Real Time Graphics Lab C.

Week 3 – Lab C

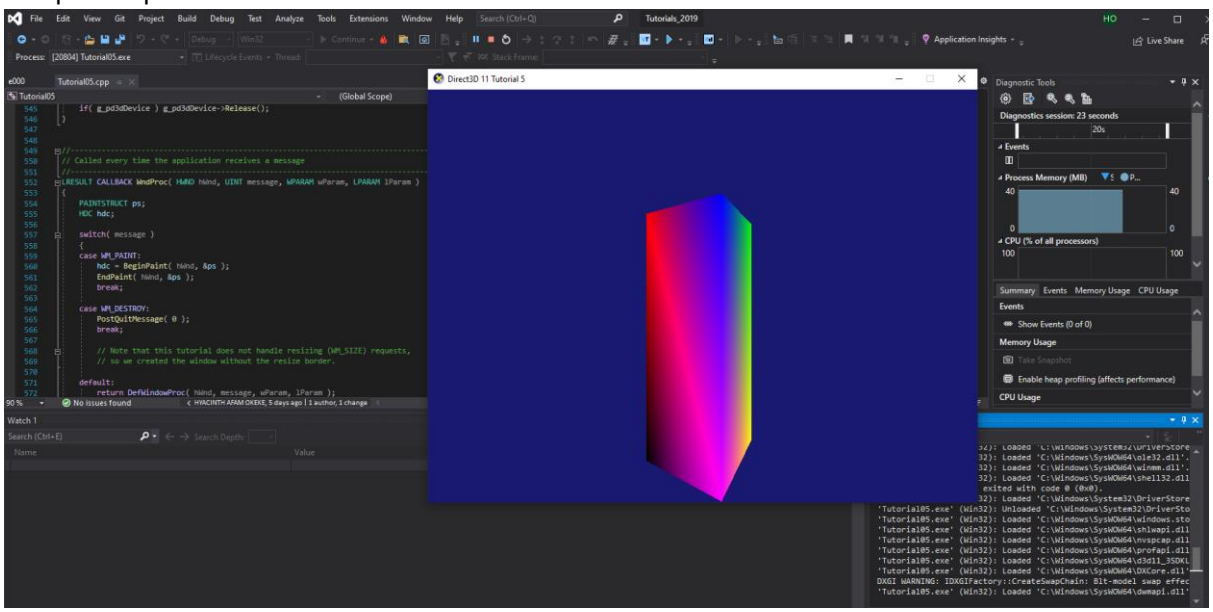
Exercise 1.

Transform the cube into following shape using scaling transformation.

Solution:

```
SimpleVertex vertices[] =
    {
        { XMFLOAT3( -1.0f, 3.0f, -1.0f ), XMFLOAT4( 0.0f, 0.0f, 1.0f, 1.0f ) },
        { XMFLOAT3( 1.0f, 3.0f, -1.0f ), XMFLOAT4( 0.0f, 1.0f, 0.0f, 1.0f ) },
        { XMFLOAT3( 1.0f, 3.0f, 1.0f ), XMFLOAT4( 0.0f, 1.0f, 1.0f, 1.0f ) },
        { XMFLOAT3( -1.0f, 3.0f, 1.0f ), XMFLOAT4( 1.0f, 0.0f, 0.0f, 1.0f ) },
        { XMFLOAT3( -1.0f, -3.0f, -1.0f ), XMFLOAT4( 1.0f, 0.0f, 1.0f, 1.0f ) },
        { XMFLOAT3( 1.0f, -3.0f, -1.0f ), XMFLOAT4( 1.0f, 1.0f, 0.0f, 1.0f ) },
        { XMFLOAT3( 1.0f, -3.0f, 1.0f ), XMFLOAT4( 1.0f, 1.0f, 1.0f, 1.0f ) },
        { XMFLOAT3( -1.0f, -3.0f, 1.0f ), XMFLOAT4( 0.0f, 0.0f, 0.0f, 1.0f ) },
    };
```

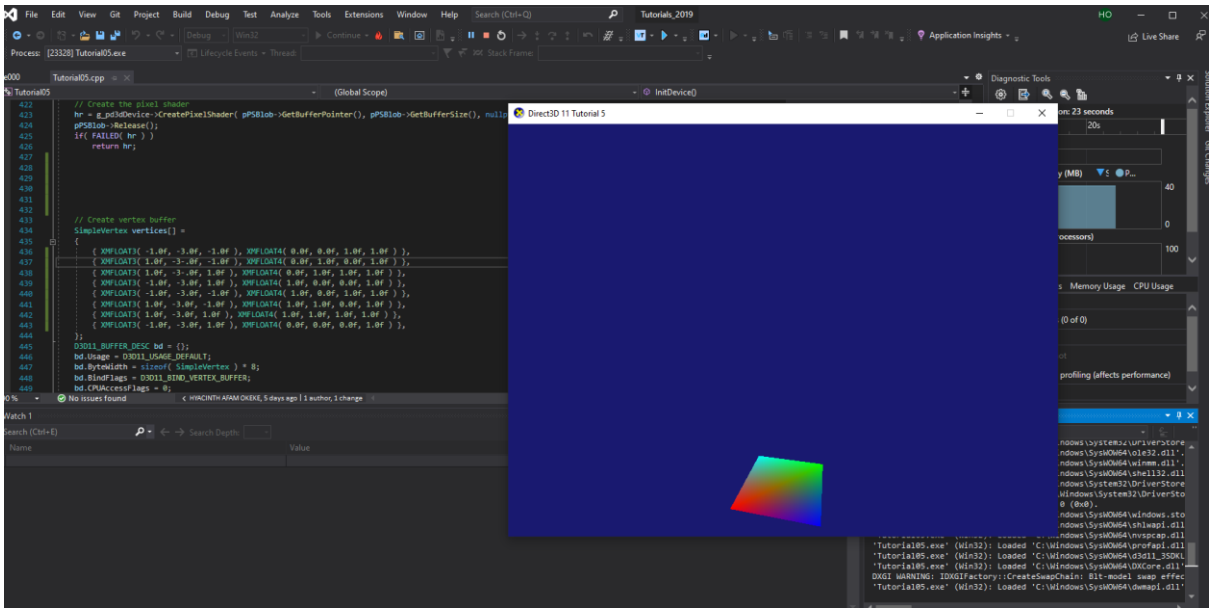
Sample Output:



Solution:

```
SimpleVertex vertices[] =
    {
        { XMFLOAT3( -1.0f, -3.0f, -1.0f ), XMFLOAT4( 0.0f, 0.0f, 1.0f, 1.0f ) },
        { XMFLOAT3( 1.0f, -3.0f, -1.0f ), XMFLOAT4( 0.0f, 1.0f, 0.0f, 1.0f ) },
        { XMFLOAT3( 1.0f, -3.0f, 1.0f ), XMFLOAT4( 0.0f, 1.0f, 1.0f, 1.0f ) },
        { XMFLOAT3( -1.0f, -3.0f, 1.0f ), XMFLOAT4( 1.0f, 0.0f, 0.0f, 1.0f ) },
        { XMFLOAT3( -1.0f, -3.0f, -1.0f ), XMFLOAT4( 1.0f, 0.0f, 1.0f, 1.0f ) },
        { XMFLOAT3( 1.0f, -3.0f, -1.0f ), XMFLOAT4( 1.0f, 1.0f, 0.0f, 1.0f ) },
        { XMFLOAT3( 1.0f, -3.0f, 1.0f ), XMFLOAT4( 1.0f, 1.0f, 1.0f, 1.0f ) },
        { XMFLOAT3( -1.0f, -3.0f, 1.0f ), XMFLOAT4( 0.0f, 0.0f, 0.0f, 1.0f ) },
    };
```

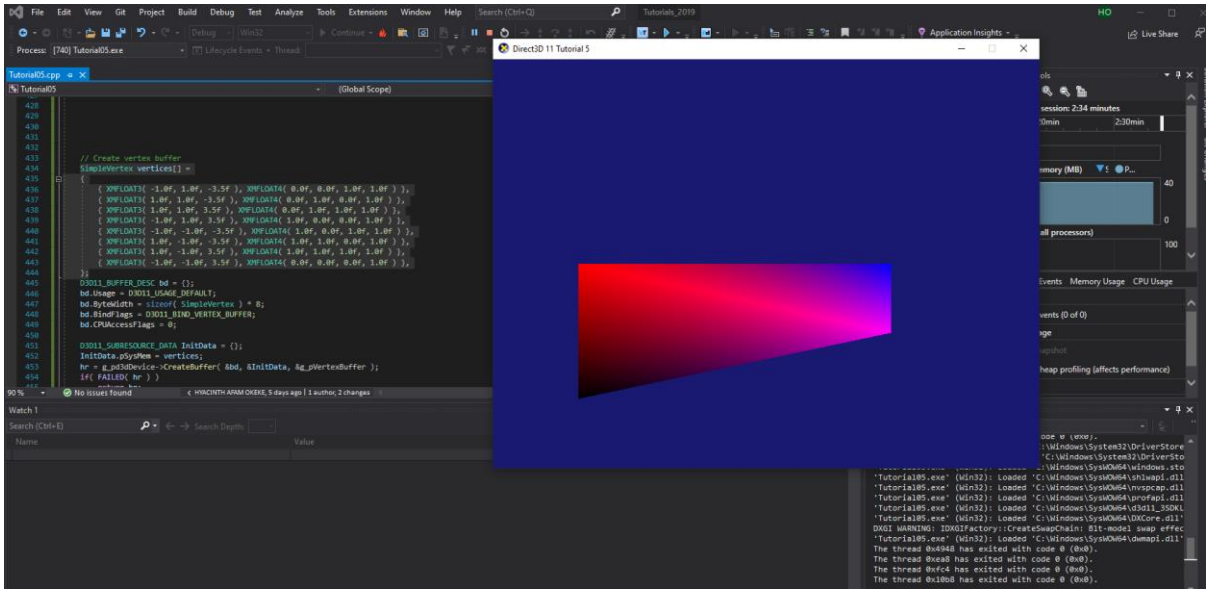
Sample Output:



Solution :

```
SimpleVertex vertices[] =
    {
        { XMFLOAT3( -1.0f, 1.0f, -3.5f ), XMFLOAT4( 0.0f, 0.0f, 1.0f, 1.0f ) },
        { XMFLOAT3( 1.0f, 1.0f, -3.5f ), XMFLOAT4( 0.0f, 1.0f, 0.0f, 1.0f ) },
        { XMFLOAT3( 1.0f, 1.0f, 3.5f ), XMFLOAT4( 0.0f, 1.0f, 1.0f, 1.0f ) },
        { XMFLOAT3( -1.0f, 1.0f, 3.5f ), XMFLOAT4( 1.0f, 0.0f, 0.0f, 1.0f ) },
        { XMFLOAT3( -1.0f, -1.0f, -3.5f ), XMFLOAT4( 1.0f, 0.0f, 1.0f, 1.0f ) },
        { XMFLOAT3( 1.0f, -1.0f, -3.5f ), XMFLOAT4( 1.0f, 1.0f, 0.0f, 1.0f ) },
        { XMFLOAT3( 1.0f, -1.0f, 3.5f ), XMFLOAT4( 1.0f, 1.0f, 1.0f, 1.0f ) },
        { XMFLOAT3( -1.0f, -1.0f, 3.5f ), XMFLOAT4( 0.0f, 0.0f, 0.0f, 1.0f ) },
```

Sample Output:



Reflection:

The vertex buffer was extensively modified to get a horizontal cube, vertical cube and a flat cube serving as a base.

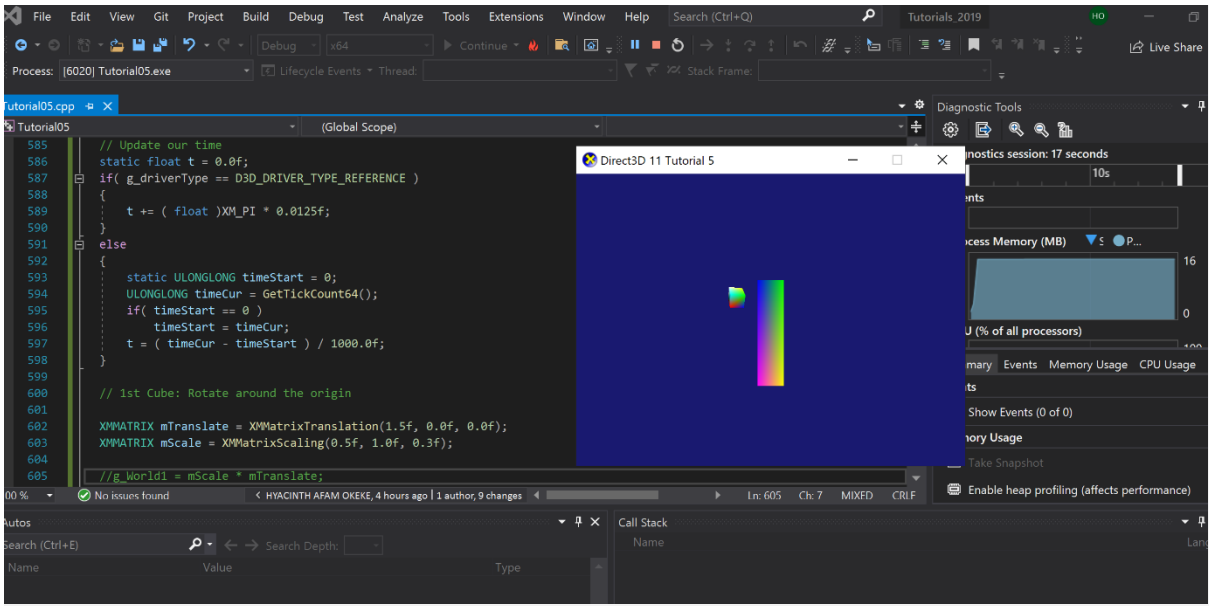
Exercise 2:

Perform scaling, translation and rotation transformation to achieve the following effects: (1) a cube
rotates by a vertical rotation axis;

Solution: 1

```
SimpleVertex vertices[] =
    {
        { XMFLOAT3( -1.0f, 5.0f, 2.5f ), XMFLOAT4( 0.0f, 0.0f, 1.0f, 1.0f ) },
        { XMFLOAT3( 1.0f, 5.0f, 2.5f ), XMFLOAT4( 0.0f, 1.0f, 0.0f, 1.0f ) },
        { XMFLOAT3( 1.0f, 5.0f, 2.5f ), XMFLOAT4( 0.0f, 1.0f, 1.0f, 1.0f ) },
        { XMFLOAT3( -1.0f, 5.0f, 2.5f ), XMFLOAT4( 1.0f, 0.0f, 0.0f, 1.0f ) },
        { XMFLOAT3( -1.0f, -5.0f, 2.5f ), XMFLOAT4( 1.0f, 0.0f, 1.0f, 1.0f ) },
        { XMFLOAT3( 1.0f, -5.0f, 2.5f ), XMFLOAT4( 1.0f, 1.0f, 0.0f, 1.0f ) },
        { XMFLOAT3( 1.0f, -5.0f, 2.5f ), XMFLOAT4( 1.0f, 1.0f, 1.0f, 1.0f ) },
        { XMFLOAT3( -1.0f, -5.0f, 2.5f ), XMFLOAT4( 0.0f, 0.0f, 0.0f, 1.0f ) },
    };
```

Sample Output :



(2). Two cubes rotate by two different
rotational axes with different rotational speeds respectively.

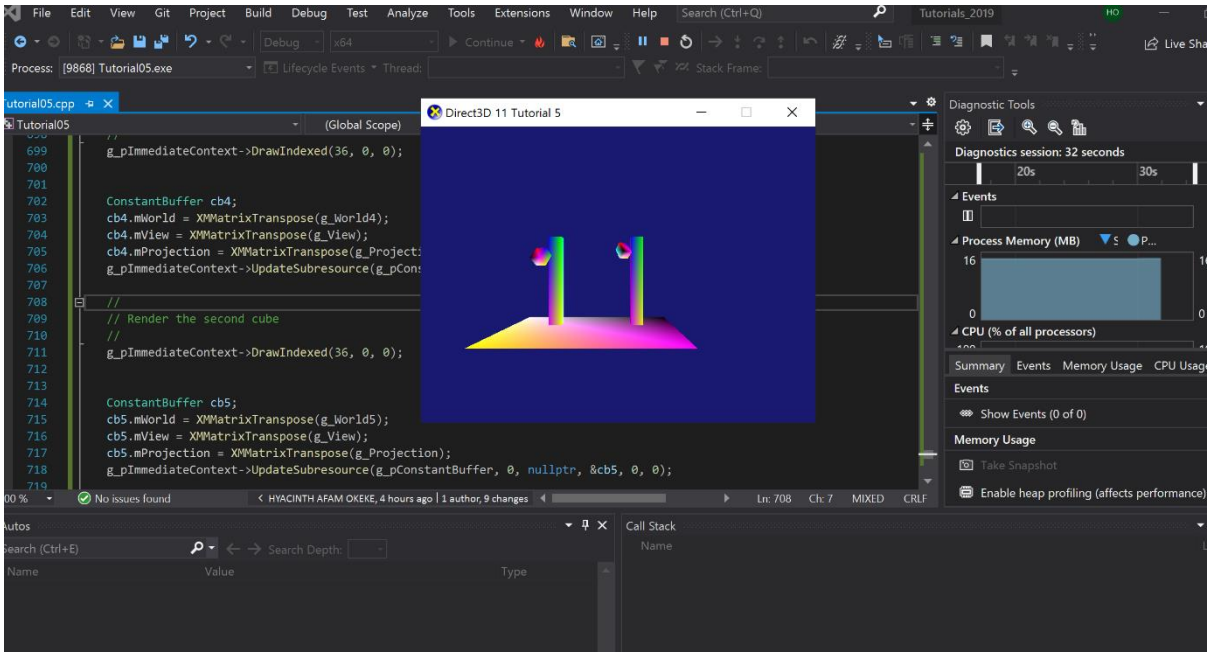
Solution :

```
// 1st Cube: Rotate around the origin

    XMMATRIX mTranslate = XMMatrixTranslation(1.5f, 0.0f, 0.0f);
    XMMATRIX mScale = XMMatrixScaling(0.5f, 1.0f, 0.3f);

    g_World1 = mScale * mTranslate;
       //g_World1 = XMMatrixRotationY(0);
```

```
    //g_World4 = XMMatrixRotationY(t);

    // 2nd Cube:  Rotate around origin
    XMMATRIX mSpin2 = XMMatrixRotationZ( -t );
    XMMATRIX mOrbit2 = XMMatrixRotationY( -t * 3.0f );
       XMMATRIX mTranslate2 = XMMatrixTranslation( -2.5f, 2.5f, 1.0f );
    XMMATRIX mScale2 = XMMatrixScaling( 0.3f, 0.1f, 0.3f );

       g_World2 = mScale2 * mSpin2 * mTranslate2 * mOrbit2;


    // 3rd Cube:  Rotate around origin
    XMMATRIX mSpin3 = XMMatrixRotationZ(-t);
    XMMATRIX mOrbit3 = XMMatrixRotationY(-t * 2.0f);
    XMMATRIX mTranslate3 = XMMatrixTranslation(-4.5f, 2.0f, 3.5f);
    XMMATRIX mScale3 = XMMatrixScaling(0.3f, 0.1f, 0.3f);

    g_World3 = mScale3 * mSpin3 * mTranslate3 * mOrbit3;



    //// 4th Cube:  Rotate around origin
    XMMATRIX mSpin4 = XMMatrixRotationZ(0);
    XMMATRIX mOrbit4 = XMMatrixRotationY( 0 * 2.0f);
    XMMATRIX mTranslate4 = XMMatrixTranslation(-2.0f, -3.0f, -1.0f);
    XMMATRIX mScale4 = XMMatrixScaling(-6.3f, 0.0f, 3.0f);

    g_World4 = mScale4 * mSpin4 * mTranslate4 * mOrbit4;


    //// 5th Cube:  Rotate around origin
    //XMMATRIX mSpin5 = XMMatrixRotationZ(0);
   // XMMATRIX mOrbit5 = XMMatrixRotationY(0 * 2.0f);
    XMMATRIX mTranslate5 = XMMatrixTranslation(-5.0f, 0.0f, 0.0f);
    XMMATRIX mScale5 = XMMatrixScaling(0.5f, 1.0f, 0.3f);

    g_World5 = mScale5  * mTranslate5 ;

    //
    // Clear the back buffer
    //
    g_pImmediateContext->ClearRenderTargetView( g_pRenderTargetView,
Colors::MidnightBlue );

    //
    // Clear the depth buffer to 1.0 (max depth)
    //
    g_pImmediateContext->ClearDepthStencilView( g_pDepthStencilView,
D3D11_CLEAR_DEPTH, 1.0f, 0 );

    //
    // Update variables for the first cube
    //
    ConstantBuffer cb1;
       cb1.mWorld = XMMatrixTranspose( g_World1 );
       cb1.mView = XMMatrixTranspose( g_View );
       cb1.mProjection = XMMatrixTranspose( g_Projection );
       g_pImmediateContext->UpdateSubresource( g_pConstantBuffer, 0, nullptr, &cb1, 0,
0 );

    //
```

```
    // Render the first cube
    //
       g_pImmediateContext->VSSetShader( g_pVertexShader, nullptr, 0 );
       g_pImmediateContext->VSSetConstantBuffers( 0, 1, &g_pConstantBuffer );
       g_pImmediateContext->PSSetShader( g_pPixelShader, nullptr, 0 );
       g_pImmediateContext->DrawIndexed( 36, 0, 0 );

    //
    // Update variables for the second cube
    //
    ConstantBuffer cb2;
       cb2.mWorld = XMMatrixTranspose( g_World2 );
       cb2.mView = XMMatrixTranspose( g_View );
       cb2.mProjection = XMMatrixTranspose( g_Projection );
       g_pImmediateContext->UpdateSubresource( g_pConstantBuffer, 0, nullptr, &cb2, 0,
0 );

    //
    // Render the second cube
    //
       g_pImmediateContext->DrawIndexed( 36, 0, 0 );



    ConstantBuffer cb3;
    cb3.mWorld = XMMatrixTranspose(g_World3);
    cb3.mView = XMMatrixTranspose(g_View);
    cb3.mProjection = XMMatrixTranspose(g_Projection);
    g_pImmediateContext->UpdateSubresource(g_pConstantBuffer, 0, nullptr, &cb3, 0, 0);

    //
    // Render the second cube
    //
    g_pImmediateContext->DrawIndexed(36, 0, 0);


    ConstantBuffer cb4;
    cb4.mWorld = XMMatrixTranspose(g_World4);
    cb4.mView = XMMatrixTranspose(g_View);
    cb4.mProjection = XMMatrixTranspose(g_Projection);
    g_pImmediateContext->UpdateSubresource(g_pConstantBuffer, 0, nullptr, &cb4, 0, 0);

    //
    // Render the second cube
    //
    g_pImmediateContext->DrawIndexed(36, 0, 0);


    ConstantBuffer cb5;
    cb5.mWorld = XMMatrixTranspose(g_World5);
    cb5.mView = XMMatrixTranspose(g_View);
    cb5.mProjection = XMMatrixTranspose(g_Projection);
    g_pImmediateContext->UpdateSubresource(g_pConstantBuffer, 0, nullptr, &cb5, 0, 0);

    //
    // Render the second cube
    //
    g_pImmediateContext->DrawIndexed(36, 0, 0);
```

Sample Output:



Reflection:

After several trials, I finally got the desired result of having two standing cubes with a base and two floating cubes orbiting each standing cube. This was achieved by initializing the **g_world** variables and **cb** variables, this allowed for the creation additional cubes with the required dimension and float pattern as shown on the sample output attachment.

Exercise 3:

In Exercise 2(a), scale the small rotating cube into a long-thin stick and rote the stick by a rotation axis such that the stick is always tangent to the rotation path, as shown below. If you are able to do this, also consider how to get the stick flying along a general curve.

Solution :

```
SimpleVertex vertices[] =
    {
        { XMFLOAT3( -1.0f, 0.0f, -1.0f ), XMFLOAT4( 0.0f, 0.0f, 1.0f, 1.0f ) },
        { XMFLOAT3( 1.0f, 0.0f, -1.0f ), XMFLOAT4( 0.0f, 1.0f, 0.0f, 1.0f ) },
        { XMFLOAT3( 1.0f, 0.0f, 1.0f ), XMFLOAT4( 0.0f, 1.0f, 1.0f, 1.0f ) },
        { XMFLOAT3( -1.0f, 0.0f, 1.0f ), XMFLOAT4( 1.0f, 0.0f, 0.0f, 1.0f ) },
        { XMFLOAT3( -1.0f, 0.0f, -1.0f ), XMFLOAT4( 1.0f, 0.0f, 1.0f, 1.0f ) },
        { XMFLOAT3( 1.0f, 0.0f, -1.0f ), XMFLOAT4( 1.0f, 1.0f, 0.0f, 1.0f ) },
        { XMFLOAT3( 1.0f, 0.0f, 1.0f ), XMFLOAT4( 1.0f, 1.0f, 1.0f, 1.0f ) },
        { XMFLOAT3( -1.0f, 0.0f, 1.0f ), XMFLOAT4( 0.0f, 0.0f, 0.0f, 1.0f ) },
    };
```

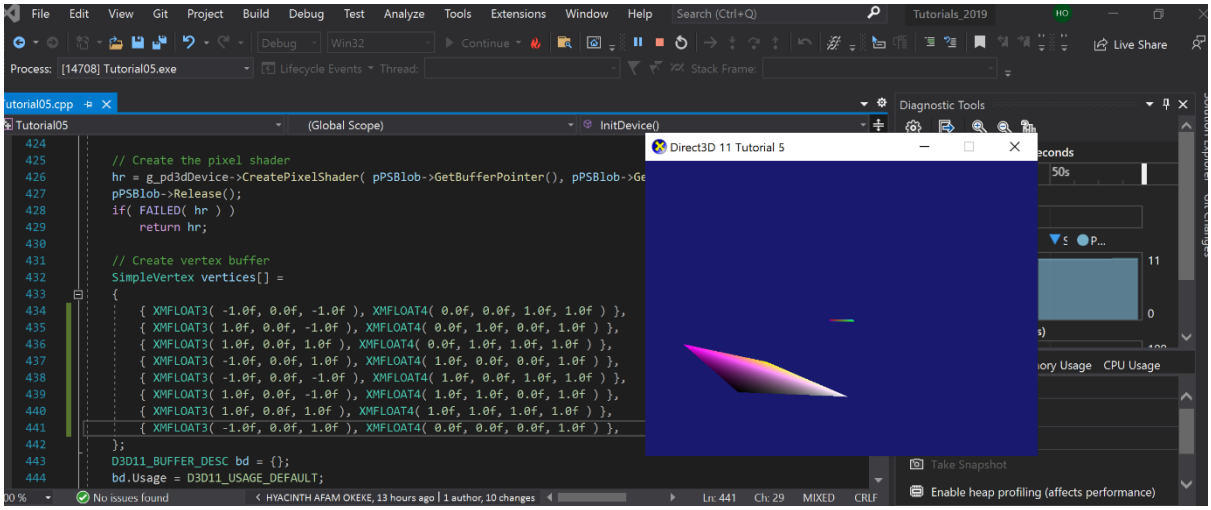
```
    D3D11_BUFFER_DESC bd = {};
    bd.Usage = D3D11_USAGE_DEFAULT;
    bd.ByteWidth = sizeof( SimpleVertex ) * 8;
    bd.BindFlags = D3D11_BIND_VERTEX_BUFFER;
       bd.CPUAccessFlags = 0;

    D3D11_SUBRESOURCE_DATA InitData = {};
    InitData.pSysMem = vertices;
    hr = g_pd3dDevice->CreateBuffer( &bd, &InitData, &g_pVertexBuffer );
    if( FAILED( hr ) )
        return hr;

    // Set vertex buffer
    UINT stride = sizeof( SimpleVertex );
    UINT offset = 0;
    g_pImmediateContext->IASetVertexBuffers( 0, 1, &g_pVertexBuffer, &stride, &offset
);

    // Create index buffer
    WORD indices[] =
    {
        3,1,0,
        2,1,3,

        0,5,4,
        1,5,0,

        3,4,7,
        0,4,3,

        1,6,5,
        2,6,1,

        2,7,6,
        3,7,2,

        6,4,5,
        7,4,6,
    };
    bd.Usage = D3D11_USAGE_DEFAULT;
    bd.ByteWidth = sizeof( WORD ) * 36;        // 36 vertices needed for 12 triangles
in a triangle list
    bd.BindFlags = D3D11_BIND_INDEX_BUFFER;
       bd.CPUAccessFlags = 0;
    InitData.pSysMem = indices;
    hr = g_pd3dDevice->CreateBuffer( &bd, &InitData, &g_pIndexBuffer );
    if( FAILED( hr ) )
        return hr;

    // Set index buffer
    g_pImmediateContext->IASetIndexBuffer( g_pIndexBuffer, DXGI_FORMAT_R16_UINT, 0 );

    // Set primitive topology
    g_pImmediateContext->IASetPrimitiveTopology( D3D11_PRIMITIVE_TOPOLOGY_TRIANGLELIST
);

       // Create the constant buffer
       bd.Usage = D3D11_USAGE_DEFAULT;
       bd.ByteWidth = sizeof(ConstantBuffer);
       bd.BindFlags = D3D11_BIND_CONSTANT_BUFFER;
       bd.CPUAccessFlags = 0;
    hr = g_pd3dDevice->CreateBuffer( &bd, nullptr, &g_pConstantBuffer );
```

```
    if( FAILED( hr ) )
        return hr;

    // Initialize the world matrix
       //g_World1 = XMMatrixIdentity();
       g_World2 = XMMatrixIdentity();
    g_World3 = XMMatrixIdentity();
    //g_World4 = XMMatrixIdentity();
    //g_World5 = XMMatrixIdentity();

    // Initialize the view matrix
       XMVECTOR Eye = XMVectorSet( 0.0f, 1.0f, -12.0f, 0.0f );
       XMVECTOR At = XMVectorSet( 0.0f, 1.0f, 0.0f, 0.0f );
       XMVECTOR Up = XMVectorSet( 0.0f, 1.0f, 0.0f, 0.0f );
       g_View = XMMatrixLookAtLH( Eye, At, Up );

    // Initialize the projection matrix
       g_Projection = XMMatrixPerspectiveFovLH( XM_PIDIV2, width / (FLOAT)height,
0.01f, 100.0f );

    return S_OK;
}


//--------------------------------------------------------------------------------------
// Clean up the objects we've created
//--------------------------------------------------------------------------------------
void CleanupDevice()
{
    if( g_pImmediateContext ) g_pImmediateContext->ClearState();

    if( g_pConstantBuffer ) g_pConstantBuffer->Release();
    if( g_pVertexBuffer ) g_pVertexBuffer->Release();
    if( g_pIndexBuffer ) g_pIndexBuffer->Release();
    if( g_pVertexLayout ) g_pVertexLayout->Release();
    if( g_pVertexShader ) g_pVertexShader->Release();
    if( g_pPixelShader ) g_pPixelShader->Release();
    if( g_pDepthStencil ) g_pDepthStencil->Release();
    if( g_pDepthStencilView ) g_pDepthStencilView->Release();
    if( g_pRenderTargetView ) g_pRenderTargetView->Release();
    if( g_pSwapChain1 ) g_pSwapChain1->Release();
    if( g_pSwapChain ) g_pSwapChain->Release();
    if( g_pImmediateContext1 ) g_pImmediateContext1->Release();
    if( g_pImmediateContext ) g_pImmediateContext->Release();
    if( g_pd3dDevice1 ) g_pd3dDevice1->Release();
    if( g_pd3dDevice ) g_pd3dDevice->Release();
}


//--------------------------------------------------------------------------------------
// Called every time the application receives a message
//--------------------------------------------------------------------------------------
LRESULT CALLBACK WndProc( HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam )
{
    PAINTSTRUCT ps;
    HDC hdc;

    switch( message )
```

```
    {
    case WM_PAINT:
        hdc = BeginPaint( hWnd, &ps );
        EndPaint( hWnd, &ps );
        break;

    case WM_DESTROY:
        PostQuitMessage( 0 );
        break;

        // Note that this tutorial does not handle resizing (WM_SIZE) requests,
        // so we created the window without the resize border.

    default:
        return DefWindowProc( hWnd, message, wParam, lParam );
    }

    return 0;
}


//--------------------------------------------------------------------------------
--
// Render a frame
//--------------------------------------------------------------------------------
--
void Render()
{
    // Update our time
    static float t = 0.0f;
    if( g_driverType == D3D_DRIVER_TYPE_REFERENCE )
    {
        t += ( float )XM_PI * 0.0125f;
    }
    else
    {
        static ULONGLONG timeStart = 0;
        ULONGLONG timeCur = GetTickCount64();
        if( timeStart == 0 )
            timeStart = timeCur;
        t = ( timeCur - timeStart ) / 1000.0f;
    }

    // 1st Cube: Rotate around the origin

    XMMATRIX mTranslate = XMMatrixTranslation(1.5f, 0.0f, 0.0f);
    XMMATRIX mScale = XMMatrixScaling(0.5f, 1.0f, 0.3f);

    //g_World1 = mScale * mTranslate;
       g_World1 = XMMatrixRotationY(0);

    //g_World4 = XMMatrixRotationY(t);

    // 2nd Cube:  Rotate around origin
    XMMATRIX mSpin2 = XMMatrixRotationZ( -t );
    XMMATRIX mOrbit2 = XMMatrixRotationY( -t * 3.0f );
       XMMATRIX mTranslate2 = XMMatrixTranslation( -2.5f, 2.5f, 1.0f );
    XMMATRIX mScale2 = XMMatrixScaling( 0.3f, 0.1f, 0.3f );

       //g_World2 = mScale2 * mSpin2 * mTranslate2 * mOrbit2;
```

```
    // 3rd Cube:  Rotate around origin
    XMMATRIX mSpin3 = XMMatrixRotationZ(-t);
    XMMATRIX mOrbit3 = XMMatrixRotationY(-t * 2.0f);
    XMMATRIX mTranslate3 = XMMatrixTranslation(-4.5f, 2.0f, 3.5f);
    XMMATRIX mScale3 = XMMatrixScaling(0.3f, 0.1f, 0.4f);

    //g_World3 = mScale3 * mSpin3 * mTranslate3 * mOrbit3;



    //// 4th Cube:  Rotate around origin
    XMMATRIX mSpin4 = XMMatrixRotationZ(-t);
    XMMATRIX mOrbit4 = XMMatrixRotationY( -t * 2.0f);
    XMMATRIX mTranslate4 = XMMatrixTranslation(-5.0f, -3.0f, 3.0f);
    XMMATRIX mScale4 = XMMatrixScaling(3.3f, 0.0f, -5.0f);

    g_World4 = mScale4 * mSpin4 * mTranslate4 * mOrbit4;
```

Sample Output:



Reflection:

I converted one of the long cubes to a floating stick around an axis, this was achieved my modifying the Translation and Scaling values to achieved the desired floating stick. This was fun to do.

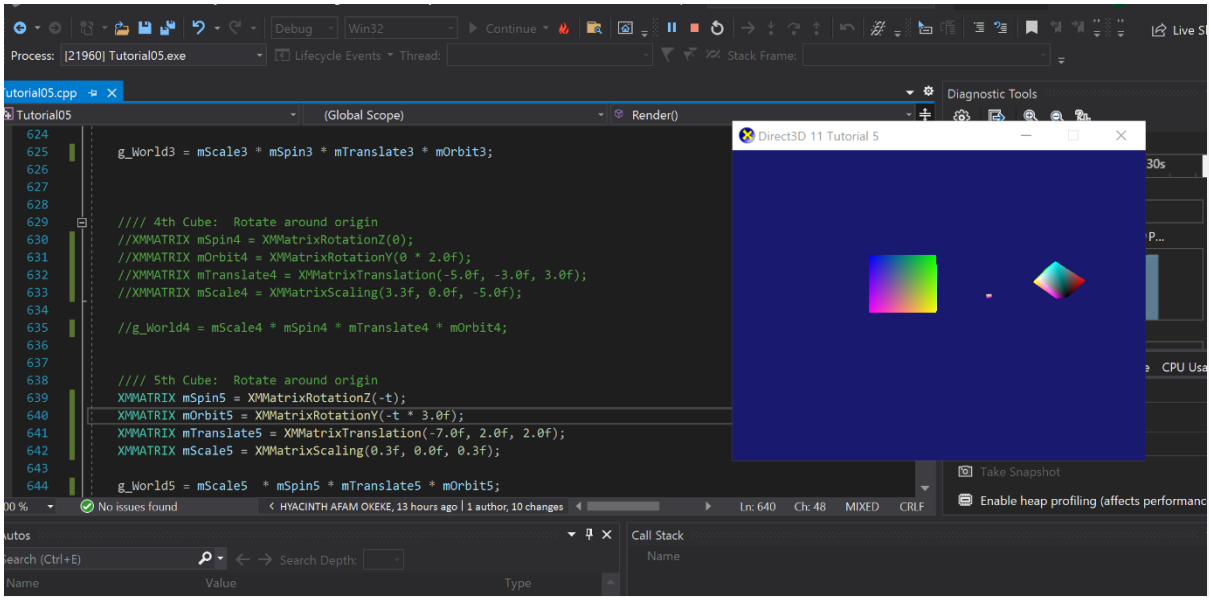
Exercise 4:

Scale the cube into different sizes corresponding to the Sun, the Earth and the Moon respectively and then combine a set of rotation and translation transformations to animate a simple solar system.

Solution:

```
XMMATRIX mSpin2 = XMMatrixRotationZ(0);
    XMMATRIX mOrbit2 = XMMatrixRotationY(0 * 3.0f );
       XMMATRIX mTranslate2 = XMMatrixTranslation( -2.5f, 2.5f, 1.0f );
    XMMATRIX mScale2 = XMMatrixScaling( 2.3f, 0.5f, 2.3f );

       g_World2 = mScale2 * mSpin2 * mTranslate2 * mOrbit2;


    // 3rd Cube:  Rotate around origin
    XMMATRIX mSpin3 = XMMatrixRotationZ(-t);
    XMMATRIX mOrbit3 = XMMatrixRotationY(-t * 0.5f);
    XMMATRIX mTranslate3 = XMMatrixTranslation(-6.5f, 2.0f, 3.5f);
    XMMATRIX mScale3 = XMMatrixScaling(0.6f, 0.1f, 0.6f);

    g_World3 = mScale3 * mSpin3 * mTranslate3 * mOrbit3;



    //// 4th Cube:  Rotate around origin
    //XMMATRIX mSpin4 = XMMatrixRotationZ(0);
    //XMMATRIX mOrbit4 = XMMatrixRotationY(0 * 2.0f);
    //XMMATRIX mTranslate4 = XMMatrixTranslation(-5.0f, -3.0f, 3.0f);
    //XMMATRIX mScale4 = XMMatrixScaling(3.3f, 0.0f, -5.0f);

    //g_World4 = mScale4 * mSpin4 * mTranslate4 * mOrbit4;


    //// 5th Cube:  Rotate around origin
    XMMATRIX mSpin5 = XMMatrixRotationZ(-t);
    XMMATRIX mOrbit5 = XMMatrixRotationY(-t * 3.0f);
    XMMATRIX mTranslate5 = XMMatrixTranslation(-7.0f, 2.0f, 2.0f);
    XMMATRIX mScale5 = XMMatrixScaling(0.3f, 0.0f, 0.3f);

    g_World5 = mScale5  * mSpin5 * mTranslate5 * mOrbit5;

    //
    // Clear the back buffer
    //
    g_pImmediateContext->ClearRenderTargetView( g_pRenderTargetView,
Colors::MidnightBlue );
```

Sample Output:



Reflection:

The Biggest object represents the Sun followed by the earth and the moon represented as the Big, small and smallest objects respectively. Just as in the solar system the Earth revolves around the sun while the moon orbits the earth. This was achieved in visual studio by modifying the different g_world parameters of the cubes shown in the program sample output.

Exercise 5:

In the Tutorial04, the view transformation and projection transformation are created from the XMMatrixLookAtLH( ) method and the XMMatrixPerspectiveFovLH() method. To develop a better understanding of the two transformations, you can directly define the matrices in your c++ program and observe if you can get exactly the same effect. You can define the two matrices directly using XMMatrixSet( ) method.