Sending data to the GPU. Textures and transforms.

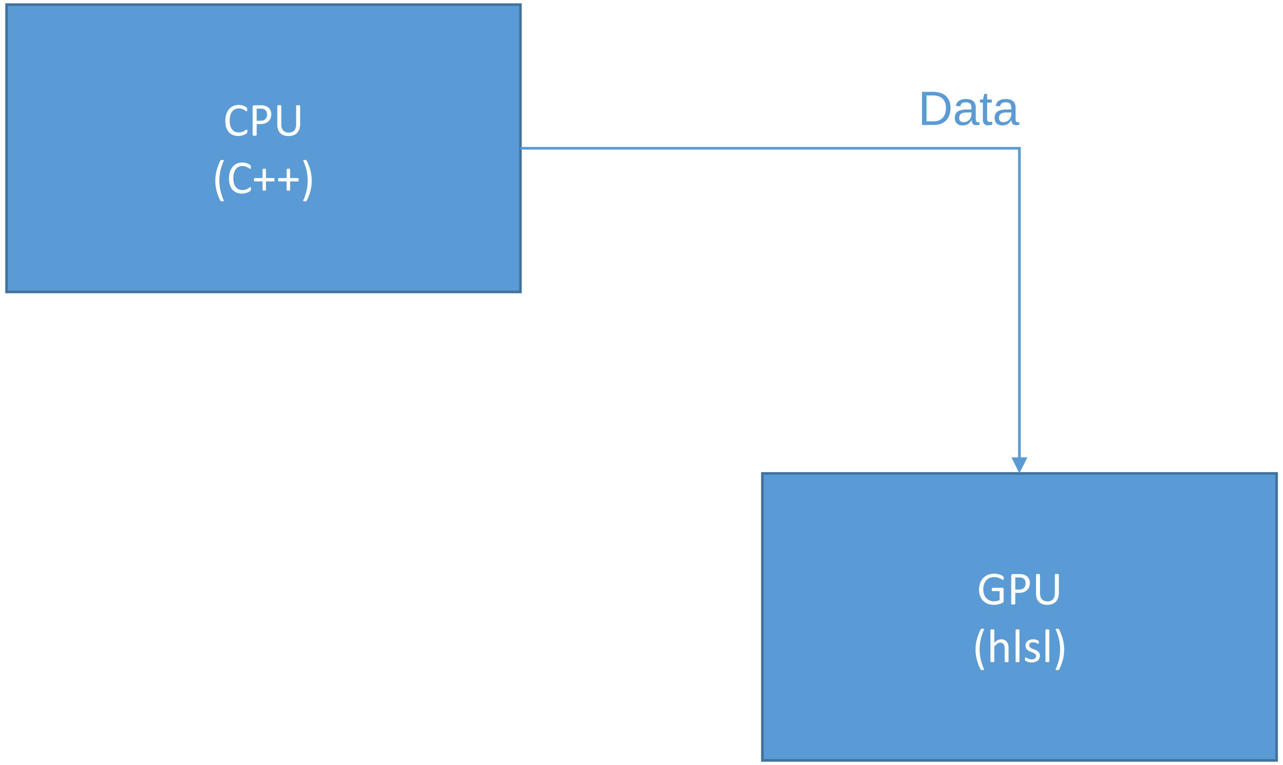
CMP301 Graphics Programming with Shaders

Overview

- Look at sending data to shaders
  - Semantics
  - Registers
- Creating a sampler
- Working with Textures
- Texture shaders / rendering
- Transforms

What data are we sending

- Geometry data
  - Vertex position
  - Texture coordinates
  - Normals
  - (could add more)
- Additional / Non-geometric data
  - Matrices
  - Lighting information
  - Blending values
  - Etc
- Texture data
  - Texture(s)
  - sampler



Geometric data

- On the CPU side this data is stored a vertex buffer and index buffer
  - In the mesh class
- Passed to the GPU via the Input-Assembler Stage
- Traditionally this data includes
  - Vertex position
  - Texture coordinates
  - Normals
- Could contain additional data such as
  - Colour
  - Binormal
  - Tangent
- Data description on both CPU and GPU must match

Geometric data

```cpp
class BaseMesh
{
protected:

    struct VertexType
    {
        XMFLOAT3 position;
        XMFLOAT2 texture;
        XMFLOAT3 normal;
    };
```

```hlsl
struct InputType
{
    float4 position : POSITION;
    float2 tex : TEXCOORD0;
    float3 normal : NORMAL;
};
```

```cpp
vertexCount = 3;
indexCount = 3;

vertices = new VertexType[vertexCount];
indices = new unsigned long[indexCount];

// Load the vertex array with data.
vertices[0].position = XMFLOAT3(0.0f, 1.0f, 0.0f);  // Top.
vertices[0].texture = XMFLOAT2(0.5f, 0.0f);
vertices[0].normal = XMFLOAT3(0.0f, 0.0f, -1.0f);

vertices[1].position = XMFLOAT3(-1.0f, 0.0f, 0.0f);  // bottom left.
vertices[1].texture = XMFLOAT2(0.0f, 1.0f);
vertices[1].normal = XMFLOAT3(0.0f, 0.0f, -1.0f);

vertices[2].position = XMFLOAT3(1.0f, 0.0f, 0.0f);  // bottom right.
vertices[2].texture = XMFLOAT2(1.0f, 1.0f);
vertices[2].normal = XMFLOAT3(0.0f, 0.0f, -1.0f);

// Load the index array with data.
indices[0] = 0;  // Top/
indices[1] = 1;  // Bottom left.
indices[2] = 2;  // Bottom right.
```

Semantics

- Define what variables will be used for
  - Allows for optimisation
- Required on all variables passed between shaders
  - Must match!
- Quite a few different types, we will look at some common ones
  - Most are arbitrary
  - Some are important system values
    - Effect the data
- Seen some last week

Semantics

```
// TYPEDEFS
struct InputType
{
    float4 position : POSITION;
    float2 tex : TEXCOORD0
    float3 normal : NORMAL;
};

struct OutputType
{
    float4 position : SV_POSITION;
    float2 tex : TEXCOORD0
    float3 normal : NORMAL;
};
```

Semantics

- Position
  - Vertex position in object space
- Texcoord[n]
  - Texture coordinates
- Normal[n]
  - Normal vector
- SV_Position
  - System value
  - Transformed position (for rasterization) (screen space)

Non-geometric data

- Sent straight to the required shader stage
- If the data is required at multiple shader stages, the data must be sent multiple times
- Typically contains
  - Matrix data
  - Lighting data
  - Manipulation data
  - Etc

Registers

- When we send data to the gfx card how does it know where to go?
  - What if I have multiple buffers or texture resources
  - How does the data end up in the correct variable
- Without registers a constant buffer is automatically mapped to one of 15 registers
  - Corresponding to the stage of the pipeline
- We can’t trust it to automatically get it right!

Registers

- Manually configure registers
  - Handy
  - Easy
  - Reduces risk of variables getting messed up
  - Flags the buffer for easy location
- Some examples

Registers

- Non-registered constant buffer

```
cbuffer MatrixBuffer
{
    matrix worldMatrix;
    matrix viewMatrix;
    matrix projectionMatrix;
};
```

Registers

- Registered constant buffer

```
cbuffer MatrixBuffer : register(cb0)
{
    matrix worldMatrix;
    matrix viewMatrix;
    matrix projectionMatrix;
};
```

- or

```
Texture2D texture0 : register(t0);
```

Registers

- When sending data to the gfx card we can specify the slot it goes in

```
deviceContext->VSSetConstantBuffers(SLOT, 1, &m_matrixBuffer);

deviceContext->PSSetShaderResources(0, 1, &texture);
```

C++ and HLSL

```cpp
class BaseShader
{
protected:
    struct MatrixBufferType
    {
        XMMATRIX world;
        XMMATRIX view;
        XMMATRIX projection;
    };
```

```hlsl
// Simple geometry pass
// texture coordinates and normals will be ignored.

cbuffer MatrixBuffer : register(cb0)
{
    matrix worldMatrix;
    matrix viewMatrix;
    matrix projectionMatrix;
};
```

Textures

- Similar to the “additional / non-geometric data” just mentioned
- Textures and samplers have specific registers for sending/receiving them

```
    deviceContext->VSSetConstantBuffers(bufferNumber, 1, &matrixBuffer);

    // Set shader texture resource in the pixel shader.
    deviceContext->PSSetShaderResources(0, 1, &texture);
}
```

```
// Texture pixel/fragment shader
// Basic fragment shader for rendering textured geometry


Texture2D texture0 : register(t0);
SamplerState Sampler0 : register(s0);
```

Sampler

- This should sound familiar
- Sampler determines how the texture is to be sampled and filtered
- Unlike last year we have to define a sampler
- Some are specified in HLSL
  - Deprecated
- Define the Sampler in C++ and pass it to the required shader like a resource

Sampler

- How?
- Configure Sampler description
  - How the texture will be sampled
  - Address options
  - And a few other setting

Sampler

- Filter examples
  - D3D11_FILTER_MIN_MAG_MIP_LINEAR
  - D3D11_FILTER_MIN_POINT_MAG_LINEAR_MIP_POINT
  - D3D11_FILTER_ANISOTROPIC
  - Etc
- http://msdn.microsoft.com/en-gb/library/windows/desktop/ff476132(v=vs.85).aspx

Sampler

- Texture addressing mode
  - Texcoords outside 0 to 1 range
  - Applied to 3 address components (U, V, W)
  - Wrap
  - Mirror
  - Clamp
  - Border
  - Mirror once
- Example
  - D3D11_TEXTURE_ADDRESS_WRAP

Sampler

- Other properties include
  - MipLODBias
  - MaxAnisotropic
    - Clamps the value used if anisotropic filtering
  - ComparisionFunc
    - Compare function used for sampled data and existing sampled data
  - BorderColor
    - Border colour used if D3D11_TEXTURE_ADDRESS_BORDER is used
  - MinLOD and Max LOD
    - Upper and lower end of mipmap range for clamping access

Sampler

```
// Create a texture sampler state description.
samplerDesc.Filter = D3D11_FILTER_MIN_MAG_MIP_LINEAR;
samplerDesc.AddressU = D3D11_TEXTURE_ADDRESS_CLAMP;
samplerDesc.AddressV = D3D11_TEXTURE_ADDRESS_CLAMP;
samplerDesc.AddressW = D3D11_TEXTURE_ADDRESS_CLAMP;
samplerDesc.MipLODBias = 0.0f;
samplerDesc.MaxAnisotropy = 1;
samplerDesc.ComparisonFunc = D3D11_COMPARISON_ALWAYS;
samplerDesc.BorderColor[0] = 0;
samplerDesc.BorderColor[1] = 0;
samplerDesc.BorderColor[2] = 0;
samplerDesc.BorderColor[3] = 0;
samplerDesc.MinLOD = 0;
samplerDesc.MaxLOD = D3D11_FLOAT32_MAX;

// Create the texture sampler state.
device->CreateSamplerState(&samplerDesc, &m_sampleState);
```

Types of textures

- Colour/Color maps
  - Commonly referred to as Decal maps
  - Used for colouring surfaces
- Specular maps
  - Also known as gloss maps
  - Used for masking highlights
- Light maps
  - Pre-rendered lighting or shadows stored in textures
  - UDK is big on this

Types of textures

- Ambient occlusion maps
  - Attempts to approximate the way light radiates in real life
  - A global method meaning illumination is determined by other geometry in the scene
- Shadow maps
  - Used for real time shadow generation and rendering
- Displacement maps
  - Commonly used for geometry displacement and deformation
  - Aka height maps
- Normal maps
  - Used during per-pixel lighting to simulate high-complexity geometry on low resolution geometry
- Alpha maps, cube maps, etc

Texture format

- The framework can load a small range of image files
  - DDS
  - JPG
  - PNG
- Using the DirectXTK
- I have provided some images for testing purposes
  - Check in the **res** folder

Shader resource

- Resources are areas in memory that can be accessed by the Direct3D pipeline
- Resources contain data
  - Geometry
  - Textures
  - Shader data
- Resources can have both read and write access
- Accessible to only the CPU, GPU, or both
- Up to 128 resources can be active for each pipeline stage

Loading the texture

- There is a texture manager class designed for loading and storing textures
- It is initialised as part of the BaseApplication
- In your application

```
    // Create Mesh object, load required textures and load shaders.
    mesh = new QuadMesh(renderer->getDevice(), renderer->getDeviceContext());

    textureMgr->loadTexture("brick", L"../res/brick1.dds");

    textureShader = new TextureShader(renderer->getDevice(), hwnd);

}
```

```
// Send geometry data (from mesh)
mesh->sendData(renderer->getDeviceContext());
// Set shader parameters (matrices and texture)
textureShader->setShaderParameters(renderer->getDeviceContext(), worldMatrix, viewMatrix, projectionMatrix, textureMgr->getTexture("brick"));
// Render object (combination of mesh geometry and shader process
textureShader->render(renderer->getDeviceContext(), mesh->getIndexCount());
```

Texture shader class

- Like the colour shader class
  - Loads our shaders
  - Configures a sampler
- Will load two new shaders
  - Texture_vs.hlsl
  - Texture_ps.hlsl

New shaders

- New Vertex shader
  - Very similar to last week
- New pixel shader
  - Samples the texture to calculate pixel colour

Texture_vs

```
cbuffer MatrixBuffer : register(cb0)
{
    matrix worldMatrix;
    matrix viewMatrix;
    matrix projectionMatrix;
};

struct InputType
{
    float4 position : POSITION;
    float2 tex : TEXCOORD0;
    float3 normal : NORMAL;
};

struct OutputType
{
    float4 position : SV_POSITION;
    float2 tex : TEXCOORD0;
    float3 normal : NORMAL;
};
```

Texture_vs

```
OutputType main(InputType input)
{
    OutputType output;

    // Change the position vector to be 4 units for proper matrix calculations.
    input.position.w = 1.0f;

    // Calculate the position of the vertex against the world, view, and projection matrices.
    output.position = mul(input.position, worldMatrix);
    output.position = mul(output.position, viewMatrix);
    output.position = mul(output.position, projectionMatrix);

    // Store the texture coordinates for the pixel shader.
    output.tex = input.tex;

    // Calculate the normal vector against the world matrix only.
    output.normal = mul(input.normal, (float3x3)worldMatrix);

    // Normalize the normal vector.
    output.normal = normalize(output.normal);

    return output;
}
```

Texture_ps

```
Texture2D texture : register(t0);
SamplerState sampler : register(s0);

struct InputType
{
    float4 position : SV_POSITION;
    float2 tex : TEXCOORD0;
    float3 normal : NORMAL;
};
```

Texture_ps

```
float4 main(InputType input) : SV_TARGET
{
    float4 textureColour;

    // Sample the pixel colour from the texture using the sampler at the        texture
coordinate location.
    textureColour = texture.Sample(sampler, input.tex);

    return textureColour;
}
```

Transforms

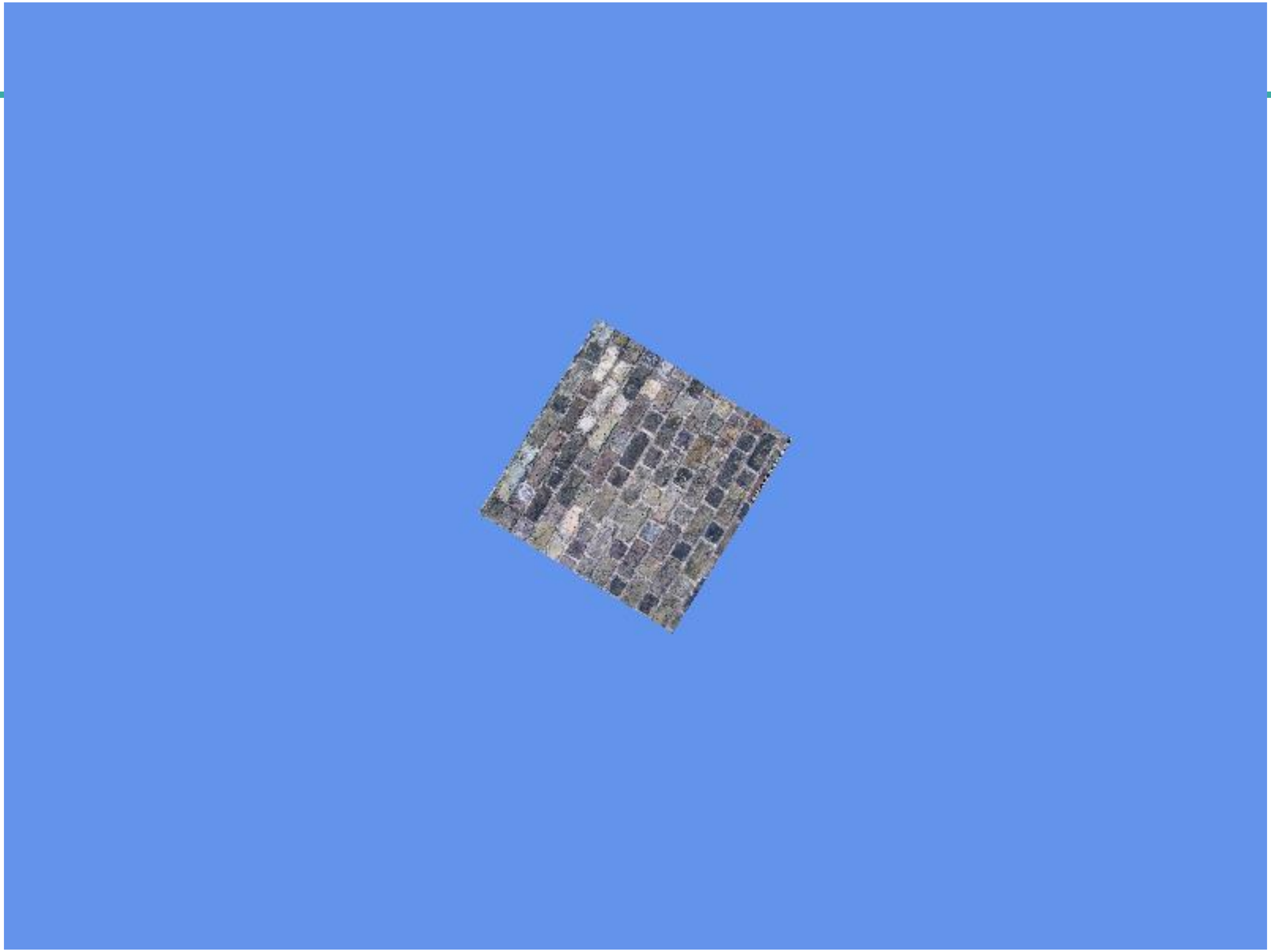
- Translation
- Rotation
- Scale
- Provided functions to generate required matrices
- We need to combine matrices and pass the correct matrix to the vertex shader

Translation

```
worldMatrix = XMMatrixTranslation(2.0, 0.0, 0.0);
```

Rotation

```
worldMatrix = XMMatrixRotationRollPitchYaw(0.0, 0.0, 45.0);
```



Scaling

```
worldMatrix = XMMatrixScaling(2.0, 1.0, 1.0);
```

Multiply

- worldMatrix = XMMatrixMultiply(matrix2, matrix1);

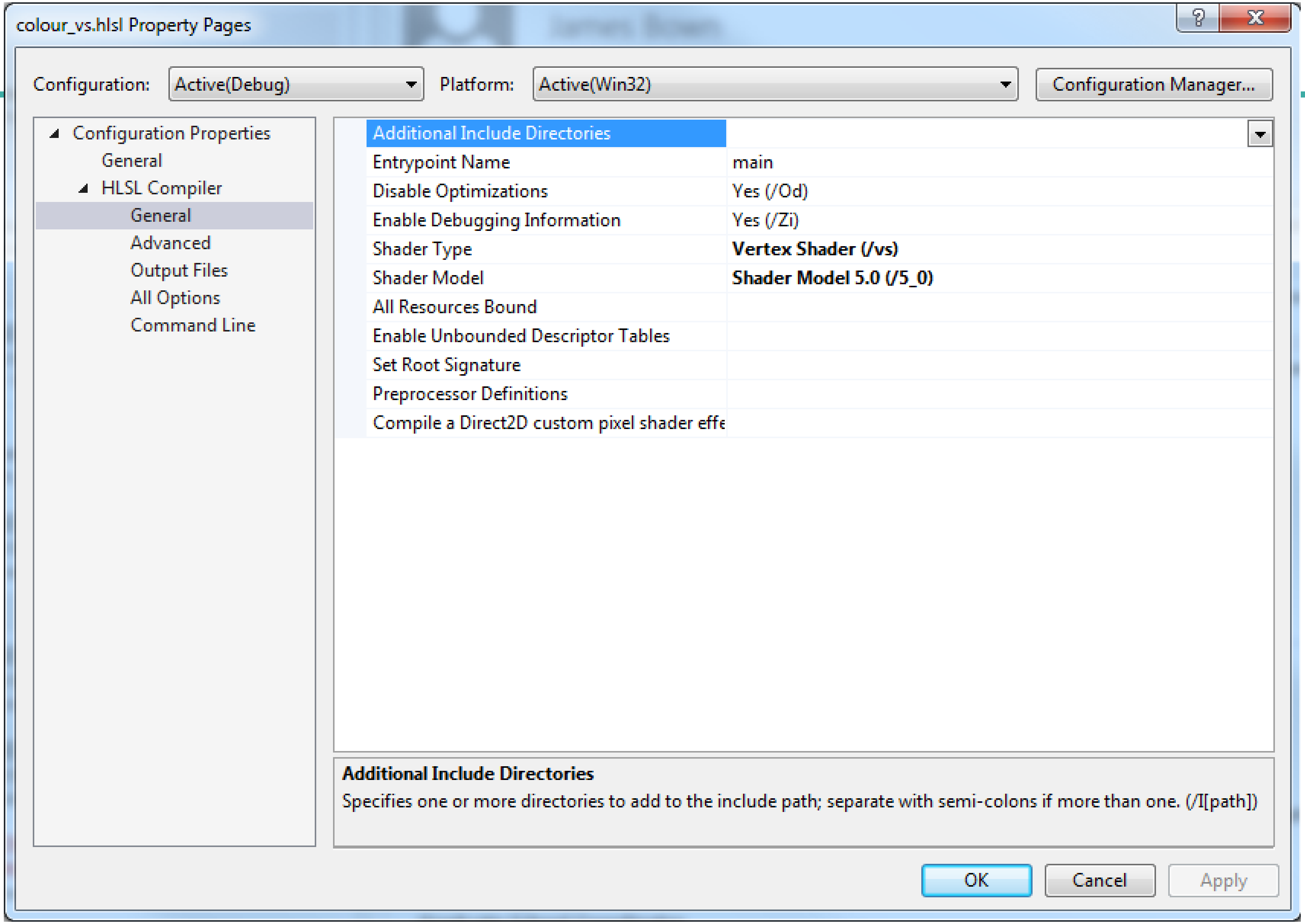
- Order is important
  - Based on order you want them applied

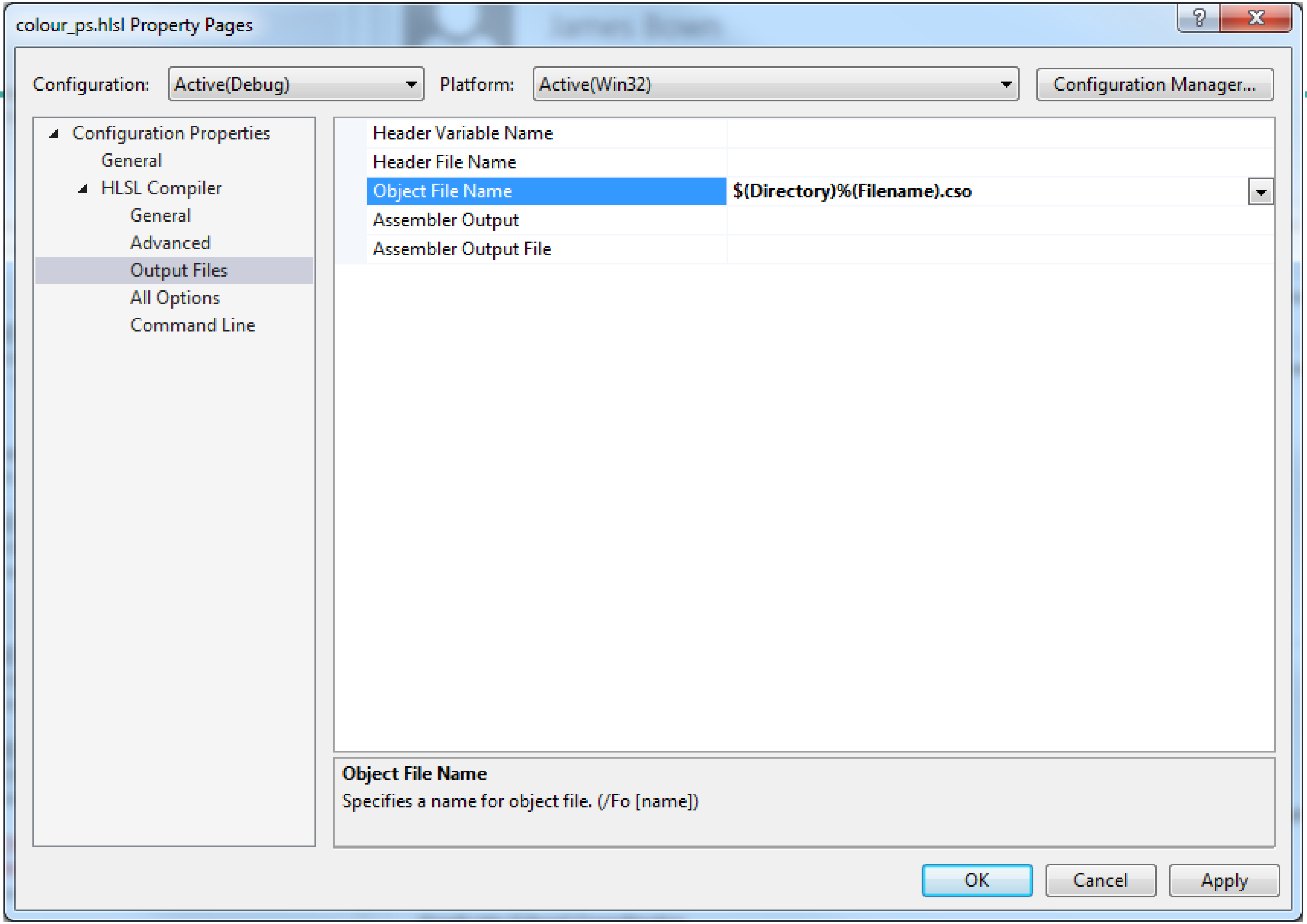
- Do you want to rotate first then translate
- Or translate first then rotate
- Same applies to matrix2 * matrix1

End

- In the labs we will be working with textures and transforms
- Some code will be provided ☺

- As a note, when creating new shader files you will need to specify there type in Visual Studio…


colour_vs.hlsl Property Pages
Configuration: Active(Debug)
Platform: Active(Win32)
Configuration Manager...
Configuration Properties
General
HLSL Compiler
General
Advanced
Output Files
All Options
Command Line
Additional Include Directories
Entrypoint Name
main
Disable Optimizations
Yes (/Od)
Enable Debugging Information
Yes (/Zi)
Shader Type
Vertex Shader (/vs)
Shader Model
Shader Model 5.0 (/5_0)
All Resources Bound
Enable Unbounded Descriptor Tables
Set Root Signature
Preprocessor Definitions
Compile a Direct2D custom pixel shader effe
Additional Include Directories
Specifies one or more directories to add to the include path; separate with semi-colons if more than one. (/I[path])
OK
Cancel
Apply


colour_ps.hlsl Property Pages
Configuration: Active(Debug)
Platform: Active(Win32)
Configuration Manager...
Configuration Properties
General
HLSL Compiler
General
Advanced
Output Files
All Options
Command Line
Header Variable Name
Header File Name
Object File Name
$(Directory)%(Filename).cso
Assembler Output
Assembler Output File
Object File Name
Specifies a name for object file. (/Fo [name])
OK
Cancel
Apply