The Cg Runtime

Cyril Zeller
Cg Pipeline

Graphics programs are written in Cg ...

... and compiled to ...

... low-level assembly code ...

... that runs on any GPU compatible with DirectX or OpenGL
Compiling Offline

At Development Time

- Cg program source code
- Shader program assembly code

\[\text{Cg Compiler}\]

\[\text{Shader Compiler (nvasm.exe, psa.exe)}\]

At Runtime

- At initialization:
  - Load assembly or binary program

- For every frame:
  - Load program parameters to hardware registers
  - Set rendering state
  - Load geometry
  - Render

Compiler Offline

// Diffuse lighting
//
float d = dot(normalize(frag.N), normalize(frag.L));
if (d < 0)
  d = 0;
c = d*tex2D(t,frag.uv)*diffuse;
...

DP3 r0.x, f[TEX0], f[TEX0];
RSQ r0.x, r0.x;
MUL r0, r0.x, f[TEX0];
DP3 r1.x, f[TEX1], f[TEX1];
RSQ r1.x, r1.x;
MUL r1, r1.x, f[TEX1];
DP3 r0, r0, r1;
MAX r0.x, r0.x, 1.0;
MUL r0, r0.x, DIFFUSE;
TEX r1, f[TEX1], 0, 2D;
MUL r0, r0, r1;
...
// Diffuse lighting
// float d = dot(normalize(frag.N), normalize(frag.L));
if (d < 0)
  d = 0;
   c = d*tex2D(t, frag.uv)*diffuse;
...
Pros and Cons of Runtime Compilation

Pros:
- Future compatibility: The application does not need to change to benefit from future compilers (future optimizations, future hardware)
- Easy parameter management

Cons:
- Loading takes more time because of compilation
- Cannot tweak the result of the compilation
Core Cg Runtime: cgCreateProgram
Direct3D Cg Runtime: cgD3D9LoadProgram
OpenGL Cg Runtime: cgGLLoadProgram
Core Cg Runtime

Does *not* make any 3D API call

Allows you to:

- **Create a context:** cgCreateContext()
- **Compile a program for a given profile** (vs_2_0, vs_2_x, ps_2_0, ps_2_x, arbvp1, arbvp1, vs_1_1, ps_1_1, fp20, etc...): cgCreateProgram(), cgGetProgramString(), etc...
- **Manage program parameters:**
  - Iterate through the parameters: cgGetFirstParameter(), cgGetNextParameter(), etc...
  - Get parameter information: type, semantic, register, ...
- **Handle errors:** cgGetError(), cgSetErrorCallback(), etc...
Direct3D Cg Runtime: Minimal Interface

Does *not* make any Direct3D call

Allows you to translate the information obtained through the Core Runtime to Direct3D data structures, so that you can:

- Create a Direct3D vertex declaration from the Cg program:
  `cgD3D9GetVertexDeclaration()`

- Validate a Direct3D vertex declaration against the Cg program:
  `cgD3D9ValidateVertexDeclaration()`

and create a Direct3D vertex or pixel shader from the Cg program
Direct3D Cg Runtime: Expanded Interface

Makes the necessary Direct3D calls for you

Allows you to:

- Pass the Direct3D device: `cgD3D9SetDevice()`
- Load a program into Direct3D: `cgD3D9LoadProgram()`
- Tell Direct3D to render with it: `cgD3D9BindProgram()`
- Set parameter values: `cgD3D9SetUniform()`, `cgD3D9SetUniformArray()`, `cgD3D9SetTexture()`, etc...
- Output debug information by using the Cg Runtime debug DLL
OpenGL Cg Runtime

Makes the necessary OpenGL calls for you

Allows you to:

- Load a program into OpenGL: `cgGLLoadProgram()`
- Enable a profile: `cgGLEnableProfile()`
- Tell OpenGL to render with it: `cgGLBindProgram()`
- Set parameter values: `cgGLSetParameter{1234}{fd}{v}()`, `cgGLSetParameterArray{1234}{fd}()`, `cgGLSetTextureParameter()`, etc...
Learning the Runtime

The documentation is in:

- Windows: C:\Program Files\NVIDIA Corporation\Cg\docs
- Linux: /usr/share/doc/Cg

It includes:

- A Using the Cg Runtime Library chapter from the Cg User’s Manual
- Manual pages for every function

Source code examples are in:

- Windows: C:\Program Files\NVIDIA Corporation\Cg\examples
- Linux: /usr/local/Cg/examples

They include self-contained Direct3D and OpenGL examples
Questions, comments, feedback?

- Cyril Zeller, czeller@nvidia.com
- For Cg support
  - Go to developer.nvidia.com/Cg
  - Go to www.cgshaders.org
  - Email cgsupport@nvidia.com