

# Introduction to Direct3D and High Level Shading Language

Charles Calkins

# Contents

- History
- Rendering pipeline
- Direct3D basics
- Vertex and pixel shaders
- References

# What is Direct3D?

- The 3D graphics component of DirectX
  - DirectSound, DirectInput, etc.
- Exposes capabilities of 3D graphics hardware
  - Geometry, texturing, lighting, etc.

# History

- Project began in November, 1994
  - To encourage game developers to move to Windows 95 from DOS
- “Game SDK”
  - Debut at the Computer Game Developers Conference (CGDC), April 1995



# History

- Timeline
  - DirectX 1.0, September 1995
  - DirectX 2.0, 3.0 in 1996
    - Difficult to use, but Microsoft continued development instead of choosing competitor OpenGL
  - No 4.0, but 5.0 in 1997
    - Fixed some of the awkwardness of execute buffers by adding the DrawPrimitive() API (simplified geometry creation)

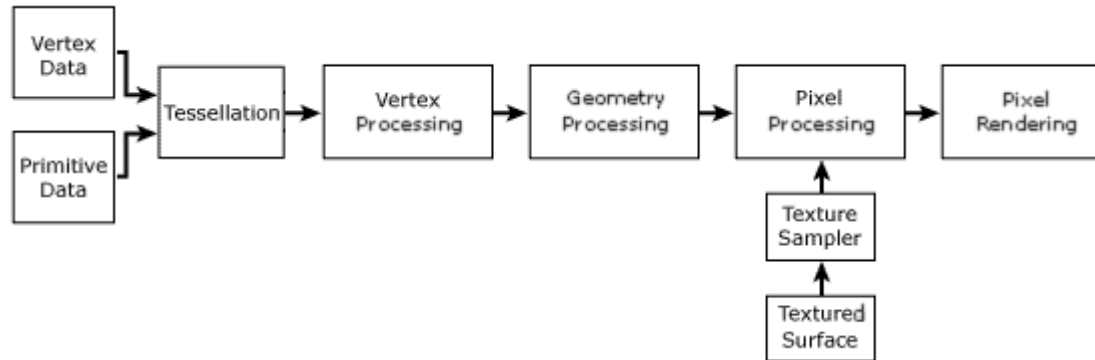
# History

- Timeline
  - 6.0 in 1998
    - Added multitexturing, vendor proprietary features, and algorithms via marketing arrangements
    - Faster development than the OpenGL Architectural Review Board, so supported by vendors
  - 7.0 in 1999
    - First version supporting hardware transform and lighting, also first to allow vertex buffers in hardware

# History

- Timeline
  - 8.0 in 2000, 8.1 in 2001
    - Introduction of vertex and pixel shaders
      - assembly language-style
    - Deviation from fixed-function pipeline of earlier Direct3D and OpenGL
  - 9.0 in 2002, 9.0a and 9.0b 2003, 9.0c 2005
    - High Level Shading Language (HLSL)
  - 10.0 in 2007 – Vista
    - No more fixed-function pipeline – only shaders

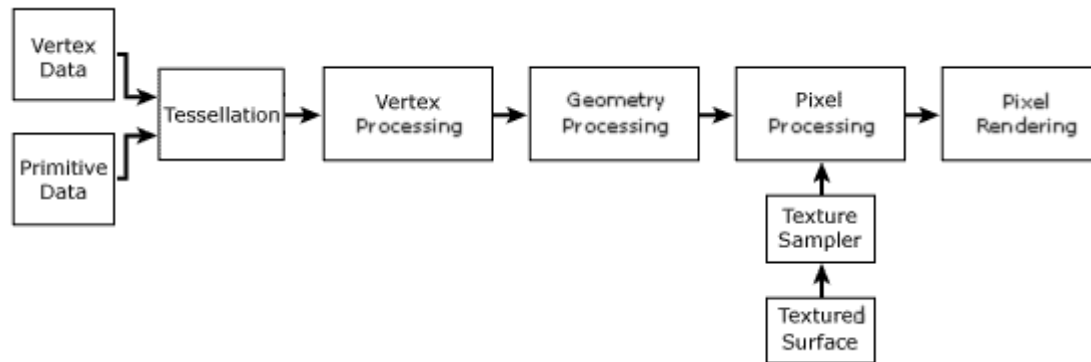
# Rendering Pipeline



- Vertex data
  - Untransformed model vertices in vertex buffers
- Primitive data
  - Points, lines, triangles, and polygons in the vertex data are referenced via index buffers

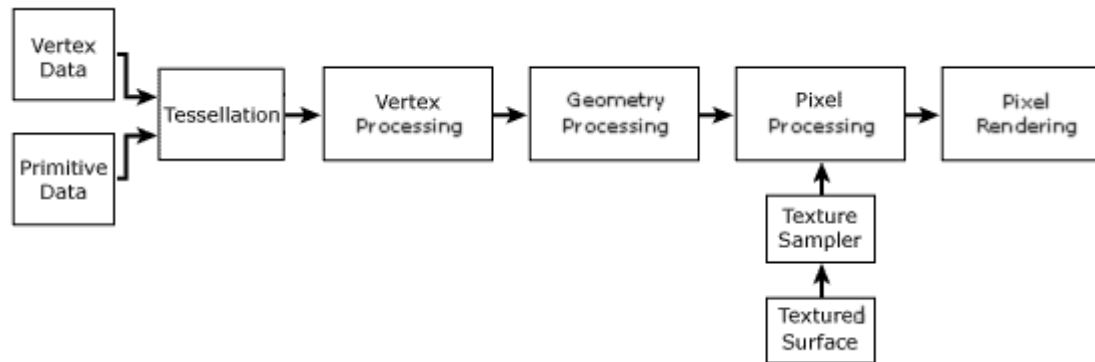


# Rendering Pipeline



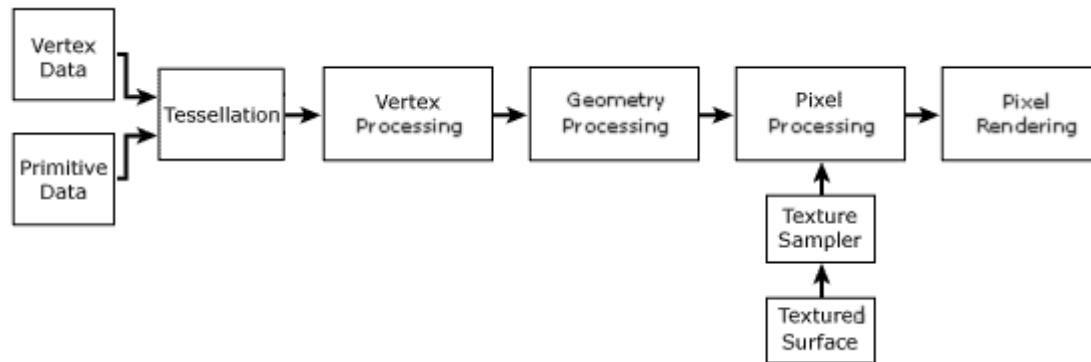
- Tessellation
  - Convert higher-order primitives to vertices in vertex buffers
- Vertex processing
  - Apply transformations to vertices
  - Vertex shader

# Rendering Pipeline



- Geometry processing
  - Clipping, backface culling, rasterization
- Textured surface
  - Textures and texture coordinates supplied
- Texture sampler
  - Level of detail filtering

# Rendering Pipeline



- Pixel processing
  - Use vertex and texture info to determine pixel color
  - Pixel shader
- Pixel rendering (to output device)
  - Alpha, depth testing, blending, fog, etc.

# Basics – Device

- Device
  - Represents the drawing surface
  - Contains drawing state
    - e.g., geometry, transformations
  - Provides events for when the display must be reset
    - e.g., device is lost when a user alt-tabs to another 3D application, so geometry, textures, etc., of first app are flushed out of the hardware and must be reloaded

# Basics – Spaces

- Model space
  - Coordinates of vertices in a geometrical object are relative to an object-specific origin
- World space
  - Objects in the 3D scene are positioned relative to a world-origin
- View space
  - How the world is viewed through a camera
- Screen space
  - The 2D screen

# Basics – World Matrix

- Transform from Model to World space
- Set for each object to orient it in the scene
- Operations
  - Translate
  - Rotate
  - Scale

# Basics – View Matrix

- Transform from World to View space
- Only needs to change when the camera moves
- Can be set using a “look at” syntax
  - Eye position
  - Target
  - “Up” vector

# Basics – Projection Matrix

- Transform from View to Screen space
- May only need to be set once
- Parameters
  - Field of view
  - Near and far clipping planes



# Basics – Mesh

- Contains groups (“subsets”) of geometry as vertex and index buffers
- Vertices can have associated normals, texture coordinates, etc.
- Can be loaded from a file (.x), as well as produced from primitive creation functions (e.g., Sphere(), Torus(), TextFromFont(), Teapot() )

# Basics – Rendering a Mesh

**Initialize() // call once**

**Device device = new Device()**

**Mesh m = Mesh.FromFile(“mymesh.x”, device)**

**Render() // call from window’s OnPaint() handler**

**device.Clear()**

**device.BeginScene()**

**device.SetTransform() // do for each of the 3 matrices**

**for each mesh subset s**

**m.DrawSubset(s)**

**device.EndScene()**

**device.Present()**

# Shaders

- Executed on the graphics hardware
- Are written in assembly-style or high level shading language-style, and compiled
  - compiler is fxc.exe
  - compiled output can be object file, or even C++ header file
- HLSL is common, but output can differ based on shader version (e.g., for loop unrolling in v1.1 shaders vs. v2.0)

# Vertex Shader

- Executes for each vertex that is rendered
- Input is based on vertex format
  - e.g., position, normal, texture coordinates
  - tied to format using “semantics”
    - e.g., DeclarationUsage.TextureCoordinate in C# corresponds to TEXCOORD semantic in shader
- Output is user defined
  - return a value with a COLOR semantic if no pixel shader, otherwise return a structure

# Pixel Shader

- Executes for each pixel that is rendered
- Input to pixel shader is output of vertex shader
- Output is a variable with a COLOR semantic

# Simple Vertex Shader – HLSL

```
struct app2vertex {  
    float4 Position : POSITION;  
};  
  
struct vertex2pixel {  
    float4 Position : POSITION;  
};  
  
void main(in app2vertex IN, out vertex2pixel OUT,  
    uniform float4x4 ModelViewMatrix) {  
    OUT.Position = mul(IN.Position, ModelViewMatrix);  
}
```

# Simple Vertex Shader – Asm

Compiling the previous shader via:

```
fxc /Fcsimple.a /Tvs_1_1 simple.vs
```

produces:

```
vs_1_1  
dcl_position v0  
dp4 oPos.x, v0, c0  
dp4 oPos.y, v0, c1  
dp4 oPos.z, v0, c2  
dp4 oPos.w, v0, c3
```

which is 4 instructions (and 2 directives)

# Vertex Shader – Loop HLSL

```
struct app2vertex {  
    float4 Position : POSITION;  
};  
  
struct vertex2pixel {  
    float4 Position : POSITION;  
};  
  
void main(in app2vertex IN, out vertex2pixel OUT,  
    uniform float4x4 ModelViewMatrix) {  
    float4 k = IN.Position;  
    for (int i=0; i<5; i++) // dumb, but for demonstration  
        k = mul(k,ModelViewMatrix);  
    OUT.Position = mul(k, ModelViewMatrix);  
}
```



# Vertex Shader – Loop Asm

Compiling the previous shader via:

```
fxc /Fcsimpleloop.a /Tvs_1_1 simpleloop.vs // 1.1 target
```

```
vs_1_1
dcl_position v0
dp4 r0.x, v0, c0
dp4 r0.y, v0, c1
dp4 r0.z, v0, c2
dp4 r0.w, v0, c3
dp4 r1.x, r0, c0
dp4 r1.y, r0, c1
dp4 r1.z, r0, c2
dp4 r1.w, r0, c3
dp4 r0.x, r1, c0
dp4 r0.y, r1, c1
dp4 r0.z, r1, c2
dp4 r0.w, r1, c3
```

```
dp4 r1.x, r0, c0
dp4 r1.y, r0, c1
dp4 r1.z, r0, c2
dp4 r1.w, r0, c3
dp4 r0.x, r1, c0
dp4 r0.y, r1, c1
dp4 r0.z, r1, c2
dp4 r0.w, r1, c3
dp4 oPos.x, r0, c0
dp4 oPos.y, r0, c1
dp4 oPos.z, r0, c2
dp4 oPos.w, r0, c3
```

24 instructions used

# Vertex Shader – Loop Asm

Compiling the previous shader via:

```
fxc /Fcsimpleloop.a /Tvs_2_0 simpleloop.vs // 2.0 target
```

```
vs_2_0
defi i0, 5, 0, 0, 0
dcl_position v0
mov r0, v0
rep i0
    dp4 r1.x, r0, c3
    dp4 r1.y, r0, c0
    dp4 r1.z, r0, c1
    dp4 r0.z, r0, c2
    mov r0.xy, r1.yzzw
    mov r0.w, r1.x
endrep
```

```
dp4 oPos.w, r0, c3
dp4 oPos.x, r0, c0
dp4 oPos.y, r0, c1
dp4 oPos.z, r0, c2
```

16 instructions used

Lower target version means  
more hardware support, but up  
against max shader lengths!

# Effects

- Shaders can be combined into effect files
- An association between vertex and pixel shaders is a “technique”
- Global variables (such as the `ModelViewMatrix`, used before) can be set from C# by name
- Effect files can be loaded and compiled in one step

# Mesh Rendering with Effects

**Initialize()**

**Device device = new Device()**

**Mesh m = new Mesh("mymesh.x", device)**

**Effect e = Effect.FromFile(device, "myeffects.fx")**

# Mesh Rendering with Effects

```
Render() // call from window's OnPaint() handler
    device.Clear()
    device.BeginScene()
    device.SetTransform() // do for each of the 3 matrices
    e.SetTechnique = "techniqueName"
    e.SetValue("valueName", valueParameter); // for each value
    for each mesh subset s
        e.Begin()
        for each effect pass p
            e.BeginPass(p); m.DrawSubset(s); e.EndPass(p)
        e.End()
    device.EndScene()
    device.Present()
```

# Example – Diffuse

```
float4x4 worldViewProj : WORLDVIEWPROJECTION;  
float time;
```

```
struct Diffuse_VSOUT {  
    float4 position : POSITION;  
    float4 diffuse : COLOR0;  
};
```

```
Diffuse_VSOUT Diffuse_VS(float4 position : POSITION)  
{  
    Diffuse_VSOUT vsout = (Diffuse_VSOUT)0;  
    vsout.position = mul(position, worldViewProj); // to screen space  
    vsout.diffuse.r = saturate(time); // clamp to 0...1  
    vsout.diffuse.b = sin(time);  
    vsout.diffuse.ga = cos(time); // swizzle  
    return vsout;  
}
```

# Example – Diffuse

**technique Diffuse**

```
{  
    pass P0  
    {  
        CullMode = None;  
        VertexShader = compile vs_1_1 Diffuse_VS();  
        PixelShader = NULL; // no pixel shader needed  
    }  
}
```

# Example – Diffuse/Depth

```
float4 eyePos;  
float4x4 world;  
float meshRadius;
```

```
struct Depth_VSOUT {  
    float4 position : POSITION;  
    float3 viewDist : TEXCOORD3;  
};
```

```
Depth_VSOUT Depth_VS(float4 position : POSITION)  
{  
    Depth_VSOUT vsout = (Depth_VSOUT)0;  
    vsout.position = mul(position, worldViewProj);  
    float distance = distance(eyePos, mul(position, world));  
    vsout.viewDist.x = ((2.5f*meshRadius)/distance)/2.0f;  
    return vsout;  
}
```



# Example – Diffuse/Depth

```
float4 Depth_PS(Depth_VSOUT psin) : COLOR
{
    float distMult = saturate((psin.viewDist.x-0.4f)/(0.7f-0.4f));
    float4 color;
    color.rgb=distMult;
    color.a = 1.0f;
    return color;
}
```

```
technique Depth
{
    pass P0
    {
        CullMode = None;
        VertexShader = compile vs_1_1 Depth_VS();
        PixelShader = compile ps_2_0 Depth_PS();
    }
}
```

# Example – Phong

```
float4x4 worldViewProj : WORLDVIEWPROJECTION;  
float4 lightDir;    // direction of the light  
float4 eyePos;      // position of the viewer  
float4 diffuseColor; // diffuse color to set
```

```
struct Phong_VSOUT {  
    float4 position : POSITION;  
    float3 lightDir : TEXCOORD0;  
    float3 normal : TEXCOORD1;  
    float3 viewDir : TEXCOORD2;  
};
```

# Example – Phong

```
Phong_VSOUT Phong_VS(float4 position : POSITION,  
    float3 normal : NORMAL)  
{  
    Phong_VSOUT vsout = (Phong_VSOUT)0;  
  
    vsout.position = mul(position, worldViewProj);  
  
    vsout.lightDir = normalize(lightDir); // L  
    vsout.viewDir = normalize(eyePos - normalize(mul(position, world))); // V  
    vsout.normal = normalize(mul(normal, world)); // N  
  
    return vsout;  
}
```

# Example – Phong

```
float4 Phong_PS(Phong_VSOUT psin) : COLOR
{
    float4 ambient = {0.2f, 0.2f, 0.2f, 1.0f};
    // diffuse – intensity is relative to the the surfac normal and view vectors
    float diffuse = saturate(dot(psin.normal, psin.lightDir)); // diffuse multiplier
    float shadow = saturate(4*diffuse); // self-shadowing test

    //  $R = 2 * (N.L) * N - L$  – reflectance to determine specular highlight
    // intensity is relative to the reflected light and view vectors
    float3 reflect = normalize(2 * diffuse * psin.normal - psin.lightDir);
    float4 specular = pow(saturate(dot(reflect, psin.viewDir)), 8); //  $R.V^n$ 

    //  $I = \text{ambient} + \text{shadow} * (\text{diffuse} * N.L + (R.V)^n)$ 
    return ambient + shadow * (diffuseColor * diffuse + specular);
}
```

# Example – Phong

**technique Phong**

```
{  
    pass P0  
    {  
        CullMode = None;  
        VertexShader = compile vs_1_1 Phong_VS();  
        PixelShader = compile ps_2_0 Phong_PS();  
    }  
}
```

**Note that the pixel shader must be v2.0 as the reflect statement can't be expressed as is in earlier shaders**

# Example – PhongDistort

```
Phong_VSOUT PhongDistort_VS(float4 position : POSITION,  
    float3 normal : NORMAL)  
{  
    Phong_VSOUT vsout = (Phong_VSOUT)0;  
  
    float4 newposition = {  
        position.x+time*cos(2.0f*position.z),  
        position.y+time*sin(2.0f*position.z),  
        position.z, position.w  
    };  
    vsout.position = mul(newposition, worldViewProj);  
    vsout.lightDir = normalize(lightDir); // L  
    vsout.viewDir = normalize(eyePos - normalize(mul(position, world))); // V  
    vsout.normal = normalize(mul(normal, world)); // N  
    return vsout;  
}
```

technique definition similar to previous

# Example – Texture

**Texture texture1;**

**// linear – bilinear interpolation to create mipmaps**

**// mirror – flip texture at each integer boundary 0...1, 1...2 (flipped), ...**

**sampler texture1Sampler = sampler\_state { texture = <texture1> ;**

**magfilter = LINEAR; minfilter = LINEAR; mipfilter = LINEAR;**

**AddressU = mirror; AddressV = mirror; };**

**struct Texture\_VSOUT**

**{**

**float4 position : POSITION;**

**float2 texCoords : TEXCOORD0;**

**};**

# Example – Texture

```
Texture_VSOUT Texture_VS(float4 position : POSITION
    float2 texCoords : TEXCOORD0)
{
    Texture_VSOUT vsout = (Texture_VSOUT)0;

    vsout.position = mul(position, worldViewProj);
    vsout.texCoords = texCoords;

    return vsout;
}

float4 Texture_PS(Texture_VSOUT psin) : COLOR
{
    float4 color = tex2D(texture1Sampler, psin.texCoords);
    return color;
}
```



# Example – Texture

**technique Texture**

```
{  
    pass P0  
    {  
        CullMode = None;  
        VertexShader = compile vs_1_1 Texture_VS();  
        PixelShader = compile ps_1_1 Texture_PS();  
    }  
}
```

# Example – Mandelbrot

```
struct Mandelbrot_VSOUT
{
    float4 position : POSITION;
    float2 texCoords : TEXCOORD0;
};
```

```
Mandelbrot_VSOUT Mandelbrot_VS(float4 position : POSITION,
    float2 texCoords : TEXCOORD0)
{
    Mandelbrot_VSOUT vsout = (Mandelbrot_VSOUT)0;
    vsout.position = mul(position, worldViewProj);
    vsout.texCoords = texCoords;
    return vsout;
}
```

# Example – Mandelbrot

```
const int maxiter = 15;
int DoMandelbrot(float2 center, float scale, float xoffset, float2 texCoords) {
    float2 z, c;
    c.y = (texCoords.x - 0.5) * scale - center.x;
    c.x = (texCoords.y - 0.5) * scale - center.y + xoffset;

    int i = 0;
    float mag = 0.0f;
    z = c;
    while ( (i<maxiter) && (mag<4.0f) ) {
        float x = (z.x * z.x - z.y * z.y) + c.x;    // expansion of  $z=z^2+c$ 
        float y = (z.y * z.x + z.x * z.y) + c.y;
        mag = x*x + y*y;
        z.x = x;
        z.y = y;
        i++;
    }
    return i;
}
```

# Example – Mandelbrot

```
float4 Color(int i) {  
    float4 color = {0.0f, 0.0f, 0.0f, 1.0f};  
  
    if (i == maxiter)  
        color.rgb = 1.0f;  
    else if (i<7)  
        color.r=i/7.0f;  
    else  
        color.rg = saturate((i-7)/25.0f+0.7f); // clamp to 0...1  
    return color;  
}
```

# Example – Mandelbrot

```
float4 Mandelbrot_PS(Mandelbrot_VSOUT psin) : COLOR
{
    const float2 center1 = {0.0f, 0.0f}, center2 = {-3.9f, 4.15f },
        center3 = {3.9f, 4.15f };
    const float scale1 = 6.0f, scale2 = 12.0f, scale3 = 12.0f;
    return saturate(
        Color(DoMandelbrot(center1, scale1, time, psin.texCoords)) +
        Color(DoMandelbrot(center2, scale2, 0.0f, psin.texCoords)) +
        Color(DoMandelbrot(center3, scale3, 0.0f, psin.texCoords)));
}

technique Mandelbrot
{
    pass P0
    {
        CullMode = None;
        VertexShader = compile vs_1_1 Mandelbrot_VS();
        PixelShader  = compile ps_3_0 Mandelbrot_PS();
    }
}
```

# References

- History
  - [http://craig.theeislers.com/2006/02/directx\\_the\\_n\\_and\\_now\\_part\\_1.php](http://craig.theeislers.com/2006/02/directx_the_n_and_now_part_1.php)
  - <http://en.wikipedia.org/wiki/DirectX>
- Rendering pipeline
  - <http://msdn2.microsoft.com/en-us/library/bb219679.aspx>

# References

- HLSL tutorials, examples
  - [http://msdn2.microsoft.com/en-us/library/ms810476\(d=printer\).aspx](http://msdn2.microsoft.com/en-us/library/ms810476(d=printer).aspx)
  - [http://nuclear.demoscene.gr/articles/sdr\\_fract/](http://nuclear.demoscene.gr/articles/sdr_fract/)
  - [http://developer.download.nvidia.com/shaderlibrary/webpages/shader\\_library.html](http://developer.download.nvidia.com/shaderlibrary/webpages/shader_library.html)
  - <http://www2.imm.dtu.dk/visiondag/VD03/grafisk/WolfgangEngel.pdf>
  - <http://msdn2.microsoft.com/en-us/library/bb173010.aspx>
  - lots and lots and lots more out there