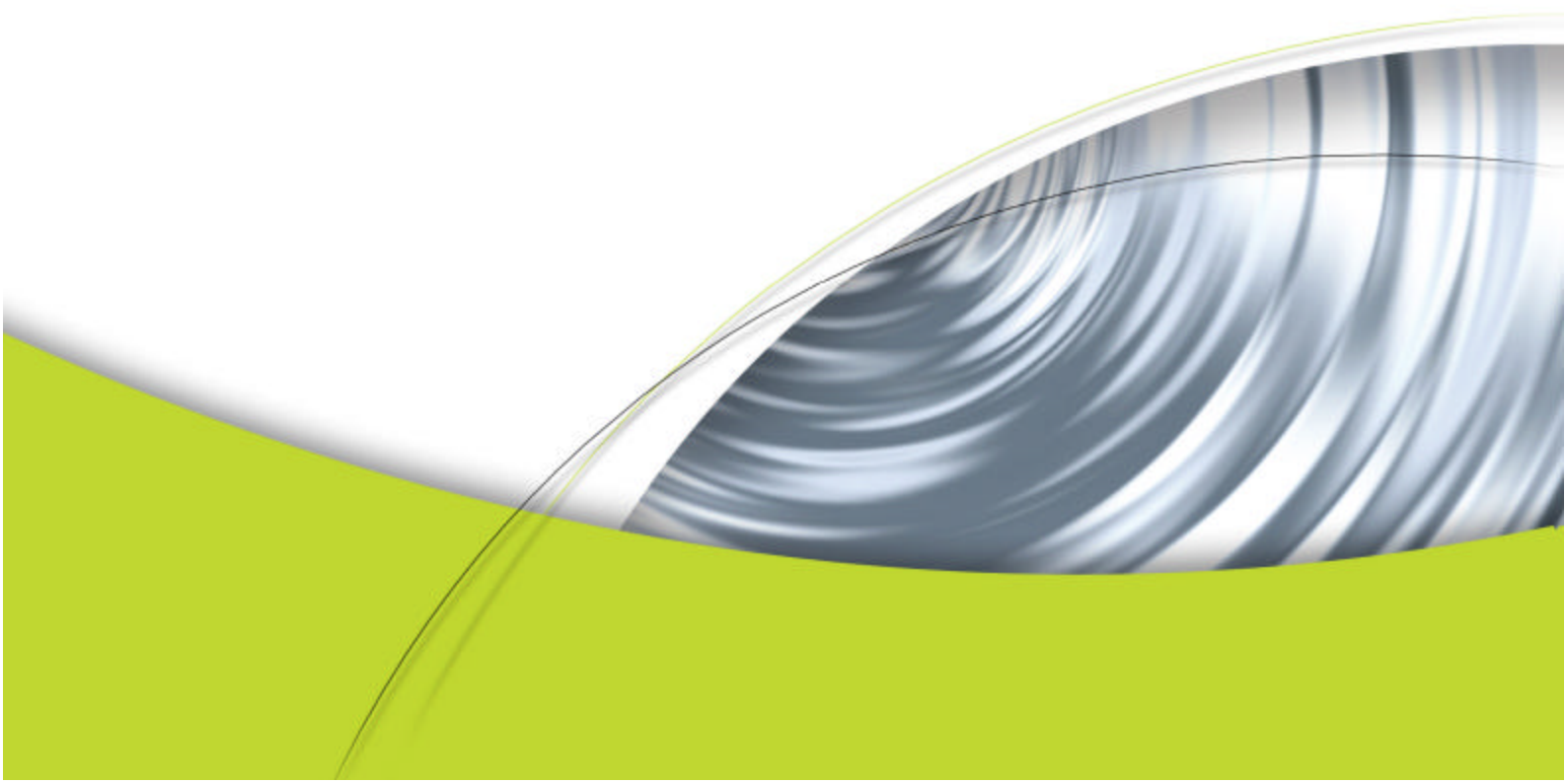

Cg Toolkit

Cg 1.3
Release Notes

December 2004

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 now supports over 20 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 1.3 release of Cg introduces several significant new features:

- New 'vp40' profile, that enables texture sampling from within vertex programs
- New 'fp40' profile, that provides a robust branching model in fragment programs, and support for output to multiple draw buffers ('MRTs')
- Support for writing more than one color output (i.e. 'MRTs') in the arbfp1 and ps_2* profiles
- New semantics to access OpenGL fixed-function state vectors from within ARB_vertex_program and ARB_fragment_program
- New '-fastprecision' option for arbfp*, fp30, and fp40 profiles, to use reduced precision storage (fp16) when appropriate.
- (New in Cg 1.2.1) The 'arbfp1' profile will attempt to generate code differently if the 'MaxTexIndirections' option is set to some non-zero value.

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. We have also not yet provided Direct3D profiles for Shader Model 3.0.

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.

A change was made to the cgD3D9GetOptimalOptions( ) function, which changes its prototype and return value. Other than this change, all Cg 1.2 programs should work with Cg 1.3 without the need to recompile the program. Applications which use the cgD3D9GetOptimalOptions( ) must be modified when moving to Cg 1.3. Please see the "Entry point changes" section, below, for more details.

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)
- vp40 (ARB_vertex_program + NV_vertex_program3 option)
- vp30 (NV_vertex_program 2.0)
- fp40 (ARB_fragment_program + NV_fragment_program2 option)
- 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

- New 'vp40' profile, that enables texture sampling (through tex2D( ), texCUBE( ), etc.) from within vertex programs. This profile generates code dependent upon the NV_vertex_program3 OpenGL extension.
- New 'fp40' profile, that provides:
  - Branching within fragment programs
  - Output of more than one color (i.e. 'MRTs') from fragment programs
  - FACE semantic to determine the front- or back-facingness of the primitive from which the fragments are created

  The 'fp40' profile generates code dependent upon the NV_fragment_program2 OpenGL extension.
- New semantics to access OpenGL fixed-function state vectors from within ARB_vertex_program and ARB_fragment_program.
- Support in the arbfp1 and ps_2* profiles for output to more than one draw buffer (i.e. 'MRTs')
- (New in Cg 1.2.1) The 'arbfp1' profile will attempt to generate code differently if the 'MaxTexIndirections' option is set to some non-zero value.

New feature: fp40 profile: branching

The branching support in ‘fp40’ allows some ‘if’ statements and looping constructs to be implemented with branching. In profiles such as fp30, conditional execution of code was always implemented with predicated instructions, and loops were always unrolled.

In the GeForce 6800 GPU, there is a cost associated with executing a branch in the fragment shading engine. As such, it is possible that the cost of the branch will out-weigh the savings from skipping over a block of conditionally-executed code, or of executing an unrolled loop. (Please refer to the NVIDIA developer web site for more information about the performance of this and other NVIDIA GPUs.) The fp40 profile, therefore, provides the following options to control whether the compiler should emit branches or conditionally-executed code for the ‘if’ statements and loops within Cg shaders:

```
-ifcvt (all | none | count=N)
   Changes if-conversion mode based on the option.
   all      All 'if' statements will be converted to conditional
            writes
   none     All 'if' statements will generate branching code
   count=N  Sets if_limit_cost to N "operations"

-unroll (all | none | count=N)
   Changes loop-unrolling mode based on the option.
   all      All loop statements that can be unrolled will be
   none     All loop statements that can be implemented with
            branching will be
   count=N  Sets loop_limit_cost to ‘N’ operations
```

Setting both –ifcvt and –unroll to “all” will yield behavior similar to the fp30 profile, where branch instructions are not available. Using “-ifcvt=none” places burden on the Cg fragment program author to use ‘if’ statements where they want true branches, and to use conditional expressions otherwise.

New feature: fp40, ps_2*, and arbfp1 profiles: MRT support

Cg 1.3 supports "Multiple Render Targets" (MRTs) under the arbfp1 and fp40 profiles. When using these profiles, up to three additional 4-component outputs may be written, in addition to the COLOR and DEPTH outputs supported in other profiles. These new outputs are available via the output semantics COLOR1 through COLOR3.

Note that MRTs are an optional feature of arbfp1 and DirectX’s PixelShader 2 spec, and as such not all hardware that supports these profiles support MRTs. The "MaxDrawBuffers" profile option may be used to explicitly set the number of draw

buffers (i.e., render targets) available on the target hardware. If the input program requires more than the specified number of draw buffers, compilation will fail.

If the MaxDrawBuffers profile option is not specified, the stand-alone Cg compiler, cgc, will assume that the target hardware supports MRTs to whatever extent required by the input program.

When compiling programs using the Cg runtime, be sure to call cgGLSetOptimalOptions() under OpenGL, or cgD3D9GetOptimalOptions() under Direct3D. These functions allow you to automatically determine the value for the MaxDrawBuffers profile option that is appropriate for the graphics hardware on the target machine.

New feature: fp40 profile: FACE semantic

The FACE semantic returns a value < 0 for back-facing fragments, > 0 for front-facing fragments, and 0.0 for fragments of lines and points.

New feature: vp40 and fp40 profile: tex*lod( ) function

The `tex{1D|2D|3D|CUBE|RECT}lod( )` function maps to the `NV_vertex_program3` and `NV_fragment_program2` extensions' TXL function. The 'w' component of the texture coordinate is used to select the level of detail used for the texture lookup.

New feature: arb* and derived profiles: State semantics

In previous versions of Cg, access to uniform GL state was via a single uniform global "glstate", which was only supported in the arbvp profile. While this variable is still supported for backwards compatibility, its use is deprecated, and will be dropped in a future version of Cg.

New code should use the name of the GL state as a semantic on a uniform global or paramter. For example, the GL model-view-projection matrix can be accessed by declaring a global such as:

```
uniform float4x4 ModelViewProjection : state.matrix.mvp;
```

transpose and inverse matrixes can also be accessed:

```
uniform float4x4 inverse : state.matrix.mvp.inverse;
```

Any GL state that is accessible to ARB_vertex_program or ARB_fragment_program can be accessed in this way. This is supported in any ARB-derived profile (arbvp1, arbfp1, vp40, and fp40). The same shaders can be used with non-arb profiles (vp30, fp30), but the application programmer will need to manually bind the appropriate values to the declared parameters using the Cg runtime.

Improvements

- New '-fastprecision' option for arbfp*, fp30, and fp40 profiles, to use reduced precision storage (fp16) when appropriate. When used with the 'arbfp*' profile, this option will generate the 'precision_hint_fastest' option in the output stream.
- The code generators used for most of the profiles have been rewritten. This has led to better code generation, better optimizations for code with branching, and better maintainability.

Removed features

- None.

Entry point changes

- cgD3D9GetOptimalOptions() now returns a NULL-terminated array of option strings, rather than a single string containing all of the profile options. This change was necessitated by changes in the way cgCreateProgram() and cgCreateProgramFromFile() process option strings containing spaces; see "Bug fixes", below, and the cgD3D9 runtime documentation for more details.

Bug fixes

- The Cg Runtime now properly supports compiler arguments which contain spaces (in the path names, etc.) A side-effect of this change is that cgCreateProgram( ) no longer supports the undocumented behavior of automatically splitting each argument containing spaces into multiple separate arguments. As per the documentation, each argument must be passed in a separate string, like the parameters passed to the execv( ) POSIX function.
- cgIsParameterReferenced( ) now correctly reports unused parameters
- Fixed issues with handling of generic 'sampler' types
- Many smaller fixes and improvements

Known issues

Known runtime issues

- The cgCopyProgram( ) function in the Cg runtime library currently does not return a valid CGprogram in some situations. Using the returned CGprogram handle may lead to segmentation faults when it is passed to other Cg runtime routines.
- The DirectX 8 and DirectX 9 runtimes have not yet been updated to support the Cg 'Interfaces' feature
- CgFX has some still-open issues with regards to use of arrays outside of basic Cg shader blocks. Major improvements to CgFX are planned for a future release.
- (Windows only) The whole Cg package is now built with Microsoft Visual C++ 2003 .NET. As a result, the Cg Runtime libraries have dependencies on the runtime C++ libraries from this newer version of Visual C++. Previous versions were built with Visual C++ 6.0sp5.
- (Linux only) The CgFX libraries are built against versions `libstdc++-libc6.2-2.so.3` (i386) and `libstdc++.so.5.0.2` (x86_64) of the GCC C++ runtime libraries. To use CgFX, you must link in these revisions of the libraries. We will work to remove this dependency in a future build of CgFX.

Known compiler issues

- Very little error checking is performed on the OpenGL state semantics string (`state.*`); it is just copied to the output assembly. As a result, a typo in the string may compile correctly, and no error will be apparent until the application attempts to load the assembly shader.
- "#" and "##" preprocessor macro operators are not supported
- Error reporting: Some error and warning messages are not as clear as they could 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 expressions ('?:') and logical expressions ('&&' and '||') are always evaluated, regardless of the condition, and currently warnings are not always issued. Hence developers need to watch out for this case.
- Only one return statement is allowed per function. There is an error issued if there is any unreachable code. Return statements in if/for blocks are not supported.
- 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.
- Only loops with single induction variables are unrolled. Loops that require more than 1 induction variable will fail to compile.
- Method calls on a struct always operate on a copy of the struct (as with an 'in' parameter), so any modifications to the struct will not affect the actual variable on which the method was originally called, just the copy. In a future release, we plan to change this to an 'inout' parameter, so modifications will be copied back to the original variable when the method returns.

Known profile-specific issues

- Branching support in the fp40 might necessitate use of the '-ifcvt' and '-unroll' profile-specific options for best performance. These options are described above, in the "New Feature: fp40 profile: Branching" section.
- 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. (also true in Cg 1.2)

Known documentation issues

- Documentation and examples for the new features are currently minimal
- The Cg Toolkit User's Manual has not yet been updated for this release

Known installer issues

- The Cg 1.3 installer currently only 'upgrades' a previous install of Cg 1.x. As a result, the installer will remove or overwrite previous versions of the Cg Toolkit when installing this version.

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