Programming 3 Tutorial 5:1

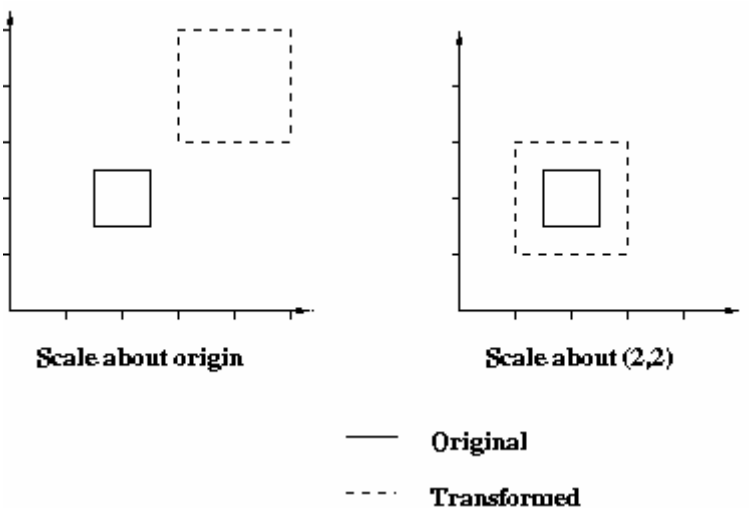
See: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/directx9_c/directx/graphics/reference/d3dx/interfaces/id3dxsprite/_ID3DXSprite.asp
For the up to date MSDN information on the sprite interface.

Rotation and Scaling of Sprites

Throughout Computer Graphics three basic transformations are used. These are:

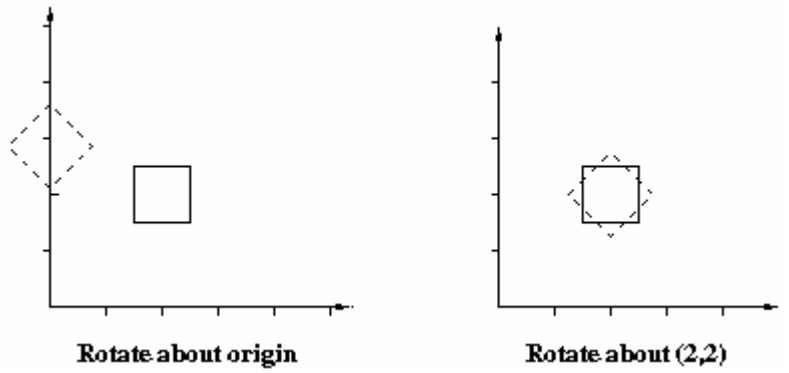
- Scaling - Scales about the origin in the x,y,(z) directions in two-(three-) dimensions.
- Rotation - Rotates about the origin in two-dimensions and one of the co-ordinate axes in three-dimensions. The positive direction of rotation is taken as anti-clockwise.
- Translation - Translates (or moves) by dx,dy,(dz) in two-(three-) dimensions.

It is important to note that the above transformations are very specific and cannot, singly, carry out all the desired transformations one might require. For example the scaling transformation takes place about the origin, that is distances from the origin are scaled by the appropriate scaling factors. Frequently one wishes to scale about a point other than the origin. Consider the diagram below:



which illustrates a square with its centre at (2,2) and side 1 which has been scaled about the origin by a factor of two in the x and y directions as compared with scaling it about its own centre at (2,2). If the latter is required then it cannot be achieved using only one basic transformation.

Similarly consider the same square rotated by 45 degrees about the origin as compared with the same square rotated by 45 degrees about its own centre at (2,2).



Again the second case cannot be achieved by using a single basic transformation.

Again the second case cannot be achieved by using a single basic transformation.

The following basic transformations are usually available:

- Scaling about the origin scale(scalex,scaley) in two-dimensions and scale(scalex,scaley,scalez) in three-dimensions.
- Rotation anti-clockwise about the origin rotate(theta) in two-dimensions. And three functions rotatex(theta), rotatey(theta) and rotatez(theta) which rotate by theta about the x,y and z co-ordinate axes respectively. Positive direction of rotation is anti-clockwise when looking from the positive axis towards the origin.
- Translation. translate(dx,dy) in two-dimensions and translate(dx,dy,dz) in three-dimensions.

This tutorial will focus on rotation and scaling of sprites. It shows how images can appear further away and be manipulated to add depth to the scene.

This effect can be used to create some well rendered DirectX scenes, to see more examples of this see the “ScrollingLayers” 2D code tutorial provided.

This tutorial builds on the code set down in the recent tutorials.

The explosions images can be found in the 2D code provided called “Explosions”.

Scaling, translation and rotation in 2D

To scale and rotate our sprites in 2D prior to the recent updates to DirectX we used be able to provide the call to draw with a series of 2D vectors representing the necessary components - Translation, Scaling, Centre – as well as a simple float value representing the rotation.
No longer, now we have to present these values as 3D vectors, this gets complicated.

To scale or rotate our sprites we need to use a matrix. The sprite provides a call SetTransform that allows us to specify the matrix to use. There are a number of D3DX functions we can use to build a matrix, in the example below I use D3DXMatrixTransformation2D which takes a scale, rotation and position and fills a matrix to do the required 2D transformation.

One of the reasons this is so difficult is because it is hard to conceptualise the how 3D translates to 2D. Luckily we can express our sprite data in 2D structures and then use a provided library function:
`D3DXMatrixTransformation2D`
This will build our matrix that we can then use to scale, translate and rotate our sprite.
After building our matrix we then call the sprite `SetTransform` method.

You can find the exercise example in this weeks code section online: scalingexample.zip

One of the things to note about this is that we now initialise two sprite interfaces, one for each of our drawings. This is because without it we would also transform out second sprite, the frame based image one.

Exercises

1) Modify the code so that the scaling of the single explosion starts of much smaller
2) Modify the code so that the explosions occur more in the middle of the screen
3) Start to build a sprite class