Linear Algebra and Computer Graphics

Grant Haines

Portland Community College

September 6, 2022

Abstract

We seek to explain several applications of linear algebra in the field of computer graphics. The primary applications that we will examine are those of storing object information in matrices, performing 2D and 3D transformations using homogeneous coordinates, applying perspective to an object, and projecting virtual objects onto a screen.

Table of Contents

1 The World of Computer Graphics

Computer screens seem to present everywhere in the modern world. Smart phones, televisions, billboards, cars, and ever-present desktop and laptop computers. We use these screens to get information about almost every part of our lives, yet it is rare for us to consider the complex mathematics that occur every time you swipe to another web page on your phone. Ever since the first digital display was created, more and more complex applications of mathematics have been discovered to power increasingly complex and intricate computer graphics.

Linear algebra forms a core element of these mathematics. In order to display changing two and three dimensional images on a screen, a computer may have to make millions or even billions of matrix calculations every second. Transformation matrices are used extensively in computer graphics to move and display objects in a simulated environment [`?linear:lay`]. Photographs and screen-shots are stored as matrices of pixels, which can be displayed or manipulated using image editing. 3-dimensional models can be represented as points in space with lines and planes connecting them. Many of these applications either directly use or can be represented using matrices and matrix operations.

2 Screens and Pixels

When you look at a computer screen, what you are actually seeing is not one continuous image, but thousands of tiny dots of light, called pixels. These pixels are told by the computer to display certain combinations of color and brightness in order to form a cohesive picture. Although this is a very complex process, we can consider a computer screen to be a matrix of pixels, the size of which determines the resolution of the image you will receive. While today it is rare for a person to work directly with onscreen pixels to create images, the ultimate purpose of all computer graphics concepts is to light up pixels in the right pattern [`?cg:janke`].

Pixels on a screen have two basic elements: color and brightness. In the common RGB system, color is represented by the intensities of red, green, and brown, combined together to form a single color. Consider Figure 1 as an example. This image has a very small size, or resolution, being only a few dozen pixels wide and tall. For comparison, a common computer screen resolution is 1366 pixels by 768 pixels. Each of the pixels in the image matrix for this picture has its values of color and brightness, which your brain interprets as a bird sitting on a branch.

If we consider each pixel as storing the location of a point on the image, it becomes possible to perform various transformations on the image. Whether this means rotating the image, moving it across the screen, or making it bigger or smaller depends on the type of transformation used [`?linear:lay`] . This method is limited by the fact that each point in the matrix has to correspond to a pixel on the screen, and transforming it may leave "gaps" that have to be filled in, reducing the quality of the image.

Figure 1: Pixelated photo

This technique of treating an image as a matrix of pixels is especially applicable in image editing software, which often have to deal with rotating and scaling images. Since a physical computer screen cannot change in size or be transformed by the computer, a matrix representation is not as useful as it is for other applications.

3 Representing 2-Dimensional Objects

While storing images as matrices of pixels may be useful for simple images, it falls short when it is necessary to model physical objects in a simulated space. In this case, it is very common to represent objects as a series of points, connected by lines and planes to form the image of an object. Because this object is not directly related to the pixels of a computer screen, it can be displayed at whatever resolution is desired by the computer. It is also much easier to transform, since you only need to apply the transformation to a relatively small number of points on the model rather than every pixel on the screen.

In 2-dimensional space, we can use coordinates in $\mathbb{R}^2$ to represent points on a coordinate system. For example, matrix D, whose columns are coordinates in $\mathbb{R}^2$, can be represented as in Figure 2.

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

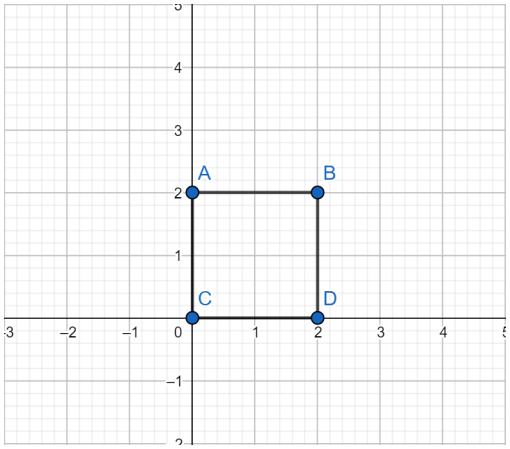


Figure 2: Graph of a square

The edges of this square would be defined in a separate data structure, indicating what points should be connected by lines, in this case AB, BD, DC, and CA. In a more complex, 3D, image, we could also indicate sets of three points to connect using triangular meshes, creating a flat surface. This concept will be discussed later in relation to 3D models and wire frames.

If we use a skew transformation matrix A, we can create a new "image" in the coordinate system, as in Figure 3.

One of the basic goals of optimization is to do use as few calculations as possible to accomplish a task. If we wanted to do multiple transformations on an object in series, such as doing a reflection and then scaling, we would have to do as many calculations as we had transformations, which is not an efficient use of a computer's processing power. Instead, we can start combining these transformations into a single matrix, reducing the number of calculations to a single transformation. Consider the example of first doing a reflection across the line $y = x$ and then scaling x by t. In series, this would be:

$$\begin{bmatrix} t & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$AD = \begin{bmatrix} 1 & 0.5 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 2 & 2 \\ 0 & 2 & 0 & 2 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 0 & 2 & 0 & 2 \end{bmatrix}$$

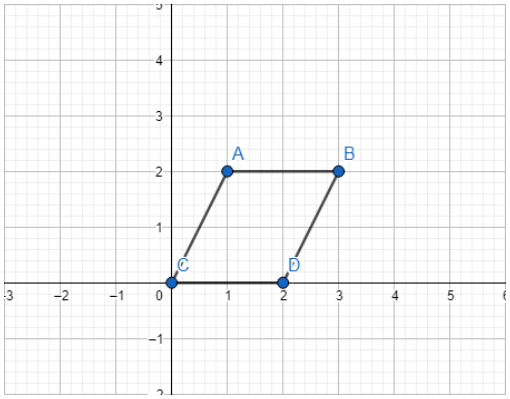


Figure 3: Graph of a square with a skew transformation applied

Note that the transformations are ordered from right to left, since place the transformation matrix on the left of the vector matrix. We can multiply the two transformation matrices together to get:

$$\begin{bmatrix} t & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 & t \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Although multiplying the transformations together first and then multiplying by the vector results in the same number of calculations, it is very rare to only have to apply a series of transformations to a single vector. In most cases, a computer will apply these transformations to hundreds or thousands of vectors, in which case pre-processing the transformations into a single matrix will save a great deal of time.

Homogeneous Coordinates

There are many limitations to using normal linear transformations like we have thus far. One of the main ones is that linear transformations cannot move points around a coordinate system. For example, if we wanted to move the box above two units the right and three units upward, we could not do so using the previous method. To solve this issue, homogeneous coordinates are used, which work by identifying each point (x, y) in $\mathbb{R}^2$ with the point $(x, y, 1)$ in $\mathbb{R}^3$ [`?linear:lay`]. We then display these three-dimensional vectors on a two-dimensional graph, effectively ignoring the third entry.

There are several consequences of this type of coordinate system. In actuality, we are viewing a plane in 3d space parallel to and exactly one unit above the xy-axis. If the third element is not 1, then to get a correct homogeneous coordinate you must scale the whole vector so that it becomes 1 [`?dalling:homogeneous`]. For example, $(15, 21, 3)$ would become the homogeneous coordinate $(5, 7, 1)$. We can compare this to a projector displaying an image on a screen. If you move it farther away from the screen,

To do a transformation of the form $(x, y) \to (x + h, y + k)$, we can rewrite it in homogeneous coordinates as $(x, y, 1) \to (x+h, y+k, 1)$, and create a 3×3 transformation matrix like the following:

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

Note that the 2×2 sub-matrix in the top-left is the linear transformation matrix we have used previously. In this case, since it is the identity matrix, no such transformation is applied. However, we can convert any linear transformation into the homogeneous coordinate systems simply by placing it into the upper-left sub-matrix.

Let's say we wanted to apply a $(x, y) \to (x + 2, y + 3)$ transformation to our box. First, we would have to convert our matrix D to homogeneous coordinates in a matrix D_H:

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

We could then create a transformation matrix B using the template from before, and apply it to D_H, as in Figure 4.

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

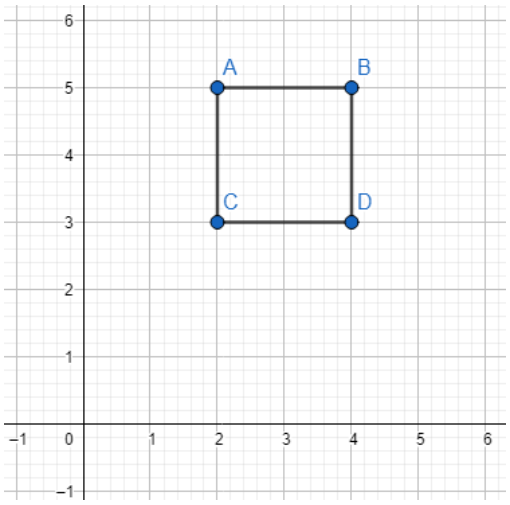


Figure 4: Graph of a square with a transformation applied using homogeneous coordinates

Homogeneous coordinates open up a great many options for matrix transformations that would not otherwise be possible. Almost every practical application within computer graphics involves transformations that standard linear transformations are not capable of, most notably translations across space. Whether moving a character across a level in a video game or dragging an application across the screen in your operating system, these transformations find places all over the field of computer graphics.

4 Representing 3-Dimensional Objects

While all of the examples and discussion up to this point have concerned only 2-dimensional images, the principles we have looked at can be applied fairly easily to 3-dimensional graphics. First of all, instead of considering points and objects within a 2-dimensional space, we are expanding into a 3-dimensional space, and as such must use 3-dimensional vectors and points. When we consider transformation matrices, we will re-size them into 3×3 matrices to fit with our new 3-dimensional vectors. As with 2D images, 3D objects are often represented by a number of points and the lines and planes between them, called wire-frames.

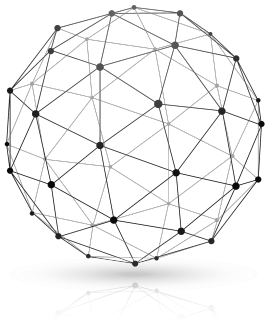
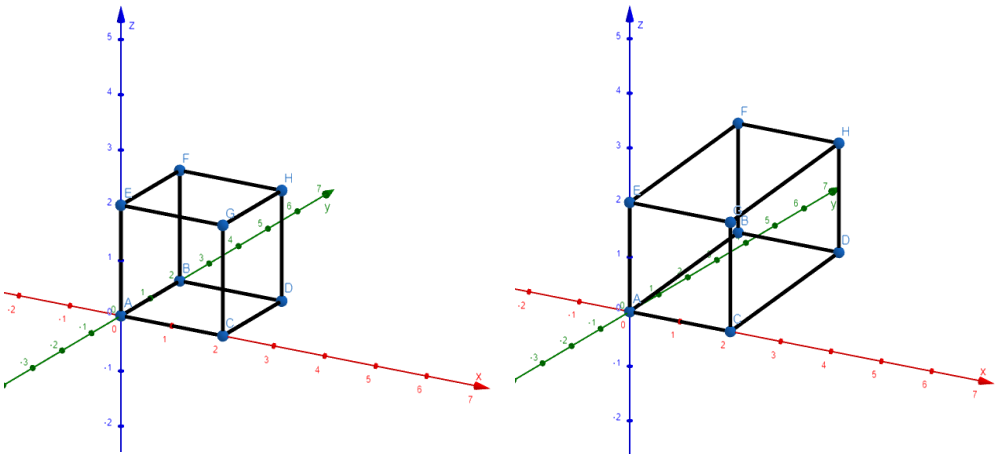


Figure 5: Wireframe image

Consider the square box shown in Figure 6a. This box is made up of eight points and twelve connecting lines, though, as with 2D space, we will only consider a matrix of the points. Note that creating a 3D objects takes many more points than a 2D object, by nature of having to identify a higher-dimensional space. This means that processing 3D graphics takes considerable resources, and is why dedicated "graphics cards" exist for high-impact applications such as gaming or 3D rendering. When it comes to transforming the box, we can create a transformation matrix just like we did in 2D space. An example of this is shown in Figure 6b with a skew transformation on the box.

$$D = \begin{bmatrix} 0 & 0 & 2 & 2 & 0 & 0 & 2 & 2 \\ 0 & 2 & 0 & 2 & 0 & 2 & 0 & 2 \\ 0 & 0 & 0 & 0 & 2 & 2 & 2 & 2 \end{bmatrix}, A = \begin{bmatrix} 1 & 0.5 & 0 \\ 0 & 1 & 0 \\ 0 & 0.5 & 1 \end{bmatrix}, AD = \begin{bmatrix} 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 \\ 0 & 2 & 0 & 2 & 0 & 2 & 0 & 2 \\ 0 & 1 & 0 & 1 & 2 & 3 & 2 & 3 \end{bmatrix}$$



(a) (b)

Figure 6: (a) A square box in a 3D coordinate system and (b) the box with a skew transformation

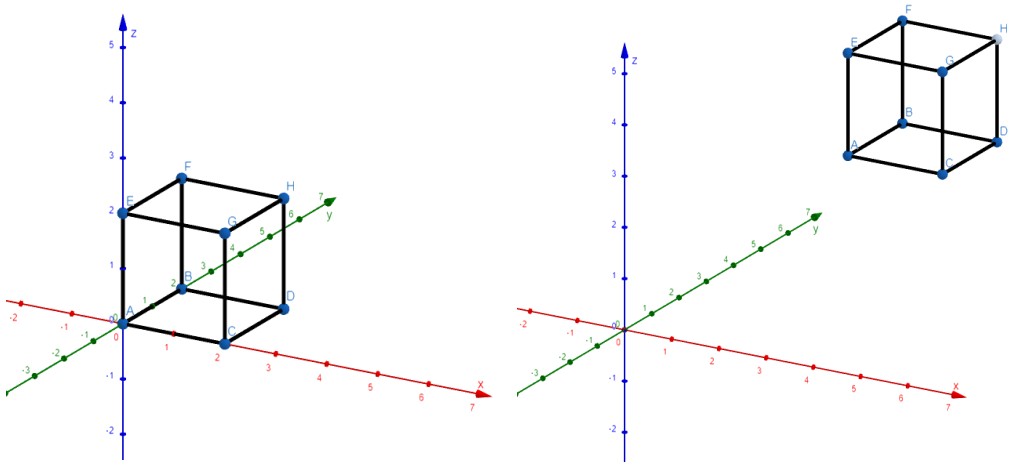
While it is fairly easy to create and represent simple square objects using points and connecting lines, circular or curved objects prove more difficult. In most cases, it is necessary to represent a curved object imperfectly, using a large number of points in order to appear curved, as in Figure 5. Today, however, computers have the processing capability to represent a curved surface using

thousands of points, making individual points almost imperceptible. It is also possible to designate a curved surface mathematically and then generate a set of representative points from that data, with the number of points depending on the capabilities of the computer [?cg:janke]. This kind of optimization grows more and more important as we seek to use computers to model increasingly complex situations with greater detail.

3-D Homogeneous Coordinates

Like in 2D space, we can use homogeneous coordinates to allow for more advanced transformations. In a similar way, we can expand a 3D coordinate to add a fourth homogeneous coordinate, which normally has a value of 1. Take, for example, the box from Figure 6. Expanding the translation matrix we looked at in two dimensions, we could produce a matrix to translate our box to a different part of the 3D space, as shown in Figure 7.

$$D = \begin{bmatrix} 0 & 0 & 2 & 2 & 0 & 0 & 2 & 2 \\ 0 & 2 & 0 & 2 & 0 & 2 & 0 & 2 \\ 0 & 0 & 0 & 0 & 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}, A = \begin{bmatrix} 1 & 0 & 0 & 3 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix}, AD = \begin{bmatrix} 3 & 3 & 5 & 5 & 3 & 3 & 5 & 5 \\ 3 & 5 & 3 & 5 & 3 & 5 & 3 & 5 \\ 3 & 3 & 3 & 3 & 5 & 5 & 5 & 5 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$



(a) (b)

Figure 7: (a) A box in a 3D homogeneous coordinate system and (b) the box with a translation

3-dimensional and 2-dimensional homogeneous coordinates are fundamental to many graphics interfaces. For example, one of the most used application programming interfaces (or API) for graphics rendering is OpenGL, which is used to create 2D and 3D vector graphics for computer-aided design, virtual reality, video games, and information visualization applications. This, and other API's like it, allow programmers to create complex visual graphics with only a basic knowledge of topics like hardware-accelerated rendering. Vector graphics are heavily dependent on homogeneous coordinates and the types of transformations we have been examining thus far.

5 Other Elements of Computer Graphics

While we have covered some of the basic applications of linear algebra to computer graphics, there are many other ways it can be applied, two of which we will examine in the following sections.

Perspective

All of the 3-dimensional examples we have done so far have concerned *isometric* images, not taking perspective into account. Perspective is the effect of distant objects appearing smaller than they actually are. This concept should not be difficult to understand, since it is one that you can observe every time you open your eyes. An isometric image, however, ignores perspective, showing all parts of the image at their real size. The difference between the two can be seen in Figure 8. While isometric images are useful in some cases, especially in engineering and design, many applications of computer graphics, such as simulating an image from a virtual camera, want to display perspective relative to some point in space.

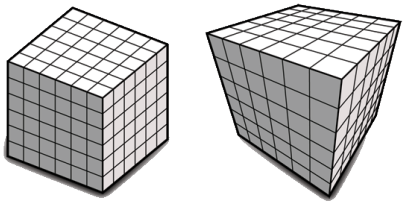


Figure 8: Isometric image (left) and Perspective image (right)

In essence, perspective can be accomplished by scaling the homogeneous coordinate (the third entry of a 2d vector, or fourth entry of a 3d vector) of a point by the distance from the observer to the point [`?dalling:homogeneous`]. For example, if an object in 3d space is 100 units away from the observer, then the homogeneous coordinate of each point in the object would become 100. To return the vectors to a homogeneous form, you would scale the other entries by $\frac{1}{100}$, making the object appear 100 times smaller, or closer, than it's actual size.

Projection

One of the main concepts of computer graphics is the attempt to display a 3D space on a 2D computer screen. Until Star Wars-style holograms are invented, we have to rely on a branch of mathematics called projective geometry. For simplicity's sake, we will imagine that the xy-plane of 3D space represents the computer screen. This is called the viewing plane, of which a portion will be displayed on the computer screen. We will also imagine that a viewer's eye exists at some distance d along the z-axis, called the *center of projection* [`?linear:lay`]. Our goal is to map each point (x, y, z) to an image $(x^*, y^*, 0)$ on the xy-plane, such that the two points and the center of projection form a line [`?linear:lay`]. This is visualized in Figure 9.

Using geometry, we can find that the point (x, y, z, w), where w is the homogeneous coordinate, maps to the coordinate $(x, y, 0, 1 - z/d)$, where d is the distance that the center projection is from the xy-plane. Using this mapping, we can create a simple transformation matrix for projection:

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

Once we have done this, we can then normalize the homogeneous coordinate to get the correctly placed xy-coordinates. For an object in 3D space, this scales it to have the correct perspective, as previously discussed.

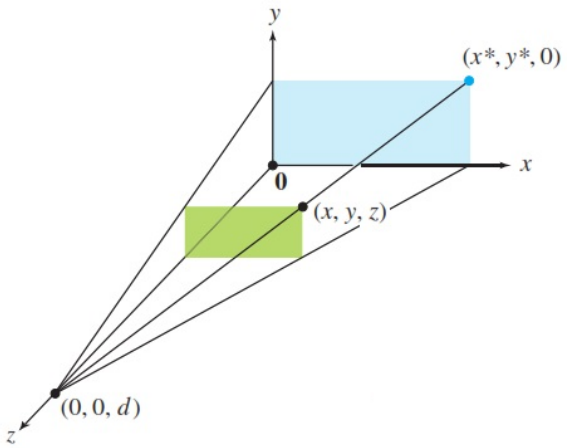


Figure 9: Projective geometry

6 Conclusion

Ever since the first computer screen was invented, the mathematics for displaying images to a user have become more and more sophisticated. Linear algebra forms a core part of these mathematics, especially as high-resolution 2D and 3D graphics have become widespread in computers everywhere. The principles of matrix operations, vector spaces, and transformations are fundamental to the applications of computer graphics and simulated spaces. It is only through understanding these crucial concepts that we can create the countless digital displays and screens that make up more and more of the human world.