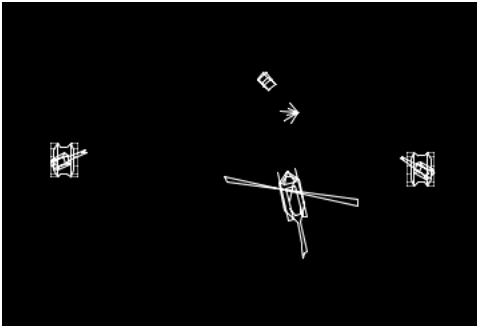
Computer Graphics and Linear Algebra

Rebecca Weber, 2007



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

Punchline: If we represent points of space in the right way, we can represent all sorts of motions and deformations of shapes by matrix multiplication.

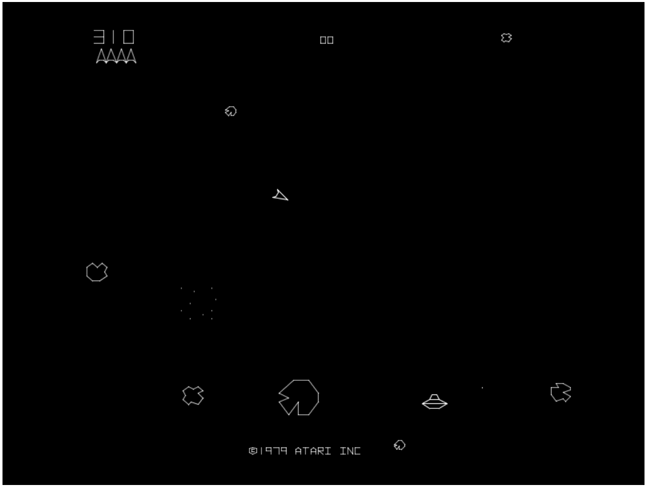
Geometrically, we can represent a polygon by a collection of vertices and the order in which to connect them. This is a lot less data to store in memory than the collection of all the pixels that are to be colored, especially as the polygon gets larger.

If we want to represent other shapes, we need to store:

- What kind of shape it is
- Enough points and distances to fully describe the shape
- (if we have this option) The style and color of the outline
- (if we have this option) The style and color of the inside

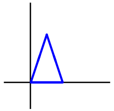
For example, for a circle we would need to know it was a circle, then have the center and radius.

Asteroids



Manipulating a Ship

Let's represent the ship by a triangle, starting with its vertices at $(0,0),(2,0),(1,3)$.



We could represent this as a matrix of vertices, with the understanding (stored as an additional piece of data) that they are connected in order and the last connects to the first.

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

A matrix of vertices allows us to represent rotation by matrix multiplication, but what about translation? If we wanted to move r units to the right and s units up, we'd have to add:

$$\begin{bmatrix} 0 & 2 & 1 \\ 0 & 0 & 3 \end{bmatrix} + \begin{bmatrix} r & r & r \\ s & s & s \end{bmatrix}$$

With a simple modification of the vertex matrix, though, we can represent translation by matrix multiplication using shear transformations (adds a multiple of one coordinate to each of the other coordinates). We just add a dummy coordinate that always holds value 1, and add multiples of that to the x and y coordinates.

Homogeneous Coordinates

In the new scheme, our triangle ship is represented by the matrix

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

Translation by (r, s) is now the product

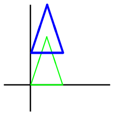
$$\begin{bmatrix} 1 & 0 & r \\ 0 & 1 & s \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 2 & 1 \\ 0 & 0 & 3 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} r & 2+r & 1+r \\ s & s & 3+s \\ 1 & 1 & 1 \end{bmatrix}$$

Flying Forward

Suppose we want to fly our ship two units in the direction it is currently facing (up). We need to translate by $(0, 2)$:

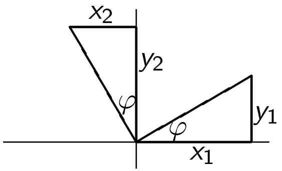
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 2 & 1 \\ 0 & 0 & 3 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 2 & 1 \\ 2 & 2 & 5 \\ 1 & 1 & 1 \end{bmatrix}$$

The new vertices, $(0, 2), (2, 2), (1, 5)$, give us the new ship image:



Turning in Place

We'll crash into an asteroid if we can't change direction. Let's find the images of the columns of I_2 under counterclockwise rotation at an angle of φ.



$$\begin{bmatrix} 1 \\ 0 \end{bmatrix} \mapsto \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} \cos\varphi \\ \sin\varphi \end{bmatrix}$$

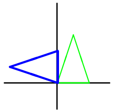
$$\begin{bmatrix} 0 \\ 1 \end{bmatrix} \mapsto \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} -\sin\varphi \\ \cos\varphi \end{bmatrix}$$

The first two columns of I_3 will do just the same thing, with a zero in the last place. Our extra coordinate will stay the same. Hence to obtain counterclockwise rotation by φ degrees, we multiply by the matrix

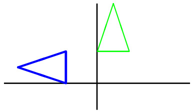
$$\begin{bmatrix} \cos\varphi & -\sin\varphi & 0 \\ \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

For example, for a 90° turn to the left, we would compute:

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

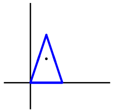


That's not the way ships turn, though! And if we rotate our ship after it drives up 2 units, it's even worse:



Center of Ship vs. Center of Screen

We want our rotation to be about the center of the ship, say, the point halfway between the front and back ends on the line straight back from the tip. In our beginning position that is $(1, 1.5)$.



One solution is to translate, rotate, and translate back: make the center of the triangle the origin, do the rotation as previously, and then move the center back where it belongs. If we move everything by the amount the center requires, everything will stay in its proper positions.

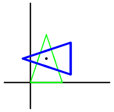
We multiply on the left by each operation's matrix in order. Hence the new vertices are given by $A^{-1}BAM$ where M is the matrix of homogeneous coordinates, A gives translation by $(-1, -1.5)$, B rotation by 90°, and A^{-1} translation by $(1, 1.5)$.

$$A^{-1}BA = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1.5 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & -1.5 \\ 0 & 0 & 1 \end{bmatrix}$$

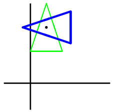
$$= \begin{bmatrix} 0 & -1 & 2.5 \\ 1 & 0 & 0.5 \\ 0 & 0 & 1 \end{bmatrix}$$

$$(A^{-1}BA)M = \begin{bmatrix} 0 & -1 & 2.5 \\ 1 & 0 & 0.5 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 2 & 1 \\ 0 & 0 & 3 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 2.5 & 2.5 & -0.5 \\ 0.5 & 2.5 & 1.5 \\ 1 & 1 & 1 \end{bmatrix}$$

The multiplication claims our vertices after rotation should be at $(2.5, 0.5), (2.5, 2.5)$, and $(-0.5, 1.5)$. Let's graph it.



Much better! The corresponding action for our ship after moving forward two units involves translating by $(-1, -3.5)$ and back with the rotation in between. The vertices obtained are $(2.5, 2.5)$, $(2.5, 4.5)$, and $(-0.5, 3.5)$ (the previous vertices with $y + 2$).



Advantages of Vector Graphics

We've mentioned that the same amount of data is required to represent an object of a given shape no matter what its size. Along those lines, resizing vector objects can be done with perfect quality maintenance. To go from 4×6 to 400×600, you simply multiply by a dilation matrix and then make the connections as usual. A raster image must be sharpened if it is blown up too much.

Another advantage is that these are device-independent descriptions. No matter what kind of screen is going to display the picture, if you give the vector description to the machine it can (with proper software, of course) convert that description into an image that will display properly on its hardware.

So Why Raster?

A little bit of history: For a brief time, displays were actually vector displays (“calligraphic” or “X-Y”). A beam would trace out the desired image on an otherwise black screen, many times per second.

Modern displays, though, are raster (back to the early 1980s or even before), so a vector graphic must be converted to raster (a bitmap) in order to display. This meant that for two-dimensional games it was often easier to deal with raster graphics from the start.

Additionally, a low-resolution display gives a lot more importance to each individual pixel, and it is not efficient to tweak individual pixels with vector representation.

Back to Vector, in part

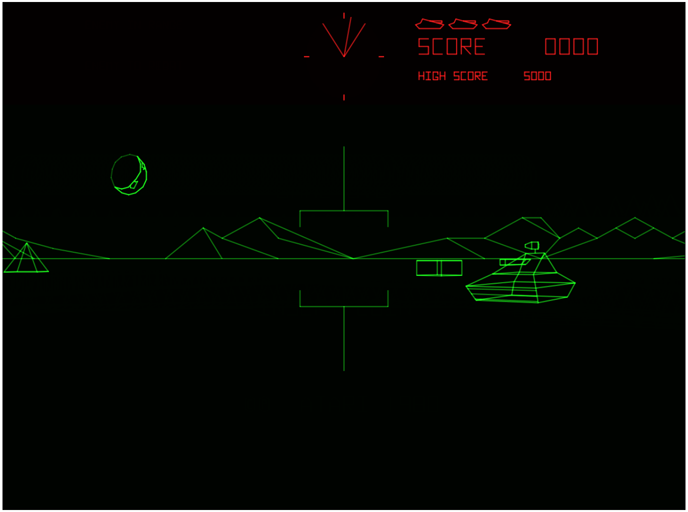
High-resolution displays decrease the need to tweak images at the pixel level, so vector representations can give quality images while maintaining efficiency.

Perhaps more importantly, many games are now 3D. Raster graphics would require representing the entire solid object, despite the fact that we only need to see the surface. With 3D games there is also a lot more manipulation of the objects, as the player moves around and looks from different angles. These combine to make a vector representation again the better choice.

However, raster representation is still more efficient for textures and patterns, so modern games use a combination of vector graphics for objects and raster graphics for surface detail.

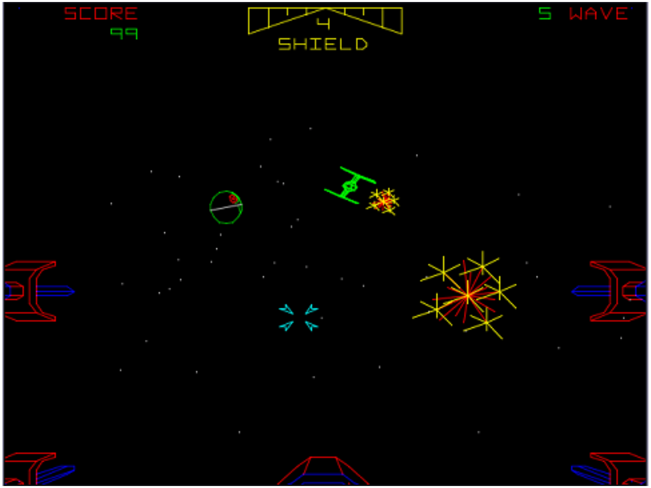
A couple more examples of vector graphics follow.

Battlezone



(Atari, 1980)

Star Wars



(Atari, 1983)

Title slide graphic is of Armor Attack (Cinematronics, 1980)

All images are from Wikipedia, `en.wikipedia.org`