INTRODUCTION

Without math we would no computer games. If you ship a computer down to its most basic form what is it? The answer is a calculator. Computers compute numbers, so it shouldn’t be surprising that without math we would have no computer graphics. Objects and images in computer games are merely points drawn together with lines that operate in space. The use of linear algebra is what allows interactions to occur in a computer. Once linear algebra has been translated into a format that the computer recognizes it can bring visuals to life.

2D Vector Graphics

Vector graphics refers to images by mathematical descriptions of geometric objects, rather than by a collection of pixels on the screen (raster graphics).

Vector Graphics vs. Raster Graphics

Vector and raster graphics are the two main forms of two-dimensional computer imagery. Raster graphics, or "bitmaps," are generally represented as a rectangular grid of pixels with each pixel being a varying level of color. This systematic collection of colors, when rendered, forms a 2D image such as digital photographs.

Vector graphics, however, use mathematical descriptions or expressions to represent images. Unlike raster images, vector graphics render images through mathematical calculations. At the core of any vector image are "control points," which have defined positions on the XY coordinate plane. These points connect lines or curves and "fill" information that make up a vector image.

The mathematical nature of vector graphics allow vector images to be resolution independent unlike raster-based images in which resolution is dependent on the number of pixels that need to be saved. Vector graphics, however, can be translated into a format that the computer recognizes it can bring visuals to life.

Vector Graphics vs. Raster Graphics

"Vector graphics refers to representing images by mathematical descriptions of objects or curves, usually in the form of parametric or geometric descriptions of lines and shapes. Vector graphics are often used for creating scalable images that can be resized without losing quality. Raster graphics, on the other hand, are created by using a grid of pixels, where each pixel is assigned a color value. The quality of raster graphics is dependent on the resolution of the image, and they can become pixelated or blurry when scaled.

The mathematical nature of vector graphics allow vector images to be resolution independent unlike raster-based images in which resolution is dependent on the number of pixels that need to be saved. Vector graphics, however, can be translated into a format that the computer recognizes it can bring visuals to life.

RESULTS

Composite Transformations

With translations and transformations the spaceship can now move and rotate around the screen. Unfortunately, our ship doesn’t rotate in the way that most people expect, which is around the center of the ship instead of around the origin.

To solve this problem, we can use "composite transformations" which combine two or more basic transformations.

So, our composite transformation to rotate around the center of the ship, (1, 1.5), should be:

\[
\begin{bmatrix}
A & B \\
0 & 1
\end{bmatrix}
\]

This composite transformation first makes the center of the ship the origin, then does the specified rotation, and finally moves the center back where it is supposed to be.

Multiplying the result with the ship matrix, \( M \), will then result in the desired rotation around the ship’s center.

CONCLUSION

Vector graphics is a linear algebraic way of storing and manipulating computer images. 2D vector graphics were especially suited to moving, rotating, and scaling images and objects within retro games. Through the use of some fundamental concepts of linear algebra, early game developers were able to create games that were revolutionary for its time. All the core of vector graphics biased games such as the 1979 classic Asteroids, are simple matrix multiplications, translations, and rotations. While those games of the past look quite primitive by today’s standards, the techniques being used laid the foundation for future advances in computer graphics, especially in 3D computer graphics.

ACKNOWLEDGEMENTS


Asteroids game screenshot courtesy of Atari / digitalrends.com

Figure 1 - Ship after translation

Figure 2 - Ship after rotation

Homogeneous Coordinates

Each point \((x, y)\) in \(\mathbb{R}^2\) can be identified with the point \((x, y, 1)\) on the plane in \(\mathbb{R}^3\) that lies one unit above the xy-plane. We say that \((x, y)\) has homogeneous coordinates \((x, y, 1)\). Homogeneous coordinates for points are not added or multiplied by scalars, but they can be transformed via multiplication by 3 x 3 matrices.

Translation

Now that our ship's matrix uses homogeneous coordinates, we can, for example, make the ship move forward by performing a translation via matrix multiplication.

This translation is done by multiplying a "movement" matrix and the ship matrix with the product being the new location of the spaceship.

The movement matrix consists of a slightly modified 3 x 3 identity matrix that contains 1's on the diagonal and zeros elsewhere. This matrix is used to represent the movement of the spaceship.

In the sample movement matrix shown on the right, \( r \) represents the number of units along the x-axis that each point will be moved by, and \( s \) represents the number of units along the y-axis that each point will be moved by.

Typically, positive \( r \)-values will result in movement to the right and negative values will result in movement to the left. Positive \( s \)-values will result in movement upward and negative values will result in movement downward.

As a result, any translation of the ship matrix will have the form shown below:

\[
\begin{bmatrix}
1 & 0 & r \\
0 & 1 & s \\
0 & 0 & 1
\end{bmatrix}
\]

For example, to move the spaceship forward (up) 2 units, we need to translate by \((0, 2)\) as shown below:

\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 2 \\
0 & 0 & 1
\end{bmatrix}
\]

The product now represents the spaceship's new position after rotation and is shown graphically in Figure 2.

Rotations

Now that the spaceship has the ability to move in 2D space, the ship needs to be able to rotate in the direction that it is traveling, which can be done by performing a transformation via matrix multiplication.

This transformation is done by multiplying a "rotation" matrix and the ship matrix with the product being the new rotation of the spaceship.

The rotation matrix consists of a partitioned matrix of the form shown on the right, where \( A \) is a 2 x 2 matrix that represents clockwise rotation about the origin at an angle.

To rotate the spaceship 90 degrees to the left, we would perform the following operation:

\[
\begin{bmatrix}
0 & -1 \\
1 & 0
\end{bmatrix}
\]

This results in multiplication of the matrix \( M \) by \( A \). With translations and transformations the spaceship can now move and rotate around the screen. Unfortunately, our ship doesn’t rotate in the way that most people expect, which is around the center of the ship instead of around the origin.

To solve this problem, we can use "composite transformations" which combine two or more basic transformations.

So, our composite transformation to rotate around the center of the ship, (1, 1.5), should be:

\[
\begin{bmatrix}
0 & -1 \\
1 & 0
\end{bmatrix}
\]

This composite transformation first makes the center of the ship the origin, then does the specified rotation, and finally moves the center back where it is supposed to be.

Multiplying the result with the ship matrix, \( M \), will then result in the desired rotation around the ship’s center.

METHODS

Matrix of vertices

In a game utilizing vector graphics, an object can be constructed using a set of points, or vertices. The coordinates of these points are stored in a "data" matrix such as in the figure below.

Matrix Multiplication

This basic triangular shape can now, for example, represent a game’s spaceship. Although a game with a stationary spaceship and nothing else isn’t much fun. So, naturally there needs to be a way move the spaceship around the screen or rotate. Fortunately, this can be accomplished with a creative use of a basic concept in linear algebra, matrix multiplication.

Matrix multiplication allows us to perform transformations (rotation and scaling) and translations (pixelate) on our objects.

This can be performed by using the New-Row Rule for Computing AB,

Definitions: If the product AB is defined, then the entry in row i of column j of AB is the sum of the products of corresponding entries from row i of A and column j of B. If \( (AB)_{ij} \) denotes the value at position \((i, j)\) of \(AB\), then

\[
(AB)_{ij} = \sum_{k=1}^{n} A_{ik} \cdot B_{kj}
\]

However, before we can perform any transformations and translations, our objects must use homogeneous coordinates. It is not possible to do a 2D translation on a 2D point using matrix multiplication.