



# Linear Algebra in Video games

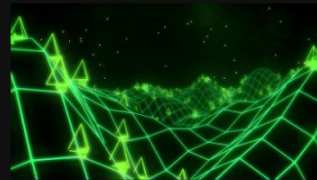
*2D animation and computer graphics*

*By Jake Durham*

*What's  
in an  
Image?*

## What is in an image?

*When video games were a brand new medium, there were generally two types of graphics. There was Raster graphics, which used pixels to create larger images, and there was vector graphics, which used points and lines to create images.*



*Vector  
Images*

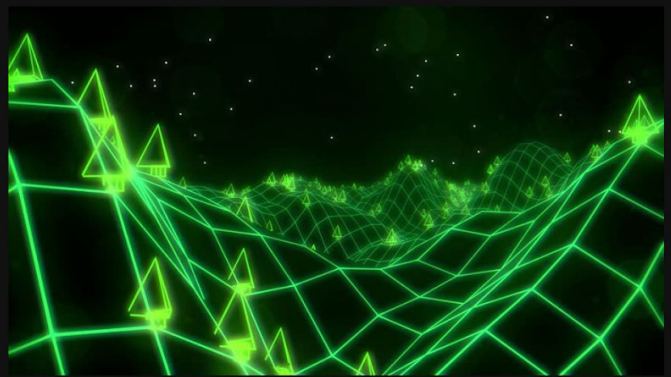
*Raster  
Images*

Raster Vs Vector



# *What is in an image?*

*When video games were a brand new medium, there were generally two types of graphics. There was Raster graphics, which used pixels to create larger images, and there was vector graphics, which used points and lines to create images.*



Raster Vs Vector

# Vector Images

Vector images can be thought of as a collection of vectors. This was a lot easier for computers to store because it took up a lot less memory than keeping track of individual pixels. The downside was being unable to keep track of color as well.

Because of this games that used Vector graphics usually only used one color and a black screen. Whereas raster graphics were much more colorful.

A great example of an early game that used Vector graphics is Asteroids.

## Asteroids

Vector graphics would keep track of shapes by knowing:

- The kind of shape.
- And the points that made up the shape.

For example: Lets say the ship was a triangle.(even though its not quite)  
For the computer to know how to display the shape it needs to be told its a triangle and be given a matrix of the points displayed.

$\begin{bmatrix} 0 & 2 & 1 \\ 0 & 0 & 3 \end{bmatrix}$  This matrix would make the triangle for the ship.

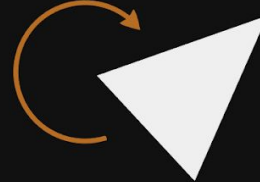
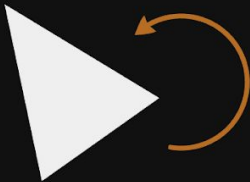


To rotate an image like the ship in asteroids all you would need to do it multiply the matrix with a rotation matrix. This would be done by one degree per frame in order to make the animation look smooth.

Counterclockwise and clockwise

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

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



To animate an image moving across the screen it was simply adding some specific vectors to the image vector:

Up  $\begin{bmatrix} 0 & 0 & 0 \\ x & x & x \end{bmatrix}$



Left  $\begin{bmatrix} -x & -x & -x \\ 0 & 0 & 0 \end{bmatrix}$

Right  $\begin{bmatrix} x & x & x \\ 0 & 0 & 0 \end{bmatrix}$



Down  $\begin{bmatrix} 0 & 0 & 0 \\ -x & -x & -x \end{bmatrix}$

If the ship needed to move diagonally then all you needed to do was add a matrix with the top row filled with x right and a bottom row filled with y up.

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



## *All of this would be true if..*

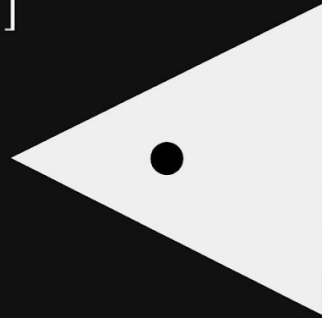
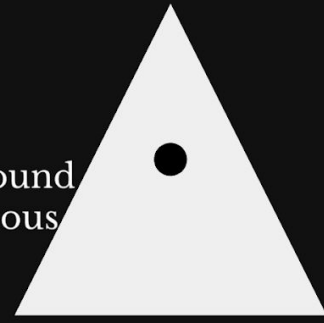
Because the player needs to rotate and move at the same time we need to use a 3x3 matrix to determine its rotation and movement from the origin. I used the 2x3 matrices to explain the basic principle but we would have to use a third row, also known as the dummy row. otherwise the image would always rotate around the origin. The dummy row shows the x and y coordinates of the point you want the image to rotate around. Otherwise the top two rows behave exactly the same!

## Example:

If we want to rotate the triangle around its center  $\begin{pmatrix} 1 \\ 1.5 \end{pmatrix}$  We multiply on the left in order. The new vertices are found with the equation:  $A^{-1}BAM$ , where  $M$  is the homogeneous coordinates. Thus the matrix:

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1.5 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & -1.5 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 2.5 \\ 1 & 0 & .5 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & -1 & 2.5 \\ 1 & 0 & .5 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 2 & 1 \\ 0 & 0 & 3 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 2.5 & 2.5 & -.5 \\ -5 & 2.5 & 1.5 \\ 1 & 1 & 1 \end{bmatrix}$$



This would rotate our triangle 90 degrees counterclockwise in it's center.




## Raster Imaging

Most of the images you see on computers today are raster images. But, if raster imaging takes up so much more memory, why do we use it? There are a few reasons:

- Raster images generally have a wider range of colors available to them.
- Vector images required a special screen when video games became popular.
- Raster images were able to get more detailed pictures.





If we get really close to this image we can see that raster images are made up of many tiny squares called pixels.

## *Rotating and Translating*

Matricis are used to rotate and to scale the size of Raster images. One of the major differences though is that instead of just three points for a triangle, you would have to have a vector for each of the pixels in the picture in order to rotate or move the picture. As you can tell it's a lot more work! Nowadays lots of images are made with vector cordinates and then converted to raster so that it can be processed by the raster screens we use. The matricis for rotating a raster image can be much larger.

## *To magnify a Raster image:*

All you do is multiply the image matrix by a scalar. super simple!



## *Colors!*

Raster images also use matrices for keeping track of colors! Each pixel is a combination of Red, Green and Blue on a scale from 0 to 255. (The size of one byte). Often these colors are just used in reference to the pixel but something interesting happens when you want to use a computer image on the TV. TV's often use a different system for color. They use the YIQ model. To convert the matrix from RGB to YIQ you use a matrix formula like this:

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} .299 & .587 & .114 \\ .596 & -.275 & -.321 \\ .212 & -.523 & .311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

This is done to each pixel to get the correct format for your TV!