GPU Shading and Rendering
Course 37
August 2, 2005


# Cg: the API-independent Language for GPU Programming

Mark J. Kilgard
Graphics Software Engineer
**NVIDIA Corporation**

# General Purpose Programming Language Bizarro World


- Linux Intel/AMD x86
  - You must use **Java** and only Java to program!
- MacOS X PowerPC
  - You must use **Pascal** and only Pascal to program!
- Solaris SPARC
  - You must use **C++** and only C++ to program!

# General Purpose Programming Language Bizarro World


- Why bizarre?
  - Because high-level languages should provide platform abstraction
  - Successful high-level languages should not tie programs to platforms
- High-level languages should provide
  - Abstraction
  - Ease of authoring and maintenance
  - Platform portability
  - **NOT** platform-tying

# 3D Graphics Content


- Consider 3D model formats
  - Direct3D-only format?
    - **No**
- Consider Texture and image formats
  - OpenGL-only formats?
    - **No**
- Consider your shading language format
  - Uh, ok.  What a minute!

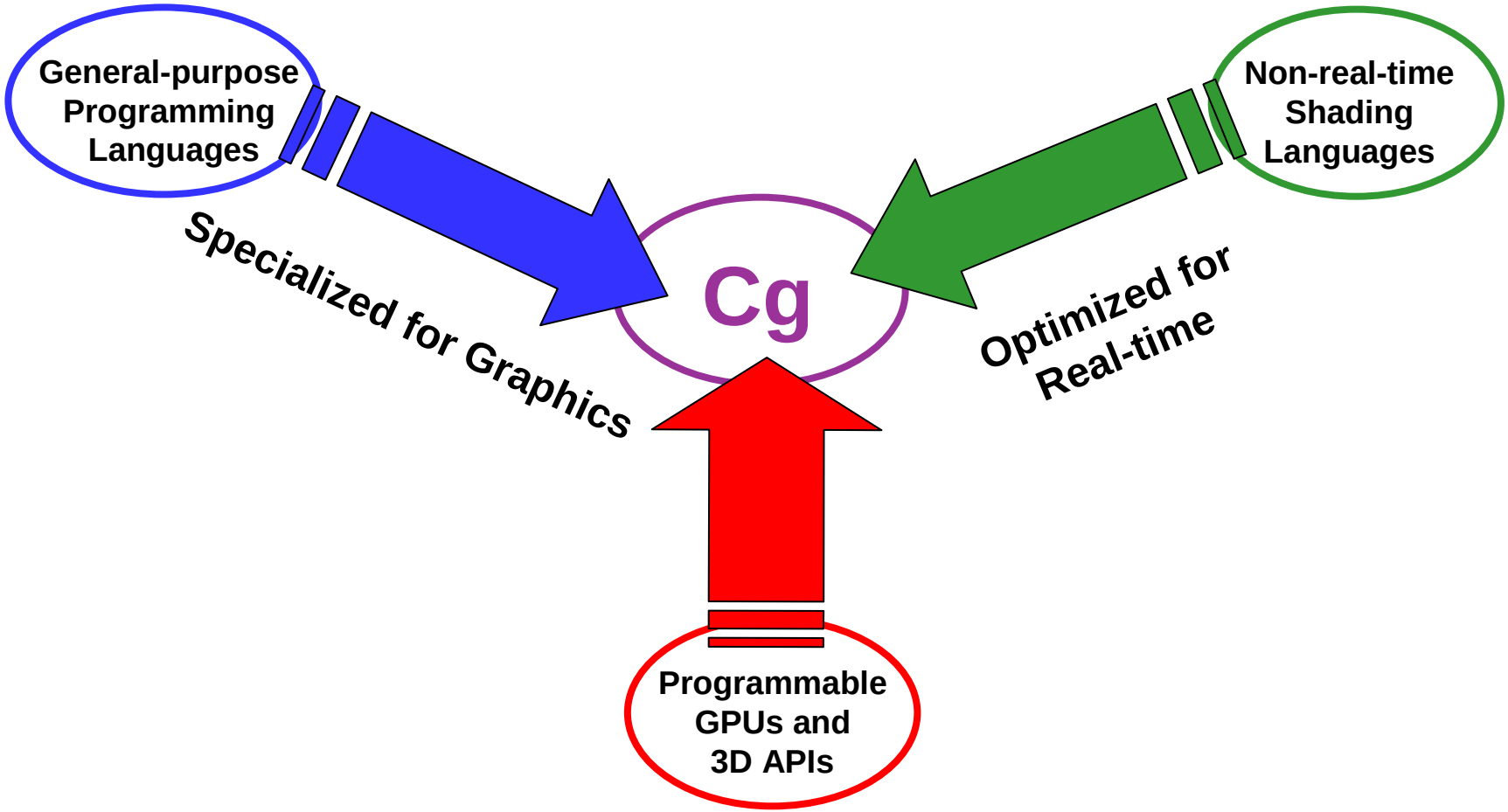
# Hardware Shading Language Standard Practice


- Current API-centric practice
  - Use OpenGL Shading Language (GLSL) just for OpenGL rendering
  - Use High-Level Shading Language (HLSL) just for Direct3D
  - Use GLSL-derivative just for OpenGL ES 2.0
- Rather bizarre
  - Why tie your 3D content to one platform?

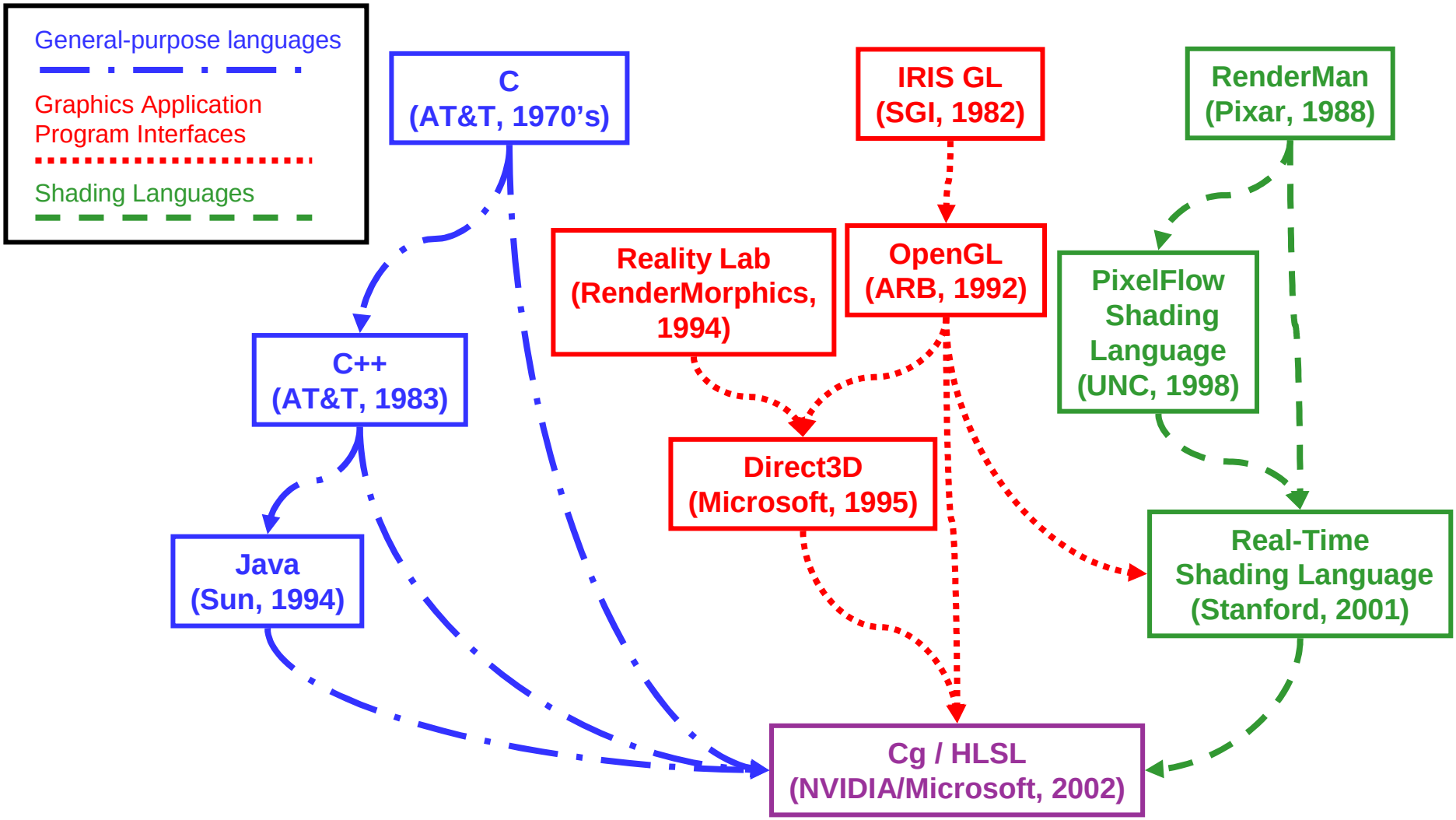
# What is Cg?


- Hardware Shading language
  - High-level, looks like C
  - API-independent
    - GLSL tied to OpenGL; HLSL tied to Direct3D
  - Platform-independent
    - Cg works on PlayStation3, ATI, NVIDIA, Linux, MacOS X, Windows, etc.
- More generally
  - GPU programming language
  - For writing constrained kernels with a data-flow

# Technologies and Forces Leading to Cg

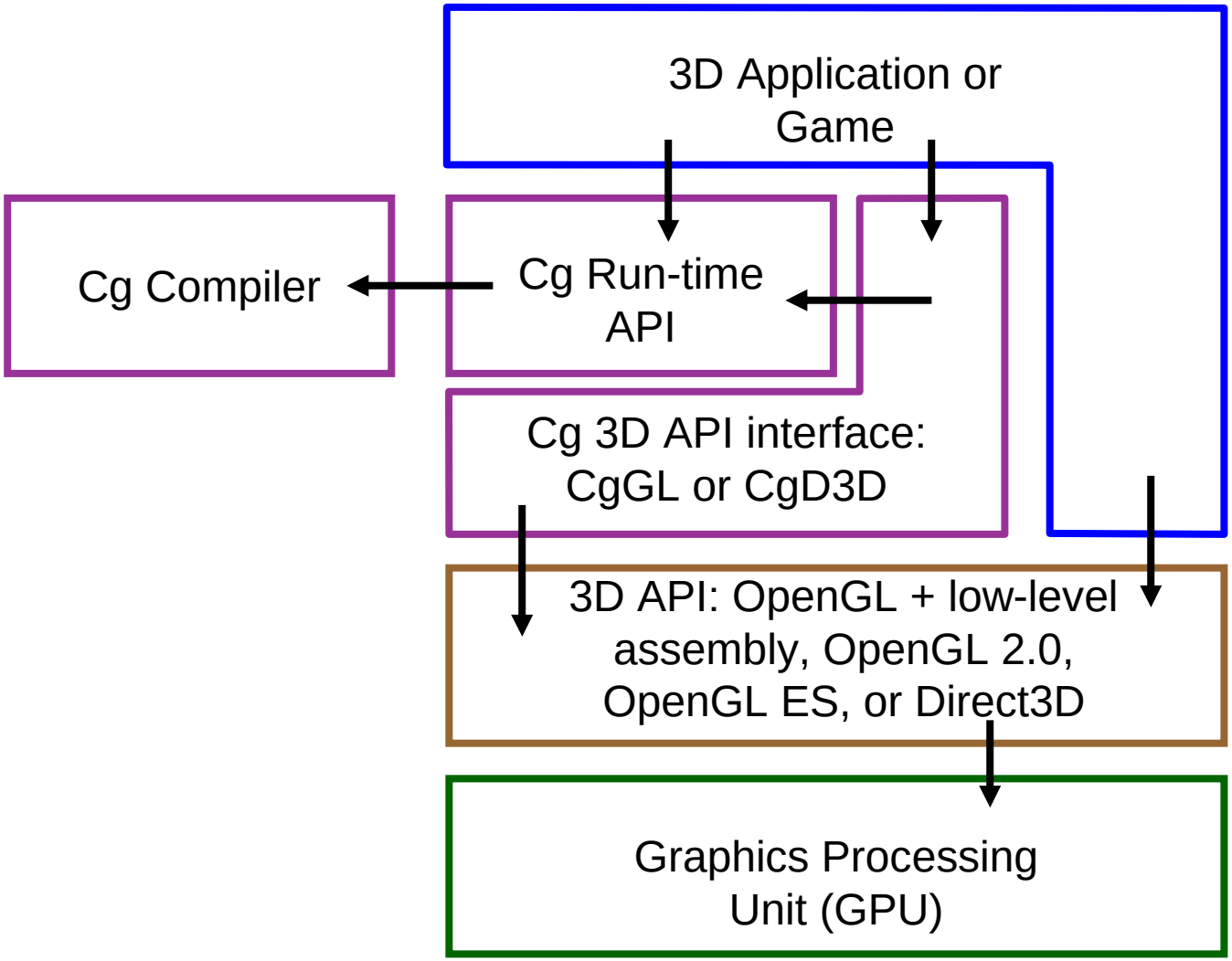


# Cg Evolution

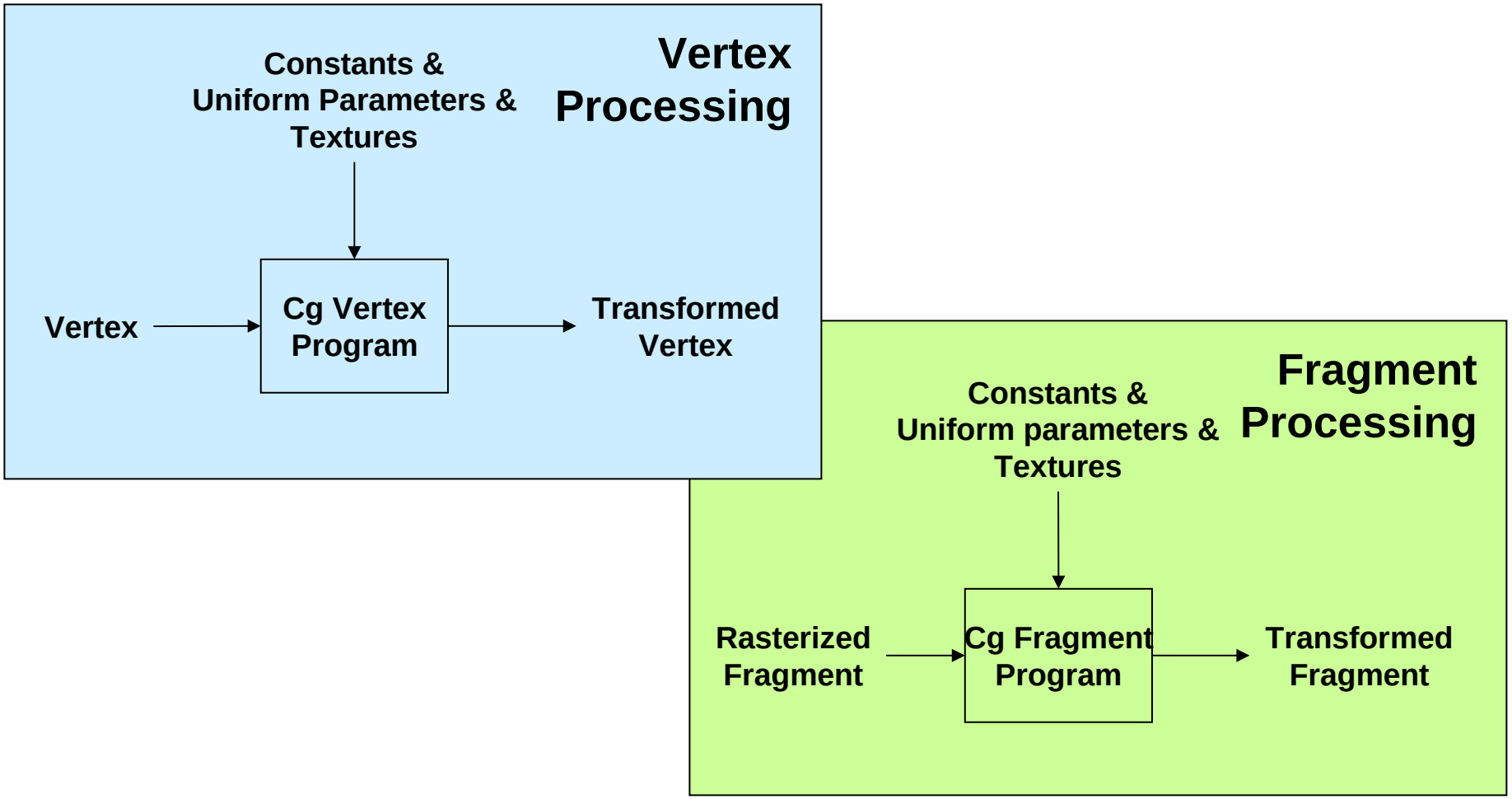


# Cg Ecosystem

SIGGRAPH2005



# Cg Vertex and Fragment Programs



# Cg Vertex Program Example


```
void arbvp1 torus(float2 parametric : POSITION,

                  out float4 position       : POSITION,
                  out float2 oTexCoord      : TEXCOORD0,
                  out float3 lightDirection : TEXCOORD1,
                  out float3 halfAngle      : TEXCOORD2,

              uniform float3 lightPosition,  // Object-space
              uniform float3 eyePosition,    // Object-space
              uniform float4x4 modelViewProj,
              uniform float2 torusInfo)
{
  const float pi2 = 6.28318530;  // 2 times Pi
  float M = torusInfo[0];
  float N = torusInfo[1];
  oTexCoord = parametric * float2(-6, 2);
  float cosS, sinS;
  sincos(pi2 * parametric.x, sinS, cosS);
  float cosT, sinT;
  sincos(pi2 * parametric.y, sinT, cosT);
  float3 torusPosition = float3((M + N * cosT) * cosS,
                                (M + N * cosT) * sinS,
                                N * sinT);
  position = mul(modelViewProj, float4(torusPosition, 1));
  float3 dPds = float3(-sinS*(M+N*cosT), cosS*(M+N*cosT), 0);
  float3 norm_dPds = normalize(dPds);
  float3 normal = float3(cosS * cosT, sinS * cosT, sinT);
  float3 dPdt = cross(normal, norm_dPds);
  float3x3 rotation = float3x3(norm_dPds,
                               dPdt,
                               normal);
  float3 eyeDirection = eyePosition - torusPosition;
  lightDirection = lightPosition - torusPosition;
  lightDirection = mul(rotation, lightDirection);
  eyeDirection = mul(rotation, eyeDirection);
  halfAngle = normalize(normalize(lightDirection) +
                        normalize(eyeDirection));
}
```

// Stretch texture coordinates counterclockwise over torus to repeat normal map in 6 by 2 pattern

// Compute torus position from its parametric equation

// Compute per-vertex rotation matrix

// Rotate object-space vectors to texture space

# Cg Fragment Program Example


```
float3 expand(float3 v) { return (v-0.5)*2; }

void arbfp1 specSurf(float2 normalMapTexCoord : TEXCOORD0,
                     float3 lightDirection    : TEXCOORD1,
                     float3 halfAngle         : TEXCOORD2,

                 out float4 color : COLOR,

             uniform float  ambient,
             uniform float4 LMd, // Light-material diffuse
             uniform float4 LMs, // Light-material specular
             uniform sampler2D   normalMap,
             uniform samplerCUBE normalizeCube,
             uniform samplerCUBE normalizeCube2)
{
  float3 normalTex = tex2D(normalMap, normalMapTexCoord).xyz;
  float3 normal = expand(normalTex);
  float3 normLightDirTex = texCUBE(normalizeCube,
                                   lightDirection).xyz;
  float3 normLightDir = expand(normLightDirTex);
  float3 normHalfAngleTex = texCUBE(normalizeCube2,
                                    halfAngle).xyz;
  float3 normHalfAngle = expand(normHalfAngleTex);

  float diffuse = saturate(dot(normal, normLightDir));
  float specular = saturate(dot(normal, normHalfAngle));
  float specular2 = specular*specular;
  float specular4 = specular2*specular2;
  float specular8 = specular4*specular4;

  color = LMd*(ambient+diffuse) + LMs*specular8;
}
```

// Fetch and expand range-compressed normal

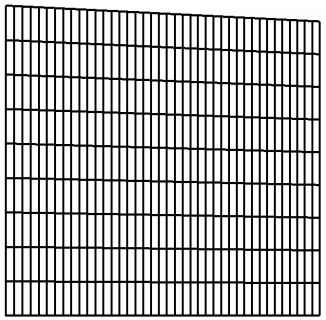
// Fetch and expand normalized light vector

// Fetch and expand normalized half-angle vector

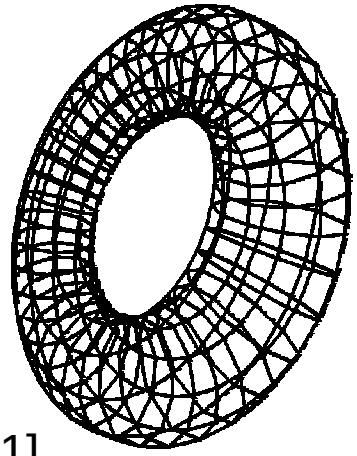
// Compute diffuse and specular lighting dot products

// Successive multiplies to raise specular to 8th power

# What Do These Cg Programs Do?



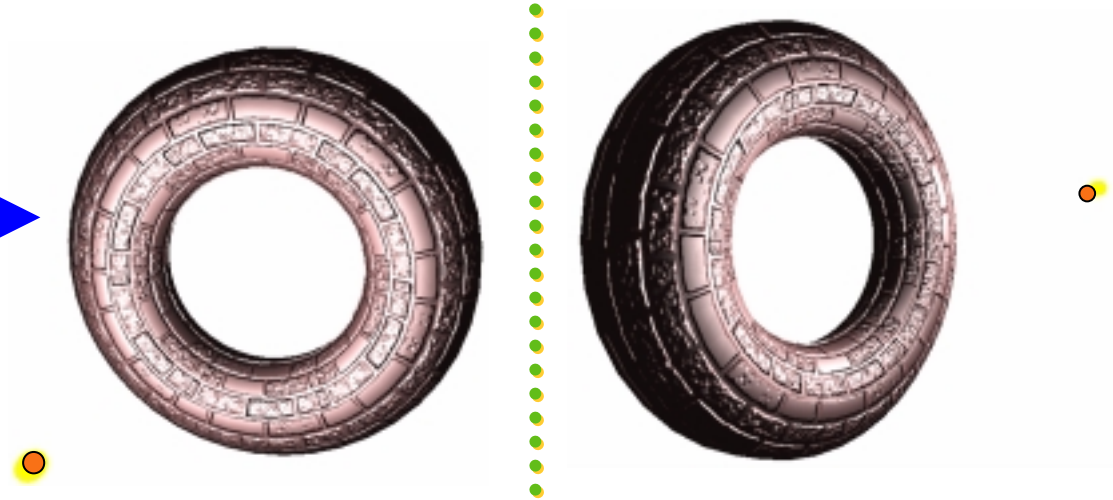
2D grid over (s,t)∈ [0,1]



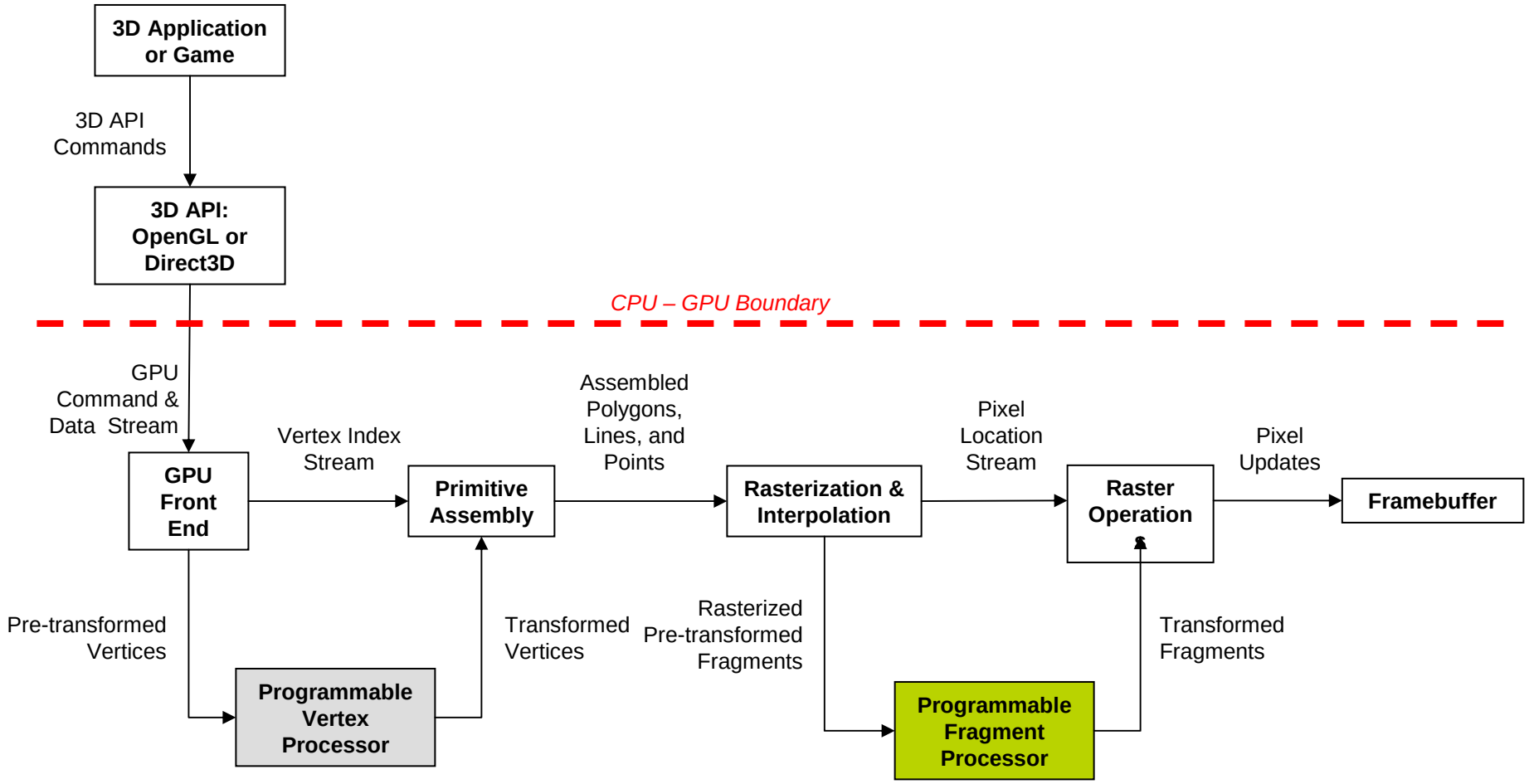
Tessellated torus

← Vertex program

Fragment program →



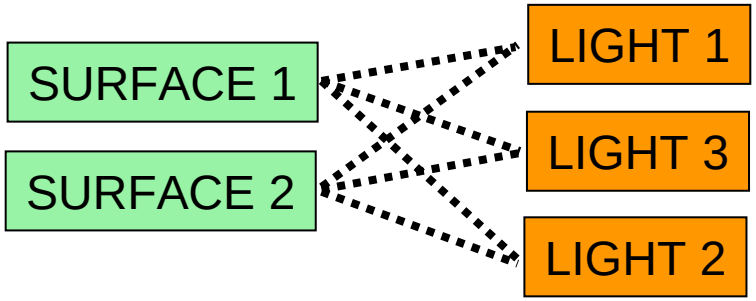
# Cg in the Programmable Graphics Hardware Pipeline



# Continuing Cg Language Advancement


- Additional functionality
  - Interfaces
  - Unsigned arrays
- Provides better abstraction and fit to graphics usage
  - Example: surface / light separability
- Future for Cg: more C++ like

# General Mechanism for Surface / Light Separability




- Convenient to put **surface** and **light** code in different module
  - Decouples surface selection from light selection
  - Proven usefulness: Classic OpenGL, RenderMan, etc.
  - Lost in low-level shader assembly, HLSL, GLSL, and original Cg
- RenderMan uses a specialized mechanism
- Cg 1.2 introduced a general-purpose mechanism
  - More flexible
  - Follows C-like philosophy

# Light / Surface Example
# First: Declare an interface


```
// Declare interface to lights
interface Light {
    float3 direction(float3 from);
    float4 illuminate(float3 p, out float3 lv);
};
```

- Mechanism adapted from Java, C#

# Second: Declare a light that implements interface


```
// Declare object type for point lights
struct PointLight : Light {
    float3 pos, color;
    float3 direction(float3 p) { return pos - p; }
    float3 illuminate(float3 p, out float3 lv) {
        lv = normalize(direction(p));
        return color;
    }
};
```

- Declare an object that implements interface

# Third: Define surface that calls interface


```
// Main program (surface shader)
float4 main(appin IN, out float4 COLOR,
                    uniform Light lights[ ]) {
    ...
    for (int i=0; i < lights.length; i++) { // for each light
        Cl = lights[i].illuminate(IN.pos, L); // get dir/color
        color += Cl * Plastic(texcolor, L, Nn, In, 30); // apply
        }
    COUT = color;
}
```

- Call objects via the interface type

# Third: Define surface that calls interface

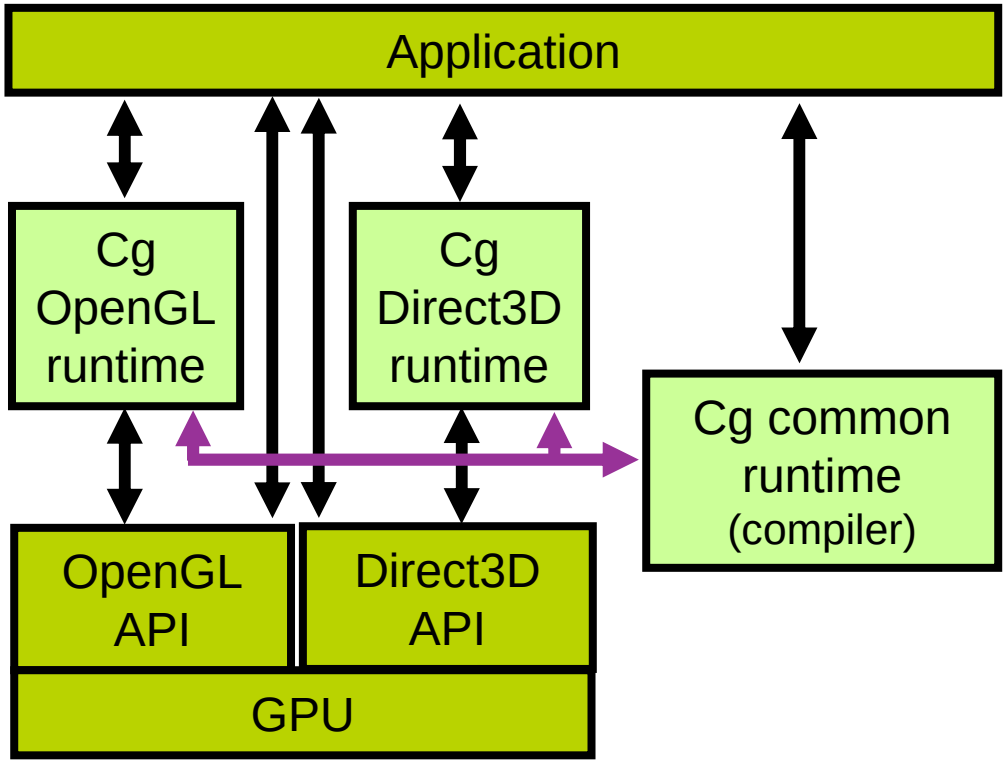

```
// Main program (surface shader)
float4 main(appin IN, out float4 COLOR,
            uniform Light lights[ ]) {
    ...
    for (int i=0; i < lights.length; i++) { // for each light
        Cl = lights[i].illuminate(IN.pos, L); // get dir/color
        color += Cl * Plastic(texcolor, L, Nn, In, 30); // apply
        }
    COUT = color;
}
```

Unsigned array length

- Call objects via the interface type

# Modular or Monolithic Architecture?


- Cg system is modular
- Provides access to assembly-level interfaces
- GLSL made different choices
- Compile **off-line**, or at **run-time**



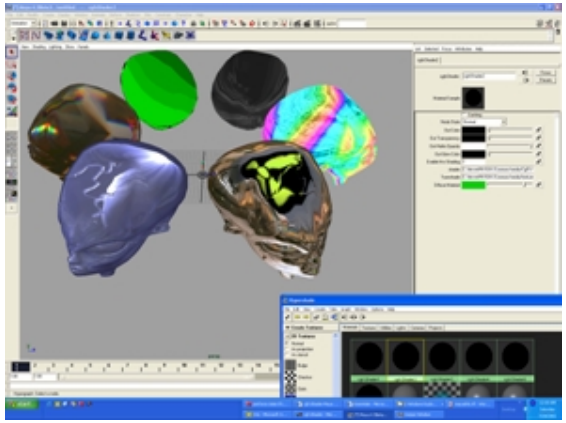

**The Cg system manages Cg programs AND the flow of data on which they operate**

# CgFX

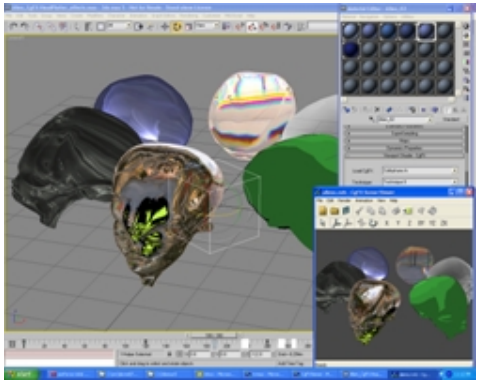

- Shaders don’t live in isolation
  - Tied to other 3D rendering state
- CgFX is meta-shading file format
  - Cg programs
  - 3D rendering state
  - Techniques and passes
  - Annotations & semantic
  - Multiple implementations
  - Texture shaders written in Cg for procedural texture generation
- Another meta-shading example: Collada

# Some Digital Content Creation Applications Using CgFX



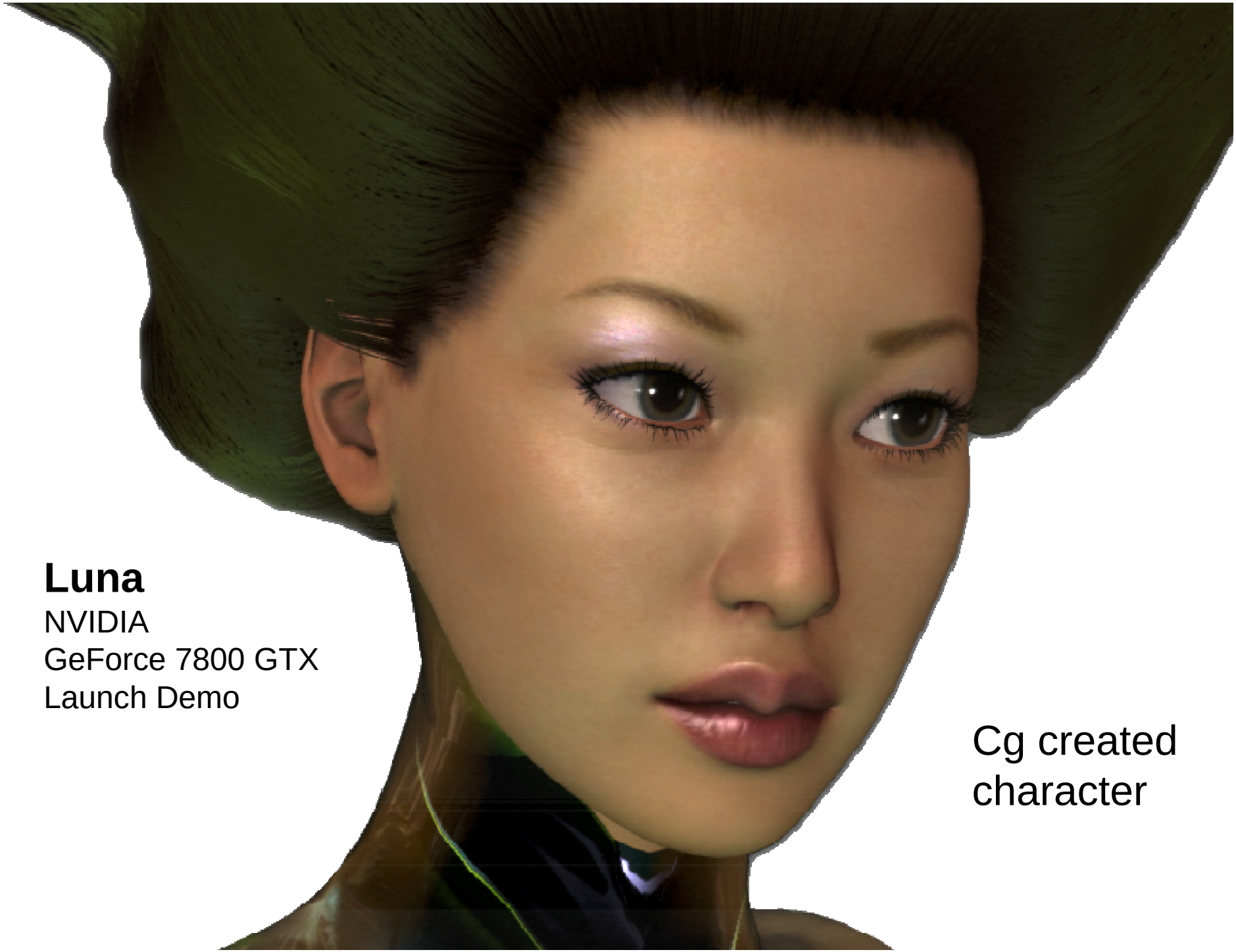


3ds max™



SOFTIMAGE|XSI™


# Cg-based Shader Rich Platform Independent Character Development



**Luna**
NVIDIA
GeForce 7800 GTX
Launch Demo

Cg created character


NVIDIA.
© 2005 NVIDIA Corporation. All Rights Reserved.

**Mad Mod Mike**
NVIDIA
GeForce 7800 GTX
Launch Demo

Cg created character

# Conclusions


- Who should use Cg?
  - **If you are concerned about shader portability**
    - Cg offers API-independent shaders: Windows, Linux, Mac, PlayStation3
  - **If you want access latest language features**
    - Examples:  interfaces and un-sized arrays
  - **If you want access to latest hardware features**
    - Example: GeForce 7800
- Key points
  - API-independent high-level shading language
  - Off-line and real-time compilation models
  - Metafiles: CgFX and Collada offer Cg combined with state