Shader Programming

BY TIGRAN GASPARIAN

Who am I?

Name: Tigran Gasparian

Age: 22

Master student Game and Media Technology

Working on a game for almost two years already

- The Flock – multiplayer horror game
- Like us on Facebook!
- Doing a lot of graphics programming in Unity

Table of contents

What are shaders?

Programs that run on the graphics card

- Somewhat limited compared to languages like C#
- Because of these limitations, very good at certain tasks
- Perform some of the computations necessary for graphics

We'll cover pixel and vertex shaders

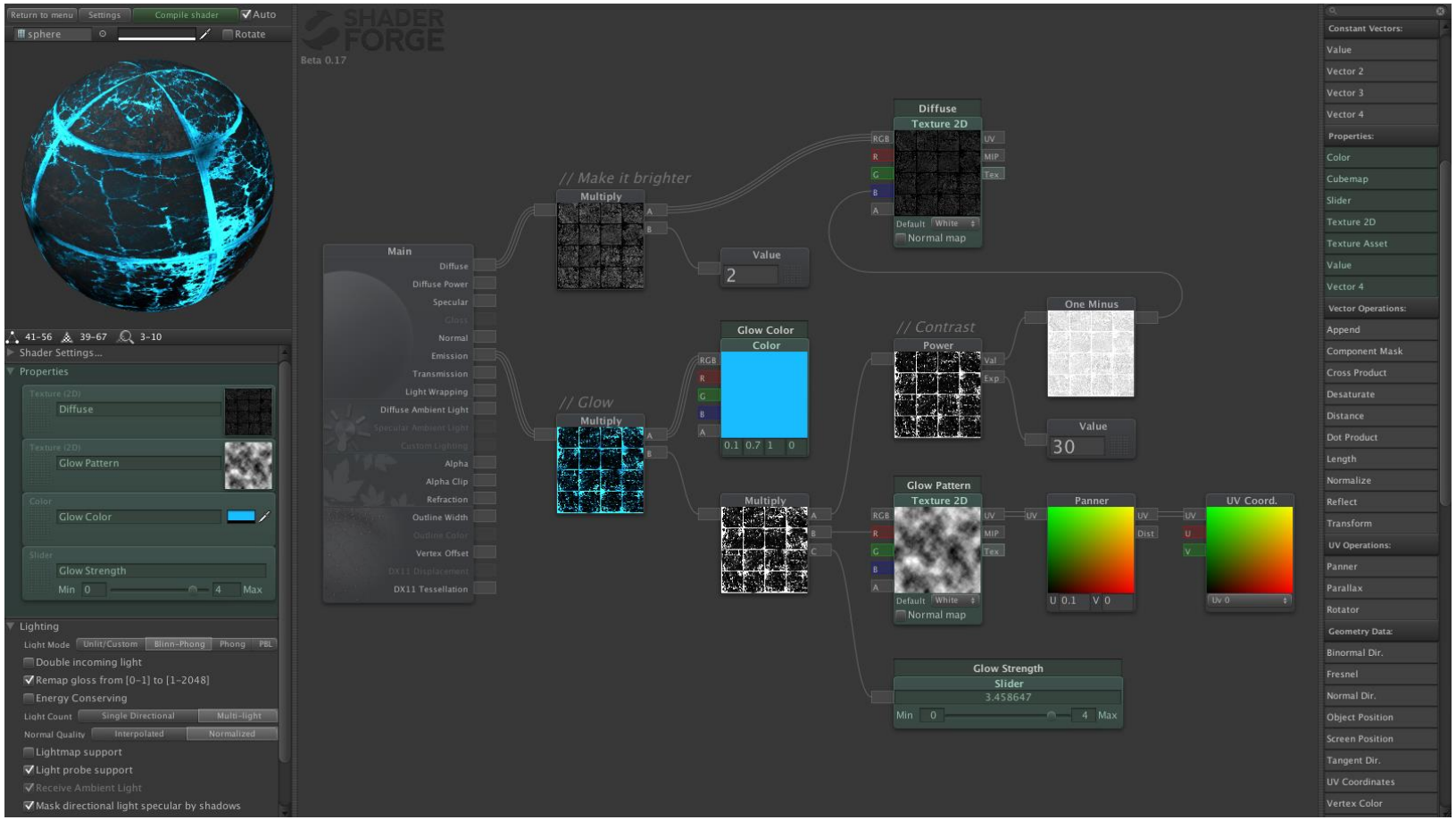
- Geometry, Tessellation and Compute shaders not covered here.

Written in some shading language

- HLSL, GLSL, CG, PSSL, MSL, etc.
- We're going to use HLSL – High Level Shading Language

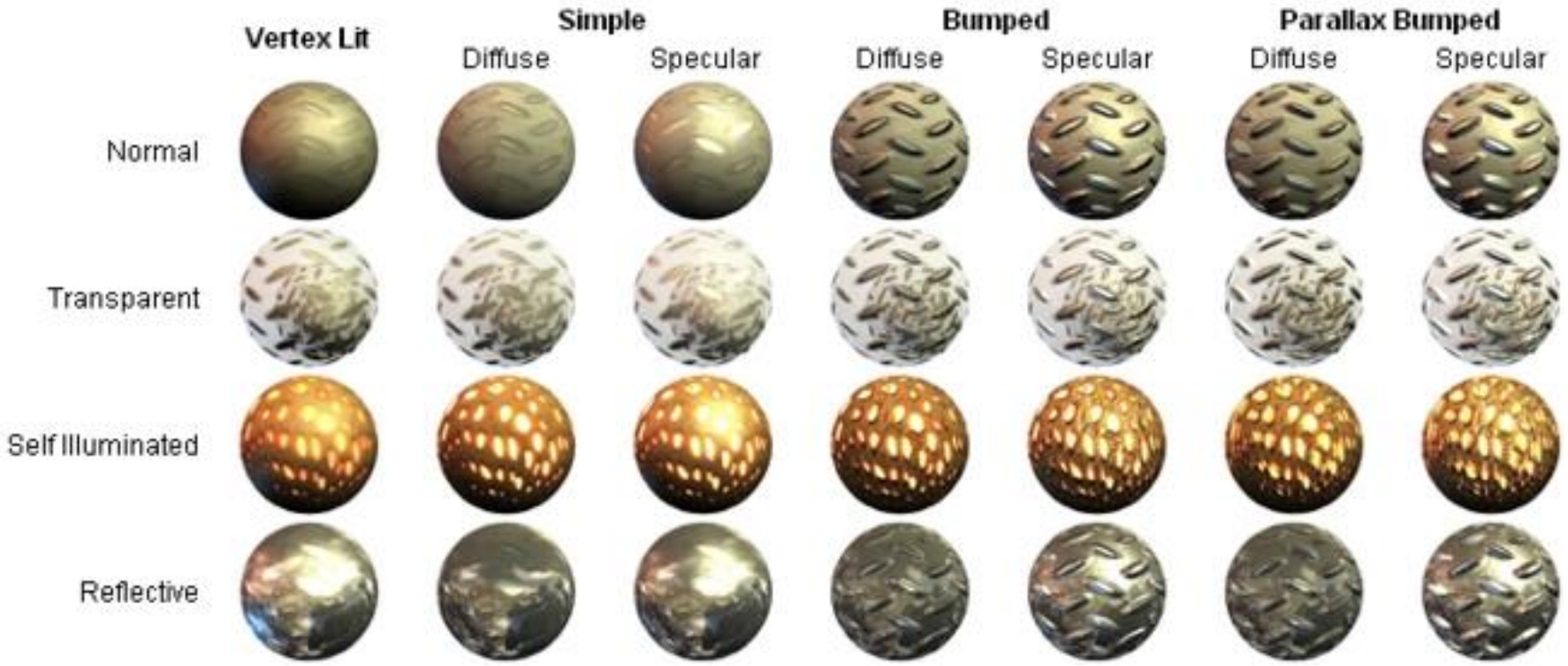
Node-based editor

What are shaders?



Why learn to write shaders?

Game engines come with lots of built-in shaders



Why learn to write shaders?

Create a unique look and feel for your game

- Want something never done before? Do it yourself!

Implement state of the art techniques

- The latest and greatest techniques don’t have standard implementations yet.

Unlock the infinite powers of the GPU!

- Well, not infinite, but it’s certainly powerful.
- 14x speedup according to Intel
- 300x speedup according to NVIDIA
- Depends on the application.

Why learn to write shaders?

It helps you pass the second practicum.

The Graphics Pipeline

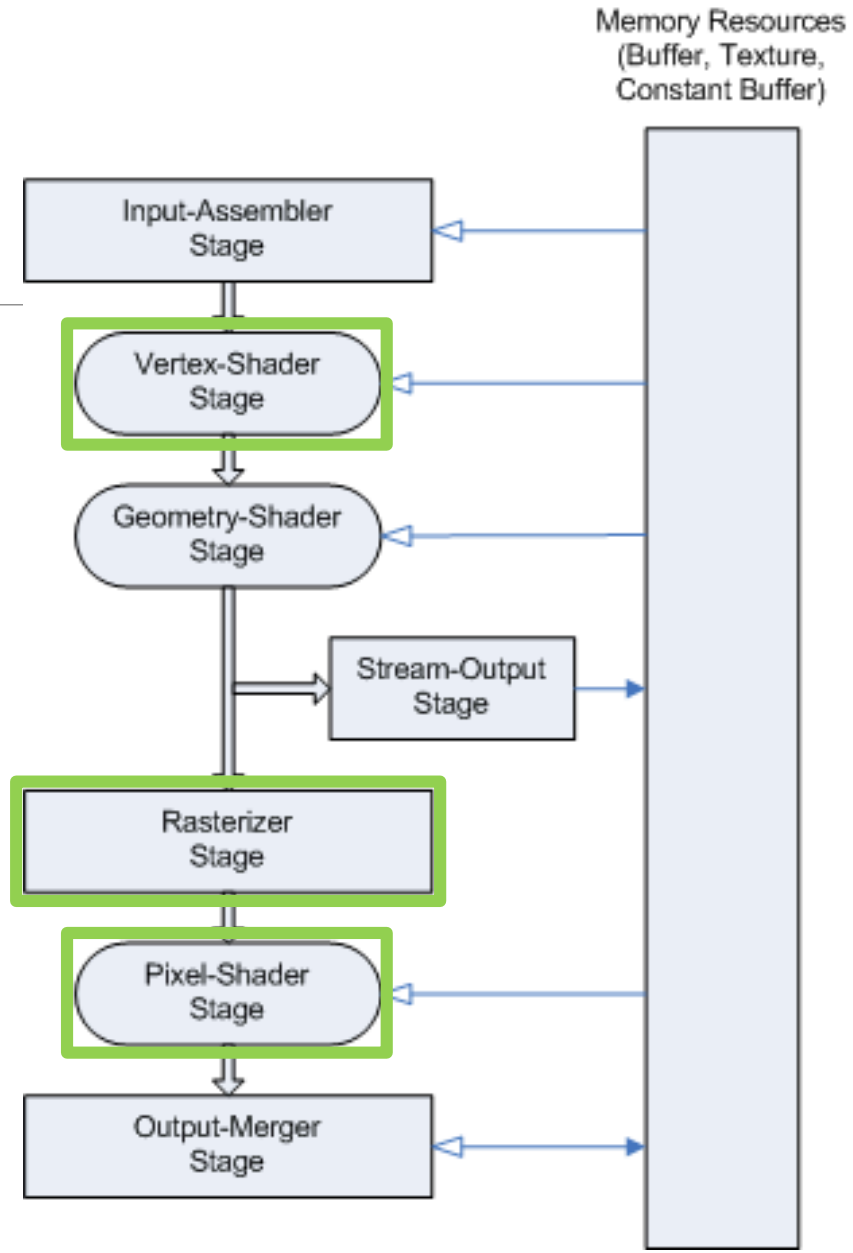
What is the graphics pipeline?

- A sequence of actions that is performed to render a 2D image from a 3D scene

We'll cover the following stages:

- Vertex Shader
- Rasterizer
  - Can't be programmed, but it's an important stage
- Pixel Shader

Will be covered in-depth in the next lecture.



The Graphics Pipeline

Vertex data + resources + graphics card = image!

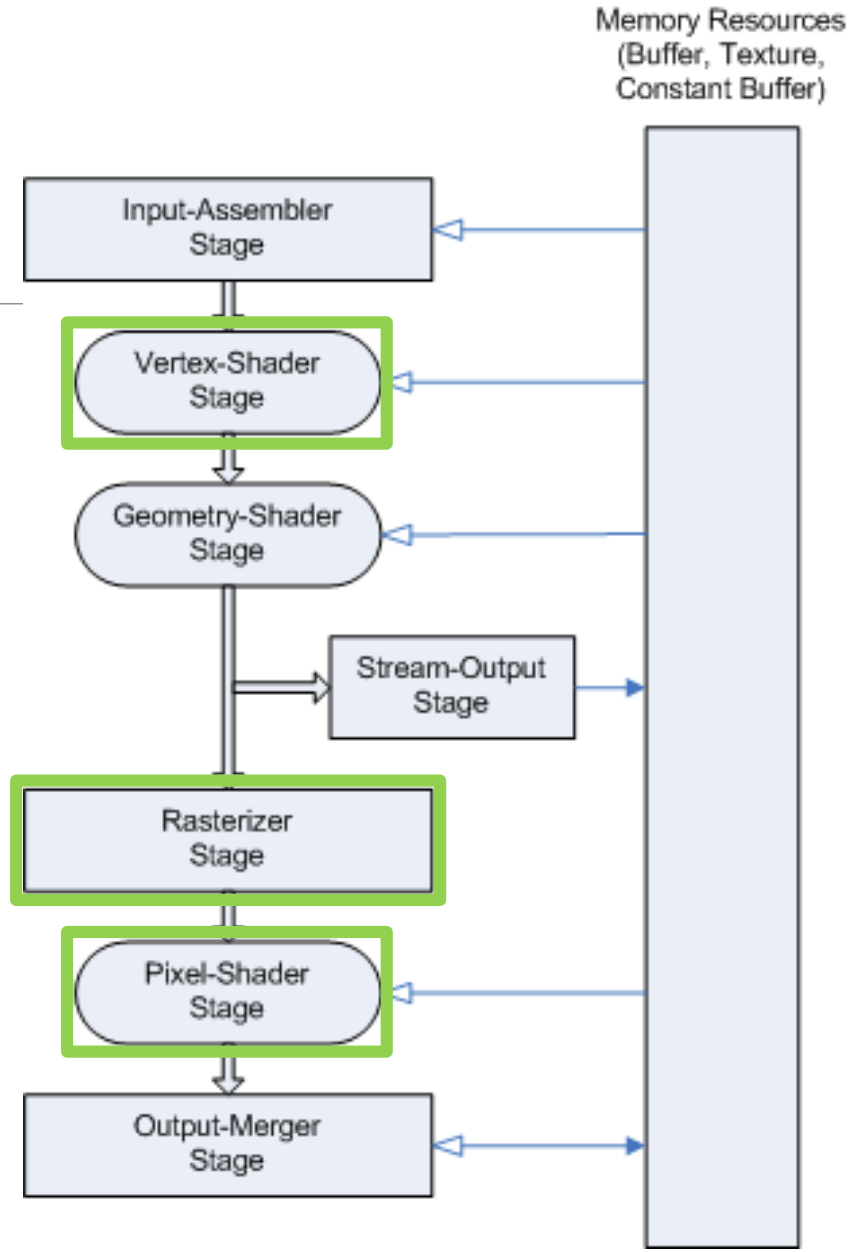
What is vertex data?

- Mostly triangles

What kind of resources?

- Textures
- Global variables
  - Like the light color
- Lots of other stuff we won't cover.

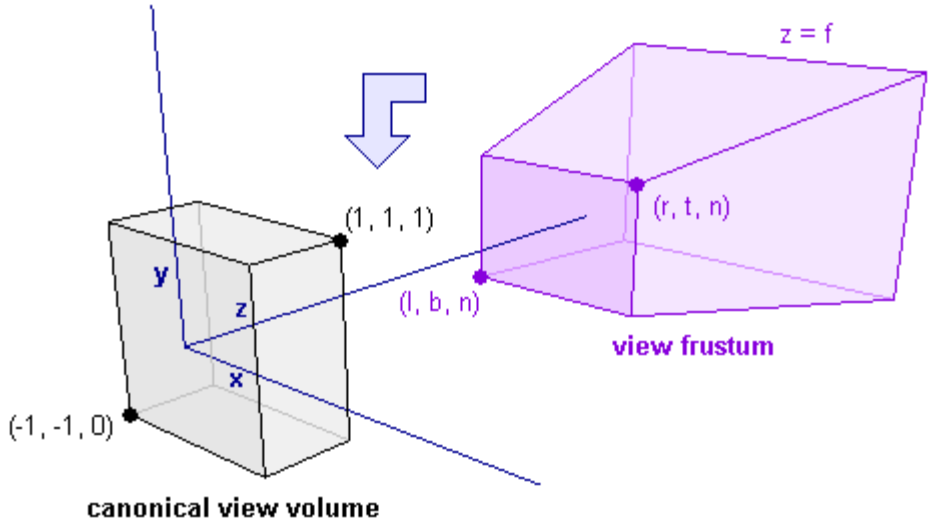
We first set the resources, then we push the vertices to the graphics card

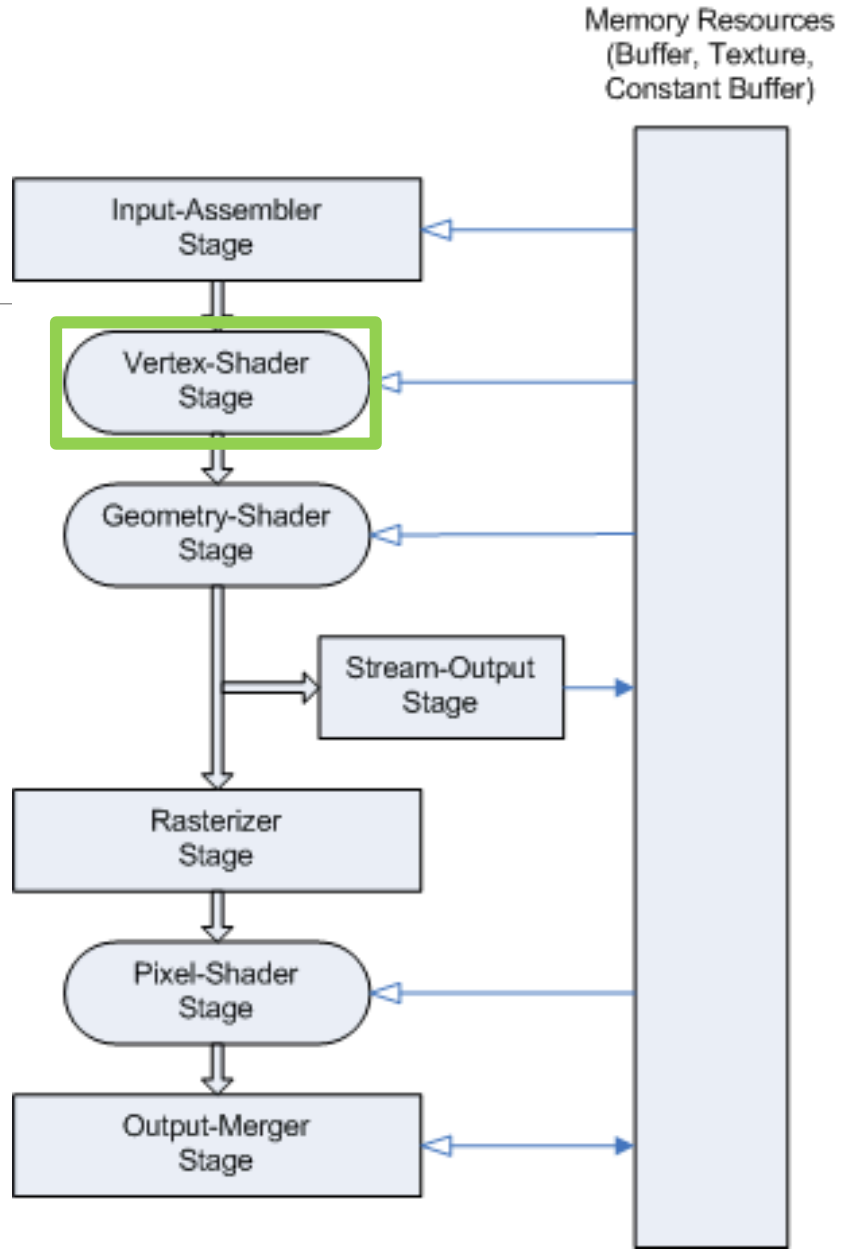


The Vertex Shader

Perform some computations per vertex.

- Can move the positions of vertices.
- Usually transforms vertices
  - Using the World, View and Projection matrices

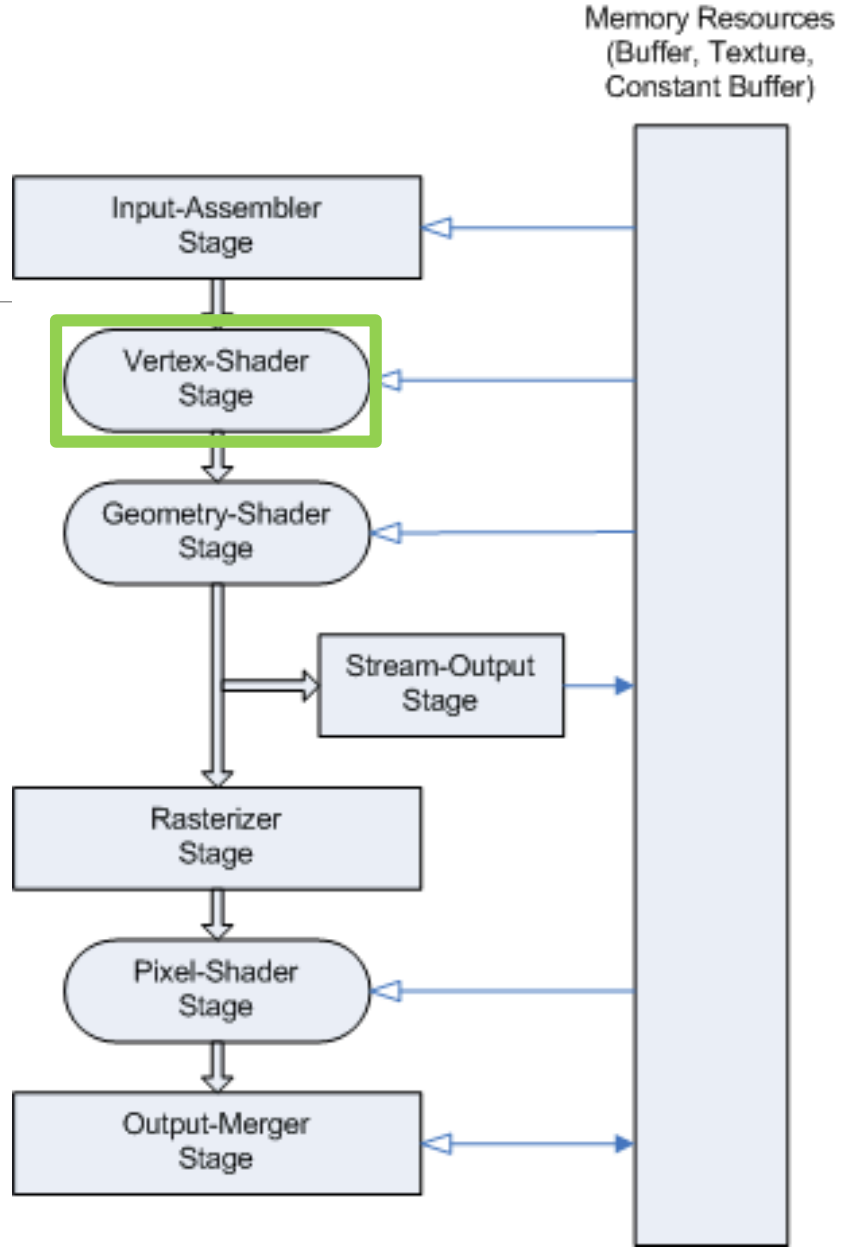




The Vertex Shader

Perform some computations per vertex.

- Can move the positions of vertices.
- Usually transforms vertices
  - Using the World, View and Projection matrices
- Other deformations

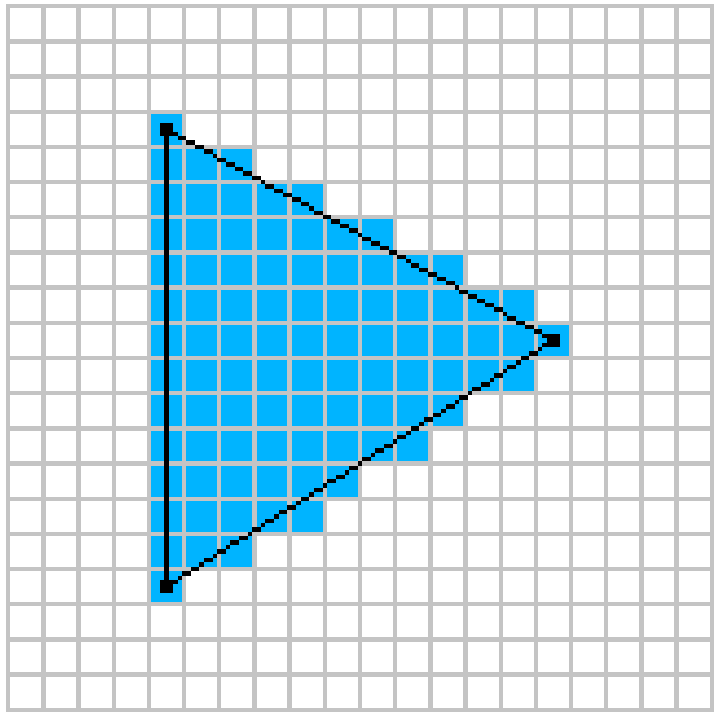


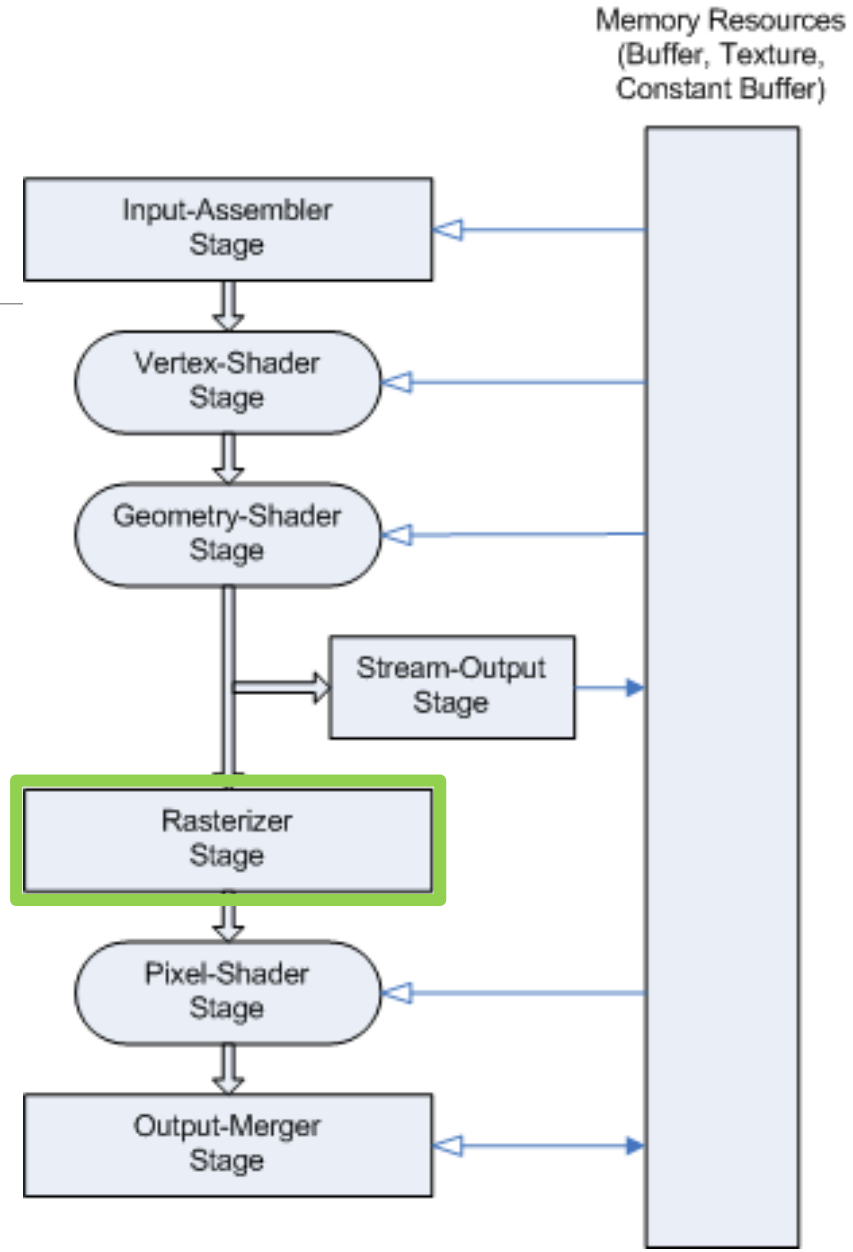
The Rasterizer

Turns triangles into pixels!

Interpolates vertex data

- More about this later





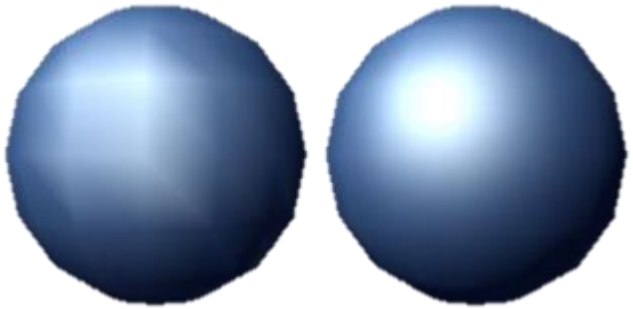
The Pixel Shader

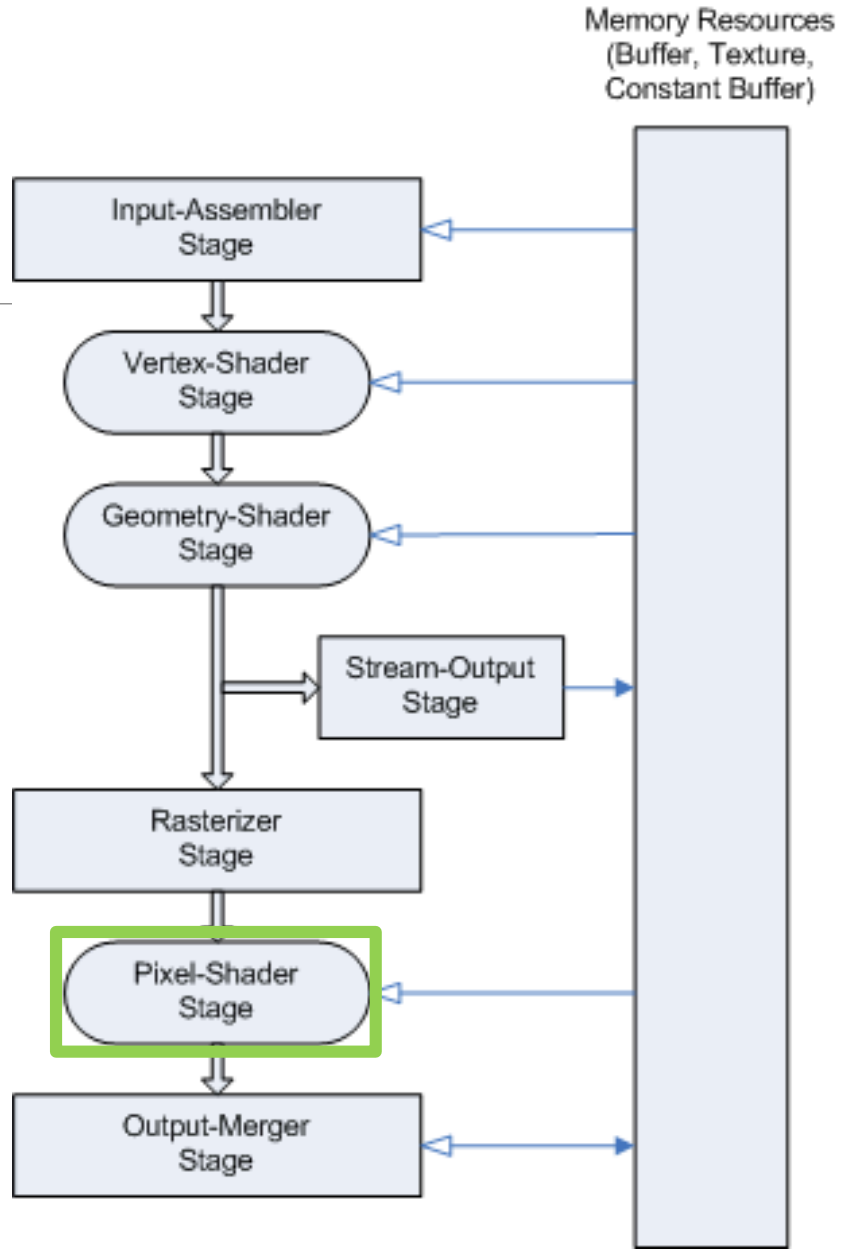
Perform computations per pixel

- More accurate than per-vertex computations
- Can’t change the shape of the geometry
  - Vertex shaders can!

Common uses

- Texture mapping
  - We need to sample the texture for every pixel
- Per pixel lighting





The Vertex Shader – In-Depth

What's a vertex shader?

- Just a function called for every vertex

Input: Vertex data

- A struct containing vertex information
- Let's call it: VertexShaderInput

Output: A struct

- Can contain anything
- Let's call it: VertexShaderOutput

Observation: Vertex shader doesn't know about triangles

The Vertex Shader – VertexShaderInput

```
struct VertexShaderInput
{
        float4 Position : POSITION0;
        float4 Color : COLOR0;
        float2 TextureCoordinate : TEXCOORD0;
        float SomeCustomData : TEXCOORD1;
};
```

The data sent from the CPU to the GPU

HLSL code

Looks like C

- New data types
- Input semantics

The Vertex Shader – VertexShaderOutput

```
struct VertexShaderOutput
{
        float4 Position : POSITION0;
        float4 Color : COLOR0;
        float3 VertexPosition : TEXCOORD0;
};
```

Must have member with POSITION0 semantic

- Contains vertex position transformed to normalized device coordinates

Sidetrack #1 - Input semantics

Required for interaction with C# code

- More on this later

Lots of remnants from the fixed function pipeline era.

Lots of semantics

- POSITION[n]
- TEXCOORD[n] (e.g. TEXCOORD0, TEXCOORD1, etc..)
  - Use this for all custom data
- COLOR[n]
  - Clamped between 0 and 1
- NORMAL[n]
- Many more…

Sidetrack #2 - HLSL Data types

HLSL has some specialized data types

- float – nothing special
- float2 – 2D vector
- float3 – 3D vector
- float4 – 4D vector

- float4x4 – 4x4 float matrix
- float2x3 – 2x3 float matrix

- Same thing with
  - half – 16 bit floats
  - fixed – 8 bit floats
  - int – 32 bit integer
  - etc.

Sidetrack #3 - HLSL Data types

```
float3 position = float3(0, 0, 0);
float3 direction = float3(1, 2, 1.2f);

float someValue = 5;

position += direction * someValue;

float yDirection = direction.y;
float2 xyDirection = direction.xy;
```

Works mostly like you'd expect it to.

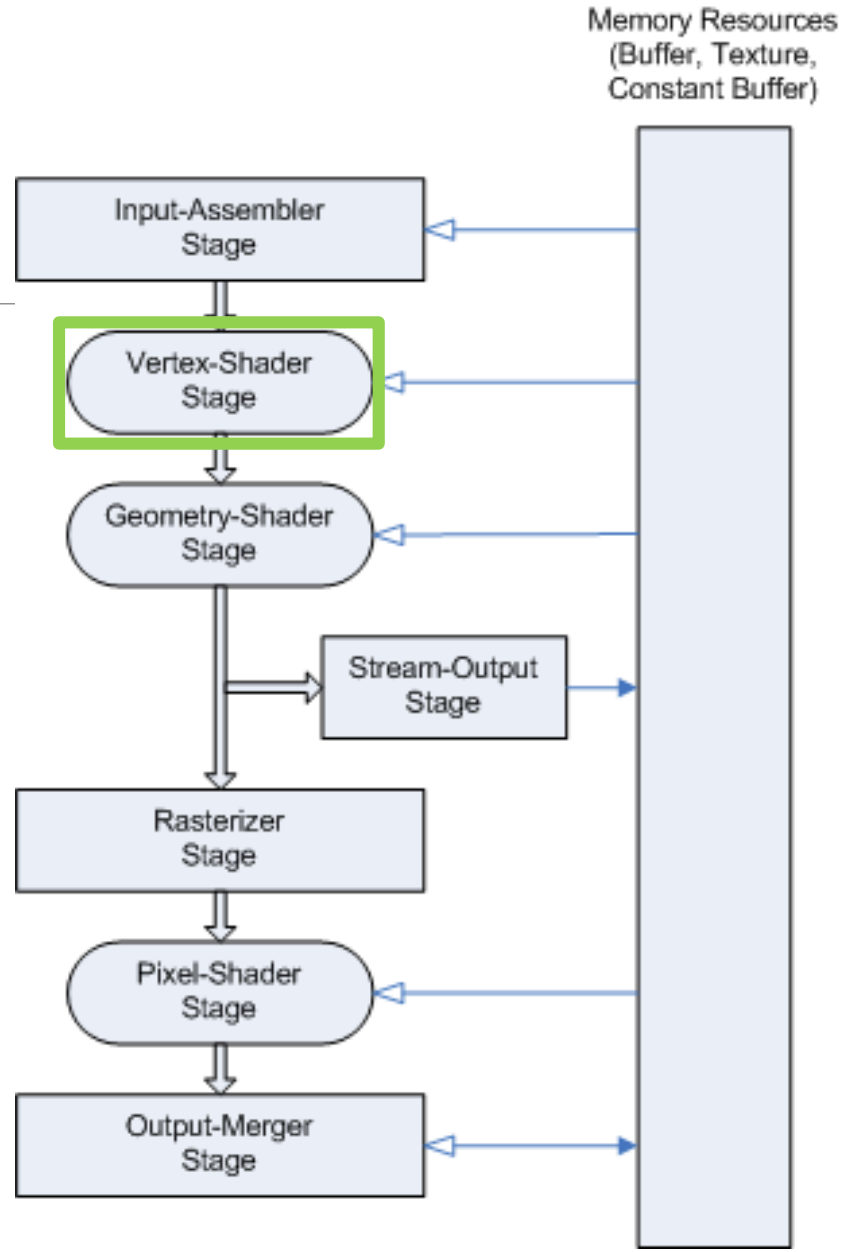
We'll cover more HLSL later on

The Graphics Pipeline

We just covered the Vertex Shader

Output:

```
struct VertexShaderOutput
{
        float4 Position : POSITION0;
        float4 Color : COLOR0;
        float3 VertexPosition : TEXCOORD0;
};
```



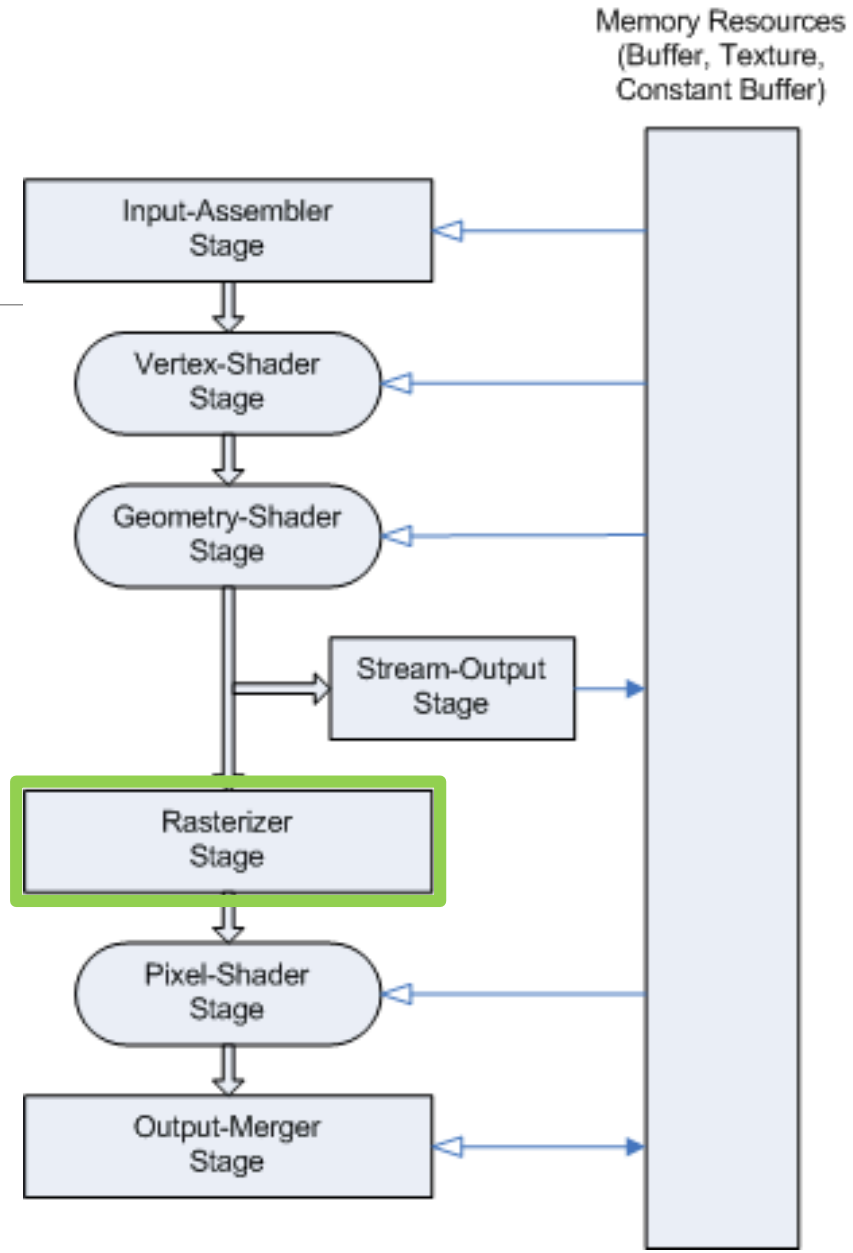
The Graphics Pipeline

We just covered the Vertex Shader

Output:

```
struct VertexShaderOutput
{
        float4 Position : POSITION0;
        float4 Color : COLOR0;
        float3 VertexPosition : TEXCOORD0;
};
```

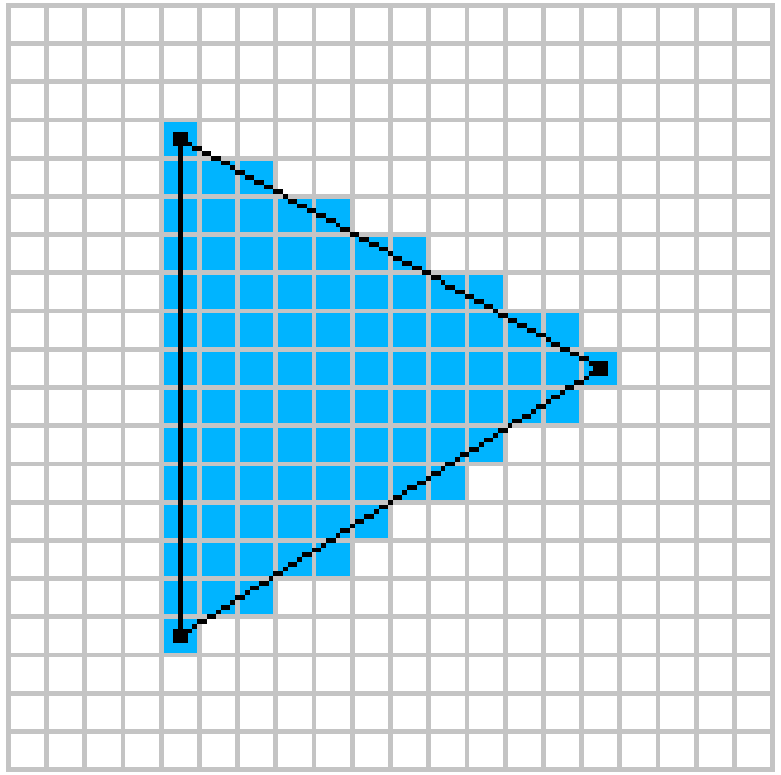
Input for Rasterizer



The Rasterizer

Converts triangles into pixels

- Or: Rasterizes triangles



The Rasterizer

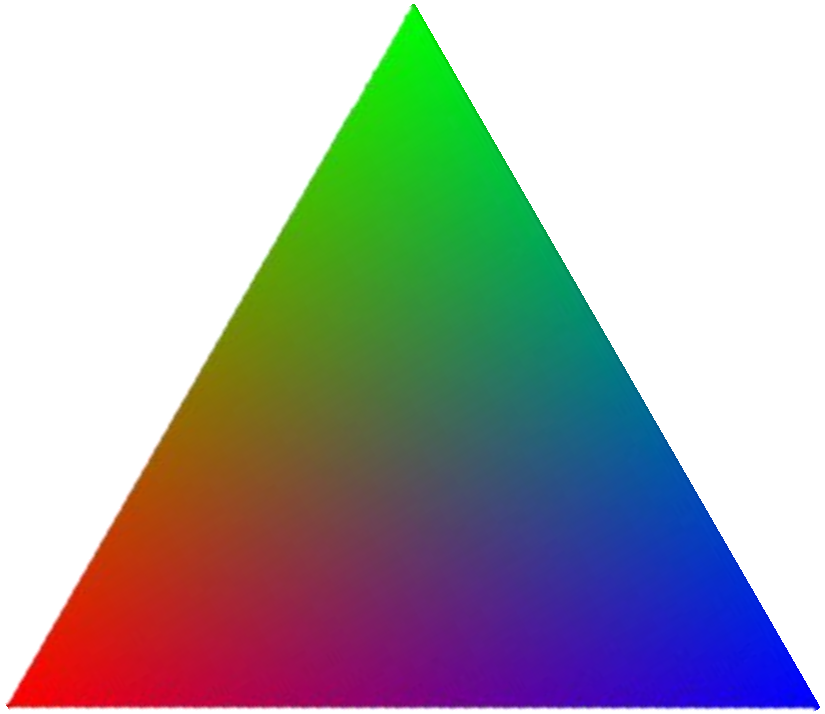
Converts triangles into pixels

- Or: Rasterizes triangles

Interpolates values between vertices

- For example the colors

```
struct VertexShaderOutput
{
        float4 Position : POSITION0;
        float4 Color : COLOR0;
        float3 VertexPosition : TEXCOORD0;
};
```



The Rasterizer – Linear Interpolation

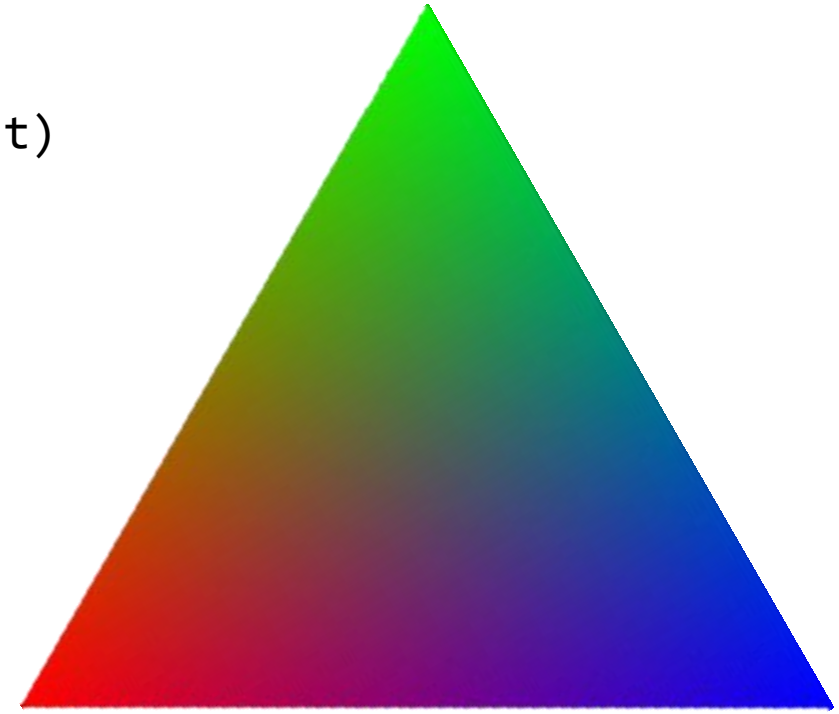
Values in the vertex structures are linearly interpolated

```
float LinearInterpolate(float a, float b, float t)
{
        return (1 - t)*a + t*b;
}
```

Weighted average

Can also be done with vectors and colors

- Interpolating vectors does not preserve length
  - Renormalize your normals after interpolation!

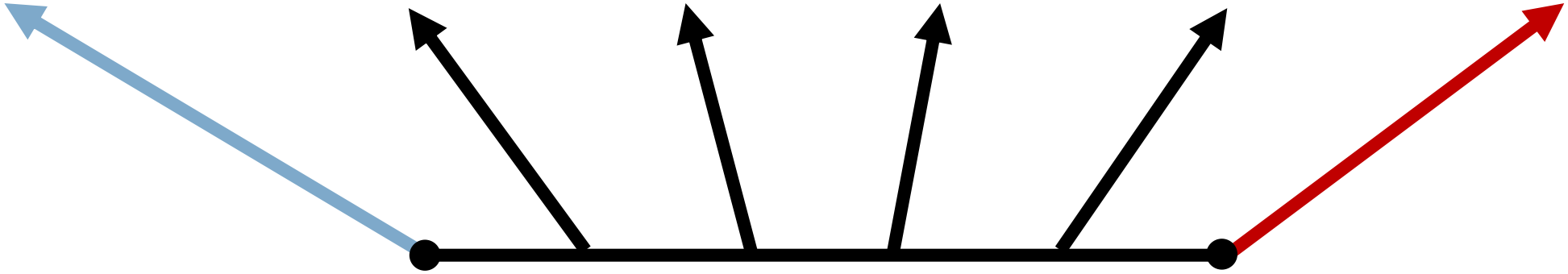


The Rasterizer – Interpolating vectors

Can also be done with vectors and colors

- Interpolating vectors does not preserve length
  - Renormalize your normals after interpolation!

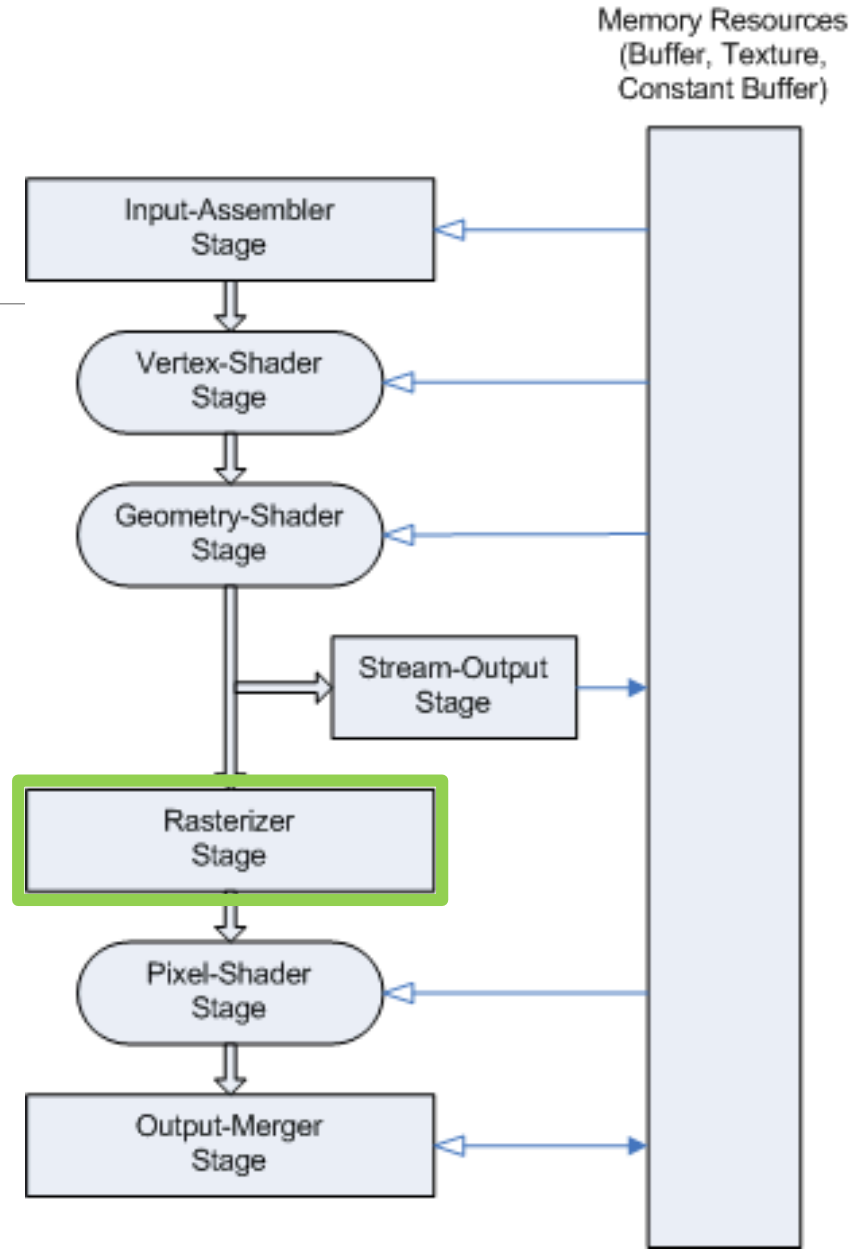
Notice what happens when we interpolate between the blue and red vectors



The Graphics Pipeline

After the rasterizer stage, we end up with pixels.

```
struct VertexShaderOutput
{
        float4 Position : POSITION0;
        float4 Color : COLOR0;
        float3 VertexPosition : TEXCOORD0;
};
```



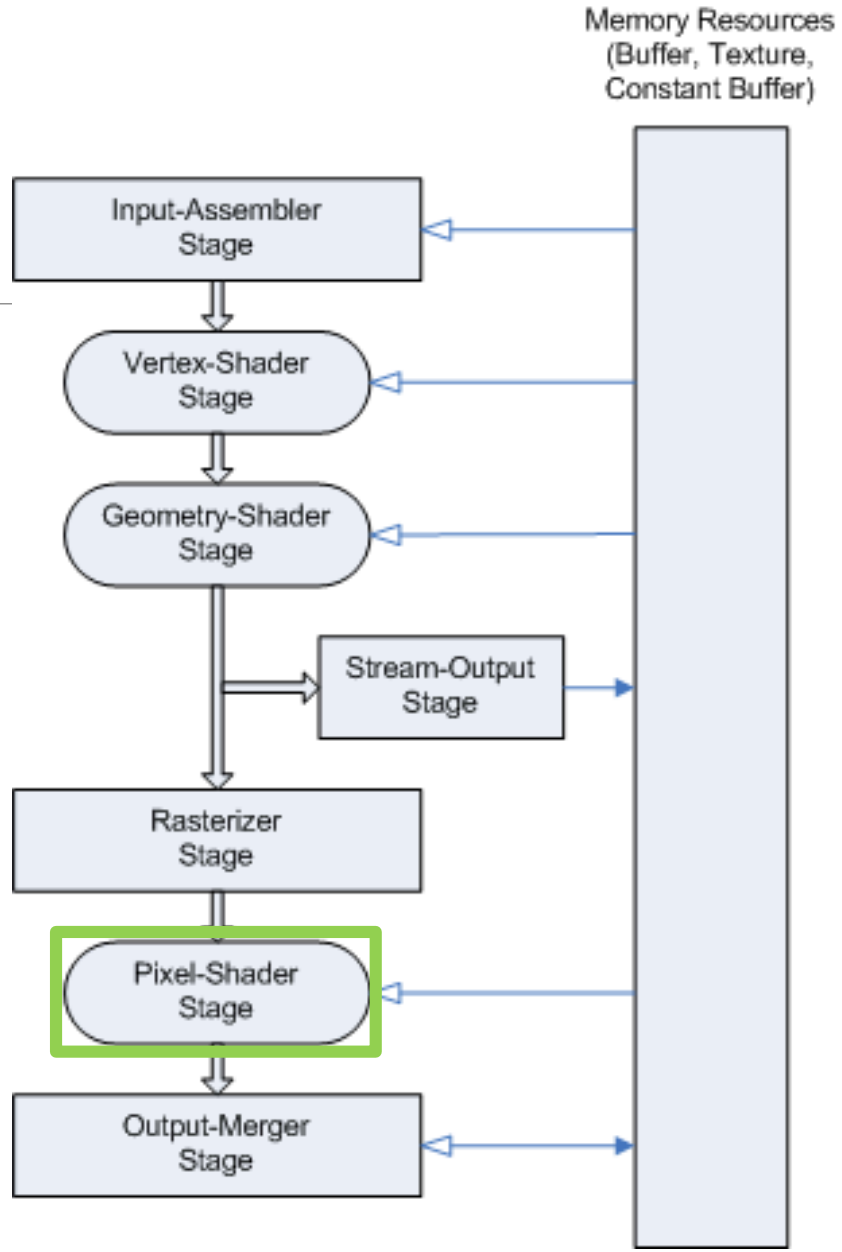
The Graphics Pipeline

After the rasterizer stage, we end up with pixels.

```
struct VertexShaderOutput
{
        float4 Position : POSITION0;
        float4 Color : COLOR0;
        float3 VertexPosition : TEXCOORD0;
};
```

For every pixel, we get a VertexShaderOutput struct

- With POSITION0 field removed



The Pixel Shader – In-Depth

What's a pixel shader?

- Just a function called for every pixel

Input: Interpolated Vertex Shader Output

- We reuse the struct VertexShaderOutput
- POSITION0 field "eaten" by the rasterizer

Output: One or more colors (or depth)

- We can put this in a struct
- Or just output a float4

The Pixel Shader – PixelShaderOutput

```
struct PixelShaderOutput
{
        float4 color : COLOR0;
};
```

COLOR0 semantic is mandatory

And that’s it!

Simple example – Vertex Color shader

Walkthrough for a simple shader

Vertex shader:

- Transform cube with World, View and Projection matrix
- Pass vertex colors to rasterizer

Pixel shader

- Output interpolated vertex colors

Simple example – Vertex Color shader

```
struct VertexShaderInput
{
	float4 Position : POSITION0;
	float4 Color : COLOR0;
};
struct VertexShaderOutput
{
	float4 Position : POSITION0;
	float4 Color : COLOR0;
};

float4x4 World;
float4x4 View;
float4x4 Projection;
```

Simple example – Vertex Color shader

```
VertexShaderOutput VertexShaderFunction(VertexShaderInput input)
{
	VertexShaderOutput output;

	float4 worldPosition = mul(input.Position, World);
	float4 viewPosition = mul(worldPosition, View);

	output.Position = mul(viewPosition, Projection);
	output.Color = input.Color;

	return output;
}
```

Simple example – Vertex Color shader

```
struct PixelShaderOutput
{
	float4 color : COLOR0;
};

PixelShaderOutput PixelShaderFunction(VertexShaderOutput input)
{
	PixelShaderOutput output;
	output.color = input.Color;
	return output;
}
```

Simple example – Vertex Color shader

```
float4 PixelShaderFunction(VertexShaderOutput input) : COLOR0
{
	return input.Color;
}
```

Simple example – Vertex Color shader

Tell the compiler which functions are the vertex shader and the pixel shader

Technique

- A shader file consists of one or more techniques (e.g. effect)

Pass

- Every technique consists of one or more passes
- Every pass consists of a vertex shader and a pixel shader

Simple example – Vertex Color shader

```
technique VertexColorsTechnique
{
    pass FirstPass
    {
        VertexShader = compile vs_2_0 VertexShaderFunction();
        PixelShader = compile ps_2_0 PixelShaderFunction();
    }
}
```

We choose a vertex shader and pixel shader profile

- Lower profile – 2_0 – less capabilities, supports older graphics cards
- Higher profile – 3_0 – more capabilities, only supports more recent graphics cards

Simple example – Vertex Color shader

That’s it!

- Not that much code!

Simple example – Vertex Color shader

```
float4x4 World;
float4x4 View;
float4x4 Projection;

struct VertexShaderInput
{
    float4 Position : POSITION0;
    float4 Color : COLOR0;
};

struct VertexShaderOutput
{
    float4 Position : POSITION0;
    float4 Color : COLOR0;
};
```

```
VertexShaderOutput VertexShaderFunction(VertexShaderInput input)
{
    VertexShaderOutput output;

    float4 worldPosition = mul(input.Position, World);
    float4 viewPosition = mul(worldPosition, View);
    output.Position = mul(viewPosition, Projection);
    output.Color = input.Color;

    return output;
}

float4 PixelShaderFunction(VertexShaderOutput input) : COLOR0
{
    return input.Color;
}

technique VertexColorsTechnique
{
    pass FirstPass
    {
        VertexShader = compile vs_2_0 VertexShaderFunction();
        PixelShader = compile ps_2_0 PixelShaderFunction();
    }
}
```

Using shaders in XNA

Vertex declarations

Loading shaders

Setting global variables

Rendering

Vertex Declarations

You've probably used this in P1

```
private VertexPositionColor[] vertices;
```

Later in P1, you define your own vertex types (see Section 7.2 of P1)

```
private VertexPositionColorNormal[] vertices;
```

Vertex Declarations

In the vertex declaration, we see this:

```
public static VertexElement[] VertexElements =
{
    new VertexElement(0, VertexElementFormat.Vector3, VertexElementUsage.Position, 0),
    new VertexElement(sizeof (float)*3, VertexElementFormat.Color, VertexElementUsage.Color, 0),
    new VertexElement(sizeof (float)*3 + 4, VertexElementFormat.Vector3, VertexElementUsage.Normal, 0),
};
```

Relates to the following shader code:

```
struct VertexShaderInput
{
    float3 Position : POSITION0;
    float4 Color : COLOR0;
    float3 Normal : NORMAL0;
};
```

Vertex Declarations

In the vertex declaration, we see this:

```
public static VertexElement[] VertexElements =
{
    new VertexElement(0, VertexElementFormat.Vector3, VertexElementUsage.Position, 0),
    new VertexElement(sizeof (float)*3, VertexElementFormat.Color, VertexElementUsage.Color, 0),
    new VertexElement(sizeof (float)*3 + 4, VertexElementFormat.Vector3, VertexElementUsage.Normal, 0),
};
```

Relates to the following shader code:

```
struct VertexShaderInput
{
    float3 Position : POSITION0;
    float4 Color : COLOR0;
    float3 Normal : NORMAL0;
};
```

Vertex Declarations

In the vertex declaration, we see this:

```
public static VertexElement[] VertexElements =
{
    new VertexElement(0, VertexElementFormat.Vector3, VertexElementUsage.Position, 0),
    new VertexElement(sizeof (float)*3, VertexElementFormat.Color, VertexElementUsage.Color, 0),
    new VertexElement(sizeof (float)*3 + 4, VertexElementFormat.Vector3, VertexElementUsage.Normal, 0),
};
```

Relates to the following shader code:

```
struct VertexShaderInput
{
    float3 Position : POSITION0;
    float4 Color : COLOR0;
    float3 Normal : NORMAL0;
};
```

Vertex Declarations

In the vertex declaration, we see this:

```
public static VertexElement[] VertexElements =
{
    new VertexElement(0, VertexElementFormat.Vector3, VertexElementUsage.Position, 0),
    new VertexElement(sizeof (float)*3, VertexElementFormat.Color, VertexElementUsage.Color, 0),
    new VertexElement(sizeof (float)*3 + 4, VertexElementFormat.Vector3, VertexElementUsage.Normal, 0),
};
```

Relates to the following shader code:

```
struct VertexShaderInput
{
    float3 Position : POSITION0;
    float4 Color : COLOR0;
    float3 Normal : NORMAL0;
};
```

Vertex Declarations

In the vertex declaration, we see this:

```
public static VertexElement[] VertexElements =
{
    new VertexElement(0, VertexElementFormat.Vector3, VertexElementUsage.Position, 0),
    new VertexElement(sizeof (float)*3, VertexElementFormat.Color, VertexElementUsage.Color, 0),
    new VertexElement(sizeof (float)*3 + 4, VertexElementFormat.Vector3, VertexElementUsage.Normal, 0),
};
```

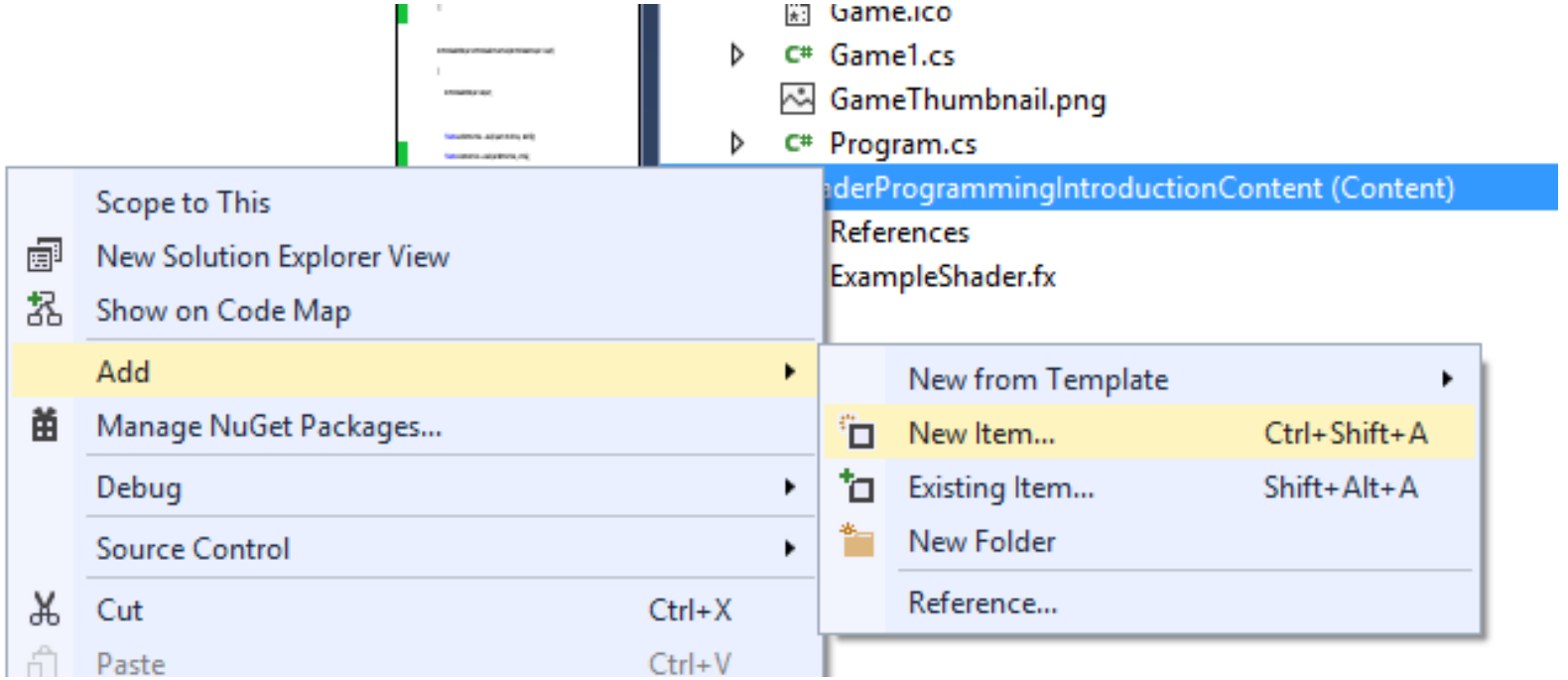
Relates to the following shader code:

```
struct VertexShaderInput
{
    float3 Position : POSITION0;
    float4 Color : COLOR0;
    float3 Normal : NORMAL0;
};
```

Creating shader file

Right click on Content Project

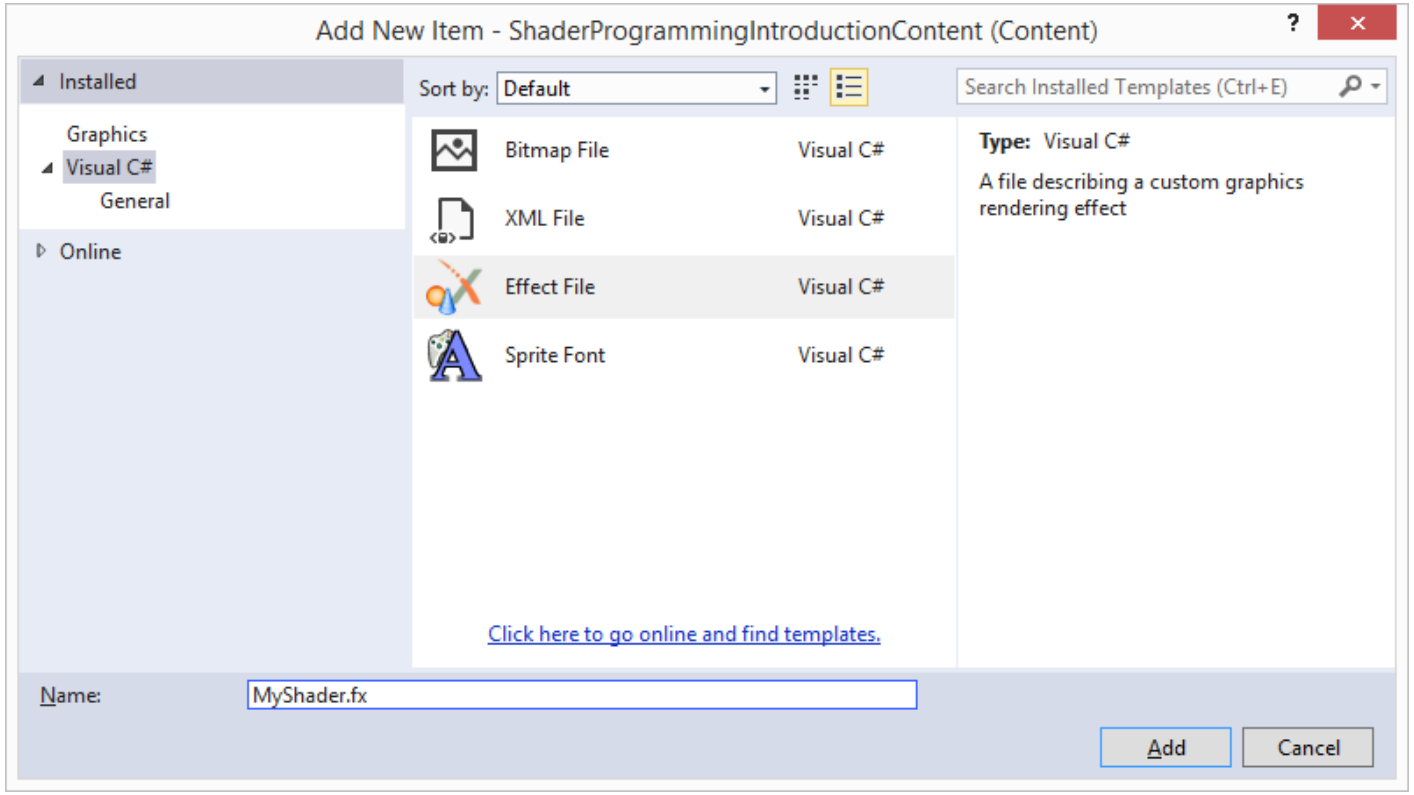
- Add -> New Item



Creating shader file

Right click on Content Project

- Add -> New Item
- Visual C# -> Effect File



Setting up the shader

Load the shader like any other resource

```
Effect effect = Content.Load<Effect>("ExampleShader");
```

Set the active technique

```
effect.CurrentTechnique = effect.Techniques["VertexColorsTechnique"];
```

Set shader global variables

```
effect.Parameters["World"].SetValue(Matrix.Identity);
effect.Parameters["View"].SetValue(Matrix.CreateLookAt(...));
effect.Parameters["Projection"].SetValue(Matrix.CreatePerspectiveFieldOfView(...));
```

Rendering using the shader

```
foreach (EffectPass pass in effect.CurrentTechnique.Passes)
{
    pass.Apply();
    // Rendering code here
}
```

Rendering using the shader

```
foreach (EffectPass pass in effect.CurrentTechnique.Passes)
{
    pass.Apply();

    // Rendering code here
    GraphicsDevice.DrawUserIndexedPrimitives(PrimitiveType.TriangleList,
                vertices, 0, vertices.Length, indices, 0, indices.Length/3);
}
```

In the draw function

```
effect.Parameters["World"].SetValue(Matrix.Identity);
effect.Parameters["View"].SetValue(Matrix.CreateLookAt(...));
effect.Parameters["Projection"].SetValue(Matrix.CreatePerspectiveFieldOfView(...));

foreach (EffectPass pass in effect.CurrentTechnique.Passes)
{
    pass.Apply();

    // Rendering code here
    GraphicsDevice.DrawUserIndexedPrimitives(PrimitiveType.TriangleList,
                vertices, 0, vertices.Length, indices, 0, indices.Length/3);
}
```

That’s it!

HLSL tips and tricks

Some useful things to know about HLSL

Just a quick glance, google it for details

Useful functions in HLSL

Vector length
- float len = length(myVector);

Normalize vector
- myNormal = normalize(myNormal);

Cross product
- float3 crossProduct = cross(forward, up);

Dot product
- float intensity = dot(normal, lightDirection);

Clamp value between [3,5]
- int x = clamp(y, 3, 5);

Clamp value between [0,1]
- int x = saturate(y);
- In some cases, this function is free!

Useful functions in HLSL

Other functions:

- abs, min, max, sin, cos, tan, pow, sqrt, exp, log, floor, lerp, smoothstep, reflect, refract, and many more..

Most common math functions are available

Just Google for ‘HLSL *function name*’ or ‘CG *function name*’

Sampling textures in HLSL

Define a texture global variable

- `Texture2D MyTexture;`
- Set its value in C#.

Then we define a texture sampler

```
sampler2D MySampler = sampler_state
{
    texture = <MyTexture>;
    magfilter = LINEAR;
    minfilter = LINEAR;
    mipfilter = LINEAR;
    AddressU = mirror;
    AddressV = mirror;
};
```

Sampling textures in HLSL

Now we need to ‘read’ the texture

- This is called sampling
- Lots of ways to do it

Sample function in HLSL

```
float4 color = tex2D(MySampler, float2(0.2, 0.3));
```

Note that we pass the sampler, not the texture.

Texture coordinates

- [0,1] range
- Usually you’d pass a variable (like interpolated texture coordinates)

Returns a float4

Sampling textures in HLSL

Lots of sampling functions available

- But you won't need most of these

And these aren't the only ones!

tex2D
tex2Dbias
tex2Dgrad
tex2Dlod
tex2Dproj

```
float4 tex2D(sampler2D samp, float2 s)
float4 tex2D(sampler2D samp, float2 s, int texelOff)
float4 tex2D(sampler2D samp, float3 s)
float4 tex2D(sampler2D samp, float3 s, int texelOff)

float4 tex2D(sampler2D samp, float2 s, float2 dx, float2 dy)
float4 tex2D(sampler2D samp, float2 s, float2 dx, float2 dy, int texelOff)
float4 tex2D(sampler2D samp, float3 s, float2 dx, float2 dy)
float4 tex2D(sampler2D samp, float3 s, float2 dx, float2 dy, int texelOff)

int4 tex2D(isampler2D samp, float2 s)
int4 tex2D(isampler2D samp, float2 s, int texelOff)

int4 tex2D(isampler2D samp, float2 s, float2 dx, float2 dy)
int4 tex2D(isampler2D samp, float2 s, float2 dx, float2 dy, int texelOff)

unsigned int4 tex2D(usampler2D samp, float2 s)
unsigned int4 tex2D(usampler2D samp, float2 s, int texelOff)

unsigned int4 tex2D(usampler2D samp, float2 s, float2 dx, float2 dy)
unsigned int4 tex2D(usampler2D samp, float2 s, float2 dx, float2 dy,
                    int texelOff)
```

Sampling textures in HLSL

1D Textures

2D Textures

3D Textures

Cubemaps

All of these have their own sampling functions

- tex1D, tex2D, tex3D, texCUBE

Swizzling

Take a look at the following code

```
float4 a;
float4 b;
a = b.wyzx;
```

You can also reuse components

```
float4 a;
float4 b;
a = b.wyyx;
```

Or reuse one components four times

```
float4 a;
float4 b;
a = b.xxxx;
```

Swizzling

Generate a 2D vector from a 4D vector

```
float2 a;
float4 b;
a = b.xz;
```

Or a 4D vector from a 2D vector

```
float4 a;
float2 b;
a = b.xxyy;
```

Instead of .xyzw, we can also use .rgba

```
float4 a;
float4 b;
a = b.rgba;
```

Common errors

You've forgotten a semicolon
- ERROR: Unexpected token 'something'
- Take a look at the line before the error

You misspelled variable names in C#
- Shader parameters and technique names are all CASE Sensitive!

Mesh doesn't show up
- It's probably the cullmode
- Or your World Matrix
- Or your View Matrix
- Or your Projection Matrix
- Or some other render state
- Or… I don't know, it's your code!

Common errors

Debugging shaders is hard

- Certainly harder than the CPU
- Can’t step through code, add breakpoints, etc.
- Can’t Console.WriteLine values

All is not lost though!

- Just don’t write any errors.

Common errors

Start with very simple shaders

- Don’t write everything and test it afterwards.
- Use a very simple pixel shader to write your vertex shader

```
float4 PixelShaderFunction(VertexShaderOutput input) : COLOR0
{
    return float4(1,0,0,0);
}
```

- Make sure you test all the code you write. Lots of sanity checks!
- If you’ve finished with your vertex shader, write simple pixel shaders to test all the input
  - Returning colors is kind of like Console.WriteLine()

```
float4 PixelShaderFunction(VertexShaderOutput input) : COLOR0
{
    return float4(input.Normal,0);
}
```

Common errors

Document every line of code

- It's not that many lines of code anyway!

Use proper variable naming

- Indicate global variables
- If a vector is a normalized vector, call it myVectorN
- If a vector is in world space, call it positionW or positionWorld or worldPosition, just indicate it.
- Same thing with view space
- Same thing with projected coordinates
- If a vector points to the camera, call it toCamera, if it's pointing to the camera and is normalized, call it toCameraN

If there's a built-in function for it, use it!

Test your vertex declaration

- Often the semantics in the C# vertex declaration and the HLSL struct don't match

Help, my shader was working and now it's not working anymore!

Changed your shader code?

- Well, there's your problem

Changed C# code?

- Some render state probably changed.

Render states changed by SpriteBatch

```
GraphicsDevice.BlendState = BlendState.AlphaBlend;
GraphicsDevice.DepthStencilState = DepthStencilState.None;
GraphicsDevice.RasterizerState = RasterizerState.CullCounterClockwise;
GraphicsDevice.SamplerStates[0] = SamplerState.LinearClamp;
```

You'll need to change it back, even though you didn't set it explicitly in the first place

```
GraphicsDevice.BlendState = BlendState.Opaque;
GraphicsDevice.DepthStencilState = DepthStencilState.Default;
```

Help, my shader was working and now it’s not working anymore!

Some other shader may also have changed the render states

- Check this first

Some global variables might be incorrect

- Example:
- Render object 1 with a 100x scaling matrix
- Forget to set the world matrix back to 1x scaling
- Render object 2, which is so big, it gets clipped away
- You can check these values easily in C#

The End

ANY QUESTIONS?


Buy The Flock!

The End

ANY QUESTIONS?