# How to pick and manipulate a 3D object using DirectX in Universal apps

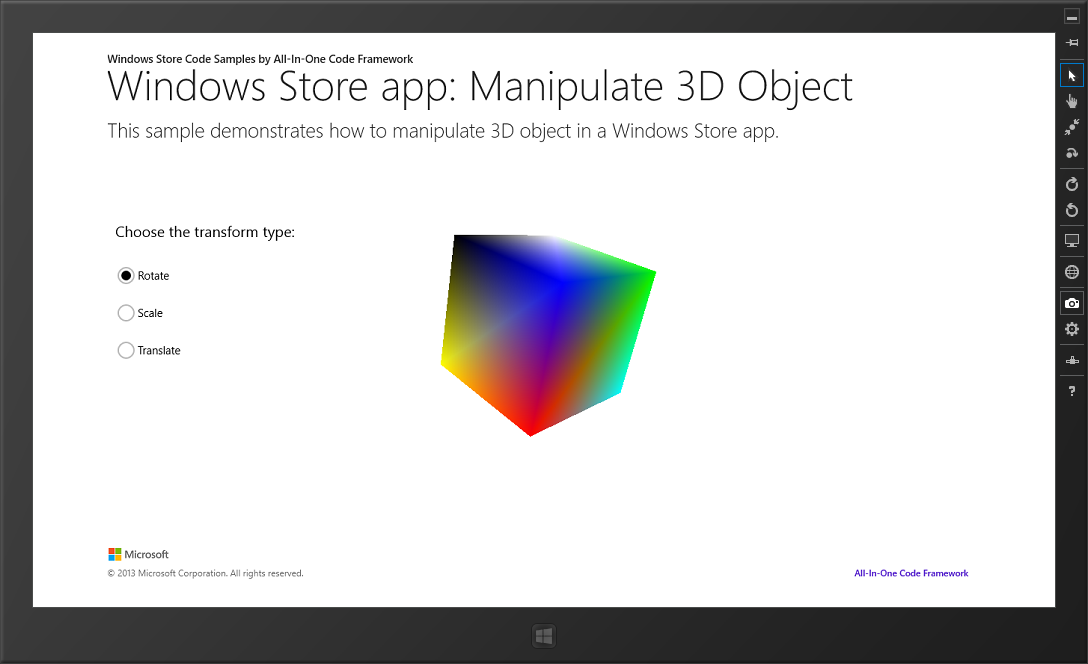
## Introduction

This sample demonstrates how to pick and manipulate a 3D object, such as rotate, scale and translate. The code also demos intersection test between bounding box and bounding frustum.

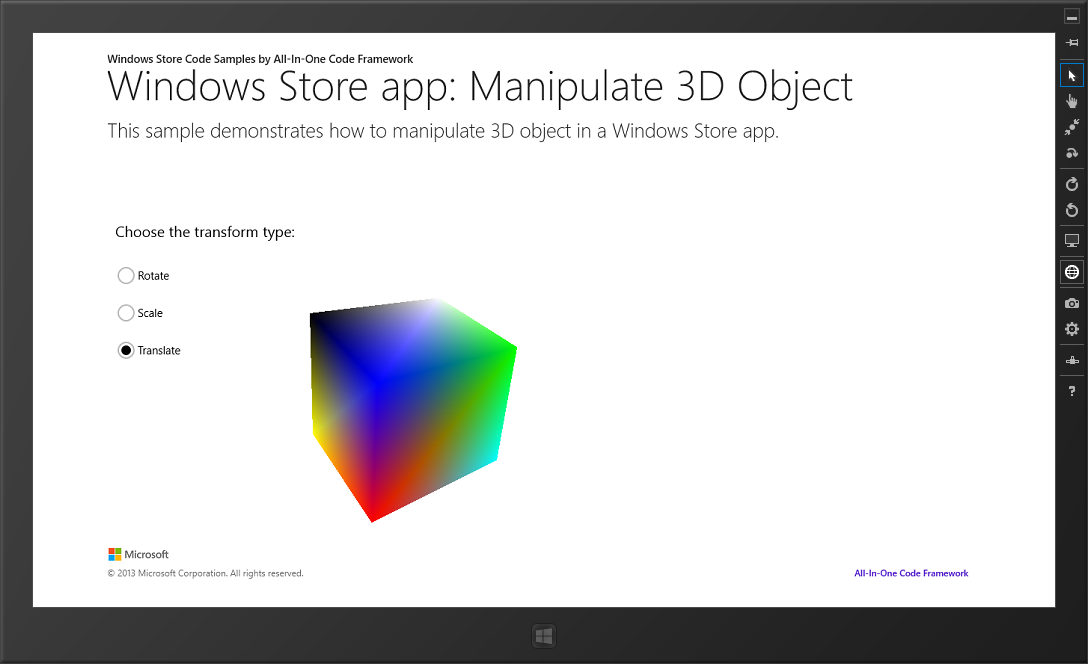
## Running the Sample

Build the sample in Visual Studio 2013, and then run it.

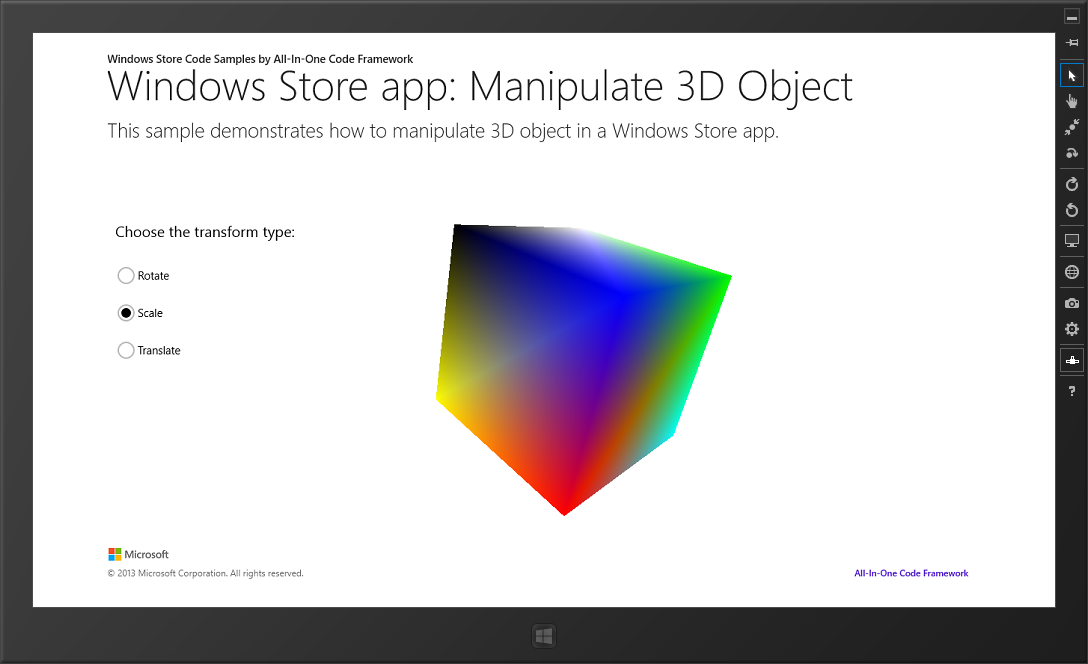
Check the “Rotate” button, and the cube will rotate with the pointer as we drag the pointer.



Check the “Translate” button, the object will transfer some distance as we drag the pointer.



Check the “Scale” button, the cube will be scaled as we drag the pointer.



## Using the Code

1. Firstly, we should transform the 2D screen coordinate to **Landscape** based space, since there are multiple orientations in windows 8 touch device.

|  |
| --- |
| -Code block start-  --C++ code snippet start--  //Transform the current point coordinate to the origin screen space  Point CubeRenderer::TransformToOrientation(Point point, bool dipsToPixels)  {  Point returnValue;    switch (m\_orientation)  {  case DisplayOrientations::Landscape:  returnValue = point;  break;  case DisplayOrientations::Portrait:  returnValue = Point(point.Y, m\_windowBounds.Width - point.X);  break;  case DisplayOrientations::PortraitFlipped:  returnValue = Point(m\_windowBounds.Height - point.Y, point.X);  break;  case DisplayOrientations::LandscapeFlipped:  returnValue = Point(m\_windowBounds.Width - point.X, m\_windowBounds.Height - point.Y);  break;  default:  throw ref new Platform::FailureException();  break;  }  // Convert DIP to Pixels, or not?  return dipsToPixels ? Point(ConvertDipsToPixels(returnValue.X),  ConvertDipsToPixels(returnValue.Y))  : returnValue;  }  --C++ code snippet end--  Insert other Programming Language Code Snippet here  -Code block end- |

1. Then we’ll transform the 2D space coordinate of the pointer to the view space. Refer to some mathematical knowledge shown below.

|  |
| --- |
| -Code block start-  --C++ code snippet start--  void CubeRenderer::ScreenToView(  \_In\_ float sx,  \_In\_ float sy,  \_Outptr\_ float \* vx,  \_Outptr\_ float \* vy  )  {  \*vx = (2.0f \* sx / m\_d3dRenderTargetSize.Width - 1.0f) / m\_cbMVPData.projection.\_11;  \*vy = (-2.0f \* sy / m\_d3dRenderTargetSize.Height + 1.0f) / m\_cbMVPData.projection.\_22;  }  void CubeRenderer::VectorToLocal(  \_In\_ XMVECTOR inVec,  \_Outptr\_ XMVECTOR \* outVec  )  {  XMMATRIX viewMx = XMLoadFloat4x4(&m\_cbMVPData.view);  XMMATRIX modelMx = XMLoadFloat4x4(&m\_cbMVPData.model);  XMMATRIX invView = XMMatrixInverse(&XMMatrixDeterminant(viewMx), viewMx);  XMMATRIX invModel = XMMatrixInverse(&XMMatrixDeterminant(modelMx), modelMx);  XMMATRIX toLocal = invView \* invModel;  XMFLOAT4 inVecF;  XMStoreFloat4(&inVecF, inVec);  if(1.0f == inVecF.w)//point vector  {  \*outVec = XMVector3TransformCoord(inVec, toLocal);  }  else  {  \*outVec = XMVector3TransformNormal(inVec, toLocal);  \*outVec = XMVector3Normalize(\*outVec);  }    }  --C++ code snippet end--  Insert other Programming Language Code Snippet here  -Code block end- |

But at here, we just make use of a method from DirectXMath, [XMVector3Unproject](http://msdn.microsoft.com/en-us/library/microsoft.directx_sdk.transformation.xmvector3unproject(v=vs.85).aspx).

We calculate two points from the screen space which have same x and y value, but difference z values.

|  |
| --- |
| -Code block start-  --C++ code snippet start--  XMVECTOR vector1 = DirectX::XMVector3Unproject(  XMVectorSet(sx, sy, 0.0f, 1.0f),  0.0f,  0.0f,  m\_d3dRenderTargetSize.Width,  m\_d3dRenderTargetSize.Height,  0.0f,  1.0f,  XMLoadFloat4x4(&m\_cbMVPData.projection),  XMLoadFloat4x4(&m\_cbMVPData.view),  XMLoadFloat4x4(&m\_cbMVPData.model)  );  XMVECTOR vector2 = DirectX::XMVector3Unproject(  XMVectorSet(sx, sy, 1.0f, 1.0f),  0.0f,  0.0f,  m\_d3dRenderTargetSize.Width,  m\_d3dRenderTargetSize.Height,  0.0f,  1.0f,  XMLoadFloat4x4(&m\_cbMVPData.projection),  XMLoadFloat4x4(&m\_cbMVPData.view),  XMLoadFloat4x4(&m\_cbMVPData.model)  );  --C++ code snippet end--  Insert other Programming Language Code Snippet here  -Code block end- |

Then we can get the origin point (vector1) and direction vector(vector2 – vector1), which would set to the XMVector3Unproject method. Note to normalize the direction vector.

1. Now we can iterate through the 3D object and test if the picking ray intersects it. We use [Intersects(…)](http://msdn.microsoft.com/en-us/library/windows/desktop/hh855922(v=vs.85).aspx) method which was new in DirectXMath.
2. After we can test intersecting, we just transform the object by dragging the pointer.
3. When we process rotating, we transform the object to an arc ball, so that it can be rotated smoothly.

|  |
| --- |
| -Code block start-  --C++ code snippet start--  void CubeRenderer::ScreenToArcBall(  \_In\_ float sx,  \_In\_ float sy,  \_Outptr\_ DirectX::XMFLOAT3 &vec  )  {  float width = m\_windowBounds.Width;  float height = m\_windowBounds.Height;  float x = ( sx - width / 2 ) / ( width / 2 );  float y = -( sy - height / 2 ) / ( height / 2 );    float z = 0.0f;  float mag = x \* x + y \* y;    if( mag > 1.0f )  {  float scale = 1.0f / sqrtf( mag );  x \*= scale;  y \*= scale;  }  else  z = -(sqrtf( 1.0f - mag ));  vec = XMFLOAT3(x, y, z);  }  --C++ code snippet end--  Insert other Programming Language Code Snippet here  -Code block end- |

1. When we execute translating, we just divide the coordinate offset by a constant.

|  |
| --- |
| -Code block start-  --C++ code snippet start--  transform = XMMatrixTranslation((x2 - x1) / 200, (y1 - y2) / 200, 0.0f);  --C++ code snippet end--  Insert other Programming Language Code Snippet here  -Code block end- |

1. When we execute scaling, we also make use of the arc ball to calculate the distance of pointer offset around the ball. Then we set data to [XMMatrixScaling(…)](http://msdn.microsoft.com/en-us/library/microsoft.directx_sdk.matrix.xmmatrixscaling(v=vs.85).aspx).

## More Information

* [Direct3D 11 Picking](http://www.braynzarsoft.net/index.php?p=D3D11PICKING)