A Whole New World with Cg

Graphics Program
Written in Cg
“C” for Graphics

Compiled & Optimized

Low Level, Graphics
“Assembly Code”
Cg is a C-like language, modified for GPUs

- Syntax, operators, functions from C
- Conditionals and flow control
- Particularly suitable for GPUs:
  - Expresses data flow of the pipeline/stream architecture of GPUs (e.g. vertex-to-pixel)
  - Vector and matrix operations
  - Supports hardware data types for maximum performance
  - Exposes GPU functions for convenience and speed:
    - Intrinsic: (mul, dot, sqrt...)
    - Built-in: extremely useful and GPU optimized math, utility and geometric functions (noise, mix, reflect, sin...)
  - Language reserves keywords to support future hardware implementations (e.g. switch, case...)
  - Compiler uses hardware profiles to subset Cg as required for particular hardware feature sets
NVIDIA Cg Usage

Three ways:

Cg Runtime – a thin API:
- ASCII .Cg shader compiled at runtime
- Convenient interface for setting shader parameters and constants

Command line compiler generates text file output:
- DX / OpenGL – vertex and pixel shader files
- Tweak the ASM yourself
- Generates comments on program params & registers

CgFX
- Effect framework with render states
Integrating Cg

Options (not mutually exclusive):

CgFX
- Manages whole rendering process
- Handles render states – cross API support
- Convenient exposure of tweakables & artist controls

Cg Shaders
- semantics (was #pragma bind) directives to match your C++ and other custom hardware shaders
- Bind textures/parameters to specific HW registers

Cg Runtime
- Thin API to compile on demand at runtime
- Optimizes & munges .Cg for range of target HW
Flexible Adoption Path

- Can use system for just fragment programs
  - Define a connector to specify how you’ll supply data to these programs
- Can use system for just vertex programs
  - Define connectors to specify how you’ll supply data to these programs; and what they have to output.
- Can use system with older OpenGL applications
  - Classical glVertex(), glNormal() can still be used
  - Use OpenGL matrix tracking to provide modelview matrix to shading program.
  - But, must load program & supply light state
Mix and Match Any Method

Vertex processing

- Fixed function
- or-
- Hand-written ASM
- or-
- Compiled Cg
- or-
- Hand-optimized Cg ASM

Fragment processing

- Fixed function
- or-
- Hand-written ASM
- or-
- Compiled Cg
- or-
- Hand-optimized Cg ASM
Using the Cg Compiler

Application Development

Cg program source code

Cg Compiler

Shader program assembly code

Shader Compiler

Shader Binary

Your Application

1) Load/bind program
2) Specify program parameters
3) Specify vertex inputs
4) Render

// Diffuse lighting
float d = dot(normalize(frag.N),
            normalize(frag.L));
if (d < 0)
  d = 0;
c = d*f4tex2D(t, frag.uv)*diffuse;
...
Using the Cg Runtime

Application Development

Cg program source code

//
// Diffuse lighting
//
float d = dot(normalize(frag.N),
normalize(frag.L));
if (d < 0)
  d = 0;
c = d*f4tex2D(t, frag.uv)*diffuse;
...

Your Application

Cg Runtime
1) Load/bind program
2) Specify parameters
3) Specify vertex inputs
4) Render

3) Specify vertex inputs
And now for the details...
// Diffuse lighting
//
float d = dot(normalize(frag.N), normalize(frag.L));
if (d < 0)
    d = 0;
c = d * f4tex2D(t, frag.uv) * diffuse;
...
Data types

- float = 32-bit IEEE floating point
- half = 16-bit IEEE-like floating point
- fixed = 12-bit fixed [-2,2) clamping *(OpenGL only)*
- bool = Boolean
- sampler* = Handle to a texture sampler
Array / vector / matrix declarations

- Declare vectors (up to length 4) and matrices (up to size 4x4) using built-in data types:

  ```
  float4   mycolor;
  float3x3 mymatrix;
  ```

- Declare more general arrays exactly as in C:

  ```
  float  lightpower[4];
  ```

- But, arrays are first-class types, not pointers

- Implementations may subset array capabilities to match HW restrictions
Extend standard arithmetic to vectors and matrices

- Component-wise + - * / for vectors
- Dot product
  - `dot(v1, v2);` // returns a scalar
- Matrix multiplications:
  - Assuming `float4x4 M` and `float4 v`
  - `matrix-vector: mul(M, v);` // returns a vector
  - `vector-matrix: mul(v, M);` // returns a vector
  - `matrix-matrix: mul(M, N);` // returns a matrix
New vector operators

- Swizzle operator extracts elements from vector
  \[ a = b.xxyy; \]
- Vector constructor builds vector
  \[ a = \text{float4}(1.0, 0.0, 0.0, 1.0); \]
“Profiles” Define Specific HW Behavior

Public NVIDIA Cg Compiler has three NV2X profiles:
- DX8 Vertex Shader (vs1.1)
- DX8 Pixel Shader (ps1.1)
- OpenGL Vertex Program (currently based on NV_vertex_program, will move to ARB_vertex_program)

Newest NVIDIA Cg Compiler currently has two NV30 profiles:
- Vertex program (vp2.0)
- Fragment Program (vp1.0)

DX9 vertex/pixel shader profile support forthcoming

Vertex profiles:
- No “half” or “fixed” data type
- No texture functions – It’s a vertex program!

Fragment/pixel profiles:
- No “for” or “while” loops (unless they’re unrollable)
- etc.
Other profile limitations for NV30

- No pointers – not supported by HW
- Function parameters are passed by value/result
  - not by reference as in C++
  - use `out` or `inout` to declare output parameter
  - aliased parameters are written in order
- No unions or bit-fields
- No `int` data type
Cg Summary

- C-like language – expressive and efficient
- HW data types
- Vector and matrix operations
- Write separate vertex and fragment programs
- Connectors enable mix & match of programs by defining data flows
- Will be supported on any DX9 hardware
- Will support future HW (beyond NV30/DX9)
Brushed Metal

- Procedural texture
- Anisotropic lighting
Melting Ice

- Procedural, animating texture
- Bumped environment map
Toon & Fur

Toon rendering without textures
Antialiasing
Great silhouettes without overdarkening

Volume fur using ray marching
Shell approach without shells
Can be self-shadowing
Vegetation & Thin Film

Translucence
Backlighting

Example of custom lighting
Simulates iridescence
Questions?

http://www.nvidia.com/Cg/
http://www.cgshaders.org/
kbjorke@nvidia.com