# Direct3D Resources

2011.05.16

Jihye Yun

# Resource

- Area in memory that can be accessed by the Direct3D pipeline
- Buffer resources
  - Vertex buffer, index buffer, constant buffer
- Texture resources
  - 1D texture and 1D texture array, 2D texture and 2D texture array, 3D texture

# Resource

- Several options that determine how resources can be used
  - Control whether resources have both read and write access
  - Make resources accessible to only the CPU, GPU or both

# Texture

```
D3D11_TEXTURE2D_DESC desc;
ZeroMemory( &desc, sizeof(desc) );
desc.Width =512;
desc.Height = 512;
desc.MipLevels = 1;
desc.ArraySize = 1;
desc.Format = DXGI_FORMAT_R16G16B16A16_SNORM;
desc.SampleDesc.Count = 1;
desc.Usage = D3D11_USAGE_DEFAULT;
desc.BindFlags = D3D11_BIND_RENDER_TARGET | D3D11_BIND_SHADER_RESOURCE;
desc.CPUAccessFlags = 0;
desc.MiscFlags = 0;

hr = g_pd3dDevice->CreateTexture2D( &desc, NULL, &g_pTex2D );
if( FAILED(hr) )
    return hr;
```

# D3D11_USAGE

- Whether a resource is accessible by the CPU and/or GPU
  - D3D11_USAGE_DEFAULT
  - D3D11_USAGE_IMMUTABLE
  - D3D11_USAGE_DYNAMIC
  - D3D11_USAGE_STAGING

# D3D11_USAGE

- Whether a resource is accessible by the CPU and/or GPU
  - **D3D11_USAGE_DEFAULT**
    - Read and write access by the GPU
  - **D3D11_USAGE_IMMUTABLE**
    - Can only be read by the GPU
    - Cannot be written by the GPU
    - Cannot be accessed at all by the CPU

# D3D11_USAGE

- Whether a resource is accessible by the CPU and/or GPU
  - **D3D11_USAGE_DYNAMIC**
    - Accessible by both the GPU(read only) and the CPU(write only)
    - Good choice for a resource that will be updated by the CPU at least once per frame
  - **D3D11_USAGE_STAGING**
    - Supports data transfer (copy) from the GPU to the CPU

# CPU→GPU

- Create dynamic texture
  - D3D11_USAGE_DYNAMIC
  - D3D11_CPU_ACCESS_WRITE
- Make your own texture
  - ID3D11DeviceContext::Map,
  - ID3D11DeviceContext :: Unmap
- Render a square using this texture

# CPU→GPU

```
D3D11_TEXTURE2D_DESC desc;
ZeroMemory( &desc, sizeof(desc) );
desc.Width =256;
desc.Height = 256;
desc.MipLevels = 1;
desc.ArraySize = 1;
desc.Format = DXGI_FORMAT_R8G8B8A8_UNORM;
desc.SampleDesc.Count = 1;
desc.Usage = D3D11_USAGE_DYNAMIC;
desc.BindFlags = D3D11_BIND_SHADER_RESOURCE;
desc.CPUAccessFlags = D3D11_CPU_ACCESS_WRITE;
desc.MiscFlags = 0;

hr = g_pd3dDevice->CreateTexture2D( &desc, NULL, &g_pTexDynamic );
if( FAILED(hr) )
    return hr;
```

# CPU→GPU

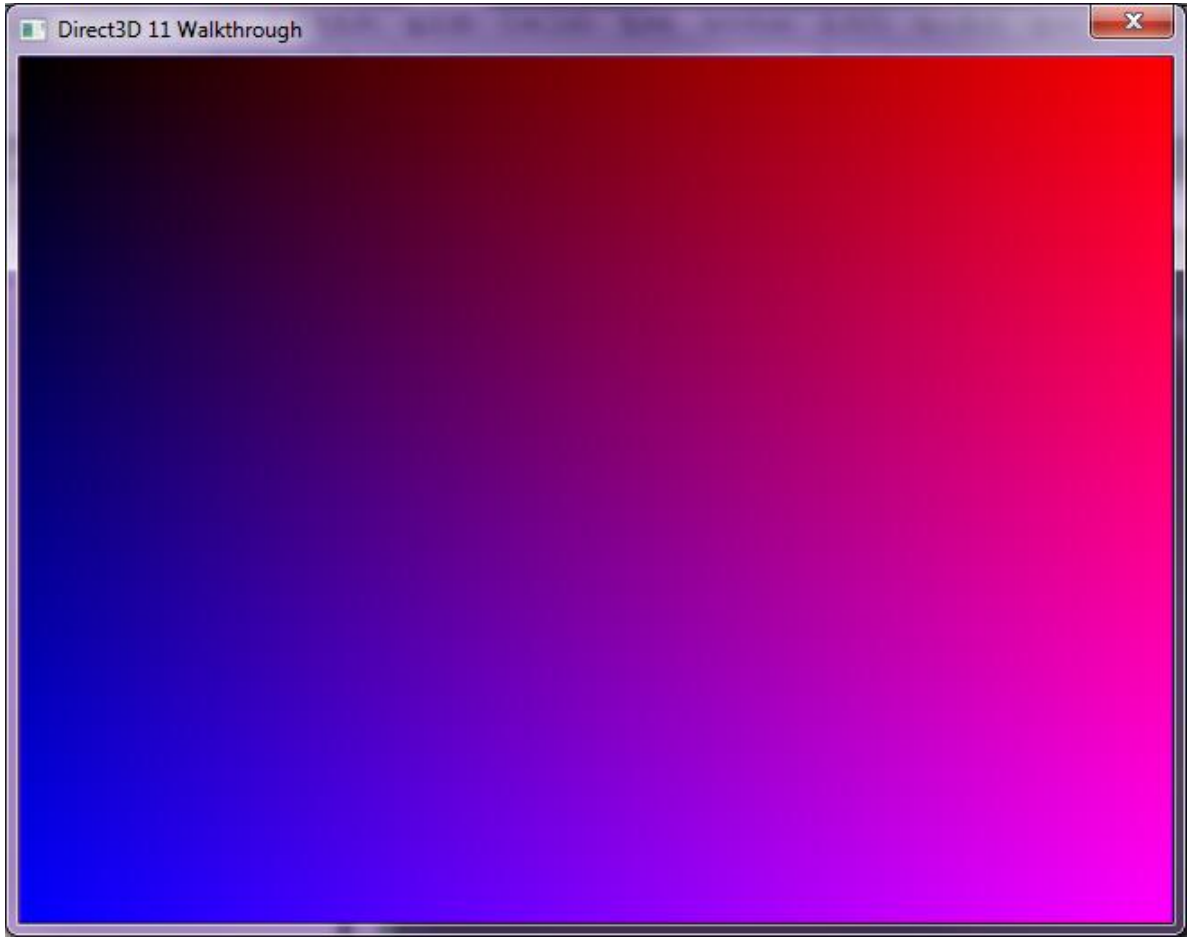
```
D3D11_MAPPED_SUBRESOURCE mapped;
hr = g_pImmediateContext->Map( g_pTexDynamic, D3D11CalcSubresource(0, 0, 1),
                               D3D11_MAP_WRITE_DISCARD, 0, &mapped );
if( FAILED(hr) )
    return hr;

BYTE* pPixel = (BYTE*)mapped.pData;

for( int y=0; y<desc.Height; y++)
{
    int iRowStart = y*mapped.RowPitch/sizeof(BYTE);
    for( int x=0; x<desc.Width; x++)
    {
        pPixel[iRowStart + x*4 + 0] = (BYTE)x;   //r
        pPixel[iRowStart + x*4 + 1] = 0;         //g
        pPixel[iRowStart + x*4 + 2] = (BYTE)y;   //b
        pPixel[iRowStart + x*4 + 3] = 255;       //a
    }
}

g_pImmediateContext->Unmap( g_pTexDynamic, D3D11CalcSubresource(0, 0, 1));
```

# CPU→GPU



# GPU→CPU

- Use previous code but change the render target
- Create staging texture
  - D3D11_USAGE_STAGING
  - D3D11_CPU_ACCESS_READ
- Copy texture from GPU to CPU
  - ID3D11DeviceContext::CopyResource
  - ID3D11DeviceContext::Map, Unmap
- Render a square using this CPU texture

# GPU→CPU

```
D3D11_TEXTURE2D_DESC desc;
ZeroMemory( &desc, sizeof(desc) );
desc.Width =640;
desc.Height = 480;
desc.MipLevels = 1;
desc.ArraySize = 1;
desc.Format = DXGI_FORMAT_R8G8B8A8_UNORM;
desc.SampleDesc.Count = 1;
desc.Usage = D3D11_USAGE_STAGING;
desc.BindFlags = 0;
desc.CPUAccessFlags = D3D11_CPU_ACCESS_READ;
desc.MiscFlags = 0;

hr = g_pd3dDevice->CreateTexture2D( &desc, NULL, &g_pTexStaging );
if( FAILED(hr) )
    return hr;
```

# GPU→CPU

```
g_pImmediateContext->CopyResource( g_pTexStaging, g_pTexDefault );

D3D11_MAPPED_SUBRESOURCE mappedSrc, mappedDst;
hr = g_pImmediateContext->Map( g_pTexStaging, D3D11CalcSubresource(0, 0, 1),
                               D3D11_MAP_READ, 0, &mappedSrc );
if( FAILED(hr) )
        return hr;
hr = g_pImmediateContext->Map( g_pTexDynamicCopyed, D3D11CalcSubresource(0, 0, 1),
                               D3D11_MAP_WRITE_DISCARD, 0, &mappedDst );
if( FAILED(hr) )
{
        g_pImmediateContext->Unmap( g_pTexStaging, D3D11CalcSubresource(0, 0, 1));
        return hr;
}

BYTE* pSrc = (BYTE*)mappedSrc.pData;
BYTE* pDst = (BYTE*)mappedDst.pData;

for( int y=0; y<640; y++)
{
        int iSrcRowStart = y*mappedSrc.RowPitch/sizeof(BYTE);
        int iDstRowStart = y*mappedDst.RowPitch/sizeof(BYTE);

        for( int x=0; x<480; x++)
        {
                pDst[iDstRowStart + x*4 + 0] = pSrc [iSrcRowStart + x*4 + 0]
                pDst[iDstRowStart + x*4 + 1] = pSrc [iSrcRowStart + x*4 + 0]
                pDst[iDstRowStart + x*4 + 2] = pSrc [iSrcRowStart + x*4 + 0]
                pDst[iDstRowStart + x*4 + 3] = pSrc [iSrcRowStart + x*4 + 0]
        }
}

g_pImmediateContext->Unmap( g_pTexDynamic, D3D11CalcSubresource(0, 0, 1));
g_pImmediateContext->Unmap( g_pTexDynamic, D3D11CalcSubresource(0, 0, 1));
```

# GPU→CPU