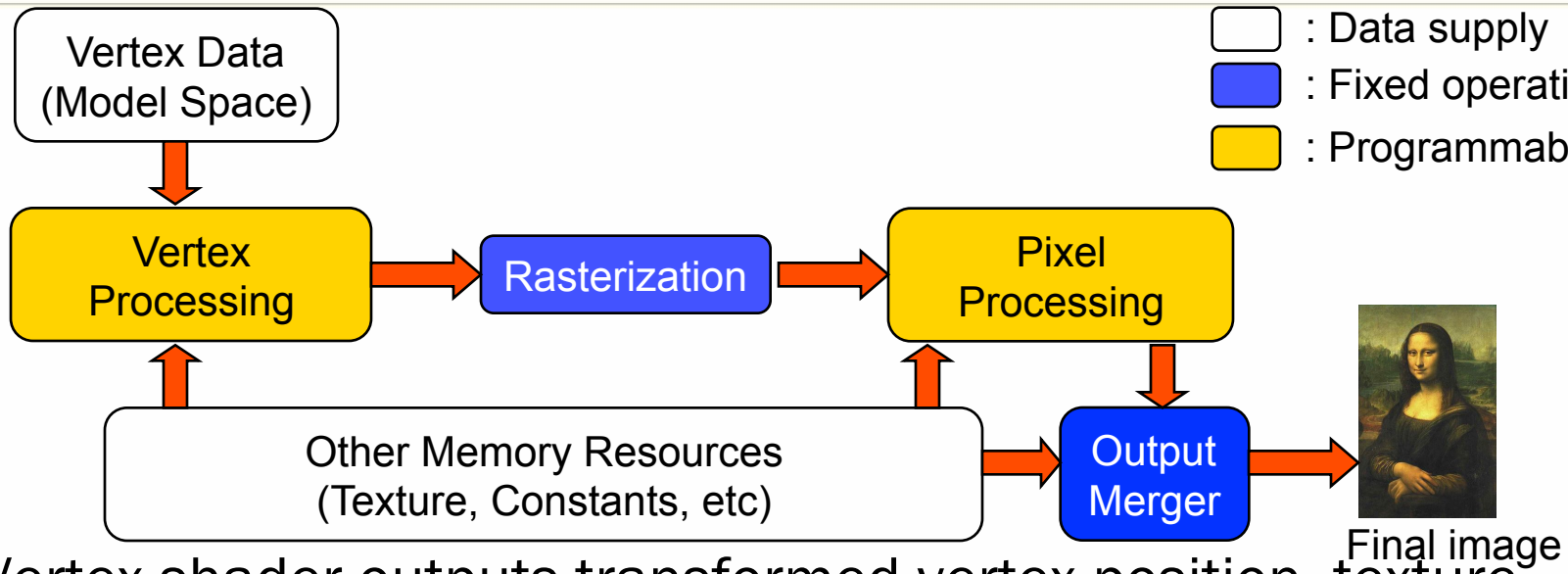
ECE4893A/CS4803MPG: Multicore and GPU Programming for Video Games

Programmable Shaders

Prof. Aaron Lanterman
(Based on slides by Prof. Hsien-Hsin Sean Lee)
School of Electrical and Computer Engineering
Georgia Institute of Technology

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.

From A. Lobao, B. Evangelista, and J.A. Leal de Farias,
"XNA 2.0 Game Programming: From Novice to Professional"

Shader languages

- HLSL/Cg most common
  - Both are compatible
  - No assembly shaders allowed in DirectX 10
- Other alternatives:
  - GLSL (for OpenGL)
  - Sh
  - Assembly?

Motivation for shader languages

- Graphics hardware has become increasingly more powerful
- Programming powerful hardware with assembly code is hard
- GeForce FX supports programs more than 1,000 assembly instructions long
- Programmers need the benefits of a high-level language:
  - Easier programming
  - Easier code reuse
  - Easier debugging

Assembly

```
…
DP3 R0, c[11].xyzx, c[11].xyzx;
RSQ R0, R0.x;
MUL R0, R0.x, c[11].xyzx;
MOV R1, c[3];
MUL R1, R1.x, c[0].xyzx;
DP3 R2, R1.xyzx, R1.xyzx;
RSQ R2, R2.x;
MUL R1, R2.x, R1.xyzx;
ADD R2, R0.xyzx, R1.xyzx;
DP3 R3, R2.xyzx, R2.xyzx;
RSQ R3, R3.x;
MUL R2, R3.x, R2.xyzx;
DP3 R2, R1.xyzx, R2.xyzx;
MAX R2, c[3].z, R2.x;
MOV R2.z, c[3].y;
MOV R2.w, c[3].y;
LIT R2, R2;
...
```

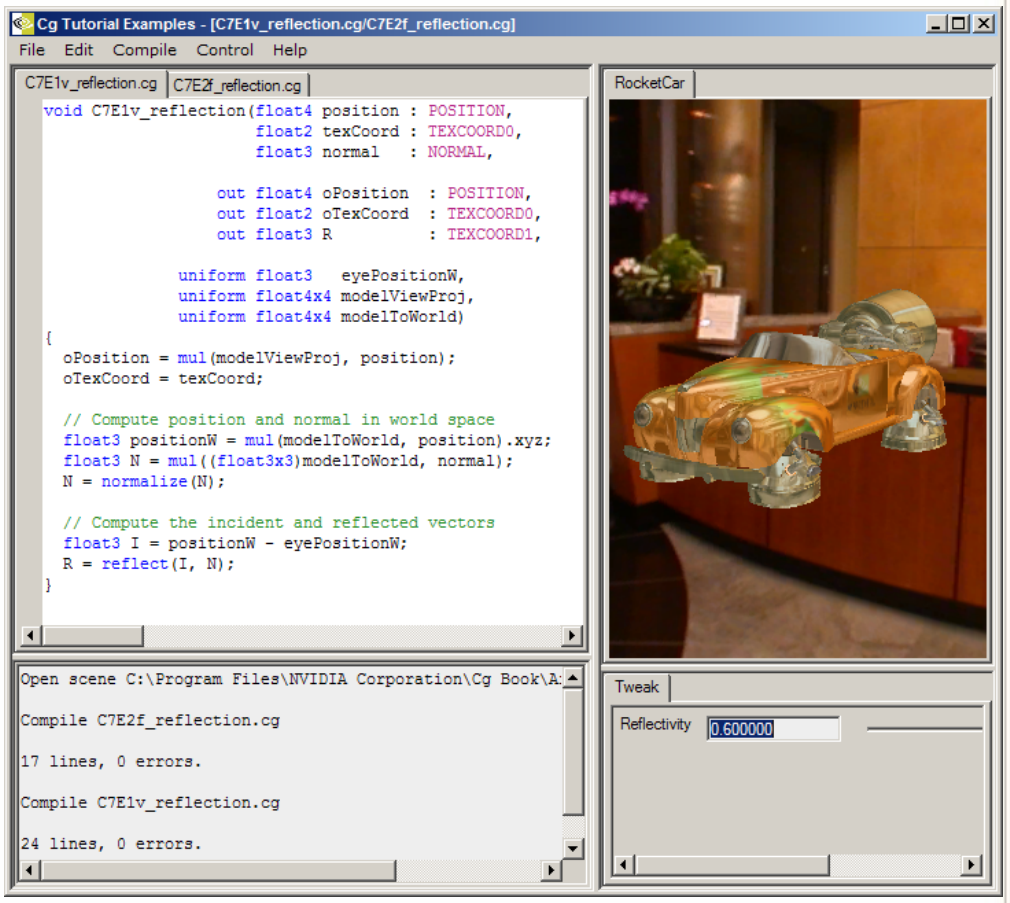
Cg

```
float3 cSpecular = pow(max(0, dot(Nf, H)),
                       phongExp).xxx;
float3 cPlastic = Cd * (cAmbient + cDiffuse) +
                  Cs * cSpecular;
```

From The Cg Tutorial

The Cg Tutorial book (NVIDIA)

- The first book about hardware shading to:
  - Discuss graphics concepts thoroughly
  - Provide complete examples
  - Provide a complete hands-on framework to try and modify the examples, out-of-the-box
- Includes end-of-chapter exercises and further reading
- Now available for free online



From The Cg Tutorial

Shader data

- 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

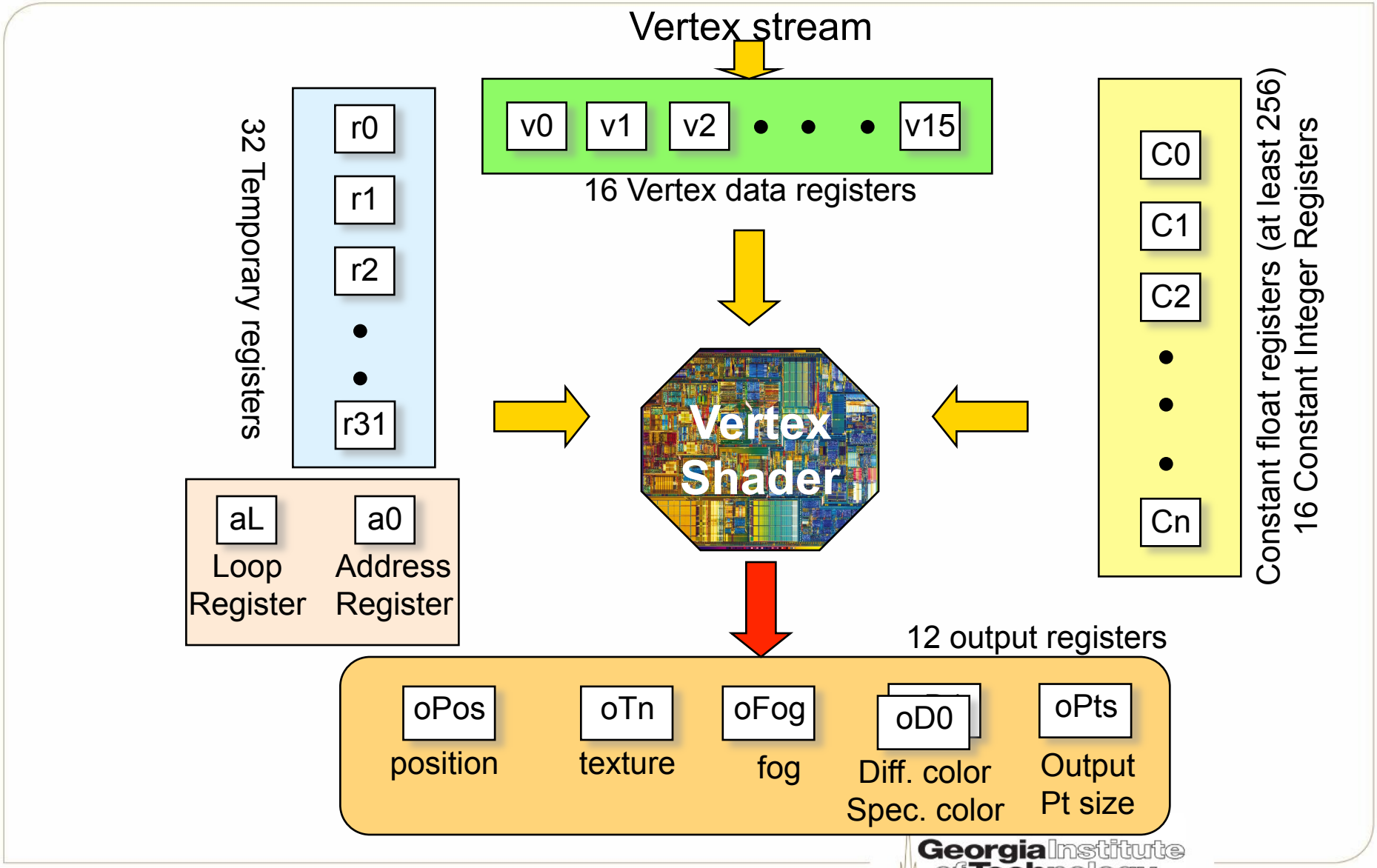
Shader flow control

- Very simple
- No recursion
- Fixed size loops for Shader Model 2.0 or earlier
- Simple if-then-else statements allowed in the latest APIs
- `Texkill` (asm) or `clip` (HLSL) or `discard` (GLSL) allows you to abort a write to a pixel (form of flow control)

Vertex shader

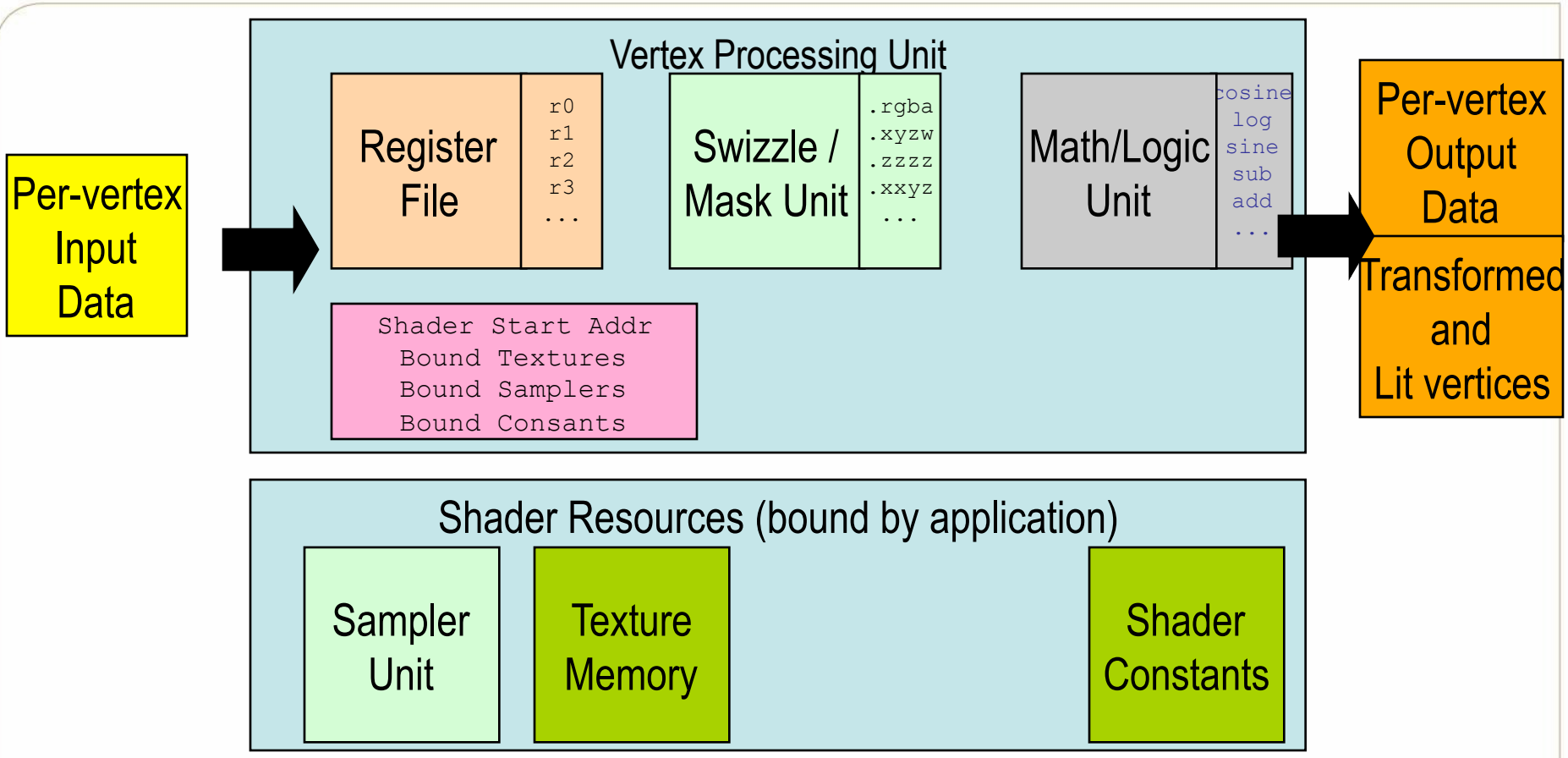
- Transform to clip-space (i.e., screen space)
- Inputs:
  - Common inputs:
    - Vertex position (x, y, z, w)
    - Texture coordinate
    - Constant inputs
    - Can also have fog, color as input, but usually passes them untouched to the pixel shader
  - Output to a pixel (fragment) shader
- Vertex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

Vertex shader data flow (3.0)



Each register is a 4-component vector register except aL

Vertex shader: logical view



Input Data		Architectural State	
Output Data		Control Logic	
State Information			
Memory			

Georgia Institute of Technology

Some uses of vertex shaders

- 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)

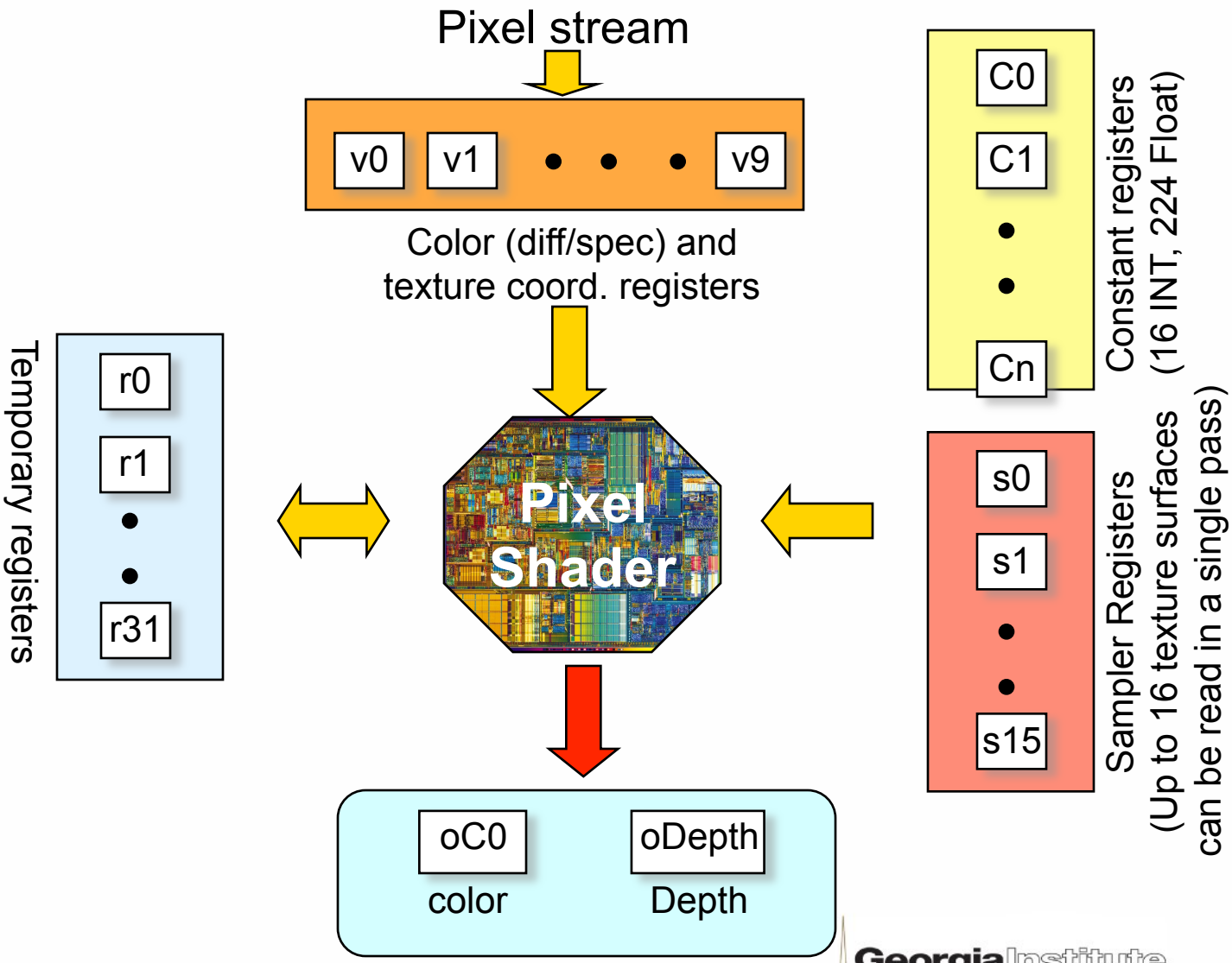
Pixel (or fragment) shader (1)

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

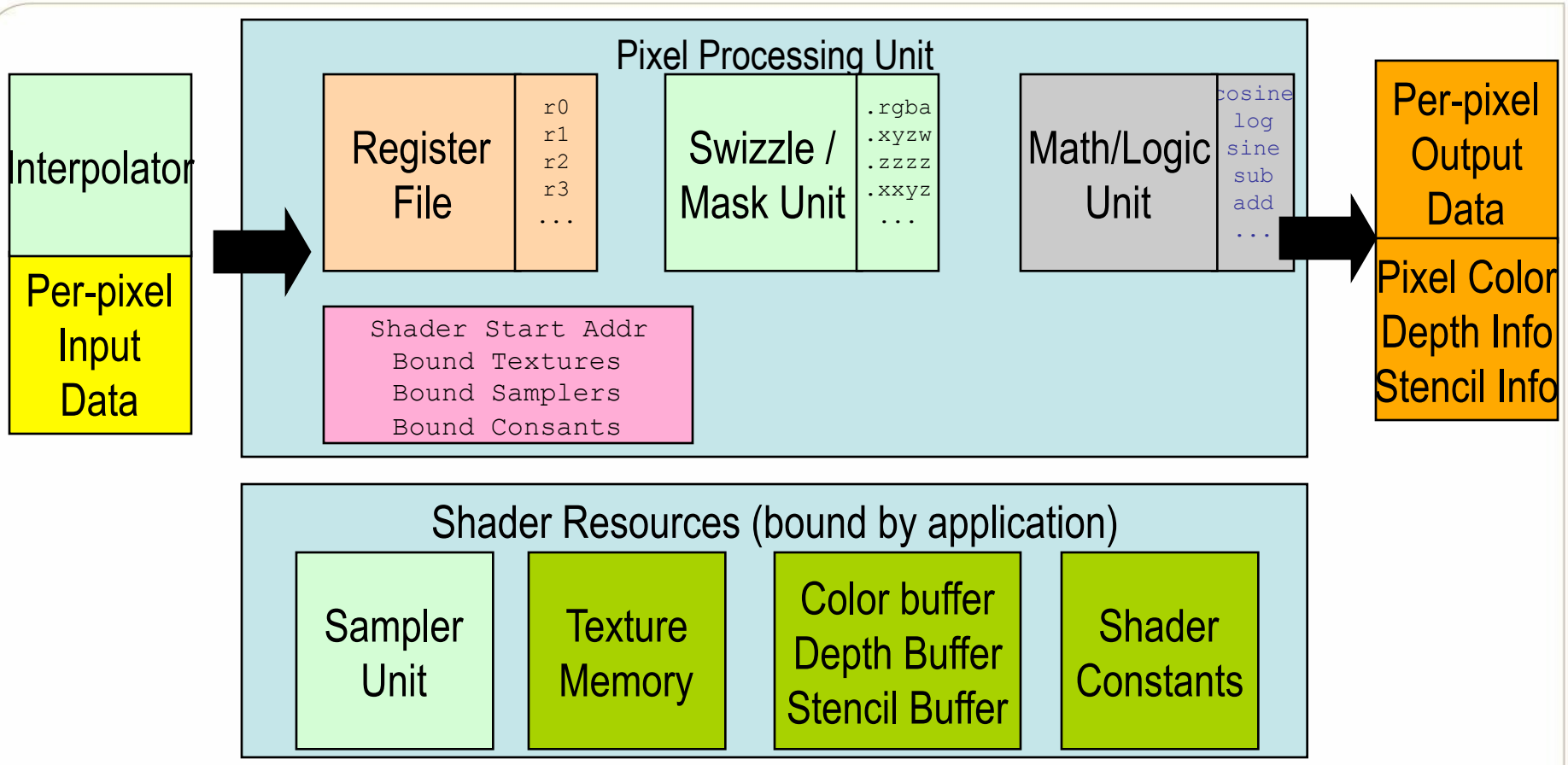
Pixel (or fragment) shader (2)

- Executed once per pixel, hence typically executed many more times than a vertex shader
- It is advantageous to compute stuff on a per-vertex basis to improve performance

Pixel shader data flow (3.0)



Pixel shader: logical view



Input Data		Architectural State	
Output Data		Control Logic	
State Information			
Memory			

Georgia Institute of Technology

Some uses of pixel shaders

- 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

HLSL / Cg

- A C-like language and syntax
- But does **not** have
  - Pointers
  - Dynamic memory allocation
  - Unstructured/complex control structure
    - e.g., goto
    - Recursion (note that functions are inlined)
- Integer & bitwise operations available in VS/PS 4.0, but not previous versions

Uniform vs. 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 at the beginning of the shader code
- Variable input
  - Attributes associated with each vertex or pixel
  - Declared using semantics

The `uniform` type qualifier

- A uniform variable value can come from an external source
  - 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 not in usual C/C++ programming
- A colon and a keyword, e.g.,
  - `MonsterPos : POSITION`
  - `VertexColor : COLOR`
  - `Vertexnormal : NORMAL`
  - `VertexUVcoord : TEXTCOORD0`

- A **glue** that
  - Binds an HLSL program to the rest of the graphics pipeline
  - Connects the semantic variables to the pipeline

A simple vertex shader

```
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;
}
```

Passed from C# apps

Semantics

Reserved data type

Input to Vertex Shader

Structure passed to fragment shader

Adapted from The Cg Tutorial

Georgia Institute of Technology

An alternative simple vertex shader

```
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

Adapted from The Cg Tutorial

A simple pixel (or fragment) shader

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


PixelOutput All_greenPS(float4 color : COLOR)
{
  PixelOutput PSout;

  PSout.color = color;

  return PSout;

}
```

Adapted from The Cg Tutorial

Georgia Institute of Technology

Math operators

- Most commonly used C/C++ operations are supported
- No pointer support or indirection in Cg/HLSL, so no * or ->
- Some ops in Shader Model 4.0 only:
  - Bitwise logic operation (&, ^, |, &=, |=, ^=…)
  - Shift: << , >>, <<=, >>=
  - Modular: %

Standard library function

- To name a few...

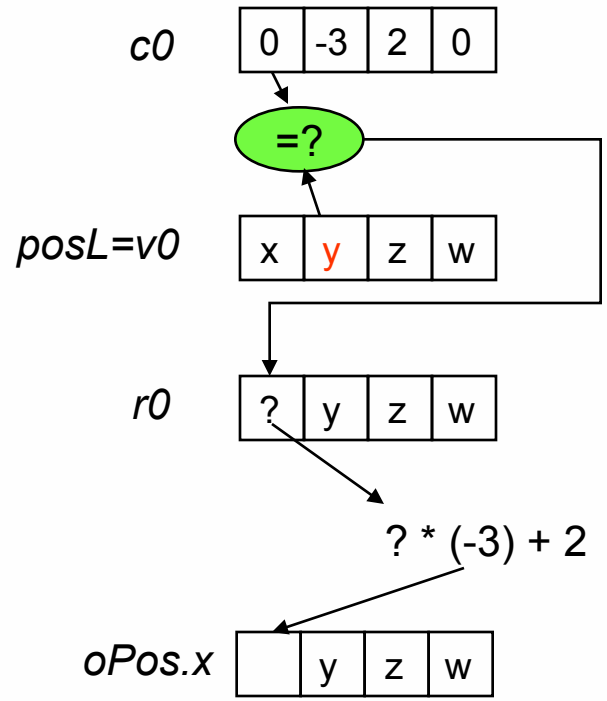
```
dot(a, b)
cross(a, b)
distance(pt1, pt2) : Euclidean distance
lerp(a, b, f) : r = (1-f)*a + f*b
lit(NL, NH, pwr) : for diffuse and specular lighting
mul(M, N)
normalize(v)
reflect(I, N) : calculate reflect vector of ray I
sincos(x, s, c) : calculate sin(x) and cos(x)
```

Flow control (predicating constants)

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

vs_2_0 compiled code

```
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
```



vs_1_1	>= 96	Constant registers
vs_2_0	>=256	Constant registers
vs_3_0	>=256	Constant registers

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();
```

XNA Effect framework

- An "Effect"
  - Encapsulates shader properties
    - E.g., Water modeling & steel modeling have 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

```
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
   }
}
```

Georgia Institute of Technology

Create an effect file

NoLightCubeHLSL - Microsoft Visual Studio

File Edit View Project Build Debug Tools Window Community Help

Debug x86

SimpleEffect.fx | Program.cs | Game1.cs

```
20
21  sampler textureSampler = sampler_state
22  {
23    Texture = <decal>;
24    mipfilter = LINEAR;
25  };
26
27  VertexShaderOutput VertexShaderFunction(VertexShaderInput input)
28  {
29      VertexShaderOutput output;
30
31      output.Position = mul(float4(input.Position, 1.0f), gWVP);
32      output.TexCoord = input.TexCoord;
33
34      return output;
35  }
36
37  float4 PixelShaderFunction(VertexShaderOutput input) : COLOR0
38  {
39      PixelShaderOutput pout;
40
41      pout.color = tex2D(textureSampler, input.TexCoord);
42
43      return pout.color;
44  }
45
46  technique myTech
47  {
48      pass Pass1
49      {
50          VertexShader = compile vs_2_0 VertexShaderFunction();
51          PixelShader = compile ps_2_0 PixelShaderFunction();
52      }
53  }
54
```

Solution Explorer - Solution 'NoLightC...

- Solution 'NoLightCubeHLSL' (1 project)
  - NoLightCubeHLSL
    - Properties
    - References
    - Content
      - References
      - GeorgiaTech.png
      - SimpleEffect.fx
    - Game.ico
    - Game1.cs
    - GameThumbnail.png
    - Program.cs

Output

Show output from: Debug

Error List | Breakpoints | Output | Find Results 1 | Find Symbol Results

Solution E... | Class View | Property ...

Item(s) Saved Ln 38 Col 2 Ch 2 INS

Again, put in the Content Pipeline

Initializing an effect in C# code

```
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

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

foreach (EffectPass pass in myEffect.CurrentTechnique.Passes)
{
       pass.Apply();
       graphics.GraphicsDevice.DrawUserIndexedPrimitives
              (PrimitiveType.TriangleList, vertex, 0, 24,
               triangleListIndices, 0, 12);
}
```

Vertex shader code for texturing

```
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 The Cg Tutorial

Pixel shader code for texturing

```
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 The Cg Tutorial

Sampler objects (in .fx file)

```
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);
}
```

Nearest Point Sampling

Bilinear Filtering

Most expensive;
Alleviate distortion when angle between normal and camera is wide

First XNA/HLSL Example

NoLightCubeHLSL
(See Demo in Visual Studio)

Structure lights and objects

- Make separate classes (C# files) for
  - Light sources
  - Objects

Defining a light source object

```
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;
        }
    }
```

In Lights.cs

Initializing the light source object

```
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

Passing in the light attributes

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

    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);
```

Drawing with the shader

```
foreach (EffectPass pass in cubeEffect.CurrentTechnique.Passes)
{
        pass.Apply();
        graphics.GraphicsDevice.DrawUserIndexedPrimitives
          (PrimitiveType.TriangleList, vertex, 0, 24,
           triangleListIndices, 0, 12);
}
base.Draw(gameTime);
```

Teapot class

```
public Teapot(Game game)
    : base(game)
{

    thisTeapot = game.Content.Load<Model>("teapot");

}

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


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

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

To associate the rendering Effect (.fx)

To associate the transformation matrix

XNA example: vertex shader

```
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;
}
```

Per-Vertex Lighting

XNA example: pixel shader

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

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

    pout.Color = temp*input.Color;

    return pout.Color;
}
```

XNA/HLSL Lighting Example

OneLightCubeHLSL
(See Demo in Visual Studio)

Ambient and Diffuse Light Teapot Example

OneLightTeapotHLSL
(See Demo in Visual Studio)

Many lights

- Use “array” in Shader code

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

uniform extern lightProperty light[2];
```

Multiple lights in the vertex shader

```
for (i=0; i<numLights; i++)
{
    ambient[i] = 0.0f;
    diffuse[i] = 0.0f;
    if (light[i].on != 0) {
        // lighting
        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]);
}
```

Initialize light sources in the game class

On the C# application side

```
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, 1.0f);
    Light[0].D_material = new Vector4(0.412f, 0.412f, 0.412f, 1.0f);
    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, 1.0f);
    Light[1].D_material = new Vector4(0.412f, 0.412f, 0.412f, 1.0f);
    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;
}
```

Multiple lights inside of the draw() call

Link the corresponding array element in the shader code on the C# application side

```
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);
}
```

Two Lights
Teapot Example

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

Morphing using two textures

```
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)

Per-vertex specular computation

```
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;
```

Transform both position and normal for the new “World” locations

Apply the half-way vector for approximating specular component

Per-vertex specular computation (1)

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

Per-vertex specular computation (2)

```
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;
        }
    }
}
```

Per-vertex specular computation (3)

```
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 shading: vertex shader

```
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);
}
```

Per-vertex shading: pixel shader

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

Per-pixel shading: vertex shading

```
PhongVertexShader(float3 posL : POSITION0,
     float3 normalL : NORMAL0,
     out float4 oPos : POSITION,
     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);
}
```

Per-pixel shading: pixel shader

```
float4 PhongPixelShader(float3 posW : TEXCOORD0,
        float3 normalW : TEXCOORD1) : COLOR {
   . . . .
   normalW = normalize(normalW);
   . . . .
   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 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 : POSITION,
             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

Turn off z-buffering

```
graphics.GraphicsDevice.DepthStencilState = DepthStencilState.None;
graphics.GraphicsDevice.BlendState = BlendState.Additive;
graphics.GraphicsDevice.RasterizerState = RasterizerState.CullNone;
```

C# Application

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

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

    return pout.Color;
}
```

Composite sampled texture color with light intensity

Pixel Shader

Transparency Example

Blending

(See Demo in Visual Studio)

Light map in “Dungeon” demo