Basic Effect’s DefaultLighting

Notes on Effects will focus on MonoGames 3.5 Windows Platform projects – GPU/OS interface specific.

AGMGSK uses a stock BasicEffect, so no Effect object is created, there are no other lights.
3 “white-ish” lights are created directed towards the origin

```csharp
private void drawModel(GameTime gameTime, Model m) {
    Matrix[] transforms = new Matrix[m.Bones.Count];
    // get model's can have embedded transforms
    m.CopyAbsoluteBoneTransformsTo(transforms);
    foreach (ModelMesh mesh in m.Meshes) {
        foreach (BasicEffect effect in mesh.Effects) {
            effect.EnableDefaultLighting();
            effect.View = view;
            effect.Projection = projection;
            effect.World = Matrix.Identity * rotateY * transforms[mesh.ParentBone.Index];
        }
        mesh.Draw();
    }
}
```
BasicEffect’s properties

  Alpha  1.0  float
+ AmbientLightColor  {X:0.053 Y:0.098 Z:0.181}
+ DiffuseColor  {X:1 Y:1 Z:1}

- DirectionalLight0
+ DiffuseColor  {X:1 Y:0.960 Z:0.807}
+ Direction  {X:-0.526 Y:-0.573 Z:-0.627}  // LDB
  Enabled  true  bool
+ SpecularColor  {X:1 Y:0.960 Z:0.807}

- DirectionalLight1
+ DiffuseColor  {X:0.964 Y:0.760 Z:0.407}
+ Direction  {X:0.719 Y:0.342 Z:0.604}  // RUF
  Enabled  true  bool
+ SpecularColor  {X:0 Y:0 Z:0}
- DirectionalLight2
  {
  }
+ DiffuseColor   {X:0.323 Y:0.360 Z:0.393}
+ Direction      {X:0.454 Y:-0.766 Z:0.454} // RDF
  Enabled        true    bool
+ SpecularColor  {X:0.32 Y:0.360 Z:0.393}

+ EmissiveColor  {X:0 Y:0 Z:0}
+ FogColor       {X:0 Y:0 Z:0}
    FogEnabled  false    bool
    FogEnd      1.0       float
    FogStart    0.0       float
  LightingEnabled true    bool
  PreferPerPixelLighting false    bool
+ SpecularColor  {X:0.55 Y:0.42 Z:0.13}
    SpecularPower 27.0    float
    TextureEnabled true    bool
    VertexColorEnabled false    bool
+ View          Microsoft.Xna.Framework.Matrix
+ World         Microsoft.Xna.Framework.Matrix
Blending

Blending – compositing, adds values from two pixels to determine the rendered pixel value.

\[
\text{NewColor} = \text{SrcBlend} \times \text{SrcColor} + \text{DestBlend} \times \text{DestColor}
\]

Both SrcBlend and DestBlend can be defined.

By default, MonoGames expects the images to contain transparency information for each pixel in the A (Alpha) channel, which contains the transparency information.

Image formats (like PNG) store this information, other (like JPG) don’t. DirectX meshes have material color values.

Modelers can set RGBA properties of materials (colors), or set in program.

Fog is a built-in blending technique using distance from camera.
Often blending uses the Alpha value as SrcBlend factor. The inverse value, (1-SrcBlend) is used as DestBlend:

```
NewColor = SrcAlpha * SrcColor + (1 - SrcAlpha) * DestColor
```

Weighting the colors of all images to add together. Equal weight sets both blending factors to 1: Blend.One

In MonoGames blending is done by creating a BlendState and setting a device’s BlendState

```
BlendState alphaBlend;
...
alphaBlend = new BlendState();
alphaBlend.ColorSourceBlend = Blend.SourceAlpha;
alphaBlend.ColorDestinationBlend = Blend.InverseSourceAlpha;
alphaBlend.ColorBlendFunction = BlendFunction.Add;
...
GraphicsDevice.BlendState = alphaBlend;
```
Blending

"B" blending on
"N" blending off
"O" Blends (source, destination) both = One
"A" Blends = SourceAlpha, InverseSourceAlpha

Agent model's BoundingSphere used to scale the "sphere"
DirectX mesh, wireSphere.x alpha is 0.243
AC3D6.1 model, wireSphere.ac transparency is 0.757

not blending, transparent sphere, lights blending
HLSL

HLSL is not used to improve the gameplay – interaction.

HLSL enhances the quality of the final image.

Every vertex that is drawn will pass through the vertex shader.

Every pixel drawn will pass through the pixel shader.

HLSL is a C-like language stored in a separate file (*.fx)

Similar to Nvidia's Cg and GLSL

Reimer's HLSL tutorial

NeatWare HLSL tutorial
Effect

Effect represents shader effects enables effect query and selection requires an effect.fx file

ApplyAnEffect's ReallySimpleEffect.fx next slide

Add shader file to Content subdirectory

Load the effect file
effect = Content.Load<Effect>("ReallySimpleEffect");

Set effect's parameters in LoadContent() & Update(…)
effect.Parameters["WorldViewProj"].SetValue(worldViewProjection);

Set effect's technique in LoadContent()
effect.CurrentTechnique =
effect.Techniques["TransformTechnique"];

Draw model with effect (just like w/ BasicEffect)

foreach (EffectPass pass in effect.CurrentTechnique.Passes) {...}
Vertex shader

programmable pipeline

model space

vertex shader

homogeneous clip space

world & view transformation

lighting

projection transformations

What happens

Vertex shader needs:

variables colors, normals, positions

Matrices view, projection, and world (transforms)

Effect object to encapsulate shader program code

*fx effect file for shader code

technique property specifies passes

every point in shader program for each pass
**Variables**

<table>
<thead>
<tr>
<th>Prefix</th>
<th>static</th>
<th>global variable like C++</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>uniform</td>
<td>input – is initialized outside shader</td>
</tr>
<tr>
<td></td>
<td>extern</td>
<td>accessible outside shader</td>
</tr>
<tr>
<td></td>
<td>const</td>
<td>like C++ constant</td>
</tr>
</tbody>
</table>

**Scalar types**

|          | bool, int (32), half (16), float (32), double (32) |

**Vectors**

|          | float4 pos; |
|          | pos[0] = 2.0f; |
|          | or pos.x = 2.0f; |

|          | vector |
|          | 4D float array |

|          | vector<T, n> |
|          | n dimensional vector of type T |

**Swizzling**

- can use combined "members" from xyzw, rgba ...
- pos.xz = 0.0f;

**Array**

|          | float M[4][4]; |
|          | int point[4]; |
Input, Output structures

```c
struct VS_INPUT {
    vector position : SV_POSITION;
}

struct VS_Output {
    vector position : SV_POSITION;
    vector diffuse : COLOR;
};
```

: syntax  is a semantic to specify usage of a variable.

SV can be used as input and output to a shader function

- **position**  the position of input and output vertex
- **diffuse**   the color of the output vertex.

Maps a shader variable to a hardware register
Matrix, structure

matrix float4x4 view;

matrix 4 x 4 float

matrix<T, m, n> m by n matrix of type T

M[i][j] = 1.0f; vector rowI = M[i];

M.11 = 2.0f to M.44 = 1.0f "one based"
M.m00 = 2.0f to M.m33 = 1.0f "zero based"

structure struct aStruct {
    matrix V, T, W;
    vector pos;
    bool valid;
};

typedef can be used as in C++
Functions

Many common graphics related functions built-in:

- abs(int), ceil(int), floor(int), sqrt(x),
- cos(radian), sin(radian), tan(radian)
- cross(u,v), dot(u,v)
- mul(M,N) //matrix multiply returns MN
  if M is vector – a row vector, N is vector – its a column vector
- normalize(v), distance(u, v), determinant(M), transpose(M)
- saturate(x), returns x in range 0.0 to 1.0
- rsrrt(x), returns 1/sqrt(x) inverse distance ... light
- lerp(x, y, s), linear interpolates x..y based on s 0..1 \( x + s(y - x) \)

User defined functions – C++ syntax

arguments passed by value, no recursion, always inlined.
argument type modifiers:

- in (default)
- out (return value via arg)
- inout
flow of control

HLSL supports standard C flow of control statements:
   if, if-else, for, while, do

Explicit casting is the same a C and Java

C operators:  +  +=  &&  ||  !=  %  []

MSDN example ApplyAnEffect.cs uses ReallySimpleEffect.fx – see links at end of slides

Vertex shader code next slide

Stock shaders code including XNA 4 BasicEffect – see links at end of slides
Effects

uniform float4x4 WorldViewProj : WORLDVIEWPROJECTION;

struct VS_OUTPUT {
    float4 position : SV_POSITION;
    float4 color : COLOR0;
};

VS_OUTPUT VertexShaderFunction(
    float4 Pos : SV_POSITION,
    float4 Color : COLOR0)
{  
    VS_OUTPUT Out = (VS_OUTPUT)0;
    Out.position = mul(Pos, WorldViewProj);
    Out.color = Color;
    return Out;
}

float4 PixelShaderFunction(VS_OUTPUT vsout) : COLOR
{  
    return vsout.color;
}

technique TransformTechnique
{  
    pass P0
    {
        vertexShader = compile vs_4_0 VertexShaderFunction();
        pixelShader = compile ps_4_0 PixelShaderFunction();
    }
}
Texture Samplers

For each different texture map that you plan to sample in a pixel shader, you must declare a sampler object.

```cpp
sampler MeshTextureSampler = sampler_state {
    Texture = <MeshTexture>; MipFilter = LINEAR;
    MinFilter = LINEAR; MagFilter = LINEAR; }
```

Texture sampling uses the texel position to look up a texel value. An offset can be applied to the position before look up. The sampler state contains the sampling and filtering options.

The drawn pixel can be selected from different samplers or can be blended from the samplers.

Get the color of the pixel's texel in pixel shader:
```cpp
Texture MeshTexture;
sampler TextureSampler = sampler_state { texture =
    <MeshTexture>; mipfilter = LINEAR; }
...
float2 TexCoords : TEXCOORD0;
...
float4 texelColor = tex2D(TextureSampler, TexCoords);
```
Light

A point on a surface is seen as reflected light from a view.
The light can have an intensity and a color that illuminates a surface.
The surface has color and reflective "material" properties:
  - color or texture
Lights have no geometry (may need "shape" to represent light source)
Pixel shader light attenuation

Point (bare bulb) and spot lights (flashlight) intensity decreases with distance of light from surface.

Point light has position but no direction.

Spot lights have position, direction, an inner radius (maximum) and outer radius (falloff) of light attenuation. Outer radius optional

Assume vertex shader output includes vertex position and normal.

Pixel shader computes light intensity to add to pixel.

Point light's contribution to pixel intensity

\[ \text{LightIntensity} = \text{dot} ( \text{normal}, - \text{lightPosition} ) \]
Ambient and Diffuse light

Ambient light intensity ($I_a$) at a surface point has been scattered by many reflections and is the source light's ambient color ($L_a$) scaled by surface's ambient (material) reflectivity constant ($K_a$).

$$I_a = L_a * K_a$$

Diffuse light intensity ($I_d$) at a surface point is a function of the source light intensity ($L_d$), the point's normal ($n$) to the surface, and the surface's diffuse reflectivity constant ($K_d$)

$$I_d = L_d * K_d * \text{dot}(\text{surfaceNormal}, -\text{lightSource})$$

Ambient and Diffuse lights are not affected by the position of the viewer. Diffuse light is omni-directionally reflected.
// LightEffects.fx, per vertex light calculation
float4x4 World;
float4x4 View;
float4x4 Projection;
float3 EyePosition;
float3 AmbientColor;
Texture MeshTexture;
sampler TextureSampler = sampler_state { texture = MeshTexture; mipfilter = LINEAR; };

struct VertexShaderInput {
    float4 Position : POSITION0;
    float2 inTexCoord : TEXCOORD0;
    float3 inNormal : NORMAL0;
};

struct VertexShaderOutput {
    float4 Position : POSITION0;
    float2 TexCoords : TEXCOORD0;
    float LightFactor : TEXCOORD1;
};
VertexShaderOutput VertexShaderFunction(
   VertexShaderInput input)    {
   VertexShaderOutput output;
   float4 worldPosition = mul(input.Position, World);
   float4 viewPosition = mul(worldPosition, View);
   // orient position and normalize normal
   output.Position = mul(viewPosition, Projection);
   float3 normal = normalize(input.inNormal);
   // get rotation part of world matrix to rotate normal
   float3x3 rotationMatrix = (float3x3)World;
   normal = mul(normal, rotationMatrix);
   // set texture coordinates - to get pixel color later
   output.TexCoords = input.inTexCoord;
   // Set point light's position closer to surface than eye
   float3 pointPosition = float3(1000, EyePosition.y - 1000, 1000);
   float3 lightDirection = input.Position - pointPosition;
   float distance = length(lightDirection);
   lightDirection = normalize(lightDirection);
   output.LightFactor = dot(normal, -lightDirection);
   // LF = LF * 1/sqrt(distance)
   output.LightFactor /= sqrt(distance);
   output.LightFactor *= 100.0f;  // magnify lighting factor
   return output;  }

Pixel shader

// pointColor is the point light's contribution
float4 PixelShaderFunction(VertexShaderOutput input) : COLOR0 {
    float4 diffuseColor = tex2D(TextureSampler, input.TexCoords);
    // initialize pointColor to BasicEffect's AmbientColor
    float4 pointColor = float4(0.053f, 0.098f, 0.181f, 1.0f);
    // calculate the point light's color contribution
    pointColor = saturate(pointColor * input.LightFactor);
    // scale for "brightness"
    diffuseColor = saturate((diffuseColor + pointColor) * 0.75f);
    return diffuseColor;
}

technique Technique1 {
    pass Pass1 {
        VertexShader = compile vs_4_0 VertexShaderFunction();
        PixelShader = compile ps_4_0 PixelShaderFunction();
    }
}
LightEffects.cs example has textured "lumpy" terrain. Eye looking down Y.

Commands:
b || e  BasicEffect or LightEffects.fx

c  BasicEffect directional light color

Arrow keys move eye position in X Z

Page Up || Down zoom eye on Y

BasicEffect ↑  LightEffects ↓

Point light in center, note even edge (per pixel) ↓
Basic Effect XNA 4.0 optimization

Stock Effect XNA 4.0 code – includes Basic Effect

ReallySimpleEffect.fx MSDN example

HLSL