h \\ 0 & 1 & k \\ 0 & 0 & 1 \end{bmatrix}$$

Rotation

The rotation matrices around the x-, y-, and z- axes counterclockwise (looking towards the origin) are defined as follows:

IN 3D:

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

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

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

In 2D:

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

NOTE:

- It is not strictly necessary to use homogeneous coordinates for rotation and scaling transformations alone in any dimension (n). These transformations can be represented as linear transformations with standard nxn matrices.
- However, when you need to combine translation with rotation or scaling, homogeneous coordinates become necessary to handle both types of transformations together in a unified framework.
- Using homogeneous coordinates for all transformations (translation, rotation, and scaling) simplifies the process of combining them, making it more efficient and consistent.

So from here on, we shall use homogeneous coordinates to represent matrices for scaling, translation, and rotation.

Additionally, the code implementation of all these transformations— scaling, translation, and rotation was carried out using homogeneous coordinates in matrix form for consistency and efficiency.

Example

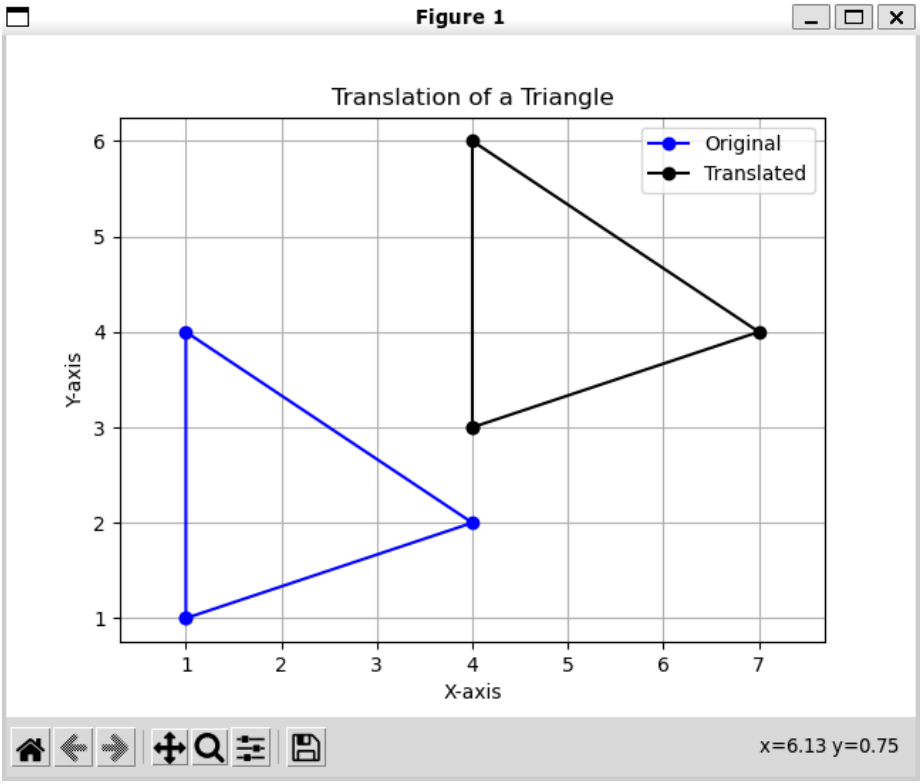
Let's say we want to move a point $(1, -2, 4)$ to location $(4, 3, 3)$ i.e $(1+3, -2+5, 4+-1)$. We represent the point in homogeneous coordinates as:

$$\begin{bmatrix} 1 \\ -2 \\ 4 \\ 1 \end{bmatrix}$$

The transformation corresponding to this 'translation' is:

$$\begin{bmatrix} 1 & 0 & 0 & 3 \\ 0 & 1 & 0 & 5 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -2 \\ 4 \\ 1 \end{bmatrix} = \begin{bmatrix} 1+3 \\ -2+5 \\ 4+-1 \\ 1 \end{bmatrix} = \begin{bmatrix} 4 \\ 3 \\ 3 \\ 1 \end{bmatrix}$$

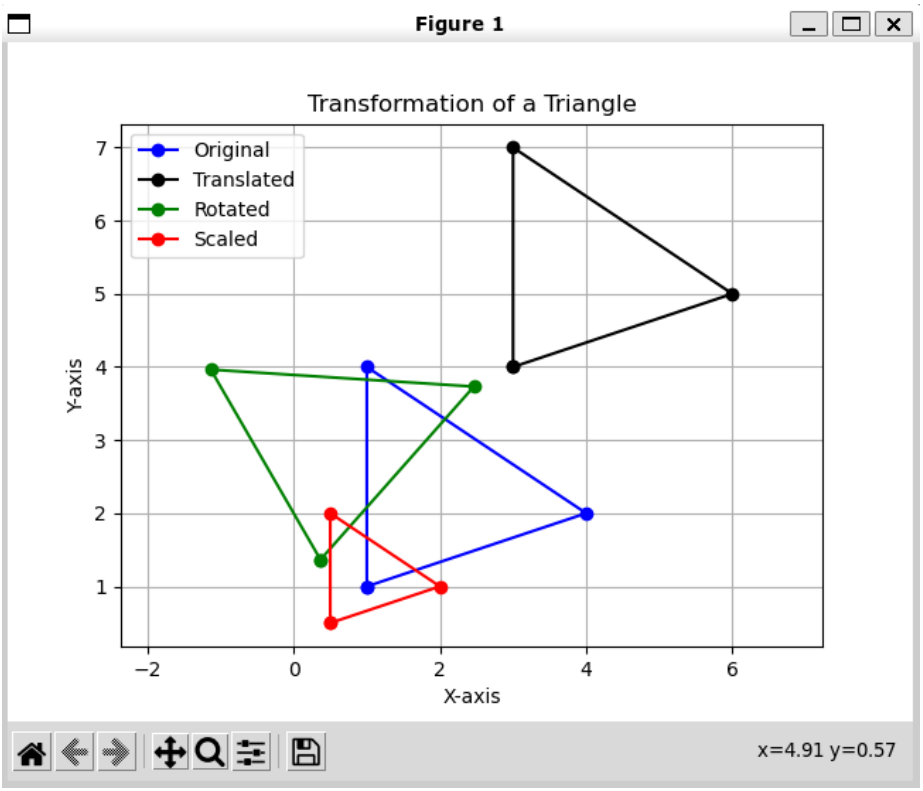
Output for a triangle from samplecode-3. Link(samplecode-3Translate.py)



If we only consider x, y, and z, this is not a linear transformation. But of course, in $\mathbb{R}^4$, this is a linear transformation. We have 'sheared' in the fourth dimension, which affects the other three. A very useful trick!

1.6 Example

Output for a triangle from samplecode-4. Link(samplecode-4TRS2d.py)
Let us now include all 3 transformations in a single frame.
Let us apply the following on the given triangle
Point A [1, 1, 1]
Point B [4, 2, 1]
Point C [1, 4, 1]
The z-coordinates are added for homogeneity.
Input:
Enter translation along x-axis (tx): 2
Enter translation along y-axis (ty): 3
Enter rotation angle in degrees (θ): 30
Enter scaling factor (s): 0.5



1.7 Composite Transformations

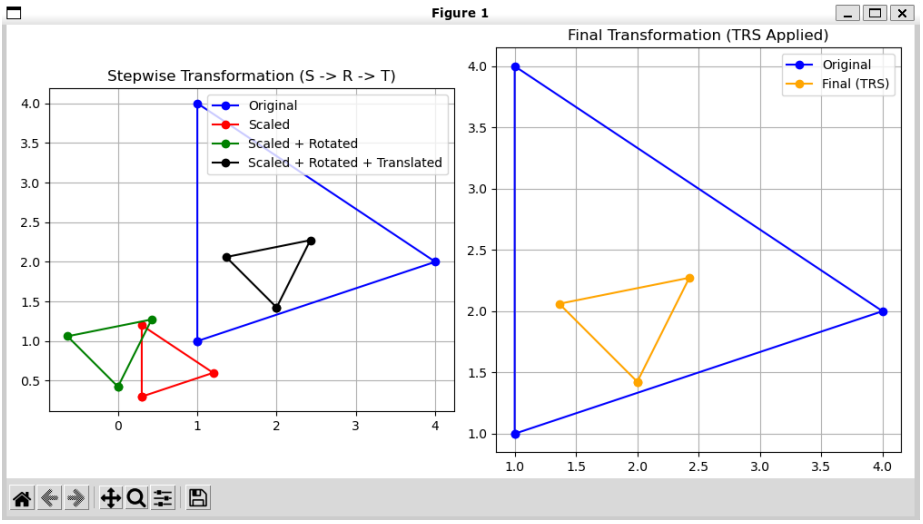
Now that we have gone over these basic transformations, it is now time to combine them to examine the net result of how multiple transformations affect an image or movement. Movement of the image is caused by the multiplication of the scaling, translational, rotational, and other transformational matrices with the coordinate matrix. The result of these matrix multiplications are called Composite Transformations.

To finish up our view in the world of 2-Dimensional graphics, let's mathematically examine a composite transformation of our example letter H using the scaling, translational, and rotational matrices we've already established. The composite transformation of H can be represented by:

$$[S][T][R(\theta)][D] = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 3 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 2 & 0 & 1 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & -2 & -4 & 0 & -2 & -4 \\ 0 & 0 & 0 & 20 & 20 & 20 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

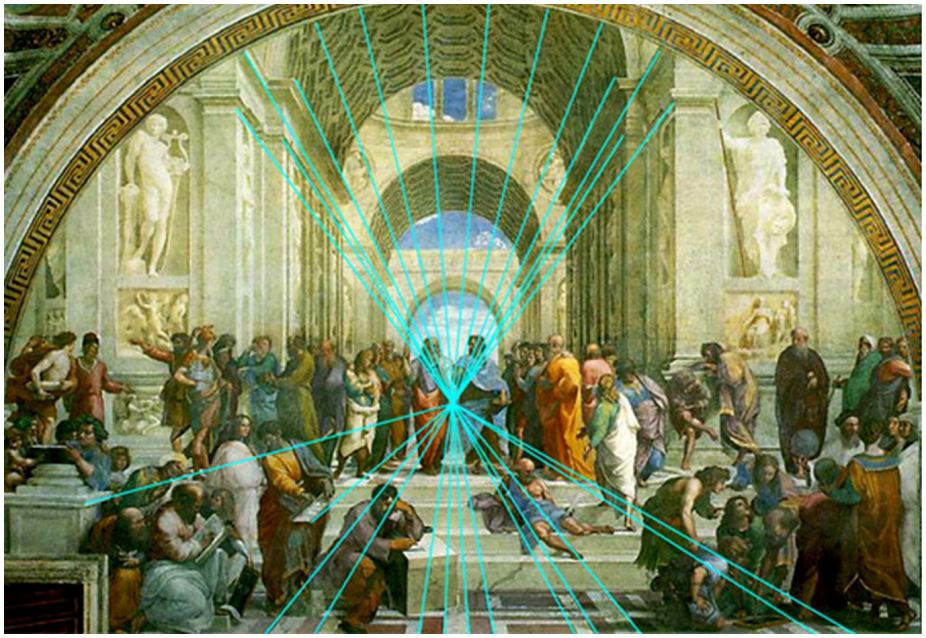
Output for a triangle from samplecode-5. Link(samplecode-5Compose.py) As we can see, the left image scaled first by 0.3, then rotated by 45° and then translated by (2,1). The right image is generated by applying the product of the translation, rotation and scaling matrices on the original triangle.



2 3-Dimensional Graphics

2.1 Perspective Projections

- There is another nonlinear transformation that is important in computer graphics: **perspective**.
- Homogeneous coordinates allow us to capture this transformation as a linear transformation in $\mathbb{R}^4$.
- The eye, or a camera, captures light (essentially) in a single location, and hence gathers light rays that are converging. To portray a scene with a realistic appearance, it is necessary to reproduce this effect.The effect can be thought of as: *"nearer objects are larger than further objects."*
- This was (re)discovered by Renaissance artists, supposedly first by Filippo Brunelleschi, around 1405.
- Here is a famous example: Raphael's *School of Athens* (1510).

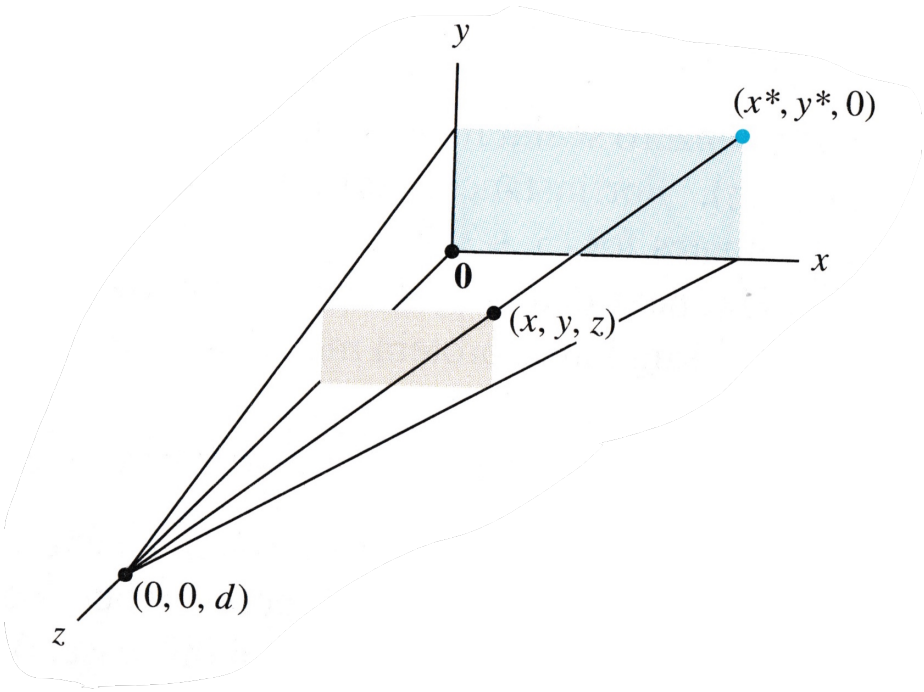


When we view 3-Dimensional objects on a computer screen, we do not actually see the images themselves, but rather we see the projections of these images onto the viewing plane, or computer screen.
The mind reconstructs a sense of three dimensions from the two dimensional

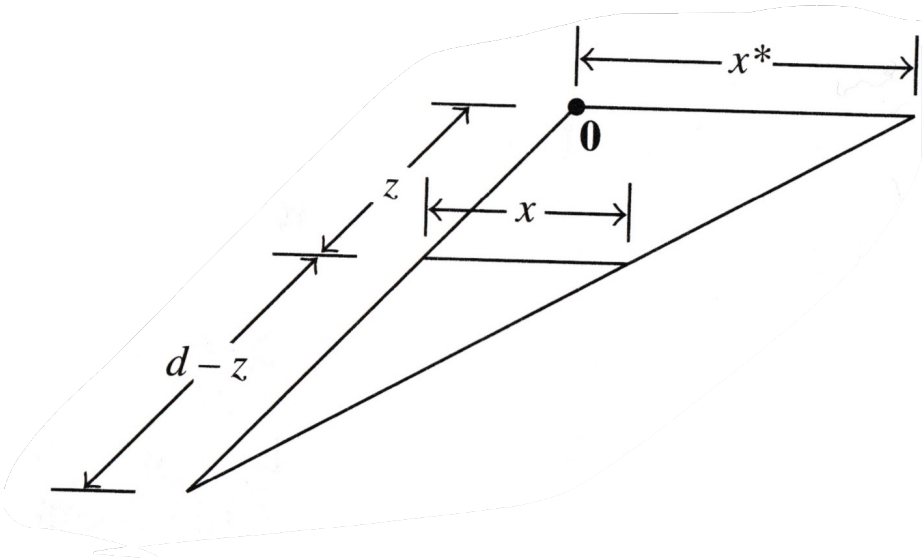
information presented to it by the retina.
This is done by sophisticated processing in the visual cortex, which is a fairly large portion of the brain.

2.1.1 Computing Perspective

A standard setup for computing a perspective transformation is as shown in this figure:



- For simplicity, we will let the xy-plane represent the screen. This is called the viewing plane.
- The eye/camera is located on the z-axis, at the point (0,0,d). This is call the center of projection.
- A perspective projection maps each point (x,y,z) onto an image point (x*,y*,0) so that the centre of projection and the two points are all on the same line.



2.1.2 Projection Using Similar Triangles

The way to compute the projection is by using similar triangles. The triangle in the xz-plane shows the lengths of corresponding line segments. Similar triangles show that:

$$\frac{x^*}{d} = \frac{x}{d-z}$$

and thus:

$$x^* = \frac{dx}{d-z} = x \cdot \frac{1}{1-\frac{z}{d}}$$

Notice that the function $T : (x, z) \mapsto \frac{x}{1-\frac{z}{d}}$ is not linear.

2.1.3 Homogeneous Coordinates for Linear Perspective

Using homogeneous coordinates, we can construct a linear version of T in $\mathbb{R}^4$.

We establish the following convention: we allow the fourth coordinate to vary away from 1. However, when we plot, for a point:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

we will plot the point:

$$\begin{bmatrix} \frac{x}{h} \\ \frac{y}{h} \\ \frac{z}{h} \end{bmatrix}.$$

In this way, by dividing the x, y, z coordinates by the h-coordinate, we can implement a nonlinear transformation in $\mathbb{R}^3$.

2.1.4 Perspective Transform in Homogeneous Coordinates

To implement the perspective transform, we want:

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \mapsto \begin{bmatrix} \frac{x}{1-\frac{z}{d}} \\ \frac{y}{1-\frac{z}{d}} \\ 0 \\ 1 \end{bmatrix}.$$

We achieve this by mapping to:

$$\begin{bmatrix} x \\ y \\ 0 \\ 1-\frac{z}{d} \end{bmatrix}.$$

Then, when we plot (dividing x and y by the h-value), we will get the proper perspective transform.

The matrix that implements this transformation is quite simple:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{d} \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Thus:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{d} & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 0 \\ 1-\frac{z}{d} \end{bmatrix}.$$

Output from samplecode-6. with video showing how a 2d figure is visualized as a 3d image using perspective projection. The image 2 is an instant of a video which rotates non stop to how the 3d image. Code: Link(samplecode-6Perspective-projection.py) **Video:**Link(perspective-vid)

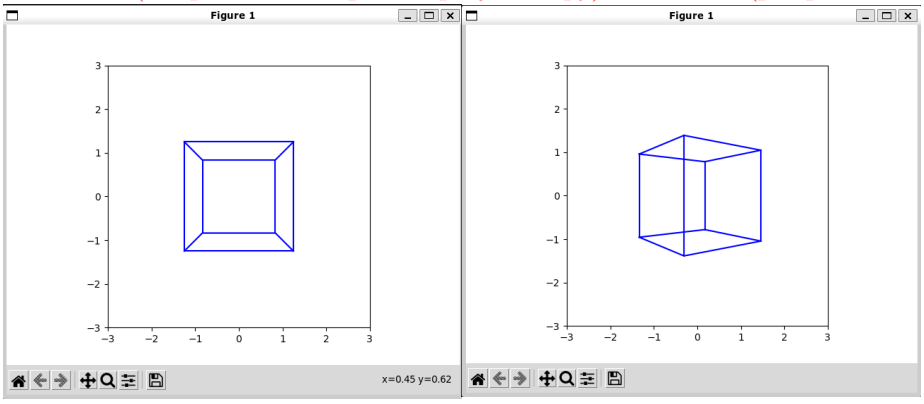


Image1 Image 2

2.2 Scaling

Scaling in 3D involves resizing an object along the X, Y, and Z axes. This is controlled by scaling factors (C1, C2, C3). You can scale along each axis independently.

$$S = \begin{bmatrix} C_1 & 0 & 0 \\ 0 & C_2 & 0 \\ 0 & 0 & C_3 \end{bmatrix}$$

The scaling transformation, like in 2-Dimensions, is represented by the matrix multiplication of the Scaling Matrix and coordinate matrix A:

$$[S]\,[A] = \begin{bmatrix} C_1 & 0 & 0 \\ 0 & C_2 & 0 \\ 0 & 0 & C_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} C_1X \\ C_2Y \\ C_3Z \end{bmatrix}$$

Example:

Give the matrix for a scaled cube with original coordinates given by

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 2 & 2 & 2 & 2 \\ 0 & 2 & 2 & 0 & 0 & 2 & 2 & 0 \\ 0 & 0 & -2 & -2 & 0 & 0 & -2 & -2 \end{bmatrix}$$

and the scaling matrix is

$$S = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$$

Solution:

$$[S]\,[A] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 0 & 2 & 2 & 2 & 2 \\ 0 & 2 & 2 & 0 & 0 & 2 & 2 & 0 \\ 0 & 0 & -2 & -2 & 0 & 0 & -2 & -2 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 0 & 0 & 2 & 2 & 2 & 2 \\ 0 & 4 & 4 & 0 & 0 & 4 & 4 & 0 \\ 0 & 0 & -6 & -6 & 0 & 0 & -6 & -6 \end{bmatrix}$$

From calculating the new matrix, we can see that the cube isn't scaled in the X-direction, scaled to twice its size in the positive Y-direction, and scaled to triple its size in the negative Z-direction.

2.3 Translation

Translation in 3D is like moving an object from one place to another. It's the same as moving an object in 2D, but you can also move it up or down (along the Z-axis).
If you only look at the object from the side (XY-plane), it might seem like it's

getting bigger or smaller, but that's just an illusion. The computer knows it's only moving forward or backward. This is true for all directions and planes.

In 3-dimensions, translation is represented by the matrix multiplication of the translation vector

$$T = \begin{bmatrix} 1 & 0 & 0 & X_0 \\ 0 & 1 & 0 & Y_0 \\ 0 & 0 & 1 & Z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

and the coordinate matrix

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

Mathematically speaking, we can represent the 3-Dimensional translation transformation with:

$$[T][A] = \begin{bmatrix} 1 & 0 & 0 & X_0 \\ 0 & 1 & 0 & Y_0 \\ 0 & 0 & 1 & Z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} X + X_0 \\ Y + Y_0 \\ Z + Z_0 \\ 1 \end{bmatrix}$$

Example: Give the matrix for the translation of a point (5, 3, 8, 1) by the vector $\mathbf{p} = (-4, -6, 3)$.
Solution: The matrix that maps $(X, Y, Z, 1) \rightarrow (X - 4, Y - 6, Z + 3, 1)$ is given by

$$[T] = \begin{bmatrix} 1 & 0 & 0 & -4 \\ 0 & 1 & 0 & -6 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A = \begin{bmatrix} 5 \\ 3 \\ -8 \\ 1 \end{bmatrix}$$

$$[T][A] = \begin{bmatrix} 1 & 0 & 0 & -4 \\ 0 & 1 & 0 & -6 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 5 \\ 3 \\ -8 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ -3 \\ -5 \\ 1 \end{bmatrix}$$

2.4 Rotation

We finally arrive at rotation in 3-Dimensions. Just like scaling and translating, rotation in three dimensions is fundamentally the same as rotation in 3-Dimensions, with the exception that we also can rotate about the Z-axis. The rotations about the X, Y, and Z axes are given by:

- $R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix}$ (Rotates the y-axis towards the z-axis)
- $R_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}$ (Rotates the z-axis towards the x-axis)
- $R_z(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$ (Rotates the x-axis towards the y-axis)

We can display the post-reflection coordinates as:

- $R(\theta)_X$: $(X, Y, Z) \longrightarrow (X, Y\cos(\theta) - Z\sin(\theta), Z\cos(\theta) + Y\sin(\theta))$
- $R(\theta)_Y$: $(X, Y, Z) \longrightarrow (X\cos(\theta) + Z\sin(\theta), Y, -X\sin(\theta) + Z\cos(\theta))$
- $R(\theta)_Z$: $(X, Y, Z) \longrightarrow (X\cos(\theta) - Y\sin(\theta), Y\cos(\theta) + X\sin(\theta), Z)$

Example: Give the final Rotational Matrix, R, for an image that is first rotated about the X-axis 47°, then about the Y-axis 52°, and then about the Z-axis −30°.

Solution: R can be mathematically represented by the matrix multiplications of the 3 rotational matrices of the X, Y, and Z axes. Mathematically speaking:

$$R = [R(\theta)_X]\,[R(\theta)_Y]\,[R(\theta)_Z]$$

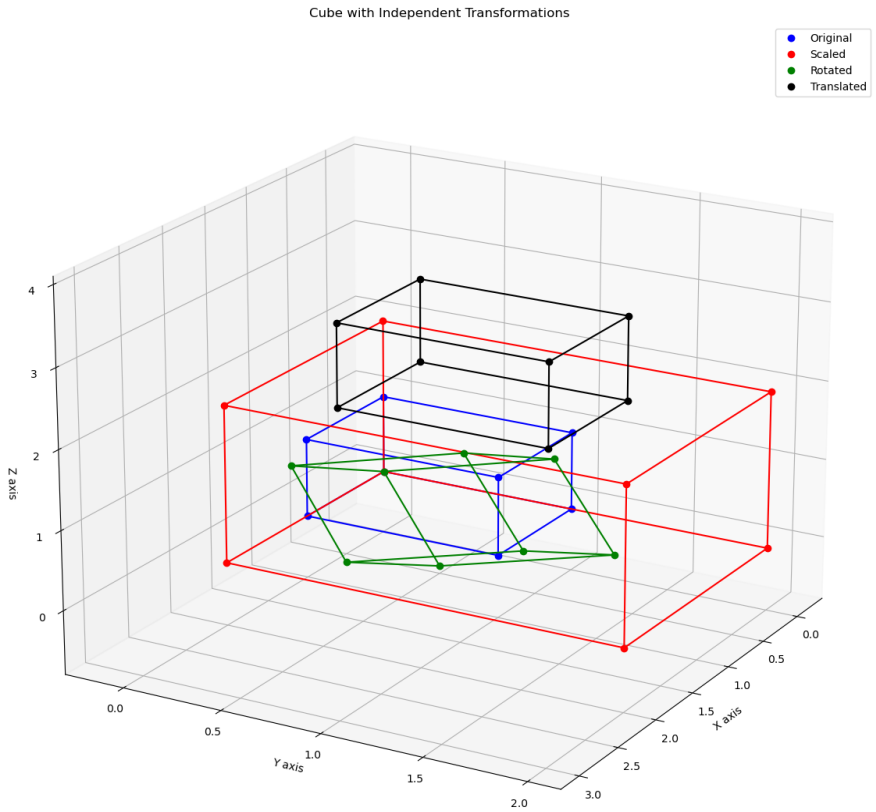
$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta)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

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(47°) & -\sin(47°) \\ 0 & \sin(47°) & \cos(47°) \end{bmatrix} \begin{bmatrix} \cos(52°) & 0 & \sin(52°) \\ 0 & 1 & 0 \\ -\sin(52°) & 0 & \cos(52°) \end{bmatrix} \begin{bmatrix} \cos(-30°) & -\sin(-30°) & 0 \\ \sin(-30°) & \cos(-30°) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.533 & 0.308 & 0.788 \\ 0.158 & 0.879 & -0.450 \\ -0.831 & 0.365 & 0.420 \end{bmatrix}$$

2.5 Example

Output for a cube from samplecode-7. Link(samplecode-7TRS-3D.py) Let us now include all 3 transformations in a single frame.
Let us apply the following on a given unit cube
Input:
Enter translation along x-axis (tx): 2
Enter translation along y-axis (ty): 1
Enter translation along z-axis (tz): 3
Enter rotation angle around x-axis in degrees: 45
Enter rotation angle around y-axis in degrees: 45
Enter rotation angle around z-axis in degrees: 45
Enter scaling factor (s): 2



3 Applications of Linear Algebra in computer graphics

Linear algebra is foundational in computer graphics, where it is used to model, manipulate, and render 2D and 3D objects. Here are some of the primary applications of linear algebra in computer graphics:

3.1 Transformations (Scaling, Rotation, and Translation)

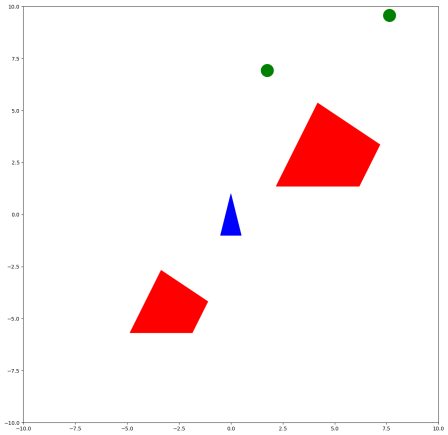
- **Scaling:** Linear algebra helps scale objects by changing their size in different directions using scaling matrices.
  - **Example:** Resizing a 3D object to make it larger or smaller.
- **Rotation:** Rotation matrices are used to rotate objects around an axis in 2D or 3D space.
  - **Example:** Rotating a spaceship in a game or the camera view.
- **Translation:** Matrices are used to translate (move) objects from one position to another.
  - **Example:** Moving an object across the screen or simulating a character's movement.

3.1.1 Example: Space Asteroid Game

It is a simple game in which a spaceship moves in space trying to avoid the asteroids (reduces score by 1) and collecting power ups (increases score by 1).

- Movement of spaceship is done using Up and Down keys. For this translation was used.
- Changing direction is done using Left and Right keys. For this rotation was used.
- On collecting a powerup, size of the asteroid reduces and on crashing the asteroid, the size of asteroid increases. For this scaling was used.

Game-samplecode-8: Game-code.py **Game-video:** Game-vid



3.2 Projections

- Using projection matrices to simulate depth and perspective when displaying 3D scenes on a 2D screen.
  - **Example:** Rendering 3D environments in video games.

NOTE: This has been discussed above in detail with videos.

3.3 3D Modeling and Animations

- Manipulating vertices, edges, and faces of 3D objects using linear transformations to animate or morph models.
  - **Example:** Transforming the vertices of a character's 3D mesh for animation.
- Applying distortions to shapes in animations and visual effects.
  - **Example:** Rendering 3D environments in video games.

3.4 Camera Movement and View Transformations

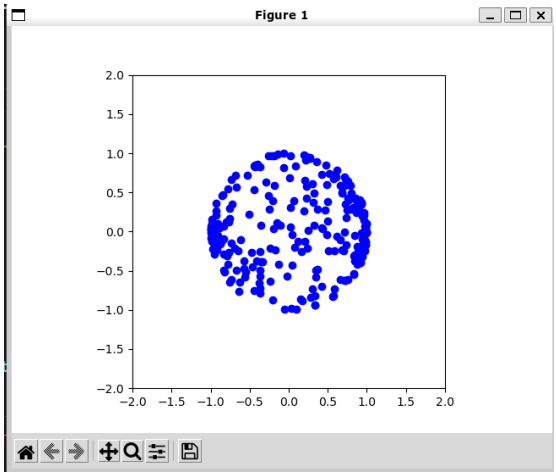
- Using view matrices to move and rotate the camera in a 3D scene.
  - **Example:** Changing camera angles in a 3D game.

3.4.1 Example:

Imagine that we want to circle a camera around a 3D figure like a ball. This is equivalent to rotating the ball around the y axis.

- A code has been implemented to do the above where we used rotation around y axis as a linear transformation on random points (x, y, z) generated on a sphere and projected onto the xy plane and then rotated continuously to achieve our goal. **Code:**Link(Ball-povcode.py)

- Here we have used concepts of 3D modelling, where vertices on the 3D model were used and then rotation was used as a linear transformation to perform the action of camera. **Video Implementation:**Link(Ball-camera-vid)



3.5 Simulations and Physics

- **Rigid Body Dynamics:** Simulating physical forces, collisions, and deformations using linear algebra.Manipulating vertices, edges, and faces of 3D objects using linear transformations to animate or morph models.
  - **Example:** Simulating car crashes in games, calculating forces and deformations.
- **Deformations:** Modeling elastic deformations such as bending, stretching, or compressing objects in simulations.
  - **Example:** Animating soft objects in films or simulations. Simulting cloth using spring mass system, simulating hair,

4 Conclusion

In conclusion, linear algebra plays a vital role in the field of computer graphics, providing the mathematical framework necessary for the representation and manipulation of visual information. Its applications, ranging from transformations

and projections to modeling, rendering, and animations, enable the creation of realistic and immersive graphics. By utilizing matrices and vectors, we can efficiently perform complex operations such as scaling, rotation, translation, and shading. As technology continues to advance, the importance of linear algebra in graphics will only grow, underpinning developments in gaming, virtual reality, computer-aided design, and more. Understanding these mathematical principles is essential for anyone looking to innovate and excel in the dynamic field of computer graphics.

5 Reader's Note

All necessary codes, videos are attached as links above.
The google drive link for the same is googledrive-link. Open the folder LA to view them.

6 References

- Anton, Howard. Elementary Linear Algebra. New York: John Wiley, 1994. 657-65. Print.
- Lay, David C. Linear Algebra and Its Applications. Boston: Addison Wesley, 2003. Print.
- Blog by PaperGen Gallery
- Boston cs 132 Spring 2021