Cg Toolkit

Cg 1.2

Release Notes

Cg Toolkit Release Notes

The Cg Toolkit allows developers to write and run Cg programs using a wide variety of hardware platforms and graphics APIs.

Originally released in December 2002, the Toolkit supports 14 different DirectX and OpenGL profile targets. It provides a compiler for the Cg language, runtime libraries for use with the OpenGL and DirectX graphics APIs, runtime libraries for CgFX, example applications, and extensive documentation.

This 1.2 release of Cg introduces several significant new features:

- Interfaces, a language construct that facilitates the creation of general, reconfigurable Cg programs, are now supported by the Cg compiler and runtime.
- Unsized arrays are now supported in the Cg compiler and runtime
- Parameter instances may be created and shared between multiple programs via the Cg runtime.
- Parameters may be marked as compile-time constants via the Cg runtime, leading to more efficient compiled code

Note that some of this new functionality is currently only supported when using the CgGL OpenGL runtime library. The Direct3D-specific Cg runtime libraries currently do not support shared parameters, for example.

An Addendum to the Cg Users' Manual covering the new language and runtime features is included in this release. In addition, documentation for the new runtime entry points may be found in the docs/manuals directory. Example source code demonstrating the use of some of these new features may be found in the examples/interfaces_ogl directory.

The Cg 1.2 core, CgGL, and CgD3D runtime libraries are backward compatible with those of the Cg 1.1 release, although some low-level implementation details have changed. The vast majority of applications compiled under Cg 1.1 will work using the Cg 1.2 runtime libraries without the need for recompilation.

The CgFX API has been substantially modified to provide better compatibility and better support for OpenGL profiles. See the CgFX Overview document for more details. The CgFX runtime libraries are not backward-compatible with previous releases of CgFX.

Cg is available for a wide variety of hardware and OS platforms. Please visit the NVIDIA Cg website at developer.nvidia.com/Cg for complete availability and compatibility information.

Please report any bugs, issues, and feedback to NVIDIA by email at cgsupport@nvidia.com. We will expeditiously address any reported problems.

Supported Profiles and Platforms

The Cg compiler currently supports the following hardware profiles:

OpenGL

- arbvp1 (ARB_vertex_program 1.0)
- arbfp1 (ARB_fragment_program 1.0)
- vp30 (NV_vertex_program 2.0)
- fp30 (NV_fragment_program 1.0)
- vp20 (NV_vertex_program 1.0)
- fp20 (NV_register_combiners and NV_texture_shader)

DirectX 8 & 9

- vs_1_1 (Vertex Shader 1.1)
- ps_1_1, ps_1_2 and ps_1_3 (Pixel Shader 1.1, 1.2, 1.3)

DirectX 9

- vs_2_0 and vs_2_x (Vertex Shader 2.0 and Extended VS 2.0)
- ps_2_0, and ps_2_x/ps_2_a (Pixel Shader PS 2.0 and Extended PS 2.0)

The Cg Runtime libraries include:

- The Cg core runtime library for managing parameters and loading programs
- The CgGL runtime library for OpenGL based applications
- The CgD3D8 runtime library for DirectX 8 based applications
- The CgD3D9 runtime library for DirectX 9 based applications

The CgFX Runtime libraries include:

- The CgFX core/parser runtime library
- The CgFXGL library for OpenGL-based applications
- The CgFXD3D8 library for Direct3D 8-based applications
- The CgFXD3D9 library for Direct3D 9-based applications

Improvements & Bug Fixes

New features

- A new “MaxTexIndirections” profile option is now supported under the arbfp1 profile. If the indicated maximum is less than the maximum allowed number of texture instructions, the Cg compiler will attempt to minimize the number of texture indirections in the generated assembly program.
- Support for interfaces, a language feature that greatly simplifies the creation of general, reconfigurable Cg programs, has been added to the compiler and core runtime.
- Structures may contain methods, and may optionally implement a single interface.
- Program parameter instances may be created via runtime and shared between multiple programs.
- A parameter may be “connected” to another, which causes it to inherit its value and variability from other parameters. This feature provides “copy by reference” parameter assignment and value setting.
- Parameter variability may be changed via the runtime, including marking uniforms as compile-time constants (literals), allowing for more targeted and efficient compiled programs.
- The core runtime now includes entry points for directly setting parameters.
- The CgGL runtime can now automatically enable texture parameters (by making calls to `cgGLEnableTextureParameter()` when a program is bound. This behavior is disabled by default, and may be toggled via the `cgGLSetManageTextureParameters()` entry point.
- Many new core Cg API entry points have been added to support these new features. See the Cg User Manual Addendum and on-line documentation for more details.
- The CgFX API has been changed substantially. Please see the CgFX Overview document for details.

Improvements

- Compilation time for longer programs has been reduced.
- The compiler now performs more aggressive constant folding and constant propagation. This often results in shorter, more efficient programs, especially in cases where parameter variability is set to CG_LITERAL via `cgSetParameterVariability()`

Removed features

- The original Cg core runtime API, previously deprecated in the Cg 1.1 release, is no longer supported.

Bug Fixes

- Problems with the order of arguments to `lerp()` have been fixed.
- ps2.X assembly language constants are now printed in a higher-precision format.
- Reads of uninitialized values in vp2.X/vp30 assembly programs have been fixed.
- Problems accessing string annotations in CgFX using GetValue() have been fixed.
- Issues with negation range analysis in ps1.X profiles have been fixed.
- Fixed issue wherein incorrect texture unit could be referenced when using parameter instances.
- `cgGLSetParameterPointer()` now takes a const pointer argument
- Fixed problem wherein compiler could 'successfully' compile some programs that used more than the maximum number of temporary registers.
- Fixed problem with incorrect swizzles being generated when e.g. dot products were interspersed with vector ops.
- Fixed issues with certain kinds of conditions in vertex programs.
- Fixed several issues with conditional matrix assignment
- Fixed bug wherein unrecognized function parameter qualifiers were silently ignored.
- Fixed scoping-related problems with identically-named variables
- Many smaller fixes and improvements

Known issues

Known runtime issues

- The CgD3D runtimes do not work in conjunction with the `cgSetParameter*()` or `cgConnectParameter()` core entry points. As a result, many of the new Cg 1.2 features, including parameter sharing and the use of top-level interface parameters, are effectively not supported when using the CgD3D runtimes.
- The `cgIsParameterReferenced()` entry point sometimes returns true even if a parameter may not be referenced in the final compiled output.
- Setting a parameter that is declared but never referenced can sometimes cause the runtime to return an error.

- The 1.2 Cg core runtime currently consumes more memory than the 1.1 runtime.

Known compiler issues

- Determining the length of a multidimensional array via "`a[].length`" is currently not supported. Instead, the syntax "`a[0].length`" must be used.
- When using interfaces or unsized arrays, the compiler sometimes emits spurious warnings about uninitialized variables.
- The compiler does not currently optimize instructions involving matrix parameters whose variability is set to `CG_LITERAL` via the Cg core runtime.
- The # and ## preprocessor macro operators are not supported
- Some error and warning messages are less clear than we would like them to be. Some of the issues to be aware of are:
  - reported line numbers do not match source code lines when standard library functions are being used,
  - in some cases, errors are not reported in the order they appear in the program,
  - errors are not reported when constants are out of range for untyped constants.
- Side effects in conditional (`?:`) and logical expressions (&& and `||`) are always evaluated, regardless of the condition; currently warnings are not always issued.
- Only one return statement is allowed per function.
- Return statements in if/for blocks are not supported.
- `while` loops, loops with multiple induction variables, and loops that use a `==` conditional test are not currently unrolled. In these cases, compilation errors will result under profiles that do not support dynamic branching.
- All matrices are assumed to be row-major. Currently, column-major matrices are not supported.
- At most one binding semantic per uniform variable is supported by the compiler. Multiple profile-specific binding semantics per uniform variable are not supported.
- The `%` operator is not supported. Integer division is not fully emulated, and is implemented as floating point division.
- Conditional assignments to arrays and matrices do not work in all cases.

Known profile-specific issues:

- Directly accessing the OpenGL state structure is not yet supported under the ARB fragment program (arbfp1) profile. This limitation can be somewhat inefficiently overcome by setting explicit uniform parameters to the OpenGL state in the application.
- Some third-party graphics drivers use a conservative method for computing the number of 'indirect' or 'dependent' texture reads in a compiled program. The Cg

compiler currently does not calculate the number of dependent texture reads in the same way; and as a result, some drivers will sometimes fail to load compiled Cg programs under the more recent fragment program profiles (e.g., arbfp1, ps_2_*).

- Writing to multiple color outputs is not yet supported under the ps_2_* profiles.
- Because the underlying hardware support for the fp20 and ps_1_* profiles is quite limited and inflexible, it is not always possible to compile seemingly simple Cg programs under these profiles. For more details on these limitations, please see the NV_register_combiners, NV_texture_shader OpenGL extension specifications, or the DirectX PixelShader 1.* specifications.
- The FOG varying input semantic is not yet supported under the fp20 profile.

NVIDIA Corporation
2701 San Tomas Expressway
Santa Clara, CA 95050
www.nvidia.com