

ECE 595 / CS 491 / CS 591
**Real-Time Rendering &
 Graphics Hardware**

Pradeep Sen
 Advanced Graphics Lab

Class 9
 February 14, 2007

Pop quiz!

- This quiz will test you on your knowledge of ARB fragment programs

Announcements

- New homework available (due Wednesday)
- GFX Café series starting March 23
- I won't be here on Wednesday (substitute: Joe Kniss)

Last time

- DirectX
- Started talking about high-level shading languages

Today

- More on high-level shading languages
- Specific kinds of shaders

Motivation



curly maple

padauk

walnut

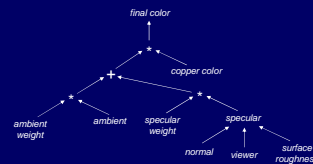
- We want the shader to do several things:
 - Model the surface color $f(s,t)$ (albedo)
 - Deform the surface
 - Model how the light interacts with the surface (glossiness, etc)

Real-time shading languages

- SGI Interactive Shading Language (SGI, 2000)
- Real-time Shading Language (Stanford, 2001)
- Cg (NVIDIA, 2003)
- HLSL (Microsoft, 2003)
 - very similar to Cg
- GLSL (3DLabs, 2003)
 - Included in OpenGL2.0 spec
- Brook, Sh (2005)
 - GPGPU

High-end shading languages

- For high-end rendering, shading languages have been around for 20 years:
 - Shade Trees (Cook 84)



High-end shading languages

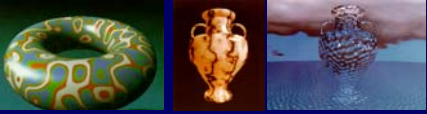
- For high-end rendering, shading languages have been around for 20 years:
 - Shade Trees (Cook 84)



From Cook 84

High-end shading languages

- For high-end rendering, shading languages have been around for 20 years:
 - Shade Trees (Cook 84)
 - Pixel stream programs (Perlin 85)



from Perlin 85


AGL Real-time Rendering & Graphics Hardware Pradeep Sen Class 9 - February 14, 2007

High-end shading languages

- For high-end rendering, shading languages have been around for 20 years:
 - Shade Trees (Cook 84)
 - Pixel stream programs (Perlin 85)
 - RenderMan Shading Language (Hanrahan and Lawson 90)

AGL Real-time Rendering & Graphics Hardware Pradeep Sen Class 9 - February 14, 2007


Renderman Examples



Terminator 2: Judgement Day - Carolco Pictures (1991)

AGL Real-time Rendering & Graphics Hardware Pradeep Sen Class 9 - February 14, 2007

Renderman Examples




Terminator

Jurassic Park - Universal (1993)

AGL Real-time Rendering & Graphics Hardware Pradeep Sen Class 9 - February 14, 2007

Renderman Examples




Terminator

Monsters, Inc. - Pixar (2001)

AGL Real-time Rendering & Graphics Hardware Pradeep Sen Class 9 - February 14, 2007

Renderman Examples



Terminator

Finding Nemo - Pixar (2003)

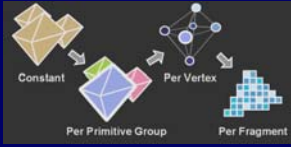
AGL Real-time Rendering & Graphics Hardware Pradeep Sen Class 9 - February 14, 2007

RTSL

- Inspired by the RenderMan shading language
- First shading language to expose capability at both the vertex and fragment levels
- Single program model
 - User does not specify what is a vertex program and what is a fragment program
 - Automatically evaluated computation frequency: constant, per-primitive, per-vertex, per-fragment

AGL Real-time Rendering & Graphics Hardware Pradeep Sen Class 9 - February 14, 2007

Computation Frequencies



Courtesy Kekoa Proudfoot

AGL Real-time Rendering & Graphics Hardware Pradeep Sen Class 9 - February 14, 2007

RTSL shader example


```

surface shader float4
anisotropic_ball (texref anisotex, texref star){
    // generate texture coordinates
    perlight float4 uv = {center(dot(B, E)), center(dot(B, L)),
        0, 1};
    // compute reflection coefficient
    perlight float4 fd = max(dot(N, L), 0);
    perlight float4 fr = fd * texture(anisotex, uv);

    // compute amount of reflected light
    float4 lightcolor = 0.2 * Ca + integrate(Cl * fr);

    // modulate reflected light color
    float4 uv_base = {center(Pobj[2]), center(Pobj[0]),
        0, 1};

    return lightcolor * texture(star, uv_base);
}
  
```



Courtesy Kekoa Proudfoot

AGL Real-time Rendering & Graphics Hardware Pradeep Sen Class 9 - February 14, 2007

Computation Frequency Analysis (constant)

```

surface shader float4
anisotropic_ball (texref anisotex, texref star){

    // generate texture coordinates
    perlight float4 uv = {center(dot(B, E)), center(dot(B, L)),
        0, 1 };


    // compute reflection coefficient
    perlight float4 fd = max(dot(N, L), 0);
    perlight float4 fr = fd * texture(anisotex, uv);

    // compute amount of reflected light
    float4 lightcolor = 0.2 * Ca + integrate(Cl * fr);

    // modulate reflected light color
    float4 uv_base = {center(Pobj[2]), center(Pobj[0]),
        0, 1 };

    return lightcolor * texture(star, uv_base);
}

```



Courtesy Kekoa Proudfoot
Class 9 – February 14, 2007

Computation Frequency Analysis (per-primitive)

```

surface shader float4
anisotropic_ball (texref anisotex, texref star){

    // generate texture coordinates
    perlight float4 uv = {center(dot(B, E)), center(dot(B, L)),
        0, 1 };


    // compute reflection coefficient
    perlight float4 fd = max(dot(N, L), 0);
    perlight float4 fr = fd * texture(anisotex, uv);

    // compute amount of reflected light
    float4 lightcolor = 0.2 * Ca + integrate(Cl * fr);

    // modulate reflected light color
    float4 uv_base = {center(Pobj[2]), center(Pobj[0]),
        0, 1 };

    return lightcolor * texture(star, uv_base);
}

```



Courtesy Kekoa Proudfoot
Class 9 – February 14, 2007

Computation Frequency Analysis (per-vertex)

```

surface shader float4
anisotropic_ball (texref anisotex, texref star){

    // generate texture coordinates
    perlight float4 uv = {center(dot(B, E)), center(dot(B, L)),
        0, 1 };


    // compute reflection coefficient
    perlight float4 fd = max(dot(N, L), 0);
    perlight float4 fr = fd * texture(anisotex, uv);

    // compute amount of reflected light
    float4 lightcolor = 0.2 * Ca + integrate(Cl * fr);

    // modulate reflected light color
    float4 uv_base = {center(Pobj[2]), center(Pobj[0]),
        0, 1 };

    return lightcolor * texture(star, uv_base);
}

```



Courtesy Kekoa Proudfoot
Class 9 – February 14, 2007

Computation Frequency Analysis (per-fragment)

```

surface shader float4
anisotropic_ball (texref anisotex, texref star){

    // generate texture coordinates
    perlight float4 uv = {center(dot(B, E)), center(dot(B, L)),
        0, 1 };


    // compute reflection coefficient
    perlight float4 fd = max(dot(N, L), 0);
    perlight float4 fr = fd * texture(anisotex, uv);

    // compute amount of reflected light
    float4 lightcolor = 0.2 * Ca + integrate(Cl * fr);

    // modulate reflected light color
    float4 uv_base = {center(Pobj[2]), center(Pobj[0]),
        0, 1 };

    return lightcolor * texture(star, uv_base);
}

```



Courtesy Kekoa Proudfoot
Class 9 – February 14, 2007

RTSL Video

AGL Real-time Rendering & Graphics Hardware
Pradeep Sen
Class 9 – February 14, 2007

RDS

- Challenge: as shaders grow larger and more complex, they no longer fit into a single rendering pass on the hardware
- We have to resort to multi-pass rendering in order to process complex shaders
- e.g. render part of the shader, save the result and use it as an input in a subsequent pass where we render the scene again using the rest of the shader

AGL Real-time Rendering & Graphics Hardware
Pradeep Sen
Class 9 – February 14, 2007

RDS

- What is the optimal way to partition shaders for multipass?
- Multipass Partitioning Problem (MPP)
- Recursive Dominator Split (RDS)

AGL Real-time Rendering & Graphics Hardware
Pradeep Sen
Class 9 – February 14, 2007

A word from Carmack (June 2002)

- "...The upcoming high level languages MUST NOT have fixed, queried resource limits if they are going to reach their full potential..."
- "... drivers must have the right and responsibility to multipass arbitrarily complex inputs to hardware with smaller limits."

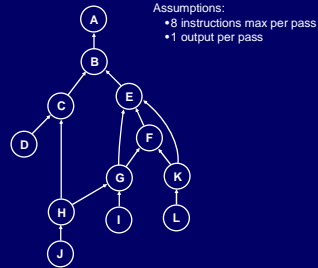
AGL Real-time Rendering & Graphics Hardware
Pradeep Sen
Class 9 – February 14, 2007

Hardware constraints

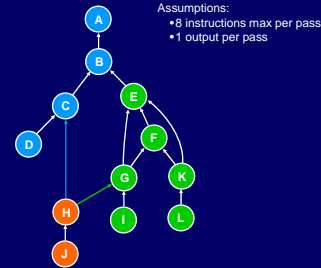
- Fixed number of interpolants
- Fixed number of instructions for vertex or fragment programs
- Fixed number of registers for storing intermediate values
- Fixed number of bound textures
- Fixed depth of texture dependency
- Fixed number of outputs per pass

AGL Real-time Rendering & Graphics Hardware
Pradeep Sen
Class 9 – February 14, 2007

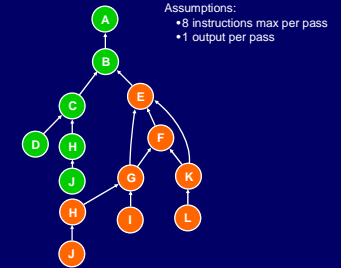
Shaders as DAGs



Shaders as DAGs



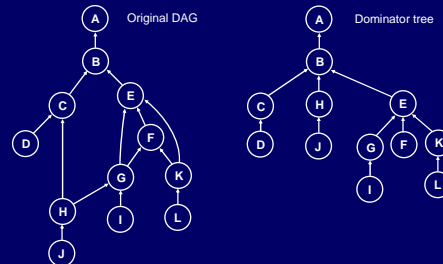
Shaders as DAGs



Things that make the search complicated...

- Computation can be duplicated
- The number of graph cuts is variable
- Limited number of outputs per pass

Dominator tree



Cg

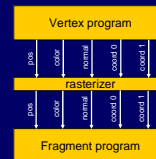
- Not an application-specific shading language
 - Does not explicitly define "surface" or "light" shaders (but you can implement these using interfaces)
 - Different from RenderMan or RTSL
- Based on syntax and philosophy of C
 - Data types and operators have a direct mapping to the hardware
 - Low overhead, minimal management of application data
- Departs from the single program model: users required to write vertex and fragment programs separately

Cg

- Compilation target controlled by user-specified profiles
- Compilation of Cg happens at run-time in the driver

Interpolants

- How do you pass interpolants between vertex/fragment programs?



Interpolant binding

- Bind by position
 - The interpolant's position is consistent from vertex to fragment program
 - Supported by Cg, not HLSL
- Bind by name
 - The vertex program identifies the output interpolant by name. The same name is used to identify the input interpolant in the fragment program.
 - Both Cg and HLSL support a more specific version of bind-by-name where the names are pre-defined. This allows compatibility between different vertex and fragment programs (plug-and-play)

Important differences from C

- No pointers!
 - Current graphics hardware have very limited indirect addressing capability
 - While C blurs the distinction between array types and pointer types, Cg is explicit about the two
- No bitwise operations!
- First class support for vector data types: float4, float4x4
- Swizzling and masking (var.xy)
- Design paradigm: expose what the hardware does best



Real-time Rendering & Graphics Hardware
Pradeep Sen

Class 9 – February 14, 2007

Comparison with C++

- Some notion of constructors, but not user-defined:
- No support for “programming in the large.” Missing classes, templates, namespaces, etc.
- No support for file I/O



Real-time Rendering & Graphics Hardware
Pradeep Sen

Class 9 – February 14, 2007

Cg runtime

- Is composed of two parts:
 - 1) Compiler which compiles the Cg programs using the specified profiles and outputs the appropriate assembly code
 - 2) Interface to the 3D API, which load and binds the Cg programs, passes environment variables in, etc.



Real-time Rendering & Graphics Hardware
Pradeep Sen

Class 9 – February 14, 2007

Simple vertex program in Cg

```
void simpleTransform(float4 objectPosition : POSITION,
float4 color : COLOR,
float4 decalCoord : TEXCOORD0,
float4 lightMapCoord : TEXCOORD1,
out float4 clipPosition : POSITION,
out float4 oColor : COLOR,
out float4 oDecalCoord : TEXCOORD0,
out float4 oLightMapCoord : TEXCOORD1,
uniform float brightness,
uniform float4x4 modelViewProjection) {
clipPosition = mul(modelViewProjection, objectPosition);
oColor = brightness * color;
oDecalCoord = decalCoord;
oLightMapCoord = lightMapCoord;
}
```



Real-time Rendering & Graphics Hardware
Pradeep Sen

Class 9 – February 14, 2007

Compiled version

- When compiled with a vs_1_1 profile (DirectX 8 vertex program)

```
vs.1.1
mov oT0, v7
mov oT1, v8
dp4 oPos.x, c1, v0
dp4 oPos.y, c2, v0
dp4 oPos.z, c3, v0
dp4 oPos.w, c4, v0
mul oD0, c0.x, v5
END
```



Real-time Rendering & Graphics Hardware
Pradeep Sen

Class 9 – February 14, 2007

Simple fragment program in Cg

```
float4 brightLightMapDecal(float4 color : COLOR,
float4 decalCoord : TEXCOORD0,
float4 lightMapCoord : TEXCOORD1,
uniform sampler2D decal,
uniform sampler2D lightMap) : COLOR {
float4 d = tex2Dproj(decal, decalCoord);
float4 lm = tex2Dproj(lightMap, lightMapCoord);
return 2.0 * color * d * lm;
}
```



Real-time Rendering & Graphics Hardware
Pradeep Sen

Class 9 – February 14, 2007