Tutorial 3: Using Matrices

This tutorial introduces the concept of matrices and shows how to use them. The Vertices sample project rendered 2D vertices to draw a triangle. However, in this tutorial you will be working with transformations of vertices in 3D. Matrices are also used to set up cameras and viewports.

Before the Matrices sample project renders geometry, it calls the *SetupMatrices* application-defined function to create and set the matrix transformations that are used to render the 3D triangle. Typically, three types of transformation are set for a 3D scene. Steps for creating each one of these typical transformations are listed below.

Steps

- Step 1 - Defining the World Transformation Matrix
- Step 2 - Defining the View Transformation Matrix
- Step 3 - Defining the Projection Transformation Matrix

The path of the Matrices sample project is:

(SDK root)\Samples\C++\Direct3D\Tutorials\Tut03_Matrices

The order in which these transformation matrices are created does not affect the layout of the objects in a scene. However, Direct3D applies the matrices to the scene in the following order:

1. World
2. View
3. Projection

The sample code in the Matrices project is nearly identical to the sample code in the Vertices project. The>Using Matrices tutorial focuses only on the code unique to matrices and does not cover initializing Direct3D, handling Windows messages, rendering, or shutting down.

Step 1 - Defining the World Transformation Matrix

The world transformation matrix defines how to translate, scale, and rotate the geometry in the 3D model space.

The following code fragment rotates the triangle on the y-axis and then sets the current world transformation for the Direct3D device.

```
D3DXMATRIX matWorld;
D3DXMatrixRotationY( &matWorld, timeGetTime()/150.0f );
g_pd3dDevice->SetTransform( D3DTS_WORLD, &matWorld );
```

The first step is to rotate the triangle around the y-axis by calling the **D3DXMatrixRotationY** method. The first parameter is a pointer to a **D3DXMATRIX** structure that is the result of the operation. The second parameter is the angle of rotation in radians.

The next step is to call **IDirect3DDevice9::SetTransform** to set the world transformation for the Direct3D device. The first parameter accepted by **IDirect3DDevice9::SetTransform** tells Direct3D which transformation to set. This sample uses the **D3DTS_WORLD** macro to specify that the world transformation should be set. The second parameter is a pointer to a matrix that is set as the current transformation.

For more information about world transformations, see **World Transform (Direct3D 9)**.

After defining the world transformation for the scene, you can prepare the view transformation matrix. Again, note that the order in which transformations are defined is not critical. However, Direct3D applies the matrices to the scene in the following order:

1. World
2. View
3. Projection

Defining the view transformation matrix is described in Step 2 - Defining the View Transformation Matrix.

Step 2 - Defining the View Transformation Matrix

The view transformation matrix defines the position and rotation of the view. The view matrix is the camera for the scene.

The following code fragment creates the view transformation matrix and then sets the current view transformation for the Direct3D device.

```
D3DXVECTOR3 vEyePt    ( 0.0f, 3.0f,-5.0f );
D3DXVECTOR3 vLookatPt( 0.0f, 0.0f, 0.0f );
D3DXVECTOR3 vUpVec    ( 0.0f, 1.0f, 0.0f );
D3DXMATRIXA16 matView;
D3DXMatrixLookAtLH( &matView, &vEyePt, &vLookatPt, &vUpVec );
g_pd3dDevice->SetTransform( D3DTS_VIEW, &matView );
```

The first step is to define the view matrix by calling **D3DXMatrixLookAtLH**. The first parameter is a pointer to a **D3DXMATRIX** structure that is the result of the operation. The second, third, and fourth parameters define the eye point, look-at point, and "up" direction, respectively. Here the eye is set back along the z-axis by five units and up three units, the look-at point is set at the origin, and "up" is defined as the y-direction.

The next step is to call **IDirect3DDevice9::SetTransform** to set the view transformation for the Direct3D device. The first parameter accepted by **IDirect3DDevice9::SetTransform** tells Direct3D which transformation to set. This sample uses the D3DTS_VIEW flag to specify that the view transformation should be set. The second parameter is a pointer to a matrix that is set as the current transformation.

For more information about view transformations, see **View Transform (Direct3D 9)**.

After defining the world transformation for the scene, you can prepare the projection transformation matrix. Again, note that the order in which transformations are defined is not critical. However, Direct3D applies the matrices to the scene in the following order:

1. World
2. View
3. Projection

Defining the projection transformation matrix is described in Step 3 - Defining the Projection Transformation Matrix.

Step 3 - Defining the Projection Transformation Matrix

The projection transformation matrix defines how geometry is transformed from 3D view space to 2D viewport space.

The following code fragment creates the projection transformation matrix and then sets the current projection

transformation for the Direct3D device.

```
D3DXMATRIX matProj;
D3DXMatrixPerspectiveFovLH( &matProj, D3DX_PI/4, 1.0f, 1.0f, 100.0f );
g_pd3dDevice->SetTransform( D3DTS_PROJECTION, &matProj );
```

The first step is to call **D3DXMatrixPerspectiveFovLH** to set up the projection matrix. The first parameter is a pointer to a **D3DXMATRIX** structure that is the result of the operation. The second parameter defines the field of view, which tells how objects in the distance get smaller. A typical field of view is 1/4 pi, which is what the sample uses. The third parameter defines the aspect ratio. The sample uses the typical aspect ratio of 1. The fourth and fifth parameters define the near and far clipping plane. This determines the distance at which geometry should no longer be rendered. The Matrices sample project has its near clipping plane set at 1 and its far clipping plane set at 100.

The next step is to call **IDirect3DDevice9::SetTransform** to apply the transformation to the Direct3D device. The first parameter accepted by **IDirect3DDevice9::SetTransform** tells Direct3D which transformation to set. This sample uses the D3DTS_PROJECTION flag to specify that the projection transformation should be set. The second parameter is a pointer to a matrix that is set as the current transformation.

For more information about projection transformations, see **Projection Transform (Direct3D 9)**.

This tutorial has shown you how to use matrices. Tutorial 4: Creating and Using Lights shows how to add lights to your scene for more realism.