ECE4893A/CS4803MPG: Multicore and GPU Programming for Video Games

Lecture 12: XNA and Programmable Shaders

Prof. Hsien-Hsin Sean Lee
School of Electrical and Computer Engineering
Georgia Institute of Technology
DirectX and GPU (Nvidia-centric)

History

- DirectX 5
  - Riva 128
  - 3dfx’s first Voodoo chip
  - 1996

- DirectX 6
  - Multitexturing Riva TNT (NV4)
  - 1998

- DirectX 7
  - T&L GeForce 256 (NV10)
  - 1999

- DirectX 8
  - SM 1.x GeForce3 (NV20)
  - 2000

- DirectX 8.1
  - Cg

- DirectX 9
  - SM 2.0 GeForceFX (NV30)
  - 2001

- DirectX 9.0c
  - SM 3.0 GeForce 6 (NV40)
  - 2002

- DirectX 10
  - SM 4.0 GeForce 8 (G80)
  - 2006

- GTX200
  - 1.4 billion Transistors
  - 2008

Legend:
- NVidia’s response to Voodoo 2
- 686 million Transistors
- 1.5GHz

Adapted from David Kirk’s slide
Why Programmable Shaders

• Hardwired pipeline
  - Produces limited effects
  - Effects look the same
  - Gamers want unique look-n-feel
  - Multi-texturing somewhat alleviates this, but not enough
  - Less interoperable, less portable

• Programmable Shaders
  - Vertex Shader
  - Pixel or Fragment Shader
  - Starting from DX 8.0 (assembly)
  - DX 9.0 added HLSL (High Level Shading Language)
  - HLSL (MS) is compatible with Cg (Nvidia)
Evolution of Graphics Processing Units

- **Pre-GPU**
  - Video controller
  - Dumb frame buffer
- **First generation GPU**
  - PCI bus
  - Rasterization done on GPU
  - ATI Rage, Nvidia TNT2, 3dfx Voodoo3 (‘96)
- **Second generation GPU**
  - AGP
  - Include **T&L** into GPU
  - Nvidia GeForce 256, ATI Radeon 7500, S3 Savage3D (‘98)
- **Third generation GPU**
  - Programmable vertex shader
  - Nvidia GeForce3, ATI Radeon 8500, Microsoft Xbox (‘01)
- **Fourth generation GPU**
  - Both programmability in vertex and fragment shaders
  - Nvidia GeForce FX, ATI Radeon 9700 (‘02)
- **Current generation GPU**
  - Include **geometry shader** (SM 4.0 and DX10)
  - Nvidia G80 and up
Programmable Graphics Pipeline

3D Apps

API commands

3D API: Direct3D

GPU cmd & data stream

Nvidia GeForce FX

Fixed Function Pipeline

GPU Frontend

Assembled polygons

Pixel location

Pixel updates

Vtx index

Transformed vertices

Transformed Fragments

Programmable Vertex Shader

Programmable Fragment Shader

Source: Cg tutorial
Graphics Programmable Pipeline

DirectX 8.0 Pipeline
Choice between programmable and fixed function pipeline
(mutually exclusive, parallel pipelines)

DirectX 10.0 Pipeline, Fully Programmable
XNA Rendering Pipeline

- Vertex shader outputs transformed vertex position, texture coordinates, color, etc.
  - Solid deforming, skeletal animation, particle motion, etc.
- Rasterization interpolates and determines what pixels to draw
- Pixel shader outputs pixel color, depth (optional)
  - Per-pixel lighting, procedural texture generation, postprocessing effects, brightness, contrast, blur, etc.

Adapted from XNA 2.0 Game Programming Book
Shader Languages

• HLSL/Cg most common
  – Both are compatible
  – No assembly shaders allowed in DX 10.0.

• Other options:
  – GLSL
  – Legacy DirectX shaders in assembly
  – Sh
  – OpenVidia (U of Toronto)
Basic Shader Mechanics

• Data types:
  – Typically floats, and vectors/matrices of floats
  – Fixed size arrays
  – Three main types:
    • Per-instance data, e.g., per-vertex position
    • Per-pixel interpolated data, e.g., texture coordinates
    • Per-batch data, e.g., light position
  – Data are tightly bound to the GPU
  – Flow control is very simple:
    • No recursion
    • Fixed size loops for v_2_0 or earlier
    • Simple if-then-else statements allowed in the latest APIs
    • Texkill (asm) or clip (HLSL) or discard (GLSL) or allows you to abort a write to a pixel (form of flow control)
Vertex Shader

- Transform to clip-space (i.e., screen space)
- Inputs:
  - Common inputs:
    - Vertex position \((x, y, z, w)\)
    - Texture coordinate
    - Constant inputs
      - Can also have fog, color as input, but usually leaves them untouched for pixel shader
  - Output to Pixel (fragment) shader
- Vertex shader is executed once per vertex, could be less expensive than pixel shader
Vertex Shader (3.0)

- **Vertex stream**
  - v0, v1, v2, ..., v15
  - 16 Vertex data registers

- **32 Temporary registers**
  - r0, r1, r2, ..., r31

- **Constant float registers (at least 256)**
  - C0, C1, C2, ..., Cn
  - 16 Constant Integer Registers

- **Loop Register**
  - aL

- **Address Register**
  - a0

- **12 output registers**
  - oPos (position)
  - oTn (texture)
  - oFog (fog)
  - oD0 (Diff. color)
  - Spec. color
  - oPts (Output Pt size)

Each register is a 4-component vector register except aL.
Pixel (or Fragment) Shader

- Determine each fragment’s color
  - custom (sophisticated) pixel operations
  - texture sampling

- Inputs
  - Interpolated output from vertex shader
  - Typically vertex position, vertex normals, texture coordinates, etc.
  - These registers could be reused for other purposes (thus called GPGPU)

- Output
  - Color (including alpha)
  - Depth value (optional)

- Executed once per pixel so is executed a lot more times than vertex shader typically
  - It is advantageous to compute stuff on a per-vertex basis to improve performance
Pixel Shader (3.0)

Pixel stream

Color (diff/spec) and texture coord. registers

Constant registers (16 INT, 224 Float)

Temporary registers

Sampler Registers (Up to 16 texture surfaces can be read in a single pass)

Pixel Shader

-oC0 color

-oDepth Depth
Use of the Vertex Shader

- Transform vertices to clip-space
- Pass normal, texture coordinates to PS
- Transform vectors to other spaces (e.g., texture space)
- Calculate per-vertex lighting (e.g., Gouraud shading)
- Distort geometry (waves, fish-eye camera)

Adapted from Mart Slot’s presentation
Use of the Pixel Shader

- Texturing objects
- Per-pixel lighting (e.g., Phong shading)
- Normal mapping (each pixel has its own normal)
- Shadows (determine whether a pixel is shadowed or not)
- Environment mapping

Adapted from Mart Slot’s presentation
HL SL / Cg

• Compatible, jointly developed by Microsoft and Nvidia
• A C-like language and syntax
• DX 10 will discontinue assembly shader

• But do not have
  – Pointers
  – Dynamic memory allocation
  – Unstructured/complex control structure
    • e.g., goto
    • Recursion (note that functions are inlined)
  – Bitwise operations (may have in the future)
A Simple Vertex Shader

```cpp
uniform extern float4x4 gWVP;

struct VtxOutput {
    float4 position : POSITION;
    float4 color    : COLOR;
};

VtxOutput All_greenVS(float2 position : POSITION) {
    VtxOutput OUT;

    OUT.position = mul(float4(position, -30.0f, 1.0f), gWVP);
    OUT.color    = float4(0, 1, 0, 1);

    return OUT;
}
```

Adapted from Cg Tutorial
A Simple Vertex Shader (Alternative)

```c
uniform extern float4x4 gWVP;

void All_greenVS(float2 position: POSITION,
                  out float4 oPosition: POSITION,
                  out float4 oColor : COLOR)
{
    oPosition = mul(float4(position, -30.0f, 1.0f), gWVP);
    oColor = float4(0, 1, 0, 1);
}
```

No structure declaration
Uniform versus Variable Input

- Two different input data types in Shader code

**uniform** (a keyword in Cg/HLSL) Input:
- Global, do not change per vertex
- Outside the scope of the shader function
- Define in the beginning of a shader code

Variables Input
- Attributes assoc. with each vertex
- Declared using Semantics
The uniform Type Qualifier

• A uniform variable value can come from external
  – E.g., your C# application

• Retrieve the initial value from a constant register (e.g., c0, read-only) in the GPU

• Uniform (or global) to all processed vertices in the entire shading process
Semantics

• Something new to C/C++ programming
• A colon and a keyword, e.g.,
  - MonsterPos : POSITION
  - VertexColor : COLOR
  - VertexNormal : NORMAL
  - VertexUVcoord : TEXCOORD0

• A glue that
  - binds an HLSL program to the rest of the graphics pipeline
  - Connects the semantic variables to the pipeline
A Simple Pixel (or Fragment) Shader

```c
struct PixelOutput {
    float4 color : COLOR;
};

PixelOutput All_greenPS(float4 color : COLOR) {
    PixelOutput PSout;
    PSout.color = color;
    return PSout;
}
```

Adapted from Cg Tutorial
Profiles

• Need a profile to compile the vertex shader and the pixel shader

• Specify shader models, for example
  – vs_3_0 for vertex shader
  – ps_3_0 for pixel shader

• Specify particular models for compilation

• Can be embedded inside technique

  vertexShader = compile vs_2_0 PhongVS();
  pixelShader  = compile ps_2_0 PhongPS();
Flow Control (Predicating Constants)

```c
if (posL.y < 0)
    outVS.posH.x = -1.0f;
else
    outVS.posH.x = 2.0f;
```

def c0, 0, -3, 2, 1
dcl_position v0
slt r0.x, v0.y, c0.x
mad oPos.x, r0.x, c0.y, c0.z

```
\[
\begin{array}{cccc}
| \text{vs}_2 \_0 \text{ compiled code} |
\end{array}
\]

```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
XNA Effect Framework

• An “Effect”
  – Encapsulates shader properties
    • E.g., Water modeling, steel modeling has their own effects
  – Reusable for the same type of modeled objects

• An **effect** consists of one or more **techniques**
  – To enable fallback mechanism on different GPUs
  – Several versions of an effect (GPU-dependent)

• A **technique** consists of one or more **passes**

• Described in an effect file (.fx) in XNA framework
  – External file
  – No application recompilation needed
An Example of an FX File

```c
uniform extern float4x4 gWVP;
uniform extern float4 gAmbMtrl;

void VShader(float4 pos : POSITION, float4 normal : NORMAL,
             out float4 oColor : COLOR)
{
    . . . . .
}

float4 PShader(float4 color : COLOR) : COLOR
{
    . . . . .
}

technique SuperShading
{
    pass P0
    {
        vtxshader = compile vs_2_0 VShader();
pxlshader = compile ps_2_0 PShader();
        FillMode = Wireframe; // default Solid
    }
}
```
Create an Effect File

Again, put in the Content Pipeline
**Initialize Effect in C# Code**

```csharp
Effect myEffect;

myEffect = Content.Load<Effect>("SimpleEffect");

GT = Content.Load<Texture2D>("Buzz");

myEffect.Parameters["gWVP"].SetValue(WVPmatrix);
myEffect.Parameters["decal"].SetValue(GT);

myEffect.CurrentTechnique =
  myEffect.Techniques["myTech"];```

- Fx filename included in the Content Pipeline
- Texture map included in the Content Pipeline
- Uniform variable defined in the .fx file
- Variable defined in C# source
- myTech is defined in the .fx file
Applying an Effect

```csharp
// Update the matrix
myEffect.Parameters["gWVP"].SetValue(gWVP);

myEffect.Begin();
foreach (EffectPass pass in myEffect.CurrentTechnique.Passes)
{
    pass.Begin();

    graphics.GraphicsDevice.DrawUserIndexedPrimitives(PrimitiveType.TriangleList,
            vertex, 0, 24, triangleListIndices, 0, 12);

    pass.End();
}
myEffect.End();
```

- If changing states after pass begins, myEffect.CommitChanges() need to be used to propagate changes to the device.
uniform extern float4x4 gWVP;

void TexVS(float3 position : POSITION,
            float3 color : COLOR,
            float2 texcoord : TEXCOORD0,
            out float4 oPos : POSITION,
            out float4 oColor : COLOR,
            out float2 oTexcoord : TEXCOORD0)
{
    oPos = mul(float4(position, 1.0f), gWVP);
    oColor = float4(color, 1.0f);
    oTexcoord = texcoord;
}

Adapted from Cg Tutorial
Pixel Shader Code for Texturing

```cpp
uniform extern texture texture_monster_skin;
sampler TexS = sampler_state {
    Texture = <texture_monster_skin>;
    MinFilter = Anisotropic;
    MagFilter = LINEAR;
    MipFilter = LINEAR;
    MaxAnisotropy = 8;
    AddressU = WRAP;
    AddressV = WRAP;
};

void TexPS(float4 pos : POSITION,
           float3 color : COLOR,
           float2 texcoord : TEXCOORD0,
           out float4 oColor : COLOR)
{
    float4 temp = tex2D(TexS, texcoord);
    color = lerp(temp, color, 0.5);
}
```

Adapted from Cg Tutorial
Sampler Objects (in .fx file)

```cpp
uniform extern texture texture_brick;

sampler textureSampler = sampler_state
{
    Texture = texture_brick;
    MinFilter = POINT;
    MagFilter = LINEAR;
    MipFilter = Anisotropic;
    MaxAnisotropy = 8;
    AddressU = WRAP;
    AddressV = WRAP;
};

void PixelShader(float2 texcoord : TEXCOORD0)
{
    float4 color = tex2D(textureSampler, texcoord);
}
```
First XNA/HLSL Example

NoLightCubeHLSL
(See Demo in Visual Studio)
Structure Lights and Objects

• Make separate classes (C# files) for
  – Light sources
  – Objects
Adding Light Source

- Declare a new light class (Lights.cs)

```csharp
public class Lights : Microsoft.Xna.Framework.GameComponent
{
    public Vector4 position;
    public Vector4 a_material;
    public Vector4 d_material;
    public Vector4 diffuseLight;
    public Vector4 ambientLight;

    public Lights(Game game)
        : base(game)
    {
    }

    public Vector4 Position
    {
        get
        {
            return position;
        }
        set
        {
            position = value;
        }
    }

    public Vector4 A_material
    {
        get
        {
            return a_material;
        }
        set
        {
            a_material = value;
        }
    }

    public Vector4 D_material
    {
        get
        {
            return d_material;
        }
        set
        {
            d_material = value;
        }
    }
}
```

```csharp
private void InitializeLightSource()
{
    Light1 = new Lights(this);
    Light1.Position = new Vector4(0.0f, 2.0f, -4.0f, 1.0f);
    Light1.A_material = new Vector4(0.412f, 0.412f, 0.412f, 1.0f);
    Light1.D_material = new Vector4(0.412f, 0.412f, 0.412f, 1.0f);
    Light1.DiffuseLight = new Vector4(1.0f, 1.0f, 1.0f, 1.0f);
    Light1.AmbientLight = new Vector4(0.4f, 0.4f, 0.4f, 0.4f);
}
```

In game.cs
Adding Light Source (cont’d)

```csharp
protected override void Draw(GameTime gameTime)
{
    graphics.GraphicsDevice.Clear(Color.Black);
    graphics.GraphicsDevice.VertexDeclaration = LeeVertexDecl;

    cubeEffect.Parameters["gWVP"].SetValue(gWVP);
    cubeEffect.Parameters["lightPosition"].SetValue(Light1.Position);
    cubeEffect.Parameters["a_material"].SetValue(Light1.A_material);
    cubeEffect.Parameters["d_material"].SetValue(Light1.D_material);
    cubeEffect.Parameters["diffuseLight"].SetValue(Light1.DiffuseLight);
    cubeEffect.Parameters["ambientLight"].SetValue(Light1.AmbientLight);

    cubeEffect.Begin();
    foreach (EffectPass pass in cubeEffect.CurrentTechnique.Passes)
    {
        pass.Begin();

        graphics.GraphicsDevice.DrawUserIndexedPrimitives(PrimitiveType.TriangleList,
            vertex,
            0,
            24,
            triangleListIndices,
            0,
            12);

        pass.End();
    }
    cubeEffect.End();
    base.Draw(gameTime);
}
```
Teapot Class (Object Initiation)

```csharp
public Teapot(Game game)  
: base(game)
{
    thisTeapot = game.Content.Load<Model>("teapot");
}

global Vector3 Initial_Position
{
    get
    {
        return initial_position;
    }
    set
    {
        initial_position = value;
    }
}

public Effect TeapotEffect
{
    get
    {
        return teapotEffect;
    }
    set
    {
        teapotEffect = value;
    }
}

global Matrix WorldMatrix
{
    get
    {
        return worldMatrix;
    }
    set
    {
        worldMatrix = value;
    }
}
```

To associate the rendering Effect (.fx)

To associate the transformation matrix
Shader Code

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

    // transformation
    output.Position = mul(float4(input.Position, 1.0f), gWVP);

    // only if texture is used
    output.TexCoord = input.TexCoord;

    // lighting
    float4 ambient = ambientLight * a_material;

    float3 TransP = mul(float4(input.Position.xyz, 1.0f), gWorld);
    float3 TransN = mul(float4(input.Normal.xyz, 0.0f), gWorld);
    float3 L = normalize(lightPosition - TransP);
    float intensity = max(dot(TransN, L), 0);
    float4 diffuse = d_material * diffuseLight * intensity;

    output.Color = ambient + diffuse;
    return output;
}

float4 PixelShaderFunction(VertexShaderOutput input) : COLORO {
    PixelShaderOutput pout;
    float4 temp = tex2D(textureSampler, input.TexCoord);
    pout.Color = temp*input.Color;
    return pout.Color;
}
```

Per-Vertex Lighting
XNA/HLSSL Lighting Example

OneLightCubeHLSSL
(See Demo in Visual Studio)
Ambient and Diffuse Light Teapot Example

OneLightTeapotHLSSL (See Demo in Visual Studio)
Many Lights

- Use “array” in Shader code
- How to pass the light property into Shader from XNA C# apps?

```c
struct lightProperty {
    float4 position;
    float4 a_material;
    float4 d_material;
    float4 diffuseLight;
    float4 ambientLight;
    int on;
};

uniform extern lightProperty light[2];

for (i=0; i<numLights; i++)
{
    ambient[i] = light[i].ambientLight * light[i].a_material;
    l = normalize(light[i].position - P);
    intensity = max(dot(N, l), 0);
    diffuse[i] = light[i].d_material * light[i].diffuseLight * intensity;
}

for (i=0; i<numLights; i++)
{
    output.Color += (ambient[i] + diffuse[i]);
}
```

Declare struct in shader code

Declare light array

Walk through the lights in Vertex Shader
Adding Light Sources in C# Applications

Inside draw() call

```csharp
private void InitializeLightSource()
{
    Light = new Lights[2];
    Light[0] = new Lights(this);
    Light[0].Position = new Vector4(2.0f, 2.0f, -4.0f, 1.0f);
    Light[0].A_material = new Vector4(0.412f, 0.412f, 0.412f, 0.412f);
    Light[0].D_material = new Vector4(0.412f, 0.412f, 0.412f, 0.412f);
    Light[0].DiffuseLight = new Vector4(1.0f, 1.0f, 1.0f, 1.0f);
    Light[0].AmbientLight = new Vector4(0.4f, 0.4f, 0.4f, 0.4f);
    Light[0].On = 1;

    Light[1] = new Lights(this);
    Light[1].Position = new Vector4(-2.0f, 2.0f, -4.0f, 1.0f);
    Light[1].A_material = new Vector4(0.412f, 0.412f, 0.412f, 0.412f);
    Light[1].D_material = new Vector4(0.412f, 0.412f, 0.412f, 0.412f);
    Light[1].DiffuseLight = new Vector4(1.0f, 0.0f, 0.0f, 1.0f);
    Light[1].AmbientLight = new Vector4(0.4f, 0.0f, 0.0f, 0.4f);
    Light[1].On = 1;

    numLights = 2;
}
```

for (i = 0; i < numLights; i++)
{

    teapotEffect.Parameters["light"].Elements[i].StructureMembers["position"].SetValue(Light[i].Position);
    teapotEffect.Parameters["light"].Elements[i].StructureMembers["a_material"].SetValue(Light[i].A_material);
    teapotEffect.Parameters["light"].Elements[i].StructureMembers["d_material"].SetValue(Light[i].D_material);
    teapotEffect.Parameters["light"].Elements[i].StructureMembers["diffuseLight"].SetValue(Light[i].DiffuseLight);
    teapotEffect.Parameters["light"].Elements[i].StructureMembers["ambientLight"].SetValue(Light[i].AmbientLight);
    teapotEffect.Parameters["light"].Elements[i].StructureMembers["on"].SetValue(Light[i].On);
}

Link the corresponding array element in the shader code
Two Lights
Teapot Example

TwoLightTeapotHLSL
(See Demo in Visual Studio)
Turn off the Red light by pressing “R”
Math Operators

- Most commonly used C/C++ operations are supported
- Some are reserved for the future implementations, e.g.,
  - Bitwise logic operation (&, ^, |, &=, |=, ^=...)
  - Shift: << , >>, <<=, >>=
  - Modular: %
  - *, -> (No pointer support or indirection in Cg/HLSL)
Standard Library Function

- Many, ... to name a few

\texttt{dot}(a, b)
\texttt{cross}(a, b)
\texttt{distance}(pt1, pt2) : Euclidean distance
\texttt{lerp}(a, b, f) : \( r = (1-f)*a + f*b \)
\texttt{lit}(NL, NH, pwr) : calculate amb, diff, spec co-efficients
\texttt{mul}(M, N)
\texttt{normalize}(v)
\texttt{reflect}(I, N) : calculate reflect vector of ray I
\texttt{sincos}(x, s, c) : calculate \( \sin(x) \) and \( \cos(x) \)
Morphing Using Two Textures

- Blending the values of two textures
- The interpolated co-efficient changes per frame

```c
float4 PixelShaderFunction(VertexShaderOutput input) : COLOR0
{
    PixelShaderOutput pout;

    float4 temp = tex2D(textureSampler, input.TexCoord);
    float4 temp2 = tex2D(textureSampler2, input.TexCoord);

    if (morphrate >= 0.0f)
    {
        pout.Color = lerp(temp, temp2, morphrate)*input.Color;
    }
    else // control timing...
    {
        pout.Color = lerp(temp, temp2, 0.0f)*input.Color;
    }

    return pout.Color;
}
```

Linear interpolation of two texture maps
Morphing Texture HLSL Examples

TextureMorphing
(See Demo in Visual Studio)
More Lighting Effect

- Per-Vertex specular computation

```c
TransP = mul(float4(input.Position, 1.0f), gWorld).xyz;
TransN = mul(float4(input.Normal, 0.0f), gWorld).xyz;
V = normalize(eyePosition - TransP);

for (i=0; i<numLights; i++)
{
    ambient[i] = 0.0f;
    diffuse[i] = 0.0f;
    spec[i] = 0.0f;
    if (light[i].on != 0)
    {
        // lighting
        ambient[i] = light[i].ambientLight * light[i].a_material;
        L = normalize(light[i].position.xyz - TransP);
        intensity = max(dot(TransN, L), 0);
        diffuse[i] = light[i].d_material * light[i].diffuseLight * intensity;
        // specular effect for just the light specified specular is on
        if (light[i].spec_on == 1)
        {
            H = normalize(L+V);
            spec_intensity = pow(max(dot(TransN, H), 0), light[i].shininess);
            spec[i] = light[i].s_material * light[i].specLight * spec_intensity;
        }
    }
}

output.Color = float4(0.0f, 0.0f, 0.0f, 0.0f);
for (i=0; i<numLights; i++)
{
    output.Color += (ambient[i] + diffuse[i] + spec[i]);
}
return output;
```
Two Lights + Per-Vertex Spec Light
Teapot Example

ThreeLightSpecTeapotHLSL
(See Demo in Visual Studio)
Toggle the Specular light by pressing “P”
Per-Vertex vs. Per-Pixel Shading

GouroudVertexShader(float3 posL : POSITION0,
    float3 normalL : NORMAL0,
    out float oPos : POSITION0,
    out float oColor : COLOR0)
{
    . . .
    lightVecW = normalize(LightPosW - posW);
    ambient = (AmbMtrl*AmbLight).rgb;
    s = max(dot(normalW, lightVecW), 0.0f);
    diffuse = s*(DiffMtrl*DiffLight).rgb;
    toEyeW = normalize(EyePosW - posW);
    reflectW = reflect(-lightVecW, normalW);
    t = pow(max(dot(reflectW, toEyeW), 0.0f), SpecPower);
    spec = t*(SpecMtrl*SpecLight).rgb;
    oColor = ambient + ((diffuse + spec) / A);
    // Transform to homogeneous clip space.
    oPos = mul(float4(posL, 1.0f), gWVP);
}

GouroudPixelShader(float4 c : COLOR0) : COLOR
{
    return c;
}

PhongVertexShader(float3 posL : POSITION0,
    float3 normalL : NORMAL0,
    out float4 oPos : POSITION0,
    out float3 posW : TEXCOORD0,
    out float3 normalW : TEXCOORD1)
{
    posW = mul(float4(posL, 1.0f), World);
    normalW = mul(float4(normalL, 0.0f), WorldInvTrans).xyz;
    // Transform to homogeneous clip space.
    oPos = mul(float4(posL, 1.0f), gWVP);
}

float4 PhongPixelShader(float3 posW : TEXCOORD0,
    float3 normalW : TEXCOORD1) : COLOR
{
    . . .
    lightVecW = normalize(LightPosW - posW);
    ambient = (AmbientMtrl*AmbientLight).rgb;
    s = max(dot(normalW, lightVecW), 0.0f);
    diffuse = s*(DiffMtrl*DiffLight).rgb;
    toEyeW = normalize(EyePosW - posW);
    reflectW = reflect(-lightVecW, normalW);
    t = pow(max(dot(reflectW, toEyeW), 0.0f), SpecPower);
    spec = t*(SpecMtrl*SpecLight).rgb;
    color = ambient + ((diffuse + spec) / A);
    return float4(color, 1.0f)
}
Per-Vertex Shaders

vs.

Per-Pixel Shaders

TwoShadingTeapot
(See Demo in Visual Studio)
Alpha Blending

- Create transparency effect

C# Application

```csharp
graphics.GraphicsDevice.RenderState.AlphaBlendEnable = true;
```

Pixel Shader

```csharp
float4 PixelShaderFunction(VertexShaderOutput input) : COLOR0
{
    PixelShaderOutput pout;

    float4 temp = tex2D(textureSampler, input.TexCoord);
    pout.Color = temp * input.Color;

    return pout.Color;
}
```

- Alpha Blending
- Turn off z-buffering
- No default Backface Culling
- Composite sampled texture color with light intensity
Transparency Example

Blending
(See Demo in Visual Studio)
Light Map
LightMap Example

LightMap
(See Demo in Visual Studio)