NAME

Cg − An multi-platform, multi-API C-based programming language for GPUs

DESCRIPTION

Cg is a high-level programming language designed to compile to the instruction sets of the programmable portions of GPUs. While Cg programs have great flexibility in the way that they express the computations they perform, the inputs, outputs, and basic resources available to those programs are dictated by where they execute in the graphics pipeline. Other documents describe how to write Cg programs. This document describes the library that application programs use to interact with Cg programs. This library and its associated API is referred to as the Cg runtime.

SEE ALSO

the cgCreateContext manpage, the cgDestroyContext manpage

Cg 1.2 RUNTIME API ADDITIONS

Version 1.2 of the Cg runtime adds a number of new capabilities to the existing set of functionality from previous releases. These new features include functionality that make it possible to write programs that can run more efficiently on the GPU, techniques that help hide some of the inherent limitations of some Cg profiles on the GPU, and entrypoints that support new language functionality in the Cg 1.2 release.

Parameter Literalization

The 1.2 Cg runtime makes it possible to denote some of the parameters to a program as having a fixed constant value. This feature can lead to substantially more efficient programs in a number of cases. For example, a program might have a block of code that implements functionality that is only used some of the time:

```
float4 main(uniform float enableDazzle, ...) : COLOR {
 if (enableDazzle) {
  // do lengthy computation
 }
 else {
  // do basic computation
 } }
```

Some hardware profiles don't directly support branching (this includes all of the fragment program profiles supported in this release), and have to handle code like the program by effectively following both sides of the *if()* test. (They still compute the correct result in the end, just not very efficiently.)

However, if the ''enableDazzle'' parameter is marked as a literal parameter and a value is provided for it, the compiler can generate an optimized version of the program with the knowledge of ''enableDazzle'''s value, just generating GPU code for one of the two cases. This can lead to substantial performance improvments. This feature also makes it easier to write general purpose shaders with a wide variety of supported functionality, while only paying the runtime cost for the functionality provided.

This feature is also useful for parameters with numeric values. For example, consider a shader that implements a diffuse reflection model:

```
float4 main(uniform float3 lightPos, uniform float3 lightColor, uniform float3 Kd, float3 pos : TEXCOORD0, float3 normal : TEXCOORD1) : COLOR { return Kd * lightColor * max(0., dot(normalize(lightPos − pos), normal)); }
```

If the ''lightColor'' and ''Kd'' parameters are set to literals, it is possible for the compiler to compute the product ''Kd * lightColor'' once, rather than once each time the program executes.

Given a parameter handle, the *cgSetParameterVariability()* entrypoint sets the variability of a parameter:

```
void cgSetParameterVariability(CGparameter param, CGenum vary);
```

To set it to a literal parameter, the CG_LITERAL enumerant should be passed as the second parameter.

After a parameter has set to be a literal, the following routines should be used to set the parameter's value.

void cgSetParameter1f(CGparameter param, float x); void cgSetParameter2f(CGparameter param, float x, float y); void cgSetParameter3f(CGparameter param, float x, float y, float z); void cgSetParameter4f(CGparameter param, float x, float y, float z, float w); void cgSetParameter1d(CGparameter param, double x); void cgSetParameter2d(CGparameter param, double x, double y); void cgSetParameter3d(CGparameter param, double x, double y, double z); void cgSetParameter4d(CGparameter param, double x, double y, double z, double w);

void cgSetParameter1fv(CGparameter param, const float *v); void cgSetParameter2fv(CGparameter param, const float *v); void cgSetParameter3fv(CGparameter param, const float *v); void cgSetParameter4fv(CGparameter param, const float *v); void cgSetParameter1dv(CGparameter param, const double *v); void cgSetParameter2dv(CGparameter param, const double *v); void cgSetParameter3dv(CGparameter param, const double *v); void cgSetParameter4dv(CGparameter param, const double *v);

void cgSetMatrixParameterdr(CGparameter param, const double *matrix); void cgSetMatrixParameterfr(CGparameter param, const float *matrix); void cgSetMatrixParameterdc(CGparameter param, const double *matrix); void cgSetMatrixParameterfc(CGparameter param, const float *matrix);

After a parameter has been set to be a literal, or after the value of a literal parameter has been changed, the program must be compiled and loaded into the GPU, regardless of whether it had already been compiled. This issue is discussed further in the section on program recompilation below.

Array Size Specification

The Cg 1.2 language also adds support for ‘‘unsized array’’ variables; programs can be written to take parameters that are arrays with an indeterminate size. The actual size of these arrays is then set via the Cg runtime. This feature is useful for writing general-purpose shaders with a minimal performance penalty.

For example, consider a shader that computes shading given some number of light sources. If the information about each light source is stored in a struct LightInfo, the shader might be written as:

```
float4 main(LightInfo lights[], ...) : COLOR {
  float4 color = float4(0,0,0,1);
  for (i = 0; i < lights.length; ++i) {
    // add lights[i]’s contribution to color
  }
  return color; }
```

The runtime can then be used to set the length of the lights[] array (and then to initialize the values of the LightInfo structures.) As with literal parameters, the program must be recompiled and reloaded after a parameter’s array size is set or changes.

These two entrypoints set the size of an unsized array parameter referenced by the given parameter handle. To set the size of a multidimensional unsized array, all of the dimensions’ sizes must be set simultaneously, by providing them all via the pointer to an array of integer values.

void cgSetArraySize(CGparameter param, int size); void cgSetMultiDimArraySize(CGparameter param, const int *sizes);

XXX what happens if these are called with an already-sized array?? XXX

To get the size of an array parameter, the *cgGetArraySize()* entrypoint can be used.

int cgGetArraySize(CGparameter param, int dimension);

Program Recompilation at Runtime

The Cg 1.2 runtime environment will allow automatic and manual recompilation of programs. This functionality is useful for multiple reasons :

- **Changing variability of parameters**
  Parameters may be changed from uniform variability to literal variability as described above.
- **Changing value of literal parameters**
  Changing the value of a literal parameter will require recompilation since the value is used at compile time.
- **Resizing parameter arrays**
  Changing the length of a parameter array may require recompilation depending on the capabilities of the profile of the program.
- **Binding sub-shader parameters**
  Sub-shader parameters are structures that overload methods that need to be provided at compile time; they are described below. Binding such parameters to program parameters will require recompilation. See the Sub-Shaders entry elsewhere in this document for more information.

Recompilation can be executed manually by the application using the runtime or automatically by the

runtime.

The entry point:

void cgCompileProgram(CGprogram program);

causes the given program to be recompiled, and the function:

CGbool cgIsProgramCompiled(CGprogram program);

reurns a boolean value indicating whether the current program needs recompilation.

By default, programs are automatically compiled when *cgCreateProgram()* or *cgCreateProgramFromFile()* is called. This behavior can be controled with the entry point :

```
void cgSetAutoCompile(CGcontext ctx, CGenum flag);
```

Where flag is one of the following three enumerants :

- **CG_COMPILE_MANUAL**
  With this method the application is responsible for manually recompiling a program. It may check to see if a program requires recompilation with the entry point cgIsProgramCompiled(). cgCompileProgram() can then be used to force compilation.
- **CG_COMPILE_IMMEDIATE**
  CG_COMPILE_IMMEDIATE will force recompilation automatically and immediately when a program enters an uncompiled state.
- **CG_COMPILE_LAZY**
  This method is similar to **CG_COMPILE_IMMEDIATE** but will delay program recompilation until the program object code is needed. The advantage of this method is the reduction of extraneous recompilations. The disadvantage is that compile time errors will not be encountered when the program is enters the uncompiled state but will instead be encountered at some later time.

For programs that use features like unsized arrays that can not be compiled until their array sizes are set, it is good practice to change the default behavior of compilation to CG_COMPILE_MANUAL so that *cgCreateProgram()* or *cgCreateProgramFromFile()* do not unnecessarily encounter and report compilation errors.

Shared Parameters (context global parameters)

Version 1.2 of the runtime introduces parameters that may be shared across programs in the same context via a new binding mechanism. Once shared parameters are constructed and bound to program parameters, setting the value of the shared parameter will automatically set the value of all of the program parameters they are bound to.

Shared parameters belong to a **CGcontext** instead of a **CGprogram**. They may be created with the following new entry points :

CGparameter cgCreateParameter(CGcontext ctx, CGtype type); CGparameter cgCreateParameterArray(CGcontext ctx, CGtype type, int length); CGparameter cgCreateParameterMultiDimArray(CGcontext ctx, CGtype type, int dim, const int *lengths);

They may be deleted with :

void cgDestroyParameter(CGparameter param);

After a parameter has been created, its value should be set with the cgSetParameter*() routines described in the literalization section above.

Once a shared parameter is created it may be associated with any number of program parameters with the call:

void cgConnectParameter(CGparameter from, CGparameter to);

where ‘‘from’’ is a parameter created with one of the *cgCreateParameter()* calls, and ‘‘to’’ is a program parameter.

Given a program parameter, the handle to the shared parameter that is bound to it (if any) can be found with the call:

CGparameter cgGetConnectedParameter(CGparameter param);

It returns NULL if no shared parameter has been connected to ‘‘param’’.

There are also calls that make it possible to find the set of program parameters to which a given shared parameter has been connected to. The entry point:

int cgGetNumConnectedToParameters(CGparameter param);

returns the total number of program parameters that ‘‘param’’ has been conencted to, and the entry point:

CGparameter cgGetConnectedToParameter(CGparameter param, int index);

can be used to get CGparameter handles for each of the program parameters to which a shared parameter is connected.

A shared parameter can be unbound from a program parameter with :

void cgDisconnectParameter(CGparameter param);

The context in which a shared parameter was created can be returned with:

CGcontext cgGetParameterContext(CGparameter param);

And the entrypoint:

CGbool cgIsParameterGlobal(CGparameter param);

can be used to determine if a parameter is a shared (global) parameter.

Shader Interface Support

From the runtime’s perspective, shader interfaces are simply struct parameters that have a **CGtype** associated with them. For example, if the following Cg code is included in some program source compiled in the runtime :

```
interface FooInterface
 {
  float SomeMethod(float x);
 }

struct FooStruct : FooInterface
 {
  float SomeMethod(float x);
   {
    return(Scale * x);
   }

  float Scale;
 };
```

The named types **FooInterface** and **FooStruct** will be added to the context. Each one will have a unique **CGtype** associated with it. The **CGtype** can be retrieved with :

CGtype cgGetNamedUserType(CGprogram program, const char *name); int cgGetNumUserTypes(CGprogram program); CGtype cgGetUserType(CGprogram program, int index);

CGbool cgIsParentType(CGtype parent, CGtype child); CGbool cgIsInterfaceType(CGtype type);

Once the **CGtype** has been retrieved, it may be used to construct an instance of the struct using cgCreateParameter(). It may then be bound to a program parameter of the parent type (in the above

example this would be FooInterface) using cgBindParameter().

Calling cgGetParameterType() on such a parameter will return the **CG_STRUCT** to keep backwards compatibility with code that recurses parameter trees. In order to obtain the enumerant of the named type the following entry point should be used :

CGtype cgGetParameterNamedType(CGparameter param);

The parent types of a given named type may be obtained with the following entry points :

int cgGetNumParentTypes(CGtype type); CGtype cgGetParentType(CGtype type, int index);

If Cg source modules with differing definitions of named types are added to the same context, an error will be thrown. XXX update for new scoping/context/program local definitions stuff XXX

Updated Parameter Management Routines

XXX wheer should these go?

Some entrypoints from before have been updated in backwards compatible ways

CGparameter cgGetFirstParameter(CGprogram prog, CGenum name_space); CGparameter cgGetFirstLeafParameter(CGprogram prog, CGenum name_space);

like cgGetNamedParameter, but limits search to the given name_space (CG_PROGRAM or CG_GLOBAL)...

CGparameter cgGetNamedProgramParameter(CGprogram prog, CGenum name_space, const char *name);

NAME

cgAddStateEnumerant − associates an integer enumerant value as a possible value for a state

SYNOPSIS

```
#include <Cg/cg.h>

void cgAddStateEnumerant(CGstate state, const char *name, int value);
```

PARAMETERS

state Specifies the state to associate the name and value with.

name Specifies the name of the enumerant.

value Specifies the value of the enumerant.

DESCRIPTION

cgAddStateEnumerant associates a given named integer enumerant value with a state definition. When that state is later used in a pass in an effect file, the value of the state assignment can optionally be given by providing the name of a named enumerant defined with **cgAddStateEnumerant**. The state assignment will then take on the value provided when the enumerant was defined.

ERRORS

CG_INVALID_STATE_HANDLE_ERROR is generated if **state** does not refer to a valid state.

SEE ALSO

the cgCreateState manpage, the cgCreateArrayState manpage, the cgCreateSamplerState manpage, the cgCreateSamplerArrayState manpage, and the cgGetStateName manpage.

NAME

cgCallStateResetCallback − calls the state resetting callback function for a state assignment.

SYNOPSIS

```
#include <Cg/cg.h>

CGbool cgCallStateResetCallback(CGstateassignment sa);
```

PARAMETERS

sa Specifies the state assignment handle.

RETURN VALUES

The boolean value returned by the callback function is returned. It should be **CG_TRUE** upon success. If no callback function was defined, **CG_TRUE** is returned.

DESCRIPTION

cgCallStateResetCallback calls the graphics state resetting callback function for the given state assignment.

The semantics of ‘‘resetting state’’ will depend on the particular graphics state manager that defined the valid state assignments; it will generally either mean that graphics state is reset to what it was before the pass, or that it is reset to the default value. The OpenGL state manager in the OpenGL Cg runtime implements the latter approach.

ERRORS

CG_INVALID_STATE_ASSIGNMENT_ERROR is generated if **sa** does not refer to a valid state assignment.

SEE ALSO

the cgResetPassState manpage, the cgSetStateCallbacks manpage

NAME

cgCallStateSetCallback − calls the state setting callback function for a state assignment.

SYNOPSIS

```
#include <Cg/cg.h>

CGbool cgCallStateSetCallback(CGstateassignment sa);
```

PARAMETERS

sa Specifies the state assignment handle.

RETURN VALUES

The boolean value returned by the callback function is returned. It should be **CG_TRUE** upon success. If no callback function was defined, **CG_TRUE** is returned.

DESCRIPTION

cgCallStateSetCallback calls the graphics state setting callback function for the given state assignment.

ERRORS

CG_INVALID_STATE_ASSIGNMENT_ERROR is generated if **sa** does not refer to a valid state assignment.

SEE ALSO

the cgSetPassState manpage, the cgSetStateCallbacks manpage

NAME

cgCallStateValidateCallback − calls the state validation callback function for a state assignment.

SYNOPSIS

```
#include <Cg/cg.h>

CGbool cgCallStateValidateCallback(CGstateassignment sa);
```

PARAMETERS

sa Specifies the state assignment handle.

RETURN VALUES

The boolean value returned by the validation function is returned. It should be **CG_TRUE** upon success. If no callback function was defined, **CG_TRUE** is returned.

DESCRIPTION

cgCallStateValidateCallback calls the state validation callback function for the given state assignment. The validation callback will return **CG_TRUE** or **CG_FALSE** depending on whether the current hardware and driver support the graphics state set by the state assignment.

ERRORS

CG_INVALID_STATE_ASSIGNMENT_ERROR is generated if **sa** does not refer to a valid state assignment.

SEE ALSO

the cgValidatePassState manpage, the cgSetStateCallbacks manpage

NAME

cgCompileProgram − compile a program object

SYNOPSIS

```
#include <Cg/cg.h>

void cgCompileProgram( CGprogram prog );
```

PARAMETERS

prog Specifies the program object to compile or inspect.

DESCRIPTION

cgCompileProgram compiles the specified Cg program for its target profile. A program must be compiled before it can be loaded (by the API-specific part of the runtime). It must also be compiled before its parameters can be inspected.

Certain actions invalidate a compiled program and the current value of all of its parameters. If one of these actions is performed, the program must be recompiled before it can be used. A program is invalidated if the program source is modified, if the compile arguments are modified, or if the entry point is changed.

If one of the parameter bindings for a program is changed, that action invalidates the compiled program, but does not invalidate the current value of the program's parameters.

RETURN VALUES

cgCompileProgram does not return any values.

EXAMPLES

```
if(!cgIsProgramCompiled(prog))
 cgCompileProgram(prog);
```

ASSOCIATED GETS

cgGetProgramString with **pname CG_COMPILED_PROGRAM**.

ERRORS

CG_PROGRAM_HANDLE_ERROR is generated if **prog** is an invalid program handle.

CG_COMPILE_ERROR is generated if the compile fails.

SEE ALSO

the cgIsProgramCompiled manpage, the cgCreateProgram manpage, the cgGetNextParameter manpage, the cgIsParameter manpage, and the cgGetProgramString manpage

NAME

cgConnectParameter − connect two parameters

SYNOPSIS

```
#include <Cg/cg.h>

void cgConnectParameter(CGparameter from, CGparameter to);
```

PARAMETERS

from The source parameter.

to The destination parameter.

DESCRIPTION

cgConnectParameter connects a source (from) parameter to a destination (to) parameter. The resulting connection forces the value and variability of the destination parameter to be identical to the source parameter. A source parameter may be connected to multiple destination parameters but there may only be one source parameter per destination parameter.

cgConnectParameter may be used to create an arbitrarily deep tree. A runtime error will be thrown if a cycle is inadvertently created. For example, the following code snipped would generate a **CG_BIND_CREATES_CYCLE_ERROR** :

```
CGcontext Context = cgCreateContext();
CGparameter Param1 = cgCreateParameter(Context, CG_FLOAT);
CGparameter Param2 = cgCreateParameter(Context, CG_FLOAT);
CGparameter Param3 = cgCreateParameter(Context, CG_FLOAT);

cgConnectParameter(Param1, Param2);
cgConnectParameter(Param2, Param3);
cgConnectParameter(Param3, Param1); // This will throw the error
```

If the source type is a complex type (e.g., struct, or array) the topology and member types of both parameters must be identical. Each correlating member parameter will be connected.

Both parameters must be of the same type unless the source parameter is a struct type, the destination parameter is an interface type, and the struct type implements the interface type. In such a case, a copy of the parameter tree under the source parameter will be duplicated, linked to the orignal tree, and placed under the destination parameter.

If a an array parameter is connected to a resizable array parameter the destination parameter array will automatically be resized to match the source array.

The source parameter may not be a program parameter. Also the variability of the parameters may not be **CG_VARYING**.

RETURN VALUES

This function does not return a value.

EXAMPLES

```
CGparameter TimeParam1 = cgGetNamedParameter(program1, "time");
CGparameter TimeParam2 = cgGetNamedParameter(program2, "time");
CGparameter SharedTime = cgCreateParameter(Context,
                                           cgGetParameterType(TimeParam1));

cgConnectParameter(SharedTime, TimeParam1);
cgConnectParameter(SharedTime, TimeParam2);
```

```
cgSetParameter1f(SharedTime, 2.0);
```

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if either of the **from** or **to** parameters are invalid handles.

CG_PARAMETER_IS_NOT_SHARED is generated if the source parameter is a program parameter.

CG_BIND_CREATES_CYCLE_ERROR is generated if the connection will result in a cycle.

CG_PARAMETERS_DO_NOT_MATCH_ERROR is generated if the parameters do not have the same type or the topologies do not match.

CG_ARRAY_TYPES_DO_NOT_MATCH_ERROR is generated if the type of two arrays being connected do not match.

CG_ARRAY_DIMENSIONS_DO_NOT_MATCH_ERROR is generated if the dimensions of two arrays being connected do not match.

SEE ALSO

the cgGetConnectedParameter manpage, the cgGetConnectedToParameter manpage, and the cgDisconnectParameter manpage

NAME

cgCopyProgram − make a copy of a program object

SYNOPSIS

```
#include <Cg/cg.h>

CGprogram cgCopyProgram( CGprogram prog );
```

PARAMETERS

prog Specifies the program to copy.

DESCRIPTION

cgCopyProgram creates a new program object that is a copy of **prog** and adds it to the same context as **prog**. This function is useful for creating a new instance of a program whose parameter properties have been modified by the run-time API.

RETURN VALUES

Returns a copy of **prog** on success.

Returns **NULL** if **prog** is invalid or allocation fails.

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if **prog** is an invalid program handle.

SEE ALSO

the cgCreateProgram manpage, the cgDestroyProgram manpage

NAME

cgCreateArraySamplerState − create a new array-typed sampler state definition

SYNOPSIS

```
#include <Cg/cg.h>

CGstate cgCreateArraySamplerState(CGcontext cgfx, const char *name, CGtype type, i
```

PARAMETERS

ctx Specifies the context to define the sampler state in.

name Specifies the name of the new sampler state.

type Specifies the type of the new sampler state.

nelems Specifies the number of elements in the array.

DESCRIPTION

cgCreateArraySamplerState adds a new array-typed sampler state definition to the context. When an effect file is added to the context, all state in **sampler_state** blocks in must have been defined ahead of time via a call to **cgCreateSamplerState** or the cgCreateArraySamplerState manpage.

Applications will typically call the cgSetStateCallbacks manpage shortly after creating a new state with **cgCreateArraySamplerState**.

RETURN VALUES

cgCreateArraySamplerState returns a handle to the newly created **CGstate**. If there is an error, **NULL** is returned.

ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if **ctx** does not refer to a valid context.

CG_INVALID_PARAMETER_ERROR is generated if **name** is **NULL** or not a valid identifier, as well as if **type** is not a simple scalar, vector, or matrix-type, or if **nelems** is not a positive number.

SEE ALSO

the cgCreateSamplerState manpage, the cgGetStateName manpage, the cgGetStateType manpage, the cgIsState manpage, the cgSetStateCallbacks manpage, and the cgGLRegisterStates manpage.

NAME

cgCreateArrayState − create a new array-typed state definition

SYNOPSIS

```
#include <Cg/cg.h>

CGstate cgCreateArrayState(CGcontext cgfx, const char *name, CGtype type, int nele
```

PARAMETERS

ctx Specifies the context to define the state in.

name Specifies the name of the new state.

type Specifies the type of the new state.

nelems Specifies the number of elements in the array.

DESCRIPTION

cgCreateArrayState adds a new array-typed state definition to the context. When a CgFX file is later added to the context, all state assignments in passes in the file must have been defined ahead of time via a call to the cgCreateState manpage or **cgCreateArrayState**.

Applications will typically call the cgSetStateCallbacks manpage shortly after creating a new state with **cgCreateState**.

RETURN VALUES

cgCreateArrayState returns a handle to the newly created **CGstate**. If there is an error, **NULL** is returned.

ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if **ctx** does not refer to a valid context.

CG_INVALID_PARAMETER_ERROR is generated if **name** is **NULL** or not a valid identifier, as well as if **type** is not a simple scalar, vector, or matrix-type, or if **nelems** is not a positive number.

SEE ALSO

the cgCreateArrayState manpage, the cgGetStateName manpage, the cgGetStateType manpage, the cgIsState manpage, the cgSetStateCallbacks manpage, and the cgGLRegisterStates manpage.

NAME

cgCreateContext − create a Cg context

SYNOPSIS

```
#include <Cg/cg.h>

CGcontext cgCreateContext();
```

DESCRIPTION

cgCreateContext creates a Cg context object and returns its handle. A Cg context is a container for Cg programs. All Cg programs must be added as part of a context.

RETURN VALUES

Returns a valid CGcontext on success. **NULL** is returned if context creation fails.

ERRORS

CG_MEMORY_ALLOC_ERROR is generated if a context couldn't be created.

SEE ALSO

the cgDestroyContext manpage

NAME

cgCreateEffect − generate a new effect object from a string

SYNOPSIS

```
#include <Cg/cg.h>

CGeffect cgCreateEffect(CGcontext ctx,
                        const char *effect,
                        const char **args)
```

PARAMETERS

ctx — Specifies the context that the new effect will be added to.

effect — A string containing the effect's source code.

args — If **args** is not **NULL** it is assumed to be an array of null-terminated strings that will be passed as directly to the compiler as arguments. The last value of the array must be a **NULL**.

DESCRIPTION

cgCreateEffect generates a new **CGeffect** object and adds it to the specified Cg context.

The following is a typical sequence of commands for initializing a new effect:

```
char *effectSource = ...;
CGcontext ctx = cgCreateContext();
CGeffect eff  = cgCreateEffect(ctx,
                               effectSource,
                               NULL);
```

RETURN VALUES

Returns a **CGeffect** handle on success.

Returns **NULL** if any error occurs. the cgGetLastListing manpage can be called to retrieve any warning or error messages from the compilation process.

ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if the **ctx** is not a valid context.

CG_COMPILER_ERROR is generated if the compile failed.

SEE ALSO

the cgCreateContext manpage, the cgCreateEffectFromFile manpage, and the cgGetLastListing manpage

NAME

cgCreateEffect − generate a new effect object from a file

SYNOPSIS

```
#include <Cg/cg.h>

CGeffect cgCreateEffect(CGcontext ctx,
                        const char *filename,
                        const char **args)
```

PARAMETERS

ctx Specifies the context that the new effect will be added to.

filename A file name containing the effect's source code.

args If **args** is not **NULL** it is assumed to be an array of null-terminated strings that will be passed as directly to the compiler as arguments. The last value of the array must be a **NULL**.

DESCRIPTION

cgCreateEffectFromFile generates a new **CGeffect** object and adds it to the specified Cg context.

The following is a typical sequence of commands for initializing a new effect:

```
CGcontext ctx = cgCreateContext();
CGeffect eff  = cgCreateEffectFromFile(ctx, "filename.cgfx", NULL);
```

RETURN VALUES

Returns a **CGeffect** handle on success.

Returns **NULL** if any error occurs. the cgGetLastListing manpage can be called to retrieve any warning or error messages from the compilation process.

ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if the **ctx** is not a valid context.

CG_FILE_READ_ERROR is generated if the given filename cannot be read.

CG_COMPILER_ERROR is generated if the compile failed.

SEE ALSO

the cgCreateContext manpage, the cgCreateEffect manpage, and the cgGetLastListing manpage

NAME

cgCreateEffectParameter − whatever

SYNOPSIS

```
#include <Cg/cg.h>

CGparameter cgCreateEffectParameter(CGeffect effect, const char *name, CGtype type
```

PARAMETERS

effect Specifies something.

name Specifies something.

type Specifies something.

DESCRIPTION

cgCreateEffectParameter does something.

EXAMPLE

write_me

RETURN VALUES

cgCreateEffectParameter returns the handle to the created parameter.

ERRORS

HISTORY

This function was introduced with Cg 1.5.

SEE ALSO

NAME

cgCreateParameter − creates a parameter

SYNOPSIS

```
#include <Cg/cg.h>

CGparameter cgCreateParameter(CGcontext ctx, CGtype type);
```

PARAMETERS

ctx Specifies the context that the new parameter will be added to.

type The type of the new parameter.

DESCRIPTION

cgCreateParameter creates context level shared parameters. These parameters are primarily used by connecting them to one or more program parameters with cgConnectParameter.

RETURN VALUES

Returns the handle to the new parameter.

EXAMPLES

```
CGcontext Context = cgCreateContext();
CGparameter MySharedFloatParam = cgCreateParameter(Context, CG_FLOAT);
```

ERRORS

CG_INVALID_VALUE_TYPE_ERROR is generated if **type** is invalid.

CG_INVALID_CONTEXT_HANDLE_ERROR if **ctx** is invalid.

SEE ALSO

the cgCreateParameterArray manpage, the cgCreateParameterMultiDimArray manpage, and the cgDestroyParameter manpage

NAME

cgCreateParameterArray − creates a parameter array

SYNOPSIS

```
#include <Cg/cg.h>

CGparameter cgCreateParameter(CGcontext ctx, CGtype type, int length);
```

PARAMETERS

ctx Specifies the context that the new parameter will be added to.

type The type of the new parameter.

length The length of the array being created.

DESCRIPTION

cgCreateParameterArray creates context level shared parameter arrays. These parameters are primarily used by connecting them to one or more program parameter arrays with cgConnectParameter.

cgCreateParameterArray works similarly to cgCreateParameter. Instead of creating a single parameter, it creates an array of them.

RETURN VALUES

Returns the handle to the new parameter array.

EXAMPLES

```
CGcontext Context = cgCreateContext();
CGparameter MyFloatArray = cgCreateParameterArray(Context, CG_FLOAT, 5);
```

ERRORS

CG_INVALID_VALUE_TYPE_ERROR is generated if **type** is invalid.

CG_INVALID_CONTEXT_HANDLE_ERROR if **ctx** is invalid.

SEE ALSO

the cgCreateParameter manpage, the cgCreateParameterMultiDimArray manpage, and the cgDestroyParameter manpage

NAME

cgCreateParameterMultiDimArray − creates a multi-dimensional parameter array

SYNOPSIS

```
#include <Cg/cg.h>

CGparameter cgCreateParameterMultiDimArray(CGcontext ctx,
                                           CGtype type,
                                           int dim,
                                           const int *lengths);
```

PARAMETERS

ctx Specifies the context that the new parameter will be added to.

type The type of the new parameter.

dim The dimension of the multi-dimensional array.

lengths An array of length values, one length per dimension.

DESCRIPTION

cgCreateParameterMultiArray creates context level shared multi-dimensional parameter arrays. These parameters are primarily used by connecting them to one or more program parameter arrays with cgConnectParameter.

cgCreateParameterMultiDimArray works similarly to cgCreateParameterMultiDimArray. Instead of taking a single length parameter it takes an array of lengths, one per dimension. The dimension of the array is defined by the **dim** parameter.

RETURN VALUES

Returns the handle to the new parameter array.

EXAMPLES

```
// Creates a three dimensional float array similar to the C declaration :
//  float MyFloatArray[5][3][4];
int Lengths[] = { 5, 3, 4 };
CGcontext Context = cgCreateContext();
CGparameter MyFloatArray =
  cgCreateParameterArray(Context, CG_FLOAT, 3, Lengths);
```

ERRORS

CG_INVALID_VALUE_TYPE_ERROR is generated if **type** is invalid.

CG_INVALID_CONTEXT_HANDLE_ERROR if **ctx** is invalid.

SEE ALSO

the cgCreateParameter manpage, the cgCreateParameterArray manpage, and the cgDestroyParameter manpage

NAME

cgCreateProgram − generate a new program object from a string

SYNOPSIS

```
#include <Cg/cg.h>

CGprogram cgCreateProgram(CGcontext ctx,
                          CGenum program_type,
                          const char *program,
                          CGprofile profile,
                          const char *entry,
                          const char **args)
```

PARAMETERS

ctx Specifies the context that the new program will be added to.

program_type

The **program_type** parameter is an enumerant that describes what the **program** parameter string contains. The following is a list of valid enumerants that may be passed in :

CG_SOURCE

If **program_type** is **CG_SOURCE**, **program** is assumed to be a string that contains Cg source code.

CG_OBJECT

If **program_type** is **CG_OBJECT**, **program** is assumed to be a string that contains object code that resulted from the precompilation of some Cg source code.

program A string containing either the programs source or object code. See the program parameter above for more information.

profile The enumerant for the profile the program.

entry The entry point to the program in the Cg source. If set to **NULL** this will default to **“main”**.

args If **args** is not **NULL** it is assumed to be an array of null-terminated strings that will be passed as directly to the compiler as arguments. The last value of the array must be a **NULL**.

DESCRIPTION

cgCreateProgram generates a new CGprogram object and adds it to the specified Cg context.

The following is a typical sequence of commands for initializing a new program:

```
CGcontext ctx = cgCreateContext();
CGprogram prog = cgCreateProgram(ctx,
                                 CG_SOURCE,
                                 mysourcepointer,
                                 CG_PROFILE_ARBVP1,
                                 "myshader",
                                 NULL);
```

RETURN VALUES

Returns a CGprogram handle on success.

Returns **NULL** if any error occurs.

ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if the **ctx** is not a valid context.

CG_INVALID_ENUMERANT_ERROR is generated if **program_type** is an invalid enumerant.

CG_UNKNOWN_PROFILE_ERROR is generated if **profile** is not a supported profile.

CG_COMPILE_ERROR is generated if the compile failed.

SEE ALSO

the cgCreateContext manpage, and the cgGetProgramString manpage

NAME

cgCreateProgramFromEffect − generate a new program object from an effect

SYNOPSIS

```
#include <Cg/cg.h>

CGprogram cgCreateProgramFromEffect(CGeffect effect,
                                    CGprofile profile,
                                    const char *entry,
                                    const char **args)
```

PARAMETERS

effect — Specifies the effect with the program source code from which to create the program.

profile — The enumerant for the profile for the program.

entry — The entry point to the program in the Cg source. If set to **NULL** this will default to **"main"**.

args — If **args** is not **NULL** it is assumed to be an array of null-terminated strings that will be passed as directly to the compiler as arguments. The last value of the array must be a **NULL**.

DESCRIPTION

cgCreateProgramFromEffect generates a new **CGprogram** object and adds it to the effect's Cg context.

RETURN VALUES

Returns a **CGprogram** handle on success.

Returns **NULL** if any error occurs.

ERRORS

CG_INVALID_EFFECT_HANDLE_ERROR is generated if the **effect** is not a valid effect.

CG_UNKNOWN_PROFILE_ERROR is generated if **profile** is not a supported profile.

CG_COMPILER_ERROR is generated if compilation failed.

SEE ALSO

the cgCreateProgram manpage, and the cgCreateProgramFromFile manpage

NAME

cgCreateProgramFromFile − generate a new program object from a file

SYNOPSIS

```
#include <Cg/cg.h>

CGprogram cgCreateProgramFromFile(CGcontext ctx,
                                  CGenum program_type,
                                  const char *program_file,
                                  CGprofile profile,
                                  const char *entry,
                                  const char **args)
```

PARAMETERS

ctx Specifies the context that the new program will be added to.

program_type

The **program_type** parameter is an enumerant that describes what the file indicated by the **program_file** parameter contains. The following is a list of valid enumerants that may be passed in :

CG_SOURCE

If **program_type** is **CG_SOURCE**, **program_file** is assumed to be a file that contains Cg source code.

CG_OBJECT

If **program_type** is **CG_OBJECT**, **program_file** is assumed to be a file that contains object code that resulted from the precompilation of some Cg source code.

program_file

A filename of a file that source or object code. See the program parameter above for more information.

profile The enumerant for the profile the program.

entry The entry point to the program in the Cg source. If set to **NULL** this will default to **“main”**.

args If **args** is not **NULL** it is assumed to be an array of null-terminated strings that will be passed as directly to the compiler as arguments. The last value of the array must be a **NULL**.

DESCRIPTION

cgCreateProgramFromFile generates a new CGprogram object and adds it to the specified Cg context.

The following is a typical sequence of commands for initializing a new program:

```
CGcontext ctx = cgCreateContext();
CGprogram prog = cgCreateProgramFromFile(ctx,
                                         CG_SOURCE,
                                         mysourcefilename,
                                         CG_PROFILE_ARBVP1,
                                         "myshader",
                                         NULL);
```

RETURN VALUES

Returns a CGprogram handle on success.

Returns **NULL** if any error occurs.

ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if the **ctx** is not a valid context.

CG_INVALID_ENUMERANT_ERROR is generated if **program_type** is an invalid enumerant.

CG_UNKNOWN_PROFILE_ERROR is generated if **profile** is not a supported profile.

CG_COMPILE_ERROR is generated if the compile failed.

SEE ALSO

the cgCreateContext manpage, and the cgGetProgramString manpage

NAME

cgCreateSamplerState − create a new sampler state definition

SYNOPSIS

```
#include <Cg/cg.h>

CGstate cgCreateSamplerState(CGcontext cgfx, const char *name, CGtype type);
```

PARAMETERS

ctx Specifies the context to define the sampler state in.

name Specifies the name of the new sampler state.

type Specifies the type of the new sampler state.

DESCRIPTION

cgCreateSamplerState adds a new sampler state definition to the context. When an effect file is added to the context, all state in **sampler_state** blocks in must have been defined ahead of time via a call to **cgCreateSamplerState** or the cgCreateArraySamplerState manpage.

Applications will typically call the cgSetStateCallbacks manpage shortly after creating a new state with **cgCreateSamplerState**.

RETURN VALUES

cgCreateSamplerState returns a handle to the newly created **CGstate**. If there is an error, **NULL** is returned.

ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if **ctx** does not refer to a valid context.

CG_INVALID_PARAMETER_ERROR is generated if **name** is **NULL** or not a valid identifier, as well as if **type** is not a simple scalar, vector, or matrix-type. (Array-typed state should be created with the cgCreateArrayState manpage.)

SEE ALSO

the cgCreateArraySamplerState manpage, the cgGetStateName manpage, the cgGetStateType manpage, the cgIsState manpage, the cgCreateSamplerStateAssignment manpage, and the cgGLRegisterStates manpage.

NAME

cgCreateState − create a new state definition

SYNOPSIS

```
#include <Cg/cg.h>

CGstate cgCreateState(CGcontext cgfx, const char *name, CGtype type);
```

PARAMETERS

ctx Specifies the context to define the state in.

name Specifies the name of the new state.

type Specifies the type of the new state.

DESCRIPTION

cgCreateState adds a new state definition to the context. When a CgFX file is later added to the context, all state assignments in passes in the file must have been defined ahead of time via a call to **cgCreateState** or the cgCreateArrayState manpage.

Applications will typically call the cgSetStateCallbacks manpage shortly after creating a new state with **cgCreateState**.

RETURN VALUES

cgCreateState returns a handle to the newly created **CGstate**. If there is an error, **NULL** is returned.

ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if **ctx** does not refer to a valid context.

CG_INVALID_PARAMETER_ERROR is generated if **name** is **NULL** or not a valid identifier, as well as if **type** is not a simple scalar, vector, or matrix-type. (Array-typed state should be created with the cgCreateArrayState manpage.)

SEE ALSO

the cgCreateArrayState manpage, the cgGetStateName manpage, the cgGetStateType manpage, the cgIsState manpage, the cgSetStateCallbacks manpage, and the cgGLRegisterStates manpage.

NAME

cgDestroyContext − delete a Cg context

SYNOPSIS

```
#include <Cg/cg.h>

void cgDestroyContext(CGcontext ctx);
```

PARAMETERS

ctx Specifies the context to be deleted.

DESCRIPTION

cgDestroyContext deletes a Cg context object and all the programs it contains.

ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if the **ctx** context handle is invalid.

SEE ALSO

the cgCreateContext manpage

NAME

cgDestroyEffect − delete an effect object

SYNOPSIS

```
#include <Cg/cg.h>

void cgDestroyEffect( CGeffect effect );
```

PARAMETERS

effect Specifies the effect object to delete.

DESCRIPTION

cgDestroyEffect removes the specified effect object and all its associated data. Any **CGeffect** handles that reference this effect will become invalid after the effect is deleted. Likewise, all techniques, passes, and parameters contained in the effect also become invalid after the effect is destroyed.

ERRORS

CG_INVALID_EFFECT_HANDLE_ERROR is generated if **effect** is an invalid effect handle.

SEE ALSO

the cgCreateEffect manpage, the cgCreateEffectFromFile manpage

NAME

cgDestroyParameter − destroys a parameter

SYNOPSIS

```
#include <Cg/cg.h>

CGparameter cgDestroyParameter(CGparameter param);
```

PARAMETERS

param The parameter to destroy.

DESCRIPTION

cgDestroyParameter destroys parameters created with cgCreateParameter, cgCreateParameterArray, or cgCreateParameterMultiDimArray.

Upon destruction, the **CGparameter** will become invalid. Any connections (see the cgConnectParameter manpage) in which **param** is the destination parameter will be disconnected. An error will be thrown if **param** is a source parameter in any connections.

The parameter being destroyed may not be one of the children parameters of a struct or array parameter. In other words it must be a **CGparameter** returned by one of the cgCreateParameter family of entry points.

RETURN VALUES

This function does not return a value.

EXAMPLES

```
CGcontext Context = cgCreateContext();
CGparameter MyFloatParam = cgCreateParameter(Context, CG_FLOAT);
CGparameter MyFloatParamArray = cgCreateParameterArray(Context, CG_FLOAT, 5);

// ...

cgDestroyParameter(MyFloatParam);
cgDestroyParameter(MyFloatParamArray);
```

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if **param** is invalid.

CG_NOT_ROOT_PARAMETER_ERROR is generated if the **param** isn't the top-level parameter of a struct or array that was created.

CG_PARAMETER_IS_NOT_SHARED_ERROR is generated if **param** does not refer to a parameter created by one of the cgCreateParameter family of entry points.

CG_CANNOT_DESTROY_PARAMETER_ERROR is generated if **param** is a source parameter in a connection made by cgConnectParameter. cgDisconnectParameter should be used before calling **cgDestroyParameter** in such a case.

SEE ALSO

the cgCreateParameter manpage, the cgCreateParameterMultiDimArray manpage, and the cgCreateParameterArray manpage

NAME

cgDestroyProgram − delete a program object

SYNOPSIS

```
#include <Cg/cg.h>

void cgDestroyProgram( CGprogram prog );
```

PARAMETERS

prog Specifies the program object to delete.

DESCRIPTION

cgDestroyProgram removes the specified program object and all its associated data. Any **CGprogram** variables that reference this program will become invalid after the program is deleted. Likewise, any objects contained by this program (e.g. **CGparameter** objects) will also become invalid after the program is deleted.

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if **prog** is an invalid program handle.

SEE ALSO

the cgCreateProgram manpage the cgCreateProgramFromFile manpage

NAME

cgDisconnectParameter − disconnects two parameters

SYNOPSIS

```
#include <Cg/cg.h>

void cgDisconnectParameter(CGparameter param);
```

PARAMETERS

param The destination parameter in the connection that will be disconnected.

DESCRIPTION

cgDisconnectParameter disconnects an existing connection made with cgConnectParameter between to parameters. Since a given parameter can only be connected to one source parameter, only the destination parameter is required as an argument to **cgDisconnectParameter**.

If the type of **param** is an interface and the struct connected to it is a struct that implements it, any sub-parameters created by the connection will also be destroyed.

RETURN VALUES

This function does not return a value.

EXAMPLES

```
CGparameter TimeParam1 = cgGetNamedParameter(program1, "time");
CGparameter SharedTime = cgCreateParameter(Context,
                                           cgGetParameterType(TimeParam1));

cgConnectParameter(SharedTime, TimeParam1);

// ...

cgDisconnectParameter(TimeParam1);
```

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if **param** is invalid.

SEE ALSO

the cgGetConnectedParameter manpage, the cgGetConnectedToParameter manpage, and the cgConnnectParameter manpage

NAME

cgEvaluateProgram − evaluates a Cg program on the CPU

SYNOPSIS

```
#include <Cg/cg.h>

void cgEvaluateProgram(CGprogram prog, float *buf, int ncomps, int nx, int ny, int
```

PARAMETERS

technique
: Specifies the program handle.

buf
: Buffer in which to store the results of program evaluation.

ncomps
: Number of components to store for each returned program value

nx
: Number of points at which to evaluate the program in the x direction

ny
: Number of points at which to evaluate the program in the y direction

nz
: Number of points at which to evaluate the program in the z direction

DESCRIPTION

cgEvaluateProgram evaluates a Cg program at a set of regularly spaced points in one, two, or three dimensions. The program must have been compiled with the **CG_PROFILE_GENERIC** profile. The value returned from the program via the **COLOR** semantic is stored in the given buffer for each evaluation point, and any varying parameters to the program with **POSITION** semantic take on the (x,y,z) position over the range zero to one at which the program is evaluated at each point. The **PSIZE** semantic can be used to find the spacing between evaluating points.

The total size of the buffer **buf** should be equal to **ncomps*nx*ny*nz**.

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if **program** does not refer to a valid program.

CG_INVALID_PROFILE_ERROR is generated if **program**'s profile is not **CG_PROFILE_GENERIC**.

CG_INVALID_PARAMETER_ERROR is generated if **buf** is **NULL**, any of **nx**, **ny**, or **nz** is less than zero, or **ncomps** is not 0, 1, 2, or 3.

SEE ALSO

the cgCreateProgram manpage, the cgCreateProgramFromFile manpage, the cgCreateProgramFromEffect manpage.

NAME

cgGetAnnotationName − get an annotation's name

SYNOPSIS

```
#include <Cg/cg.h>

const char * cgGetAnnotationName( CGannotation ann );
```

PARAMETERS

ann Specifies the annotation.

DESCRIPTION

cgGetAnnotationName allows the application to retrieve the name of a annotation. This name can be used later to retrieve the annotation using **cgGetNamedPassAnnotation**, **cgGetNamedParameterAnnotation**, **cgGetNamedPTechniqueAnnotation**, or **cgGetNamedProgramAnnotation**.

RETURN VALUES

Returns the null-terminated name string for the annotation.

Returns null if **ann** is invalid.

ERRORS

CG_INVALID_ANNOTATION_HANDLE_ERROR is generated if **prog** does not refer to a valid annotation.

SEE ALSO

the cgGetNamedPassAnnotation manpage, the cgGetNamedParameterAnnotation manpage, the cgGetNamedTechniqueAnnotation manpage, and the cgGetNamedProgramAnnotation manpage

NAME

cgGetAnnotationType − get an annotation's type

SYNOPSIS

```
#include <Cg/cg.h>

CGtype cgGetAnnotationType( CGannotation ann );
```

PARAMETERS

ann Specifies the annotation.

DESCRIPTION

cgGetAnnotationType allows the application to retrieve the type of a annotation in a Cg effect.

cgGetAnnotationType will return **CG_STRUCT** if the annotation is a struct and **CG_ARRAY** if the annotation is an array. Otherwise it will return the data type associated with the annotation.

RETURN VALUES

Returns the type enumerant of **ann**. If an error occurs, **CG_UNKNOWN_TYPE** will be returned.

ERRORS

CG_INVALID_ANNOTATION_HANDLE_ERROR is generated if **ann** does not refer to a valid annotation.

SEE ALSO

the cgGetType manpage, and the cgGetTypeString manpage

NAME

cgGetArrayDimension − get the dimension of an array parameter

SYNOPSIS

```
#include <Cg/cg.h>

int cgGetArrayDimension( CGparameter param );
```

PARAMETERS

param Specifies the array parameter handle.

DESCRIPTION

cgGetArrayDimension returns the dimension of the array specified by **param**. This function is used when inspecting an array parameter in a program.

RETURN VALUES

Returns the dimension of **param** if **param** references an array.

Returns 0 otherwise.

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if **param** is invalid.

CG_ARRAY_PARAM_ERROR is generated if the handle **param** does not refer to an array parameter.

SEE ALSO

the cgGetNextParameter manpage, the cgGetArraySize manpage, and the cgGetArrayParameter manpage

NAME

cgGetArrayParameter − get a parameter from an array

SYNOPSIS

```
#include <Cg/cg.h>

CGparameter cgGetArrayParameter( CGparameter param, int index );
```

PARAMETERS

param Specifies the array parameter handle.

index Specifies the index into the array.

DESCRIPTION

cgGetArrayParameter returns the parameter of array **param** specified by the index. This function is used when inspecting elements of an array parameter in a program.

EXAMPLE

```
CGparameter array = ...; /* some array parameter */
int array_size = cgGetArraySize( array );
for(i=0; i < array_size; ++i)
 {
  CGparameter element = cgGetArrayParameter(array, i);
  /* Do stuff to element */
 }
```

RETURN VALUES

Returns the parameter at the specified index of **param** if **param** references an array, and the index is valid.

Returns NULL otherwise.

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if **param** is invalid.

CG_ARRAY_PARAM_ERROR is generated if **param** is not an array.

CG_OUT_OF_ARRAY_BOUNDS_ERROR is generated if **index** are outside the bounds of the array.

SEE ALSO

the cgGetArrayDimension manpage, the cgGetArraySize manpage, the cgGetArrayParameter manpage, the cgGetParameterType manpage

NAME

cgGetArraySize − get the size of one dimension of an array parameter

SYNOPSIS

```
#include <Cg/cg.h>

int cgGetArraySize( CGparameter param, int dimension );
```

PARAMETERS

param Specifies the array parameter handle.

dimension

Specifies the dimension whose size will be returned.

DESCRIPTION

cgGetArraySize returns the size one the specified dimension of the array specified by **param**. This function is used when inspecting an array parameter in a program.

EXAMPLE

```
// Compute the number of elements in an array, in the
// style of cgGetArrayTotalSize(param)
if (cgIsArray(param)) {
  int dim = cgGetArrayDimension(param);
  int elements = cgGetArraySize(param, 0);
  for (int i = 1; i < dim; i++) {
     elements *= cgGetArraySize(param, i);
  }
  return elements;
}
```

RETURN VALUES

Returns the size of **param** if **param** is an array.

Returns 0 is **param** is not an array, or other error.

ERRORS

CG_INVALID_DIMENSION_ERROR is generated if **dimension** is less than 0.

CG_INVALID_PARAM_HANDLE_ERROR is generated if **param** is an invalid parameter handle.

SEE ALSO

the cgGetArrayTotalSize manpage, the cgGetArrayDimension manpage, the cgGetArrayParameter manpage, the cgGetMatrixSize manpage, the cgGetTypeSizes manpage

NAME

cgGetArrayTotalSize − get the total size of an array parameter

SYNOPSIS

```
#include <Cg/cg.h>

int cgGetArrayTotalSize( CGparameter param );
```

PARAMETERS

param Specifies the array parameter handle.

DESCRIPTION

cgGetArrayTotalSize returns the total number of elements of the array specified by **param**. The total number of elements is equal to the product of the size of each dimension of the array.

EXAMPLE

Given a handle to a parameter declared as:

float2x3 array[6][1][4];

cgGetArrayTotalSize will return 24.

RETURN VALUES

If **param** is an array, its total size is returned.

If **param** is not an array, or if an error occurs, 0 is returned.

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if **param** is an invalid parameter handle.

SEE ALSO

the cgGetArraySize manpage, the cgGetArrayDimension manpage, the cgGetArrayParameter manpage

NAME

cgGetArrayType − get the type of an array parameter

SYNOPSIS

```
#include <Cg/cg.h>

CGtype cgGetArrayType(CGparameter param);
```

PARAMETERS

param Specifies the array parameter handle.

DESCRIPTION

cgGetArrayType returns the type of the members of an array. If the given array is multi-dimensional, it will return the type of the members of the inner most array.

EXAMPLE

```
CGcontext Context = cgCreateContext();
CGparameter MyArray = cgCreateParameterArray(Context, CG_FLOAT, 5);

CGtype MyArrayType = cgGetArrayType(MyArray); // This will return CG_FLOAT
```

RETURN VALUES

Returns the the type of the inner most array. Returns **CG_UNKNOWN_TYPE** if an error is thrown.

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if **param** is invalid.

CG_ARRAY_PARAM_ERROR is generated if **param** is not an array.

SEE ALSO

the cgGetArraySize manpage, and the cgGetArrayDimension manpage,

NAME

cgGetAutoCompile − get the auto-compile enumerant for a context

SYNOPSIS

```
#include <Cg/cg.h>

CGenum    cgGetAutoCompile(CGcontext ctx);
```

PARAMETERS

ctx Specifies the context.

DESCRIPTION

Returns the auto-compile enumerant for the context **ctx**. See cgSetAutoCompile for more information.

ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated, and **CG_UNKNOWN** returned, if **ctx** is not a valid context.

SEE ALSO

the cgSetAutoCompile manpage

NAME

cgGetBoolAnnotationValues − get an boolean-valued annotation's values

SYNOPSIS

```
#include <Cg/cg.h>

const int *cgGetBoolAnnotationValues(CGannotation ann,
                                     int *nvalues);
```

PARAMETERS

ann Specifies the annotation.

nvalues Pointer to integer where the number of values returned will be stored.

DESCRIPTION

cgBoolAnnotationValues allows the application to retrieve the *value* (s) of a boolean typed annotation.

RETURN VALUES

Returns a pobooler to an array of **int** values. The number of values in the array is returned via the **nvalues** parameter.

If no values are available, **NULL** will be returned and **nvalues** will be **0**.

ERRORS

CG_INVALID_ANNOTATION_HANDLE_ERROR is generated if the handle **ann** is invalid.

CG_INVALID_PARAMETER_ERROR is generated if **nvalues** is **NULL**.

SEE ALSO

the cgGetAnnotationType manpage, the cgGetFloatAnnotationValues manpage, the cgGetStringAnnotationValues manpage, and the cgGetBoolAnnotationValues manpage,

NAME

cgGetBoolStateAssignmentValues − get a bool-valued state assignment's values

SYNOPSIS

```
#include <Cg/cg.h>

const CGbool *cgGetBoolStateAssignmentValues(CGstateassignment sa,
                                             int *nvalues);
```

PARAMETERS

sa Specifies the state assignment.

nvalues Pointer to integer where the number of values returned will be stored.

DESCRIPTION

cgGetBoolStateAssignmentValues allows the application to retrieve the *value*(s) of a boolean typed state assignment.

RETURN VALUES

Returns a pointer to an array of **CGbool** values. The number of values in the array is returned via the **nvalues** parameter.

If no values are available, **NULL** will be returned and **nvalues** will be **0**.

ERRORS

CG_INVALID_STATE_ASSIGNMENT_HANDLE_ERROR is generated if the handle **sa** is invalid.

CG_INVALID_PARAMETER_ERROR is generated if **nvalues** is **NULL**.

CG_INVALID_PARAMETER_TYPE_ERROR is generated if the state assignment is not bool-typed.

SEE ALSO

the cgGetStateAssignmentState manpage, the cgGetStateType manpage, the cgGetFloatStateAssignmentValues manpage, the cgGetIntStateAssignmentValues manpage, the cgGetStringStateAssignmentValue manpage, the cgGetProgramStateAssignmentValue manpage, the cgGetSamplerStateAssignmentValue manpage, and the cgGetTextureStateAssignmentValue manpage

NAME

cgGetConnectedParameter − gets the connected source parameter

SYNOPSIS

```
#include <Cg/cg.h>

CGparameter cgGetConnectedParameter(CGparameter param);
```

PARAMETERS

param Specifies the destination parameter.

DESCRIPTION

Returns the source parameter of a connection to **param**.

RETURN VALUES

Returns the connected source parameter if **param** is connected to one. Otherwise **(CGparameter)0** is returned.

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if **param** does not refer to a valid parameter.

SEE ALSO

the cgConnectParameter manpage, and the cgGetConnectedToParameter manpage

NAME

cgGetConnectedToParameter − gets a connected destination parameter

SYNOPSIS

```
#include <Cg/cg.h>

CGparameter cgGetConnectedToParameter(CGparameter param, int index);
```

PARAMETERS

param Specifies the source parameter.

index Since there may be multiple destination (to) parameters connected to **param**, **index** is need to specify which one is returned. **index** must be within the range of **0** to **N − 1** where **N** is the number of connected destination parameters.

DESCRIPTION

Returns one of the destination parameters connected to **param**. cgGetNumConnectedToParameters should be used to determine the number of destination parameters connected to **param**.

EXAMPLE

```
int i;
int NParams = cgGetNumConnectedToParameters(SourceParam);

for(int i=0; i < NParams; ++i)
 {
  CGparameter ToParam = cgGetConnectedToParameter(SourceParam, i);
  // Do stuff to ToParam ...
 }
```

RETURN VALUES

Returns one of the connected destination parameters to **param**. **(CGparameter)0** is returned if an error is thrown.

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if **param** does not refer to a valid parameter.

CG_OUT_OF_ARRAY_BOUNDS_ERROR is generated if **index** is not greater than equal to **0** or less than what cgGetNumConnectedToParameters returns.

SEE ALSO

the cgConnectParameter manpage, and the cgGetNumConnectedParameters manpage

NAME

cgGetDependentAnnotationParameter − returns one of the parameters that a annotation's value depends on.

SYNOPSIS

```
#include <Cg/cg.h>

CGparameter cgGetDependentAnnotationParameter(CGannotation ann, int index);
```

PARAMETERS

ann Specifies the annotation handle.

index Specifies the index of the parameter to return.

DESCRIPTION

Annotations in CgFX files may include references to one or more effect parameters on the right hand side of the annotation that are used for computing the annotation's value. **cgGetDependentAnnotationParameter** returns one of these parameters, as indicated by the index given. the cgGetNumDependentAnnotationParameters manpage can be used to determine the total number of such parameters.

This information can be useful for applications that wish to cache the values of annotations so that they can determine which annotations may change as the result of changing a particular parameter's value.

ERRORS

CG_INVALID_ANNOTATION_HANDLE_ERROR is generated if **ann** does not refer to a valid annotation.

CG_OUT_OF_ARRAY_BOUNDS_ERROR is generated if **index** is less than zero or greater than the number of dependent parameters, as returned by the cgGetNumDependentStateAssignmentParameters manpage.

SEE ALSO

the cgGetDependentStateAssignmentParameter manpage, the cgGetFirstAnnotation manpage, the cgGetNamedAnnotation manpage, the cgGetNumDependentAnnotationParameters manpage

NAME

cgGetDependentStateAssignmentParameter − returns one of the parameters that a state assignment's value depends on.

SYNOPSIS

```
#include <Cg/cg.h>

CGparameter cgGetDependentStateAssignmentParameter(CGstateassignment sa, int index
```

PARAMETERS

sa Specifies the state assignment handle.

index Specifies the index of the parameter to return.

DESCRIPTION

State assignments in CgFX files may include references to one or more effect parameters on the right hand side of the state assignment that are used for computing the state assignment's value. **cgGetDependentStateAssignmentParameter** returns one of these parameters, as indicated by the index given. the cgGetNumDependentStateAssignmentParameters manpage can be used to determine the total number of such parameters.

This information can be useful for applications that wish to cache the values of annotations so that they can determine which annotations may change as the result of changing a particular parameter's value.

ERRORS

CG_INVALID_ANNOTATION_HANDLE_ERROR is generated if **ann** does not refer to a valid annotation.

CG_OUT_OF_ARRAY_BOUNDS_ERROR is generated if **index** is less than zero or greater than the number of dependent parameters, as returned by the cgGetNumDependentStateAssignmentParameters manpage.

SEE ALSO

the cgGetDependentAnnotationParameter manpage, the cgGetFirstAnnotation manpage, the cgGetNamedAnnotation manpage, the cgGetNumDependentStateAssignmentParameters manpage

NAME

cgGetEffectContext − get a effect's context

SYNOPSIS

```
#include <Cg/cg.h>

CGcontext cgGetEffectContext( CGeffect effect );
```

PARAMETERS

effect Specifies the effect.

DESCRIPTION

cgGetEffectContext allows the application to retrieve a handle to the context a given effect belongs to.

RETURN VALUES

Returns a **CGcontext** handle to the context. In the event of an error **NULL** is returned.

ERRORS

CG_INVALID_EFFECT_HANDLE_ERROR is generated if **effect** does not refer to a valid effect.

SEE ALSO

the cgCreateEffect manpage, the cgCreateEffectFromFile manpage, and the cgCreateContext manpage

NAME

cgGetEffectParameterBySemantic − get the a parameter in an effect via its semantic

SYNOPSIS

```
#include <Cg/cg.h>

CGparameter cgGetEffectParameterBySemantic( CGeffect effect, const char *semantic
```

PARAMETERS

effect Specifies the effect to retrieve the parameter from.

semantic
Specifies the name of the semantic.

DESCRIPTION

cgGetEffectParameterBySemantic returns the parameter in an effect with the given semantic associated with it. It more than one parameter in the effect has the same semantic, an arbitrary one of them will be returned.

RETURN VALUES

cgGetEffectParameterBySemantic returns a **CGparameter** object in **effect** that has the given semantic. **NULL** is returned if **effect** is invalid or if **effect** does not have any parameters with the given semantic.

ERRORS

CG_INVALID_EFFECT_HANDLE_ERROR is generated if **effect** does not refer to a valid effect.

CG_INVALID_PARAMETER_ERROR is generated if **semantic** is **NULL** or the empty string.

SEE ALSO

the cgGetNamedEffectParameter manpage

NAME

cgGetEnum − get the enumerant assigned with the given string name

SYNOPSIS

```
#include <Cg/cg.h>

CGenum cgGetEnum(const char *enum_string);
```

PARAMETERS

enum_string

A string containing the enum name. The name is case-sensitive.

DESCRIPTION

cgGetEnum returns the enumerant assigned to a enum name.

The following is an example of how cgGetEnum might be used.

```
CGenum VaryingEnum = cgGetEnum("CG_VARYING");
```

RETURN VALUES

Returns the enumerant of **enum_string**. If no such enumerant exists **CG_UNKNOWN** is returned.

ERRORS

CG_INVALID_PARAMETER_ERROR is generated if **enum_string** is **NULL**.

SEE ALSO

the cgGetEnumString manpage

NAME

cgGetEnumString − get the name string associated with an enumerant

SYNOPSIS

```
#include <Cg/cg.h>

const char *cgGetEnumString(CGenum en);
```

PARAMETERS

en The enumerant.

DESCRIPTION

cgGetEnumString returns the name string associated with an enumerant enumerant. It's primarily useful for printing out debugging information.

EXAMPLE

The following example will print ''CG_UNIFORM'' to the **stdout**.

```
const char *EnumString = cgGetEnumString(CG_UNIFORM);
printf("%s\n", EnumString);
```

RETURN VALUES

Returns the enum string of the enumerant **enum**. **NULL** is returned in the event of an error.

ERRORS

This function generates no errors.

SEE ALSO

the cgGetEnum manpage

NAME

cgGetError, **cgGetFirstError**, **cgGetErrorString** − get error conditions

SYNOPSIS

```
#include <Cg/cg.h>

CGerror cgGetError();
CGerror cgGetFirstError()
const char * cgGetErrorString(CGerror error);
```

PARAMETERS

error Specifies the error number.

DESCRIPTION

cgGetError returns the last error condition that has occured. The error condition is reset after **cgGetError** is called.

cgGetFirstError returns the first error condition that has occured since **cgGetFirstError** was previously called.

In both cases, an error condition of zero (or, equivalently, **CG_NO_ERROR**) means that no error has occurred.

cgGetErrorString returns a human readable error string for the given error condition.

ERRORS

SEE ALSO

the cgSetErrorHandler manpage

NAME

cgGetErrorCallBack, **cgSetErrorCallBack** − get and set the error callback

SEE

Please see the cgSetErrorCallback manpage for more information.

NAME

cgGetError, **cgGetErrorString** − get the current error condition

SEE

Please see the cgGetError manpage for more information.

NAME

cgGetFirstDependentParameter − get the first dependent parameter from a parameter

SYNOPSIS

```
#include <Cg/cg.h>

CGparameter cgGetFirstDependentParameter(CGparameter param);
```

PARAMETERS

param Specifies the parameter.

DESCRIPTION

cgGetFirstDependentParameter returns the first member dependent parameter associated with a given parameter. The rest of the members may be retrieved from the first member by iterating with the cgGetNextParameter() function.

Dependent parameters are parameters that have the same name as a given parameter but different resources. They only exist in profiles that have multiple resources associated with one parameter.

RETURN VALUES

cgGetFirstDependentParameter returns a handle to the first member parameter.

NULL is returned if **param** is not a struct or if some other error occurs.

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if **param** is not a valid parameter.

SEE ALSO

the cgGetNextParameter manpage, and the cgGetFirstParameter manpage

NAME

cgGetFirstEffect − get the first effect in a context

SYNOPSIS

```
#include <Cg/cg.h>

CGprogram cgGetFirstEffect( CGcontext ctx );
```

PARAMETERS

ctx Specifies the context to retrieve the first effect from.

DESCRIPTION

cgGetFirstEffect is used to begin iteration over all of the effects contained within a context. See the cgGetNextEffect manpage for more information.

RETURN VALUES

cgGetFirstEffect returns a the first CGeffect object in **ctx**. If **ctx** cointains no effects, **NULL** is returned.

ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if **ctx** does not refer to a valid context.

SEE ALSO

the cgGetNextEffect manpage, the cgCreateEffect manpage, the cgCreateEffectFromFile manpage, the cgDestroyEffect manpage, and the cgIsEffect manpage

NAME

cgGetFirstEffectParameter − get the first parameter in an effect

SYNOPSIS

```
#include <Cg/cg.h>

CGparameter cgGetFirstEffectParameter( CGeffect effect );
```

PARAMETERS

effect Specifies the effect to retrieve the first parameter from.

DESCRIPTION

cgGetFirstEffectParameter returns the first top-level parameter in an effect. This function is used for recursing through all parameters in an effect. See the cgGetNextParameter manpage for more information on parameter traversal.

RETURN VALUES

cgGetFirstEffectParameter returns a the first **CGparameter** object in **effect**. **NULL** is returned if **effect** is invalid or if **effect** does not have any parameters.

ERRORS

CG_INVALID_EFFECT_HANDLE_ERROR is generated if **effect** does not refer to a valid effect.

SEE ALSO

the cgNextParameter manpage

NAME

cgGetFirstLeafEffectParameter − get the first leaf parameter in an effect

SYNOPSIS

```
#include <Cg/cg.h>

CGparameter cgGetFirstLeafEffectParameter( CGeffect effect );
```

PARAMETERS

effect Specifies the effect to retrieve the first leaf parameter from.

DESCRIPTION

cgGetFirstLeafEffectParameter returns the first leaf parameter in an effect. The combination of **cgGetFirstLeafEffectParameter** and **cgGetNextLeafParameter** allow the iteration through all of the parameters of basic data types (not structs or arrays) without recursion. See the cgGetNextLeafParameter manpage for more information.

RETURN VALUES

cgGetFirstLeafEffectParameter returns a the first leaf CGparameter object in **effect**. **NULL** is returned if **effect** is invalid or if **effect** does not have any parameters.

ERRORS

CG_INVALID_EFFECT_HANDLE_ERROR is generated if **effect** does not refer to a valid effect.

SEE ALSO

the cgNextLeafParameter manpage, the cgGetFirstLeafParameter manpage

NAME

cgGetFirstLeafParameter − get the first leaf parameter in a program

SYNOPSIS

```
#include <Cg/cg.h>

CGparameter cgGetFirstLeafParameter( CGprogram prog, CGenum name_space );
```

PARAMETERS

prog Specifies the program to retrieve the first leaf parameter from.

name_space

Specifies the namespace of the parameter to iterate through. Currently **CG_PROGRAM** and **CG_GLOBAL** are supported.

DESCRIPTION

cgGetFirstLeafParameter returns the first leaf parameter in a program. The combination of cgGetFirstLeafParameter and cgGetNextLeafParameter allow the iteration through all of the parameters of basic data types (not structs or arrays) without recursion. See the cgGetNextLeafParameter manpage for more information.

RETURN VALUES

cgGetFirstLeafParameter returns a the first leaf CGparameter object in **prog**. **NULL** is returned if **prog** is invalid or if **prog** does not have any parameters.

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if **prog** does not refer to a valid program.

CG_INVALID_ENUMERANT_ERROR is generated if **name_space** is not **CG_PROGRAM** or **CG_GLOBAL**.

SEE ALSO

the cgNextLeafParameter manpage

NAME

cgGetFirstParameter − get the first parameter in a program

SYNOPSIS

```
#include <Cg/cg.h>

CGparameter cgGetFirstParameter( CGprogram prog, CGenum name_space );
```

PARAMETERS

prog Specifies the program to retrieve the first parameter from.

name_space

Specifies the namespace of the parameter to iterate through. Currently **CG_PROGRAM** and **CG_GLOBAL** are supported.

DESCRIPTION

cgGetFirstParameter returns the first top-level parameter in a program. This function is used for recursing through all parameters in a program. See the cgGetNextParameter manpage for more information on parameter traversal.

RETURN VALUES

cgGetFirstParameter returns a the first CGparameter object in **prog**. **NULL** is returned if **prog** is invalid or if **prog** does not have any parameters.

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if **prog** does not refer to a valid program.

CG_INVALID_ENUMERANT_ERROR is generated if **name_space** is not **CG_PROGRAM** or **CG_GLOBAL**.

SEE ALSO

the cgNextParameter manpage

NAME

cgGetFirstParameterAnnotation − get the first annotation of a parameter

SYNOPSIS

```
#include <Cg/cg.h>

CGannotation cgGetFirstParameterAnnotation( CGparameter param );
```

PARAMETERS

param Specifies the parameter to retrieve the annotation from.

DESCRIPTION

The annotations associated with a parameter can be retrieved using the **cgGetFirstParameterAnnotation** function. The remainder of the parameter's annotations can be discovered by iterating through the parameters, calling the cgGetNextAnnotation manpage to get to the next one.

RETURN VALUES

Returns the first annotation. If the parameter has no annotations, **NULL** is returned.

ERRORS

CG_INVALID_PARAMETER_HANDLE_ERROR is generated if **param** does not refer to a valid parameter.

SEE ALSO

the cgGetNamedParameterAnnotation manpage, the cgGetNextAnnotation manpage

NAME

cgGetFirstPass − get the first pass in a technique

SYNOPSIS

```
#include <Cg/cg.h>

CGpass cgGetFirstPass( CGtechnique technique );
```

PARAMETERS

technique

Specifies the technique to retrieve the first pass from.

DESCRIPTION

cgGetFirstPass is used to begin iteration over all of the passs contained within a technique. See the cgGetNextPass manpage for more information.

RETURN VALUES

cgGetFirstPass returns a the first CGpass object in **technique**. If **technique** cointains no passs, **NULL** is returned.

ERRORS

CG_INVALID_TECHNIQUE_HANDLE_ERROR is generated if **technique** does not refer to a valid technique.

SEE ALSO

the cgGetNextPass manpage, the cgGetNamedPass manpage, and the cgIsPass manpage

NAME

cgGetFirstPassAnnotation − get the first annotation of a pass

SYNOPSIS

```
#include <Cg/cg.h>

CGannotation cgGetFirstPassAnnotation( CGpass pass );
```

PARAMETERS

pass Specifies the pass to retrieve the annotation from.

DESCRIPTION

The annotations associated with a pass can be retrieved using the **cgGetFirstPassAnnotation** function. The remainder of the pass's annotations can be discovered by iterating through the parameters, calling the cgGetNextAnnotation manpage to get to the next one.

RETURN VALUES

Returns the first annotation. If the pass has no annotations, **NULL** is returned.

ERRORS

CG_INVALID_PASS_HANDLE_ERROR is generated if **pass** does not refer to a valid pass.

SEE ALSO

the cgGetNamedPassAnnotation manpage, the cgGetNextAnnotation manpage

NAME

cgGetFirstProgram − get the first program in a context

SYNOPSIS

```
#include <Cg/cg.h>

CGprogram cgGetFirstProgram( CGcontext ctx );
```

PARAMETERS

ctx Specifies the context to retrieve the first program from.

DESCRIPTION

cgGetFirstProgram is used to begin iteration over all of the programs contained within a context. See the cgGetNextProgram manpage for more information.

RETURN VALUES

cgGetFirstProgram returns a the first CGprogram object in **ctx**. If **ctx** cointains no programs, **NULL** is returned.

ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if **ctx** does not refer to a valid context.

SEE ALSO

the cgGetNextProgram manpage, the cgCreateProgram manpage, the cgDestroyProgram manpage, and the cgIsProgram manpage

NAME

cgGetFirstProgramAnnotation − get the first annotation of a program

SYNOPSIS

```
#include <Cg/cg.h>

CGannotation cgGetFirstProgramAnnotation( CGprogram program );
```

PARAMETERS

program Specifies the program to retrieve the annotation from.

DESCRIPTION

The annotations associated with a program can be retrieved using the **cgGetFirstProgramAnnotation** function. The remainder of the program's annotations can be discovered by iterating through the parameters, calling the cgGetNextAnnotation manpage to get to the next one.

RETURN VALUES

Returns the first annotation. If the program has no annotations, **NULL** is returned.

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if **program** does not refer to a valid program.

SEE ALSO

the cgGetNamedProgramAnnotation manpage, the cgGetNextAnnotation manpage

NAME

cgGetFirstSamplerState − get the first sampler state definition in a context

SYNOPSIS

```
#include <Cg/cg.h>

CGstate cgGetFirstSamplerState( CGcontext ctx );
```

PARAMETERS

ctx Specifies the context to retrieve the first sampler state definition from.

DESCRIPTION

cgGetFirstSamplerState is used to begin iteration over all of the sampler state definitions contained within a context. See the cgGetNextSamplerState manpage for more information.

RETURN VALUES

cgGetFirstSamplerState returns a the first CGstate object in **ctx**. If **ctx** cointains no programs, **NULL** is returned.

ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if **ctx** does not refer to a valid context.

SEE ALSO

the cgGetNextSamplerState manpage, the cgGetNamedSamplerState manpage, and the cgIsSamplerState manpage

NAME

cgGetFirstSamplerStateAssignment − get the first state assignment in a sampler_state block

SYNOPSIS

```
#include <Cg/cg.h>

CGstateassignment cgGetFirstSamplerStateAssignment( CGparameter param  );
```

PARAMETERS

param Specifies the sampler parameter to retrieve the first state assignment from.

DESCRIPTION

cgGetFirstSamplerStateAssignment is used to begin iteration over all of the state assignments contained within a **sampler_state** block assigned to a parameter in an effect file. See the cgGetNextStateAssignment manpage for more information.

RETURN VALUES

cgGetFirstSamplerStateAssignment returns a the first **CGstateassignment** object assigned to **param**. If **param** has no **sampler_state** block, **NULL** is returned.

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if **param** does not refer to a valid parameter.

SEE ALSO

the cgGetNextStateAssignment manpage, and the cgIsStateAssignment manpage

NAME

cgGetFirstState − get the first state definition in a context

SYNOPSIS

```
#include <Cg/cg.h>

CGstate cgGetFirstState( CGcontext ctx );
```

PARAMETERS

ctx Specifies the context to retrieve the first state definition from.

DESCRIPTION

cgGetFirstState is used to begin iteration over all of the state definitions contained within a context. See the cgGetNextState manpage for more information.

RETURN VALUES

cgGetFirstState returns a the first CGstate object in **ctx**. If **ctx** cointains no programs, **NULL** is returned.

ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if **ctx** does not refer to a valid context.

SEE ALSO

the cgGetNextState manpage, the cgGetNamedState manpage, and the cgIsState manpage

NAME

cgGetFirstStateAssignment − get the first state assignment in a pass

SYNOPSIS

```
#include <Cg/cg.h>

CGstateassignment cgGetFirstStateAssignment( CGpass pass );
```

PARAMETERS

pass Specifies the pass to retrieve the first state assignment from.

DESCRIPTION

cgGetFirstStateAssignment is used to begin iteration over all of the state assignment contained within a pass. See the cgGetNextStateAssignment manpage for more information.

RETURN VALUES

cgGetFirstStateAssignment returns a the first CGstateassignment object in **pass**. If **pass** cointains no programs, **NULL** is returned.

ERRORS

CG_INVALID_PASS_HANDLE_ERROR is generated if **pass** does not refer to a valid pass.

SEE ALSO

the cgGetNextStateAssignment manpage, and the cgIsStateAssignment manpage

NAME

cgGetFirstStructParameter − get the first child parameter from a struct parameter

SYNOPSIS

```
#include <Cg/cg.h>

CGparameter cgGetFirstStructParameter(CGparameter param);
```

PARAMETERS

param Specifies the struct parameter. This parameter must be of type **CG_STRUCT** (returned by cgGetParameterType()).

DESCRIPTION

cgGetFirstStructParameter returns the first member parameter of a struct parameter. The rest of the members may be retrieved from the first member by iterating with the cgGetNextParameter() function.

RETURN VALUES

cgGetFirstStructParameter returns a handle to the first member parameter.

NULL is returned if **param** is not a struct or if some other error occurs.

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if **param** is not a struct or valid parameter.

SEE ALSO

the cgGetNextParameter manpage, and the cgGetFirstParameter manpage

NAME

cgGetFirstTechnique − get the first technique in an effect

SYNOPSIS

```
#include <Cg/cg.h>

CGtechnique cgGetFirstTechnique( CGeffect effect );
```

PARAMETERS

effect Specifies the effect to retrieve the first technique from.

DESCRIPTION

cgGetFirstTechnique is used to begin iteration over all of the techniques contained within a effect. See the cgGetNextTechnique manpage for more information.

RETURN VALUES

cgGetFirstTechnique returns a the first CGtechnique object in **effect**. If **effect** cointains no techniques, **NULL** is returned.

ERRORS

CG_INVALID_EFFECT_HANDLE_ERROR is generated if **effect** does not refer to a valid effect.

SEE ALSO

the cgGetNextTechnique manpage, the cgGetNamedTechnique manpage, and the cgIsTechnique manpage

NAME

cgGetFirstTechniqueAnnotation − get the first annotation of a technique

SYNOPSIS

```
#include <Cg/cg.h>

CGannotation cgGetFirstTechniqueAnnotation( CGtechnique tech );
```

PARAMETERS

tech Specifies the technique to retrieve the annotation from.

DESCRIPTION

The annotations associated with a technique can be retrieved using the **cgGetFirstTechniqueAnnotation** function. The remainder of the technique's annotations can be discovered by iterating through the parameters, calling the cgGetNextAnnotation manpage to get to the next one.

RETURN VALUES

Returns the first annotation. If the technique has no annotations, **NULL** is returned.

ERRORS

CG_INVALID_TECHNIQUE_HANDLE_ERROR is generated if **tech** does not refer to a valid technique.

SEE ALSO

the cgGetNamedTechniqueAnnotation manpage, the cgGetNextAnnotation manpage

NAME

cgGetFloatAnnotationValues − get a float-valued annotation's values

SYNOPSIS

```
#include <Cg/cg.h>

const float *cgGetFloatAnnotationValues(CGannotation ann,
                                        int *nvalues);
```

PARAMETERS

ann Specifies the annotation.

nvalues Pointer to integer where the number of values returned will be stored.

DESCRIPTION

cgFloatAnnotationValues allows the application to retrieve the *value*(s) of a floating-point typed annotation.

RETURN VALUES

Returns a pointer to an array of **float** values. The number of values in the array is returned via the **nvalues** parameter.

If no values are available, **NULL** will be returned and **nvalues** will be **0**.

ERRORS

CG_INVALID_ANNOTATION_HANDLE_ERROR is generated if the handle **ann** is invalid.

CG_INVALID_PARAMETER_ERROR is generated if **nvalues** is **NULL**.

SEE ALSO

the cgGetAnnotationType manpage, the cgGetIntAnnotationValues manpage, the cgGetStringAnnotationValues manpage, and the cgGetBooleanAnnotationValues manpage,

NAME

cgGetFloatStateAssignmentValues − get a float-valued state assignment's values

SYNOPSIS

```
#include <Cg/cg.h>

const float *cgGetFloatStateAssignmentValues(CGstateassignment sa,
                                             int *nvalues);
```

PARAMETERS

sa Specifies the state assignment.

nvalues Pointer to integer where the number of values returned will be stored.

DESCRIPTION

cgGetFloatStateAssignmentValues allows the application to retrieve the *value* (s) of a floating-point typed state assignment.

RETURN VALUES

Returns a pointer to an array of **float** values. The number of values in the array is returned via the **nvalues** parameter.

If no values are available, **NULL** will be returned and **nvalues** will be **0**.

ERRORS

CG_INVALID_STATE_ASSIGNMENT_HANDLE_ERROR is generated if the handle **sa** is invalid.

CG_INVALID_PARAMETER_ERROR is generated if **nvalues** is **NULL**.

CG_INVALID_PARAMETER_TYPE_ERROR is generated if the state assignment is not float-typed.

SEE ALSO

the cgGetStateAssignmentState manpage, the cgGetStateType manpage, the cgGetIntStateAssignmentValues manpage, the cgGetBoolStateAssignmentValues manpage, the cgGetStringStateAssignmentValue manpage, the cgGetProgramStateAssignmentValue manpage, the cgGetSamplerStateAssignmentValue manpage, and the cgGetTextureStateAssignmentValue manpage

NAME

cgGetIntAnnotationValues − get an integer-valued annotation's values

SYNOPSIS

```
#include <Cg/cg.h>

const int *cgGetIntAnnotationValues(CGannotation ann,
                                    int *nvalues);
```

PARAMETERS

ann Specifies the annotation.

nvalues Pointer to integer where the number of values returned will be stored.

DESCRIPTION

cgIntAnnotationValues allows the application to retrieve the *value* (s) of an int typed annotation.

RETURN VALUES

Returns a pointer to an array of **int** values. The number of values in the array is returned via the **nvalues** parameter.

If no values are available, **NULL** will be returned and **nvalues** will be **0**.

ERRORS

CG_INVALID_ANNOTATION_HANDLE_ERROR is generated if the handle **ann** is invalid.

CG_INVALID_PARAMETER_ERROR is generated if **nvalues** is **NULL**.

SEE ALSO

the cgGetAnnotationType manpage, the cgGetFloatAnnotationValues manpage, the cgGetStringAnnotationValues manpage, and the cgGetBooleanAnnotationValues manpage,

NAME

cgGetIntStateAssignmentValues − get an int-valued state assignment's values

SYNOPSIS

```
#include <Cg/cg.h>

const int *cgGetIntStateAssignmentValues(CGstateassignment sa,
                                         int *nvalues);
```

PARAMETERS

sa Specifies the state assignment.

nvalues Pointer to integer where the number of values returned will be stored.

DESCRIPTION

cgGetIntStateAssignmentValues allows the application to retrieve the *value*(s) of an integer typed state assignment.

RETURN VALUES

Returns a pointer to an array of **int** values. The number of values in the array is returned via the **nvalues** parameter.

If no values are available, **NULL** will be returned and **nvalues** will be **0**.

ERRORS

CG_INVALID_STATE_ASSIGNMENT_HANDLE_ERROR is generated if the handle **sa** is invalid.

CG_INVALID_PARAMETER_ERROR is generated if **nvalues** is **NULL**.

CG_INVALID_PARAMETER_TYPE_ERROR is generated if the state assignment is not integer-typed.

SEE ALSO

the cgGetStateAssignmentState manpage, the cgGetStateType manpage, the cgGetFloatStateAssignmentValues manpage, the cgGetBoolStateAssignmentValues manpage, the cgGetStringStateAssignmentValue manpage, the cgGetProgramStateAssignmentValue manpage, the cgGetSamplerStateAssignmentValue manpage, and the cgGetTextureStateAssignmentValue manpage

NAME

cgGetLastErrorString − get the current error condition

SYNOPSIS

```
#include <Cg/cg.h>

const char *cgGetLastErrorString(CGerror *error);
```

PARAMETERS

error A pointer to a CGerror variable for returning the last error code.

DESCRIPTION

cgGetLastErrorString returns the current error condition and error condition string. It's similar to calling cgGetErrorString with the result of cgGetLastError. However in certain cases the error string may contain more information about the specific error that ocurred last than what cgGetErrorString would return.

RETURN VALUES

Returns the last error string. Returns **NULL** if there was no error.

If **error** is not null, it will return the last error code (the same value cgGetError would return in the location specified by **error**.

ERRORS

This function generates no errors.

SEE ALSO

the cgGetError manpage

NAME

cgGetLastListing, **cgSetListing** − get and set listing text

SYNOPSIS

```
#include <Cg/cg.h>

const char * cgGetLastListing( CGcontext context );
void cgSetLastListing( CGhandle handle, const char *listing );
```

PARAMETERS

context Specifies the context handle.

handle A CGcontext, CGstateassignment, CGeffect, CGpass, or CGtechnique belonging to the context whose listing text is to be set.

listing Specifies the new listing text.

DESCRIPTION

Each Cg context maintains a NULL-terminated string containing warning and error messages generated by the Cg compiler, state managers and the like. **cgGetlastListing** and **cgSetlastListing** allow applications and custom state managers to query and set the listing text.

cgGetLastListing returns the currrent listing string for the given **CGcontext**. When a Cg runtime error occurs, applications can use the appropriate context's listing text in order to provide the user with detailed information about the error.

cgSetLastListing is not normally used directly by applications. Instead, custom state managers can use **cgSetLastListing** to provide detailed technique validation error messages to the application.

RETURN VALUES

cgGetListListing return a NULL-terminated string containing the current listing text. If no listing text is available, or the listing text string is empty, **NULL** is returned.

In all cases, the pointer returned by **cgGetLastListing** is only guaranteed to be valid until the next Cg entry point not related to error reporting is called. For example, calls to **cgCreateProgram**, **cgCompileProgram**, **cgCreateEffect**, or **cgValidateEffect** will invalidate any previously-returned listing pointer.

ERRORS

CG_INVALID_PARAMETER_ERROR is generated if **context** or **handle** are invalid handles.

SEE ALSO

the cgCreateContext manpage, the cgSetErrorHandler manpage

NAME

cgGetMatrixParameter − gets the value of matrix parameters

SYNOPSIS

```
#include <Cg/cg.h>

/* type is one of int, float, or double */
cgGetMatrixParameter{ifd}{rc}(CGparameter param, type *matrix);
```

PARAMETERS

param Specifies the parameter for which the values will be returned.

matrix An array of values into which the parameter's value will be written. The array must have size equal to the number of rows in the matrix times the number of columns in the matrix.

DESCRIPTION

The cgGetMatrixParameter functions retrieve the value of a given matrix parameter. The functions are available in various combinations.

There are versions of each function that take **int**, **float** or **double** values signified by the **i**, **f** or **d** in the function name.

There are versions of each function that specify the order in which matrix values should be written to the array. Row-major copying is indicated by **r**, while column-major is indicated by **c**.

RETURN VALUES

The cgGetMatrixParameter functions do not return any values.

ERRORS

CG_NOT_MATRIX_PARAM_ERROR is generated if **param** is not a matrix parameter.

CG_INVALID_PARAM_HANDLE_ERROR is generated if **param** is not a valid parameter handle.

SEE ALSO

the cgGetParameterRows manpage the cgGetParameterColumsn manpage, the cgGetMatrixParameterArray manpage, the cgGetParameterValues manpage

NAME

cgGetNamedParameter − get a program parameter by name

SYNOPSIS

```
#include <Cg/cg.h>

CGparameter cgGetNamedParameter( CGprogram prog, const char * name );
```

PARAMETERS

prog Specifies the program to retrieve the parameter from.

name Specifies the name of the parameter to retrieve.

DESCRIPTION

The parameters of a program can be retrieved directly by name using the cgGetNamedParameter function. The names of the parameters in a program can be discovered by iterating through the program's parameters (see cgGetNextParameter), calling cgGetParameterName for each one in turn.

The parameter name does not have to be complete name for a leaf node parameter. For example, if you have Cg program with the following parameters :

```
struct FooStruct
 {
  float4 A;
  float4 B;
 };

struct BarStruct
 {
  FooStruct Foo[2];
 };

void main(BarStruct Bar[3])
 {
  // ...
 }
```

The following leaf-node parameters will be generated :

```
Bar[0].Foo[0].A
Bar[0].Foo[0].B
Bar[0].Foo[1].A
Bar[0].Foo[1].B
Bar[1].Foo[0].A
Bar[1].Foo[0].B
Bar[1].Foo[1].A
Bar[1].Foo[1].B
Bar[2].Foo[0].A
Bar[2].Foo[0].B
Bar[2].Foo[1].A
Bar[2].Foo[1].B
```

A handle to any of the non-leaf arrays or structs can be directly obtained by using the appropriate name. The following are a few examples of names valid names that may be used with cgGetNamedParameter given the above example Cg program :

```
"Bar"
"Bar[1]"
"Bar[1].Foo"
"Bar[1].Foo[0]"
"Bar[1].Foo[0].B"
...
```

RETURN VALUES

Returns the named parameter from the program. If the program has no parameter corresponding to **name**, a **NULL** is returned (*cgIsParameter()* returns CG_FALSE for invalid parameters).

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if **prog** does not refer to a valid program.

SEE ALSO

the cgIsParameter manpage, the cgGetFirstParameter manpage, the cgGetNextParameter manpage, the cgGetNextStructParameter manpage, the cgGetArrayParameter manpage, the cgGetParameterName manpage

NAME

cgGetNamedParameterAnnotation − get a parameter annotation by name

SYNOPSIS

```
#include <Cg/cg.h>

CGannotation cgGetNamedParameterAnnotation( CGparameter param, const char * name )
```

PARAMETERS

param Specifies the parameter to retrieve the annotation from.

name Specifies the name of the annotation to retrieve.

DESCRIPTION

The annotations associated with a parameter can be retrieved directly by name using the **cgGetNamedParameterAnnotation** function. The names of a parameter's annotations can be discovered by iterating through the annotations (see the cgGetFirstParameterAnnotation manpage and the cgGetNextAnnotation manpage), calling **cgGetAnnotationName** for each one in turn.

RETURN VALUES

Returns the named annotation. If the parameter has no annotation corresponding to **name**, **NULL** is returned.

ERRORS

CG_INVALID_PARAMETER_HANDLE_ERROR is generated if **param** does not refer to a valid parameter.

SEE ALSO

the cgGetFirstParameterAnnotation manpage, the cgGetNextParameterAnnotation manpage, the cgGetAnnotationName manpage

NAME

cgGetNamedPass − get a technique pass by name

SYNOPSIS

```
#include <Cg/cg.h>

CGpass cgGetNamedPass( CGtechnique tech, const char * name );
```

PARAMETERS

tech Specifies the technique to retrieve the pass from.

name Specifies the name of the pass to retrieve.

DESCRIPTION

The passes of a technique can be retrieved directly by name using the **cgGetNamedPass** function. The names of the passes in a technique can be discovered by iterating through the technique's passes (see the cgGetFirstPass manpage and the cgGetNextPass manpage), calling **cgGetPassName** for each one in turn.

RETURN VALUES

Returns the named pass from the technique. If the technique has no pass corresponding to **name**, **NULL** is returned.

ERRORS

CG_INVALID_TECHNIQUE_HANDLE_ERROR is generated if **tech** does not refer to a valid technique.

SEE ALSO

the cgIsPass manpage, the cgGetFirstPass manpage, the cgGetNextPass manpage, the cgGetPassName manpage

NAME

cgGetNamedPassAnnotation − get a pass annotation by name

SYNOPSIS

```
#include <Cg/cg.h>

CGannotation cgGetNamedPassAnnotation( CGpass pass, const char * name );
```

PARAMETERS

pass Specifies the pass to retrieve the annotation from.

name Specifies the name of the annotation to retrieve.

DESCRIPTION

The annotations associated with a pass can be retrieved directly by name using the **cgGetNamedPassAnnotation** function. The names of a pass's annotations can be discovered by iterating through the annotations (see the cgGetFirstPassAnnotation manpage and the cgGetNextAnnotation manpage), calling **cgGetAnnotationName** for each one in turn.

RETURN VALUES

Returns the named annotation. If the pass has no annotation corresponding to **name**, **NULL** is returned.

ERRORS

CG_INVALID_PASS_HANDLE_ERROR is generated if **pass** does not refer to a valid pass.

SEE ALSO

the cgGetFirstPassAnnotation manpage, the cgGetNextPassAnnotation manpage, the cgGetAnnotationName manpage

NAME

cgGetNamedProgramAnnotation − get a program annotation by name

SYNOPSIS

```
#include <Cg/cg.h>

CGannotation cgGetNamedProgramAnnotation( CGprogram prog, const char * name );
```

PARAMETERS

prog Specifies the program to retrieve the annotation from.

name Specifies the name of the annotation to retrieve.

DESCRIPTION

The annotations associated with a program can be retrieved directly by name using the **cgGetNamedProgramAnnotation** function. The names of a program's annotations can be discovered by iterating through the annotations (see the cgGetFirstProgramAnnotation manpage and the cgGetNextAnnotation manpage), calling **cgGetAnnotationName** for each one in turn.

RETURN VALUES

Returns the named annotation. If the program has no annotation corresponding to **name**, **NULL** is returned.

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if **prog** does not refer to a valid program.

SEE ALSO

the cgGetFirstProgramAnnotation manpage, the cgGetNextProgramAnnotation manpage, the cgGetAnnotationName manpage

NAME

cgGetNamedProgramParameter − get a program parameter by name

SYNOPSIS

```
#include <Cg/cg.h>

CGparameter cgGetNamedProgramParameter(CGprogram prog,
                                       CGenum name_space,
                                       const char *name);
```

PARAMETERS

prog Specifies the program to retrieve the parameter from.

name_space
: Specifies the namespace of the parameter to iterate through. Currently **CG_PROGRAM** and **CG_GLOBAL** are supported.

name Specifies the name of the parameter to retrieve.

DESCRIPTION

cgGetNamedProgramParameter is essentially identical to the cgGetNamedParameter manpage except it limits the search of the parameter to the name space specified by **name_space**.

RETURN VALUES

Returns the named parameter from the program. If the program has no parameter corresponding to **name**, a **NULL** is returned (*cgIsParameter()* returns CG_FALSE for invalid parameters).

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if **prog** does not refer to a valid program.

SEE ALSO

the cgGetNamedParameter manpage

NAME

cgGetNamedSamplerStateAssignment − get a sampler state assignment by name

SYNOPSIS

```
#include <Cg/cg.h>

CGstateassignment cgGetNamedSamplerStateAssignment( CGparameter param, const char
```

PARAMETERS

param Specifies the sampler parameter to retrieve the sampler state assignment from.

name Specifies the name of the state assignment to retrieve.

DESCRIPTION

The sampler state assignments associated with a **sampler** parameter, as specified with a **sampler_state** block in an effect file, can be retrieved directly by name using the **cgGetNamedSamplerStateAssignment** function. The names of the sampler state assignments can be discovered by iterating through the sampler's state assignments (see the cgGetFirstSamplerStateAssignment manpage and the cgGetNextSamplerStateAssignment manpage), calling **cgGetSamplerStateAssignmentName** for each one in turn.

RETURN VALUES

Returns the named sampler state assignment. If the pass has no sampler state assignment corresponding to **name**, **NULL** is returned.

ERRORS

CG_INVALID_PARAMETER_HANDLE_ERROR is generated if **param** does not refer to a valid parameter.

SEE ALSO

the cgIsSamplerStateAssignment manpage, the cgGetFirstSamplerStateAssignment manpage, the cgGetNextSamplerStateAssignment manpage, the cgGetSamplerStateAssignmentName manpage

NAME

cgGetNamedStateAssignment − get a pass state assignment by name

SYNOPSIS

```
#include <Cg/cg.h>

CGstateassignment cgGetNamedStateAssignment( CGpass pass, const char * name );
```

STATE ASSIGNMENTS

pass Specifies the pass to retrieve the state assignment from.

name Specifies the name of the state assignment to retrieve.

DESCRIPTION

The state assignments of a pass can be retrieved directly by name using the **cgGetNamedStateAssignment** function. The names of the state assignments in a pass can be discovered by iterating through the pass's state assignments (see the cgGetFirstStateAssignment manpage and the cgGetNextStateAssignment manpage), calling **cgGetStateAssignmentName** for each one in turn.

RETURN VALUES

Returns the named state assignment from the pass. If the pass has no state assignment corresponding to **name**, **NULL** is returned.

ERRORS

CG_INVALID_PASS_HANDLE_ERROR is generated if **pass** does not refer to a valid pass.

SEE ALSO

the cgIsStateAssignment manpage, the cgGetFirstStateAssignment manpage, the cgGetNextStateAssignment manpage, the cgGetStateAssignmentName manpage

NAME

cgGetNamedParameter − get a struct parameter by name

SYNOPSIS

```
#include <Cg/cg.h>

CGparameter cgGetNamedStructParameter(CGprogram param, const char *name);
```

PARAMETERS

prog Specifies the struct parameter to retrieve the member parameter from.

name Specifies the name of the member parameter to retrieve.

DESCRIPTION

The member parameters of a struct parameter my be retrieved directly by name using the cgGetNamedStructParameter function.

The names of the parameters in a struct may be discovered by iterating through the struct's member parameters (see cgGetFirstStructParameter), and calling cgGetParameterName for each one in turn.

RETURN VALUES

Returns the member parameter from of the given struct. If the struct has no member parameter corresponding to **name**, a **NULL** is returned (*cgIsParameter()* returns **CG_FALSE** for invalid parameters).

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if **param** does not refer to a valid parameter.

CG_INVALID_PARAMETER_ERROR is generated if **name** is **NULL**.

SEE ALSO

the cgGetFirstStructParameter manpage, the cgGetNextParameter manpage, the cgIsParameter manpage, and the cgGetParameterName manpage

NAME

cgGetNamedTechnique − get an effect's technique by name

SYNOPSIS

```
#include <Cg/cg.h>

CGtechnique cgGetNamedTechnique( CGeffect effect, const char * name );
```

PARAMETERS

effect Specifies the effect to retrieve the technique from.

name Specifies the name of the technique to retrieve.

DESCRIPTION

The techniques of an effect can be retrieved directly by name using the **cgGetNamedTechnique** function. The names of the techniques in a effect can be discovered by iterating through the effect's techniques (see the cgGetFirstTechnique manpage and the cgGetNextTechnique manpage), calling **cgGetTechniqueName** for each one in turn.

RETURN VALUES

Returns the named technique from the effect. If the effect has no technique corresponding to **name**, **NULL** is returned.

ERRORS

CG_INVALID_EFFECT_HANDLE_ERROR is generated if **effect** does not refer to a valid effect.

SEE ALSO

the cgIsTechnique manpage, the cgGetFirstTechnique manpage, the cgGetNextTechnique manpage, the cgGetTechniqueName manpage

NAME

cgGetNamedTechniqueAnnotation − get a technique annotation by name

SYNOPSIS

```
#include <Cg/cg.h>

CGannotation cgGetNamedTechniqueAnnotation( CGtechnique tech, const char * name );
```

PARAMETERS

tech Specifies the technique to retrieve the annotation from.

name Specifies the name of the annotation to retrieve.

DESCRIPTION

The annotations associated with a technique can be retrieved directly by name using the **cgGetNamedTechniqueAnnotation** function. The names of a technique's annotations can be discovered by iterating through the annotations (see the cgGetFirstTechniqueAnnotation manpage and the cgGetNextAnnotation manpage), calling **cgGetAnnotationName** for each one in turn.

RETURN VALUES

Returns the named annotation. If the technique has no annotation corresponding to **name**, **NULL** is returned.

ERRORS

CG_INVALID_TECHNIQUE_HANDLE_ERROR is generated if **tech** does not refer to a valid technique.

SEE ALSO

the cgGetFirstTechniqueAnnotation manpage, the cgGetNextTechniqueAnnotation manpage, the cgGetAnnotationName manpage

NAME

cgGetNamedUserType − get enumerant associated with type name

SYNOPSIS

```
#include <Cg/cg.h>

CGtype cgGetNamedUserType(CGhandle handle, const char *name);
```

PARAMETERS

handle The **CGprogram** or **CGeffect** in which the type is defined.

name A string containing the type name. The name is case-sensitive.

DESCRIPTION

cgGetNamedUserType returns the enumerant associated with the named type defined in the constuct associated with **handle**. **handle** may be a **CGprogram** or **CGeffect**.

For a given type name, the enumerant returned by this entry point is guaranteed to be identical if called wither either an **CGeffect** handle, or a **CGprogram** that is defined within that effect.

If two programs in the same context define a type using identical names and definitions, the associated enumerants are also guaranteed to be identical.

RETURN VALUES

Returns the type enumerant associated with **name**. If no such type exists **CG_UNKNOWN_TYPE** will be returned.

ERRORS

CG_INVALID_PARAMETER_ERROR is generated if the supplied handle is not a valid **CGprogram** or **CGeffect**.

SEE ALSO

the cgGetUserType manpage, and the cgGetType manpage

NAME

cgGetNextAnnotation − iterate through annotations

SYNOPSIS

```
#include <Cg/cg.h>

CGannotation cgGetNextAnnotation( CGannotation ann );
```

PARAMETERS

ann Specifies the current annotation.

DESCRIPTION

The annotations associated with a parameter, pass, technique, or program can be iterated over by using the cgGetNextAnnotation function. The following example code illustrates one way to do this:

```
CGannotation ann = cgGetFirstParameterAnnotation( param );
while( ann )
{
 /* do something with ann */
 ann = cgGetNextAnnotation( ann );
}
```

Note that no specific order of traversal is defined by this mechanism. The only guarantee is that each annotation will be visited exactly once.

RETURN VALUES

cgGetNextAnnotation returns the next annotation in the sequence of annotations associated with the annotated object. Returns 0 when **prog** is the last annotation.

ERRORS

CG_INVALID_ANNOTATION_HANDLE_ERROR is generated if **prog** does not refer to a valid annotation.

SEE ALSO

the cgGetFirstParameterAnnotation manpage, the cgGetFirstPassAnnotation manpage, the cgGetFirstTechniqueAnnotation manpage, the cgGetFirstProgramAnnotation manpage, the cgGetNamedParameterAnnotation manpage, the cgGetNamedPassAnnotation manpage, the cgGetNamedTechniqueAnnotation manpage, the cgGetNamedProgramAnnotation manpage, and the cgIsAnnotation manpage

NAME

cgGetNextEffect − iterate through effects in a context

SYNOPSIS

```
#include <Cg/cg.h>

CGeffect cgGetNextEffect( CGeffect effect );
```

PARAMETERS

effect Specifies the current effect.

DESCRIPTION

The effects within a context can be iterated over by using the cgGetNextEffect function. The following example code illustrates one way to do this:

```
CGeffect effect = cgGetFirstEffect( ctx );
while( effect )
{
 /* do something with effect */
 effect = cgGetNextEffect( effect );
}
```

Note that no specific order of traversal is defined by this mechanism. The only guarantee is that each effect will be visited exactly once. No guarantees can be made if effects are created or deleted during iteration.

RETURN VALUES

cgGetNextEffect returns the next effect in the context's internal sequence of effects. Returns 0 when **prog** is the last effect in the context.

ERRORS

CG_INVALID_EFFECT_HANDLE_ERROR is generated if **effect** does not refer to a valid effect.

SEE ALSO

the cgGetFirstEffect manpage,

NAME

cgGetNextLeafParameter − get the next leaf parameter in a program or effect

SYNOPSIS

```
#include <Cg/cg.h>

CGparameter cgGetNextLeafParameter( CGparameter param );
```

PARAMETERS

param Specifies the current leaf parameter.

DESCRIPTION

cgGetNextLeafParameter returns the next leaf parameter (not struct or array parameters) following a given leaf parameter.

The following is an example of how to iterate through all the leaf parameters in a program:

```
CGparameter leaf = cgGetFirstLeafParameter( prog );
while(leaf)
 {
  /* Do stuff with leaf */
  leaf = cgGetNextLeafParameter( leaf );
 }
```

In a similar manner, the leaf parameters in an effect can be iterated over starting with a call to the cgGetFirstLeafEffectParameter manpage.

RETURN VALUES

the cgGetNextLeafParameter manpage returns a the next leaf **CGparameter** object. **NULL** is returned if **param** is invalid or if the program or effect that iteration started from does not have any more leaf parameters.

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if **param** does not refer to a valid parameter.

SEE ALSO

the cgGetFirstLeafParameter manpage, the cgGetFirstLeafEffectParameter manpage

NAME

cgGetNextParameter − iterate through a program's or effect's parameters

SYNOPSIS

```
#include <Cg/cg.h>

CGparameter cgGetNextParameter( CGparameter current );
```

PARAMETERS

current Specifies the ''current'' parameter.

DESCRIPTION

The parameters of a program or effect can be iterated over using the **cgGetNextParameter**, the cgGetFirstParameter manpage, the cgGetNextStructParameter manpage, and the cgGetArrayParameter manpage functions.

The following example code illustrates one way to do this:

```
void RecurseParams( CGparameter param ) {
  if(!param)
   return;

  do {
    switch(cgGetParameterType(param))
     {
      case CG_STRUCT :
       RecurseParams(cgGetFirstStructParameter(param));
       break;

      case CG_ARRAY :
       {
        int ArraySize = cgGetArraySize(param, 0);
        int i;

        for(i=0; i < ArraySize; ++i)
         RecurseParams(cgGetArrayParameter(param, i));
       }
       break;

      default :
        /* Do stuff to param */
    }
   } while((param = cgGetNextParameter(param)) != 0);
}

void RecurseParamsInProgram( CGprogram prog )
 {
  RecurseParams( cgGetFirstParameter( prog ) );
 }
```

Similarly, the parameters in an effect can be iterated over starting with a call to the cgGetFirstEffectParameter manpage.

Note that no specific order of traversal is defined by this mechanism. The only guarantee is that each parameter will be visited exactly once.

RETURN VALUES

cgGetNextParameter returns the next parameter in the program's internal sequence of parameters. It returns **NULL** when **current** is the last parameter in the program.

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if **current** does not refer to a valid parameter.

SEE ALSO

the cgFirstParameter manpage, the cgFirstEffectParameter manpage, the cgGetFirstStructParameter manpage, the cgGetArrayParameter manpage, the cgGetParameterType manpage

NAME

cgGetNextPass − iterate through passs in a technique

SYNOPSIS

```
#include <Cg/cg.h>

CGpass cgGetNextPass( CGpass pass );
```

PARAMETERS

pass Specifies the current pass.

DESCRIPTION

The passs within a technique can be iterated over by using the cgGetNextPass function. The following example code illustrates one way to do this:

```
CGpass pass = cgGetFirstPass( technique );
while( pass )
{
 /* do something with pass */
 pass = cgGetNextPass( pass )
}
```

Passes are returned in the order defined in the technique.

RETURN VALUES

cgGetNextPass returns the next pass in the technique's internal sequence of passes. Returns 0 when **pass** is the last pass in the technique.

ERRORS

CG_INVALID_PASS_HANDLE_ERROR is generated if **pass** does not refer to a valid pass.

SEE ALSO

the cgGetFirstPass manpage, the cgGetNamedPass manpage, and the cgIsPass manpage

NAME

cgGetNextProgram − iterate through programs in a context

SYNOPSIS

```
#include <Cg/cg.h>

CGprogram cgGetNextProgram( CGprogram prog );
```

PARAMETERS

prog Specifies the program.

DESCRIPTION

The programs within a context can be iterated over by using the cgGetNextProgram function. The following example code illustrates one way to do this:

```
CGprogram prog = cgGetFirstProgram( ctx );
while( prog )
{
 /* do something with prog */
 prog = cgGetNextProgram( prog )
}
```

Note that no specific order of traversal is defined by this mechanism. The only guarantee is that each program will be visited exactly once. No guarantees can be made if programs are generated or deleted during iteration.

RETURN VALUES

cgGetNextProgram returns the next program in the context's internal sequence of programs. Returns 0 when **prog** is the last program in the context.

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if **prog** does not refer to a valid program.

SEE ALSO

the cgGetNextProgram manpage, the cgCreateProgram manpage, the cgDestroyProgram manpage, and the cgIsProgram manpage

NAME

cgGetNextState − iterate through states in a context

SYNOPSIS

```
#include <Cg/cg.h>

CGstate cgGetNextState( CGstate state );
```

PARAMETERS

state Specifies the current state.

DESCRIPTION

The states within a context can be iterated over by using the cgGetNextState function. The following example code illustrates one way to do this:

```
CGstate state = cgGetFirstState( ctx );
while( state )
{
 /* do something with state */
 state = cgGetNextState( state )
}
```

Note that no specific order of traversal is defined by this mechanism. The only guarantee is that each state will be visited exactly once. No guarantees can be made if states are created or deleted during iteration.

RETURN VALUES

cgGetNextState returns the next state in the context's internal sequence of states. Returns 0 when **state** is the last state in the context.

ERRORS

CG_INVALID_STATE_HANDLE_ERROR is generated if **state** does not refer to a valid state.

SEE ALSO

the cgGetNextState manpage, the cgGetNamedState manpage, the cgCreateState manpage, and the cgIsState manpage

NAME

cgGetNextStateAssignment − iterate through state assignments in a pass

SYNOPSIS

```
#include <Cg/cg.h>

CGstateassignment cgGetNextStateAssignment( CGstateassignment sa );
```

PARAMETERS

sa Specifies the current state assignment.

DESCRIPTION

The state assignments within a pass can be iterated over by using the cgGetNextStateAssignment function. The following example code illustrates one way to do this:

```
CGstateassignment sa = cgGetFirstStateAssignment( pass );
while( sa )
{
 /* do something with sa */
 sa = cgGetNextStateAssignment( sa )
}
```

State assignments are returned in the same order specified in the pass in the effect.

RETURN VALUES

cgGetNextStateAssignment returns the next state assignment in the context's internal sequence of state assignments. It returns 0 when **prog** is the last state assignment in the context.

ERRORS

CG_INVALID_STATE ASSIGNMENT_HANDLE_ERROR is generated if **sa** does not refer to a valid state assignment.

SEE ALSO

the cgGetFirstStateAssignment manpage, the cgGetNamedStateAssignment manpage, and the cgIsStateAssignment manpage

NAME

cgGetNextTechnique − iterate through techniques in a effect

SYNOPSIS

```
#include <Cg/cg.h>

CGtechnique cgGetNextTechnique( CGtechnique tech );
```

PARAMETERS

tech Specifies the current technique.

DESCRIPTION

The techniques within a effect can be iterated over by using the cgGetNextTechnique function. The following example code illustrates one way to do this:

```
CGtechnique tech = cgGetFirstTechnique( effect );
while( tech )
{
 /* do something with tech */
 tech = cgGetNextTechnique( tech )
}
```

Note that no specific order of traversal is defined by this mechanism. The only guarantee is that each technique will be visited exactly once.

RETURN VALUES

cgGetNextTechnique returns the next technique in the effect's internal sequence of techniques. Returns 0 when **prog** is the last technique in the effect.

ERRORS

CG_INVALID_TECHNIQUE_HANDLE_ERROR is generated if **prog** does not refer to a valid technique.

SEE ALSO

the cgGetFirstTechnique manpage, and the cgGetNamedTechnique manpage,

NAME

cgGetNumConnectedToParameters − gets the number of connected destination parameters

SYNOPSIS

```
#include <Cg/cg.h>

int cgGetNumConnectedToParameters(CGparameter param);
```

PARAMETERS

param Specifies the source parameter.

DESCRIPTION

cgGetNumConnectedToParameters returns the number of destination parameters connected to the source parameter **param**. It's primarily used with cgGetConnectedToParameter.

RETURN VALUES

Returns one of the connected destination parameters to **param**. **(CGparameter)0** is returned if an error is thrown.

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if **param** does not refer to a valid parameter.

SEE ALSO

the cgConnectParameter manpage, and the cgGetConnectedParameter manpage

NAME

cgGetNumDependentAnnotationParameters − returns the number of effect parameters an annotation depends on.

SYNOPSIS

```
#include <Cg/cg.h>

int cgGetNumDependentAnnotationParameters(CGannotation ann);
```

PARAMETERS

ann Specifies the annotation handle.

DESCRIPTION

Annotations in CgFX files may include references to one or more effect parameters on the right hand side of the annotation that are used for computing the state assignment's value. **cgGetNumDependentAnnotationParameters** returns the total number of such parameters. the cgGetDependentAnnotationParameter manpage can then be used to iterate over the parameters individually.

This information can be useful for applications that wish to cache the values of annotations so that they can determine which annotations may change as the result of changing a particular parameter's value.

ERRORS

CG_INVALID_ANNOTATION_HANDLE_ERROR is generated if **ann** does not refer to a valid annotation.

SEE ALSO

the cgGetDependentAnnotationParameter manpage, the cgGetFirstAnnotation manpage, the cgGetNamedAnnotation manpage, the cgGetNumDependentStateAssignmentParameters manpage

NAME

cgGetNumDependentStateAssignmentParameters − returns the number of effect parameters a state assignment depends on.

SYNOPSIS

```
#include <Cg/cg.h>

int cgGetNumDependentStateAssignmentParameters(CGstateassignment sa);
```

PARAMETERS

sa Specifies the state assignment handle.

DESCRIPTION

State assignments in CgFX passes may include references to one or more effect parameters on the right hand side of the state assignment that are used for computing the state assignment's value. **cgGetNumDependentStateAssignmentParameters** returns the total number of such parameters. the cgGetDependentStateAssignmentParameter manpage can then be used to iterate over the parameters individually.

This information can be useful for applications that wish to cache the values of state assignments for customized state maangement so that they can determine which state assignments may change as the result of changing a parameter's value.

ERRORS

CG_INVALID_STATE_ASSIGNMENT_HANDLE_ERROR is generated if **sa** does not refer to a valid state assignment.

SEE ALSO

the cgGetDependentStateAssignmentParameter manpage, the cgGetFirstStateAssignment manpage, the cgGetNamedStateAssignment manpage, the cgGetNumDependentAnnotationParameters manpage

NAME

cgGetNumParentTypes − gets the number of parent types of a given type

SYNOPSIS

```
#include <Cg/cg.h>

int cgGetNumParentTypes(CGtype type);
```

PARAMETERS

type Specifies the child type.

DESCRIPTION

cgGetNumParentTypes returns the number of parents the child type **type** inherits from.

A parent type is one from which the given type inherits, or an interface type that the given type implements. For example, given the type definitions:

```
interface myiface {
    float4 eval(void);
};

struct mystruct : myiface {
    float4 value;
    float4 eval(void ) { return value; }
};
```

mystruct has a single parent type, **myiface**.

Note that the current Cg language specification implies that a type may only have a single parent type — an interface implemented by the given type.

RETURN VALUES

Returns the number of parent types. **(CGparameter)0** is returned if there are no parents.

ERRORS

This function does not generate any errors.

SEE ALSO

the cgGetParentType manpage

NAME

cgGetNumUserTypes − get number of user-defined types in a program or effect

SYNOPSIS

```
#include <Cg/cg.h>

int cgGetNumUserTypes(CGhandle handle);
```

PARAMETERS

handle The **CGprogram** or **CGeffect** in which the types are defined.

DESCRIPTION

cgGetNumUserTypes returns the number of user-defined types in a given **CGprogram** or **CGeffect**.

RETURN VALUES

Returns the number of user defined types.

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if **handle** is not a valid **CGprogram** or **CGeffect**.

SEE ALSO

the cgGetUserType manpage, and the cgGetNamedUserType manpage

NAME

cgGetParameterBaseResource − get a program parameter's base resource

SYNOPSIS

```
#include <Cg/cg.h>

CGresource cgGetParameterBaseResource( CGparameter param );
```

PARAMETERS

param Specifies the program parameter.

DESCRIPTION

cgGetParameterBaseResource allows the application to retrieve the base resource for a parameter in a Cg program. The base resource is the first resource in a set of sequential resources. For example, if a given parameter has a resource of **CG_ATTR7**, it's base resource would be **CG_ATTR0**. Only parameters with resources whose name ends with a number will have a base resource. All other parameters will return the undefined resource **CG_UNDEFINED** when calling cgGetParameterBaseResource.

The numerical portion of the resource may be retrieved with the cgGetParameterResourceIndex function. For example, if the resource for a given parameter is **CG_ATTR7**, cgGetParameterResourceIndex will return **7**.

RETURN VALUES

Returns the base resource of **param**. If no base resource exists for the given parameter, **CG_UNDEFINED** is returned.

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if the handle **param** is invalid.

CG_INVALID_PARAMETER_ERROR is generated if the parameter is not a leaf node.

SEE ALSO

the cgGetParameterResource manpage, the cgGetParameterResourceIndex manpage, and the cgGetResourceString manpage

NAME

cgGetParameterBaseType − get a program parameter's base type

SYNOPSIS

```
#include <Cg/cg.h>

CGtype cgGetParameterBaseType( CGparameter param );
```

PARAMETERS

param Specifies the parameter.

DESCRIPTION

cgGetParameterBaseType allows the application to retrieve the base type of a parameter.

If **param** is of a numeric type (scalar, vector, or matrix), the scalar enumerant corresponding to **param**'s type will be returned. For example, if **param** is of type **CG_FLOAT4x3**, **cgGetParameterBaseType** will return **CG_FLOAT**.

If **param** is an array, the base type of the array elements will be returned.

If **param** is a structure, its type-specific enumerant will be returned, as per cgGetParameterNamedType.

Otherwise, **param**'s type enumerant will be returned.

RETURN VALUES

Returns the base type enumerant of **param**. If an error occurs, **CG_UNKNOWN_TYPE** will be returned.

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if **param** does not refer to a valid parameter.

SEE ALSO

the cgGetParameterType manpage the cgGetType manpage, the cgGetTypeString manpage the cgGetParameterClass manpage

NAME

cgGetParameterClass − get a parameter's class

SYNOPSIS

```
#include <Cg/cg.h>

CGparameterclass cgGetParameterClass( CGparameter param );
```

PARAMETERS

param Specifies the parameter.

DESCRIPTION

cgGetParameterClass allows the application to retrieve the class of a parameter.

The returned **CGparameterclass** value enumerates the high-level paramater classes:

CG_PARAMETERCLASS_SCALAR
The parameter if of a scalar type, such as CG_INT, or CG_FLOAT.

CG_PARAMETERCLASS_VECTOR
The parameter is of a vector type, such as CG_INT1, or CG_FLOAT4.

CG_PARAMETERCLASS_MATRIX
The parameter is of a matrix type, such as CG_INT1x1, or CG_FLOAT4x4.

CG_PARAMETERCLASS_STRUCT
The parameter is a struct or interface.

CG_PARAMETERCLASS_ARRAY
The parameter is an array.

CG_PARAMETERCLASS_SAMPLER
The parameter is a sampler.

CG_PARAMETERCLASS_OBJECT
The parameter is a texture, string, or program.

RETURN VALUES

Returns the parameter class enumerant of **param**. If an error occurs, **CG_PARAMETERCLASS_UNKNOWN** will be returned.

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if **param** does not refer to a valid parameter.

SEE ALSO

the cgGetParameterType manpage, the cgGetType manpage, the cgGetTypeString manpage, the cgGetParameterClass manpage

NAME

cgGetParameterColumns − get number of parameter columns

SYNOPSIS

```
#include <Cg/cg.h>

int cgGetParameterColumns( CGparameter param );
```

PARAMETERS

param Specifies the parameter.

DESCRIPTION

cgGetParameterColumns return the number of columns associated with the given parameter's type.

If **param** is an array, the number of columns associated with each element of the array is returned.

RETURN VALUES

If **param** is a numeric type, or an array of numeric types, the number of columns associated with the type is returned.

Otherwise, 0 is returned.

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if **param** does not refer to a valid parameter.

SEE ALSO

the cgGetParameterType manpage and the cgGetParameterRows manpage

NAME

cgGetParameterContext − get a parameter's parent context

SYNOPSIS

```
#include <Cg/cg.h>

CGcontext cgGetParameterContext(CGparameter ctx);
```

PARAMETERS

param Specifies the parameter.

DESCRIPTION

cgGetParameterContext allows the application to retrieve a handle to the context a given parameter belongs to.

RETURN VALUES

Returns a **CGcontext** handle to the parent context. In the event of an error **NULL** is returned.

ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if **ctx** does not refer to a valid context.

SEE ALSO

the cgGetParameterProgram manpage

NAME

cgGetParameterDirection − get a program parameter's direction

SYNOPSIS

```
#include <Cg/cg.h>

CGenum cgGetParameterDirection( CGparameter param );
```

PARAMETERS

param Specifies the program parameter.

DESCRIPTION

cgGetParameterDirection allows the application to distinguish program input parameters from program output parameters. This information is necessary for the application to properly supply the program inputs and use the program outputs.

cgGetParameterDirection will return one of the following enumerants :

CG_IN Specifies an input parameter.

CG_OUT Specifies an output parameter.

CG_INOUT
Specifies a parameter that is both input and output.

CG_ERROR
If an error occurs.

RETURN VALUES

Returns the direction of **param**. Returns **CG_ERROR** if an error occurs.

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if **prog** does not refer to a valid parameter.

SEE ALSO

the cgGetNamedParameter manpage, the cgGetNextParameter manpage, the cgGetParameterName manpage, the cgGetParameterType manpage, the cgGetParameterVariability manpage, the cgGetParameterBinding manpage, the cgGetParameterDirectionalBinding manpage, the cgIsArray manpage, the cgSetParameterVariablity manpage, the cgSetParameterBinding manpage, the cgSetParameterDirectionalBinding manpage

NAME

cgGetParameterIndex − get an array member parameter's index

SYNOPSIS

```
#include <Cg/cg.h>

int cgGetParameterIndex(CGparameter param);
```

PARAMETERS

param Specifies the parameter.

DESCRIPTION

cgGetParameterIndex returns an integer that represents the index of an array parameter.

For example if you have the following Cg program :

RETURN VALUES

Returns the index associated with an array member parameter. If the parameter is not in an array a **−1** will be returned.

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if the handle **param** is invalid.

CG_ARRAY_PARAM_ERROR is generated if the handle **param** is not a handle to an array parameter.

SEE ALSO

the cgGetArrayParameter manpage

NAME

cgGetParameterName − get a program parameter's name

SYNOPSIS

```
#include <Cg/cg.h>

const char * cgGetParameterName( CGparameter param );
```

PARAMETERS

param Specifies the program parameter.

DESCRIPTION

cgGetParameterName allows the application to retrieve the name of a parameter in a Cg program. This name can be used later to retrieve the parameter from the program using **cgGetNamedParameter**.

RETURN VALUES

Returns the null-terminated name string for the parameter.

Returns null if **param** is invalid.

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if **param** does not refer to a valid parameter.

SEE ALSO

the cgGetNamedParameter manpage, the cgGetNextParameter manpage, the cgGetParameterType manpage, the cgGetParameterVariablity manpage, the cgGetParameterDirection manpage, the cgIsArray manpage, the cgSetParameterVariablity manpage,

NAME

cgGetParameterNamedType − get a program parameter's type

SYNOPSIS

```
#include <Cg/cg.h>

CGtype cgGetParameterNamedType(CGparameter param);
```

PARAMETERS

param Specifies the parameter.

DESCRIPTION

cgGetParameterNamedType returns the type of **param** similarly to cgGetParameterType. However, if the type is a user defined struct it will return the unique enumerant associated with the user defined type instead of **CG_STRUCT**.

RETURN VALUES

Returns the type of **param**.

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if **prog** does not refer to a valid parameter.

SEE ALSO

the cgGetParameterType manpage, the cgGetParameterBaseType manpage

NAME

cgGetParameterOrdinalNumber − get a program parameter's ordinal number

SYNOPSIS

```
#include <Cg/cg.h>

int cgGetParameterOrdinalNumber(CGparameter param);
```

PARAMETERS

param Specifies the program parameter.

DESCRIPTION

cgGetParameterOrdinalNumber returns an integer that represents the order in which the parameter was declared within the Cg program.

Ordinal numbering begins at zero, starting with a program's first local leaf parameter. The subsequent local leaf parameters are enumerated in turn, followed by the program's global leaf parameters.

For example, the following Cg program:

```
struct MyStruct { float a; sampler2D b; };
float globalvar1;
float globalvar2
float4 main(float2 position : POSITION,
            float4 color    : COLOR,
            uniform MyStruct mystruct,
            float2 texCoord : TEXCOORD0) : COLOR
 {
  // etc ...
 }
```

Would result in the following parameter ordinal numbering:

```
position    -> 0
color       -> 1
mystruct.a  -> 2
mystruct.b  -> 3
texCoord    -> 4
globalvar1  -> 5
globalvar2  -> 6
```

RETURN VALUES

Returns the ordinal number associated with a parameter. The parameter must not be a constant. If it is a constant (cgGetParameterVariability returns **CG_CONSTANT**) then **0** is returned and no error is generated.

When **cgGetParameterOrdinalNumber** is passed an array, the ordinal number of the first array element is returned. When passed a struct, the ordinal number of first struct data member is returned.

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if the handle **param** is invalid.

SEE ALSO

the cgGetParameterVariability manpage

NAME

cgGetParameterProgram − get a parameter's parent program

SYNOPSIS

```
#include <Cg/cg.h>

CGprogram cgGetParameterProgram( CGparameter prog );
```

PARAMETERS

param Specifies the parameter.

DESCRIPTION

cgGetParameterProgram allows the application to retrieve a handle to the program a given parameter belongs to.

RETURN VALUES

Returns a **CGprogram** handle to the parent program. In the event of an error or if the parameter is not a child of a program, **NULL** is returned.

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if **prog** does not refer to a valid program.

SEE ALSO

the cgCreateProgram manpage, the cgGetParameterEffect manpage, the cgCreateContext manpage

NAME

cgGetParameterResource − get a program parameter's resource

SYNOPSIS

```
#include <Cg/cg.h>

CGresource cgGetParameterResource( CGparameter param );
```

PARAMETERS

param Specifies the program parameter.

DESCRIPTION

cgGetParameterResource allows the application to retrieve the resource for a parameter in a Cg program. This resource is necessary for the application to be able to supply the program's inputs and use the program's outputs.

The resource enumerant is a profile-specific hardware resource.

RETURN VALUES

Returns the resource of **param**.

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if the handle **param** is invalid.

CG_INVALID_PARAMETER_ERROR is generated if the parameter is not a leaf node.

SEE ALSO

the cgGetParameterResourceIndex manpage, the cgGetParameterBaseResource manpage, and the cgGetResourceString manpage

NAME

cgGetParameterResourceIndex − get a program parameter's resource index

SYNOPSIS

```
#include <Cg/cg.h>

unsigned long cgGetParameterResourceIndex( CGparameter param );
```

PARAMETERS

param Specifies the program parameter.

DESCRIPTION

cgGetParameterResourceIndex allows the application to retrieve the resource index for a parameter in a Cg program. This index value is only used with resources that are linearly addressable.

RETURN VALUES

Returns the resource index of **param**.

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if the handle **param** is invalid.

CG_INVALID_PARAMETER_ERROR is generated if the parameter is not a leaf node.

SEE ALSO

the cgGetParameterResource manpage, the cgGetResource manpage, the cgGetResourceString manpage

NAME

cgGetParameterRows − get number of parameter rows

SYNOPSIS

```
#include <Cg/cg.h>

int cgGetParameterRows( CGparameter param );
```

PARAMETERS

param Specifies the parameter.

DESCRIPTION

cgGetParameterRows return the number of rows associated with the given parameter's type.

If **param** is an array, the number of rows associated with each element of the array is returned.

RETURN VALUES

If **param** is a numeric type, or an array of numeric types, the number of rows associated with the type is returned.

Otherwise, 0 is returned.

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if **param** does not refer to a valid parameter.

SEE ALSO

the cgGetParameterType manpage and the cgGetParameterColumns manpage

NAME

cgGetParameterSemantic − get a program parameter's semantic

SYNOPSIS

```
#include <Cg/cg.h>

const char * cgGetParameterSemantic( CGparameter param );
```

PARAMETERS

param Specifies the program parameter.

DESCRIPTION

cgGetParameterSemantic allows the application to retrieve the semantic of a parameter in a Cg program. If the parameter does not have a semantic assigned to it, an empty string will be returned.

RETURN VALUES

Returns the null-terminated semantic string for the parameter.

Returns null if an error occurs.

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if the handle **param** is invalid.

SEE ALSO

the cgGetParameterResource manpage, the cgGetParameterResourceIndex manpage, the cgGetParameterName manpage, and the cgGetParameterType manpage

NAME

cgGetParameterType − get a program parameter's type

SYNOPSIS

```
#include <Cg/cg.h>

CGtype cgGetParameterType( CGparameter param );
```

PARAMETERS

param Specifies the parameter.

DESCRIPTION

cgGetParameterType allows the application to retrieve the type of a parameter in a Cg program. This type is necessary for the application to be able to supply the program's inputs and use the program's outputs.

cgGetParameterType will return **CG_STRUCT** if the parameter is a struct and **CG_ARRAY** if the parameter is an array. Otherwise it will return the data type associated with the parameter.

RETURN VALUES

Returns the type enumerant of **param**. If an error occurs, **CG_UNKNOWN_TYPE** will be returned.

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if **param** does not refer to a valid parameter.

SEE ALSO

the cgGetType manpage, the cgGetParameterBaseType manpage, the cgGetTypeString manpage, and the cgGetParameterClass manpage

NAME

cgGetParameterValue − get the value of any numeric parameter

SYNOPSIS

```
#include <Cg/cg.h>

/* type may be int, float, or double */
int cgGetParameterValue{i,f,d}{r,c}(CGparameter param,
                                    int nvals, type *v);
```

PARAMETERS

param Specifies the program parameter whose value will be retrieved.

nvals The length of the **v** array, in number of elements.

v Destination buffer to which the parameter values will be written.

DESCRIPTION

cgGetParameterValue allows the application to get the value of any numeric parameter or parameter array.

The given parameter must be a scalar, vector, matrix, or a (possibly-multidimensional) array of scalars, vectors, or matrices.

There are versions of each function that take **int**, **float** or **double** values signified by the **i**, **f** or **d** in the function name.

There are versions of each function that will cause any matrices referenced by **param** to be copied in either row-major or column-major order, as signified by the **r** or **c** in the function name.

For example, **cgGetParameterValueic** retrieves the values of the given parameter using the supplied array of integer data, and copies matrix data in column-major order.

If **v** is smaller than the total number of values in the given source parameter, **CG_NOT_ENOUGH_DATA_ERROR** is generated.

The total number of values in a parameter, ntotal, may be computed as follow:

```
int nrows = cgGetParameterRows(param);
int ncols = cgGetParameterColumns(param);
int asize = cgGetArrayTotalSize(param);
int ntotal = nrows*ncols;
if (asize > 0) ntotal *= asize;
```

RETURN VALUES

The total number of values written to **v** is returned.

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if the handle **param** is invalid.

CG_INVALID_POINTER_ERROR is generated if **v** is **NULL**.

CG_NOT_ENOUGH_DATA_ERROR is generated if **nvalues** is less than the total size of **param**.

CG_NON_NUMERIC_PARAMETER_ERROR is generated if **param** is of a non-numeric type.

SEE ALSO

the cgGetParameterRows manpage, the cgGetParameterColumns manpage, the cgGetArrayTotalSize manpage, and the cgSetParameterValue manpage

NAME

cgGetParameterValues − get a program parameter's values

SYNOPSIS

```
#include <Cg/cg.h>

const double *cgGetParameterValues(CGparameter param,
                                   CGenum value_type,
                                   int *nvalues);
```

PARAMETERS

param Specifies the program parameter.

value_type

Determines what type of value to return. Valid enumerants are :

- **CG_CONSTANT**
  Returns the constant values for parameters that have constant variability. See the cgGetParameterVariability manpage for more information.
- **CG_DEFAULT**
  Returns the default values for a uniform parameter.
- **CG_CURRENT**
  Returns the current values for a uniform parameter.

nvalues Pointer to integer that will be initialized to store the number of values returned.

DESCRIPTION

cgGetParameterValues allows the application to retrieve default or constant values from uniform parameters.

RETURN VALUES

Returns a pointer to an array of **double** values. The number of values in the array is returned via the **nvalues** parameter.

If no values are available, **NULL** will be returned and **nvalues** will be **0**.

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if the handle **param** is invalid.

CG_INVALID_PARAMETER_ERROR is generated if **nvalues** is **NULL**.

CG_INVALID_ENUMERANT_ERROR if the **value_type** parameter is invalid.

SEE ALSO

cgGetParameterVariability

NAME

cgGetParameterVariability − get a parameter's variability

SYNOPSIS

```
#include <Cg/cg.h>

CGenum cgGetParameterVariability( CGparameter param );
```

PARAMETERS

param Specifies the program parameter.

DESCRIPTION

cgGetParameterVariability allows the application to retrieve the variablilty of a parameter in a Cg program. This variability is necessary for the application to be able to supply the program's inputs and use the program's outputs.

cgGetParameterVariability will return one of the following variabilities:

CG_VARYING
A varying parameter is one whose value changes with each invocation of the program.

CG_UNIFORM
A uniform parameter is one whose value does not change with each invocation of a program, but whose value can change between groups of program invocations.

CG_LITERAL
A literal parameter is folded out at compile time. Making a uniform parameter literal with cgSetParameterVariability will often make a program more efficient at the expense of requiring a compile every time the value is set.

CG_CONSTANT
A constant parameter is never changed by the user. It's generated by the compiler by certain profiles that require immediate values to be placed in certain resource locations.

CG_MIXED
A structure parameter that contains parameters that differ in variability.

RETURN VALUES

Returns the variability of **param**. Returns **CG_ERROR** if an error occurs.

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if the handle **param** is invalid.

SEE ALSO

the cgGetNamedParameter manpage, the cgGetNextParameter manpage, the cgGetParameterName manpage, the cgGetParameterType manpage, the cgGetParameterDirection manpage, the cgGetParameterBinding manpage, the cgGetParameterDirectionalBinding manpage, the cgIsArray manpage, the cgSetParameterVariablity manpage, the cgSetParameterDirection manpage, the cgSetParameterBinding manpage, the cgSetParameterDirectionalBinding manpage

NAME

cgGetParentType − gets a parent type of a child type

SYNOPSIS

```
#include <Cg/cg.h>

CGtype cgGetParentType(CGtype type, int index);
```

PARAMETERS

type Specifies the child type.

index The index of the parent type. **index** must be greater than or equal to **0** and less than **N** where N is the value returned by cgGetNumParentTypes.

DESCRIPTION

cgGetParentType returns a parent type of **type**.

A parent type is one from which the given type inherits, or an interface type that the given type implements. For example, given the type definitions:

```
interface myiface {
    float4 eval(void);
};

struct mystruct : myiface {
    float4 value;
    float4 eval(void ) { return value; }
};
```

mystruct has a single parent type, **myiface**.

Note that the current Cg language specification implies that a type may only have a single parent type — an interface implemented by the given type.

RETURN VALUES

Returns the number of parent types. **(CGparameter)0** is returned if there are no parents.

CG_UNKNOWN_TYPE if **type** is a built-in type or an error is thrown.

ERRORS

CG_OUT_OF_ARRAY_BOUNDS_ERROR is generated if **index** is outside the proper range.

SEE ALSO

the cgGetNumParentTypes manpage

NAME

cgGetPassName − get a technique pass's name

SYNOPSIS

```
#include <Cg/cg.h>

const char * cgGetPassName( CGpass pass );
```

PARAMETERS

pass Specifies the pass.

DESCRIPTION

cgGetPassName allows the application to retrieve the name of a pass in a Cg program. This name can be used later to retrieve the pass from the program using **cgGetNamedPass**.

RETURN VALUES

Returns the null-terminated name string for the pass.

Returns null if **pass** is invalid.

ERRORS

CG_INVALID_PASS_HANDLE_ERROR is generated if **pass** does not refer to a valid pass.

SEE ALSO

the cgGetNamedPass manpage, the cgGetFirstPass manpage, and the cgGetNextPass manpage,

NAME

cgGetPassTechnique − get a pass's technique

SYNOPSIS

```
#include <Cg/cg.h>

CGtechnique cgGetPassTechnique( CGpass pass );
```

PARAMETERS

pass Specifies the pass.

DESCRIPTION

cgGetPassTechnique allows the application to retrieve a handle to the technique a given pass belongs to.

RETURN VALUES

Returns a **CGtechnique** handle to the technique. In the event of an error **NULL** is returned.

ERRORS

CG_INVALID_PASS_HANDLE_ERROR is generated if **pass** does not refer to a valid pass.

SEE ALSO

NAME

cgGetProfile − get the profile enumerant from a the profile name

SYNOPSIS

```
#include <Cg/cg.h>

CGprofile cgGetProfile( const char *profile_string );
```

PARAMETERS

profile_string

A string containing the profile name. The name is case-sensitive.

DESCRIPTION

cgGetProfile returns the enumerant assigned to a profile name.

The following is an example of how cgGetProfile might be used.

```
CGprofile ARBVP1Profile = cgGetProfile("arbvp1");

if(cgGetProgramProfile(myprog) == ARBVP1Profile)
 {
  /* Do stuff */
 }
```

RETURN VALUES

Returns the profile enumerant of **profile_string**. If no such profile exists **CG_UNKNOWN** will be returned.

ERRORS

SEE ALSO

the cgGetProfileString manpage, the cgGetProgramProfile manpage

NAME

cgGetProfileString − get the profile name associated with a profile enumerant

SYNOPSIS

```
#include <Cg/cg.h>

const char * cgGetProfileString( CGprofile profile );
```

PARAMETERS

profile The profile enumerant.

DESCRIPTION

cgGetProfileString returns the profile named associated with a profile enumerant. =back

RETURN VALUES

Returns the profile string of the enumerant **profile**.

ERRORS

SEE ALSO

the cgGetProfile manpage, the cgGetProgramProfile manpage

NAME

cgGetProgramContext − get a programs parent context

SYNOPSIS

```
#include <Cg/cg.h>

CGcontext cgGetProgramContext( CGprogram prog );
```

PARAMETERS

prog Specifies the program.

DESCRIPTION

cgGetProgramContext allows the application to retrieve a handle to the context a given program belongs to.

RETURN VALUES

Returns a **CGcontext** handle to the parent context. In the event of an error **NULL** is returned.

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if **prog** does not refer to a valid program.

SEE ALSO

the cgCreateProgram manpage, and the cgCreateContext manpage

NAME

cgGetProgramOptions − get strings from a program object

SYNOPSIS

```
#include <Cg/cg.h>

char const * const * cgGetProgramOptions( CGprogram prog );
```

PARAMETERS

prog Specifies the Cg program to query.

DESCRIPTION

cgGetProgramOptions allows the application to retrieve the set of options used to compile the program.

The options are returned in an array of ASCII-encoded NULL-terminated character strings. Each string contains a single option. The last element of the string array is guaranteed to be NULL.

NULL is returned if no options exist, or if an error occurs.

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if **prog** does not refer to a valid program.

SEE ALSO

the cgGetProgramString manpage

NAME

cgGetProgramProfile, **cgSetProgramProfile** − get or set a program's profile

SYNOPSIS

```
#include <Cg/cg.h>

CGprofile cgGetProgramProfile(CGprogram prog);
void cgSetProgramProfile(CGprogram prog, CGprofile profile);
```

PARAMETERS

prog Specifies the program.

profile Specifies the profile to be used when compiling the program.

DESCRIPTION

cgSetProgramProfile allows the application to specify the profile to be used when compiling the given program. When called, the program will be unloaded if it is currently loaded, and marked as uncompiled. When the program is next compiled (see cgSetAutoCompile), the given **profile** will be used. **cgSetProgramProfile** can be used to override the profile specified in a CgFX **compile** statement, or to change the profile associated with a program created by a call to cgCreateProgram.

cgGetProgramProfile allows the application to retrieve the profile enumerant currently associated with the program.

RETURN VALUES

cgGetProgramProfile returns the profile enumerant associated with **prog**.

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if **prog** does not refer to a valid program.

CG_INVALID_PROFILE_ERROR is generated if **profile** is not a valid profile enumerant.

SEE ALSO

the cgGetProfile manpage, the cgGetProfileString manpage, the cgCreateProgram manpage, and the cgSetAutoCompile manpage

NAME

cgGetProgramStateAssignmentValue − get a program-valued state assignment's values

SYNOPSIS

```
#include <Cg/cg.h>

CGprogram cgGetProgramStateAssignmentValue(CGstateassignment sa);
```

PARAMETERS

sa Specifies the state assignment.

DESCRIPTION

cgGetProgramStateAssignmentValues allows the application to retrieve the *value*(s) of a state assignment that stores a **CGprogram**.

RETURN VALUES

Returns a **CGprogram** handle.

ERRORS

CG_INVALID_STATE_ASSIGNMENT_HANDLE_ERROR is generated if the handle **sa** is invalid.

CG_INVALID_PARAMETER_TYPE_ERROR is generated if the state assignment is not **CGprogram**−typed.

SEE ALSO

the cgGetStateAssignmentState manpage, the cgGetStateType manpage, the cgGetFloatStateAssignmentValues manpage, the cgGetIntStateAssignmentValues manpage, the cgGetBoolStateAssignmentValues manpage, the cgGetStringStateAssignmentValue manpage, the cgGetProgramStateAssignmentValue manpage, the cgGetSamplerStateAssignmentValue manpage, and the cgGetTextureStateAssignmentValue manpage

NAME

cgGetProgramString − get strings from a program object

SYNOPSIS

```
#include <Cg/cg.h>

const char * cgGetProgramString( CGprogram prog, CGenum pname );
```

PARAMETERS

prog Specifies the Cg program to query.

pname Specifies the string to retrieve. **pname** can be one of **CG_PROGRAM_SOURCE**, **CG_PROGRAM_ENTRY**, **CG_PROGRAM_PROFILE**, or **CG_COMPILED_PROGRAM**.

DESCRIPTION

cgGetProgramString allows the application to retrieve program strings that have been set via functions that modify program state.

When **pname** is **CG_PROGRAM_SOURCE** the original Cg source program is returned.

When **pname** is **CG_PROGRAM_ENTRY** the main entry point for the program is returned.

When **pname** is **CG_PROGRAM_PROFILE** the profile for the program is returned.

When **pname** is **CG_COMPILED_PROGRAM**, the string for the compiled program is returned.

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if **prog** does not refer to a valid program.

CG_INVALID_ENUMERANT_ERROR is generated if **pname** is an invalid enumerant.

SEE ALSO

the cgCreateProgram manpage, the cgGetProgramOptions manpage

NAME

cgGetResource − get the resource enumerant assigned to a resource name

SYNOPSIS

```
#include <Cg/cg.h>

CGresource cgGetResource( const char *resource_string );
```

PARAMETERS

resource_string

A string containing the resource name.

DESCRIPTION

cgGetResource returns the enumerant assigned to a resource name.

The following is an example of how cgGetResource might be used.

```
CGresource PositionResource = cgGetResource("POSITION");

if(cgGetParameterResource(myparam) == PositionResource)
 {
  /* Do stuff */
 }
```

RETURN VALUES

Returns the resource enumerant of **resource_string**. If no such resource exists **CG_UNKNOWN** will be returned.

ERRORS

SEE ALSO

the cgGetResourceString manpage, the cgGetParameterResource manpage

NAME

cgGetResourceString − get the resource name associated with a resource enumerant

SYNOPSIS

```
#include <Cg/cg.h>

const char * cgGetResourceString( CGresource resource );
```

PARAMETERS

resource The resource enumerant.

DESCRIPTION

cgGetResourceString returns the resource named associated with a resource enumerant. =back

RETURN VALUES

Returns the resource string of the enumerant **resource**.

ERRORS

SEE ALSO

the cgGetResource manpage, the cgGetParameterResource manpage

NAME

cgGetSamplerStateAssignmentParameter − get the sampler parameter being set up given a state assignment in its sampler_state block.

SYNOPSIS

```
#include <Cg/cg.h>

CGparameter cgGetSamplerStateAssignmentParameter(CGstateassignment sa);
```

PARAMETERS

sa Specifies the state assignment in a **sampler_state** block

DESCRIPTION

Given the handle to a state assignment in a **sampler_state** block in an effect file, **cgGetSamplerStateAssignmentParameter** returns a handle to the sampler parameter being initialized. For example, given an effect file with:

```
sampler2D foo = sampler_state { GenerateMipmap = true; }
```

If **sa** is a handle to the **GenerateMipmap** state assignment, then **cgGetSamplerStateAssignmentParameter** returns a handle to **foo**.

RETURN VALUES

cgGetSamplerStateAssignmentParameter returns a handle to a parameter. If **sa** is not a state assignment in a **sampler_state** block, **NULL** is returned.

ERRORS

CG_INVALID_STATE_ASSIGNMENT_HANDLE_ERROR is generated if **sa** does not refer to a valid state assignment.

NAME

cgGetSamplerStateAssignmentValue − get a sampler-valued state assignment's values

SYNOPSIS

```
#include <Cg/cg.h>

CGparameter cgGetSamplerStateAssignmentValue(CGstateassignment sa);
```

PARAMETERS

sa Specifies the state assignment.

DESCRIPTION

cgGetSamplerStateAssignmentValues allows the application to retrieve the *value* (s) of a state assignment that stores a sampler.

RETURN VALUES

Returns a **CGparameter** handle for the sampler.

ERRORS

CG_INVALID_STATE_ASSIGNMENT_HANDLE_ERROR is generated if the handle **sa** is invalid.

CG_INVALID_PARAMETER_TYPE_ERROR is generated if the state assignment is not a sampler parameter.

SEE ALSO

the cgGetStateAssignmentState manpage, the cgGetStateType manpage, the cgGetFloatStateAssignmentValues manpage, the cgGetIntStateAssignmentValues manpage, the cgGetBoolStateAssignmentValues manpage, the cgGetStringStateAssignmentValue manpage, the cgGetProgramStateAssignmentValue manpage, and the cgGetTextureStateAssignmentValue manpage

NAME

cgGetStateAssignmentIndex − get the array index of a state assignment for array-valued state

SYNOPSIS

```
#include <Cg/cg.h>

int cgGetStateAssignmentIndex(CGstateassignment sa);
```

PARAMETERS

sa Specifies the state assignment.

DESCRIPTION

cgGetStateAssignmentIndex returns the array index of a state assignment if the state it is based on is an array type. For example, if there is a ‘‘LightPosition’’ state defined as an array of eight **float3** values, then given an effect file with the state assignment:

```
pass { LightPosition[3] = float3(10,0,0); }
```

Then when **cgGetStateAssignmentIndex** is passed a handle to this state assignment, it will return the value three.

RETURN VALUES

Returns an integer index value. If the **CGstate** for this state assignment is not an array type, zero is returned.

ERRORS

CG_INVALID_STATE_ASSIGNMENT_HANDLE_ERROR is generated if the handle **sa** is invalid.

NAME

cgGetStateAssignmentPass − get a state assignment's pass

SYNOPSIS

```
#include <Cg/cg.h>

CGpass cgGetStateAssignmentPass( CGstateassignment sa );
```

PARAMETERS

sa Specifies the state assignment.

DESCRIPTION

cgGetStateassignmentPass allows the application to retrieve a handle to the pass a given stateassignment belongs to.

RETURN VALUES

Returns a **CGpass** handle to the pass. In the event of an error **NULL** is returned.

ERRORS

CG_INVALID_STATE_ASSIGNMENT_HANDLE_ERROR is generated if **sa** does not refer to a valid state assignment.

SEE ALSO

NAME

cgGetStateAssignmentState − returns the state type of a particular state assignment

SYNOPSIS

```
#include <Cg/cg.h>

CGstate cgGetStateAssignmentState(CGstateassignment sa);
```

PARAMETERS

sa Specifies the state assignment handle.

DESCRIPTION

cgGetStateAssignmentState returns the **CGstate** object that corresponding to a particular state assignment in a pass. This object can then be queried to find out its type, giving the type of the state assignment.

ERRORS

CG_INVALID_STATE_ASSIGNMENT_HANDLE_ERROR is generated if **sa** is not a valid state assignment handle.

SEE ALSO

the cgGetStateType manpage

NAME

cgGetStateName − get a state's name

SYNOPSIS

```
#include <Cg/cg.h>

const char * cgGetStateName( CGstate state );
```

PARAMETERS

state Specifies the state.

DESCRIPTION

cgGetStateName allows the application to retrieve the name of a state defined in a Cg context. This name can be used later to retrieve the state from the context using **cgGetNamedState**.

RETURN VALUES

Returns the null-terminated name string for the state.

Returns null if **state** is invalid.

ERRORS

CG_INVALID_STATE_HANDLE_ERROR is generated if **state** does not refer to a valid state.

SEE ALSO

the cgGetNamedState manpage, the cgGetFirstState manpage, and the cgGetNextState manpage,

NAME

cgGetStateResetCallback − get the state resetting callback function for a state

SYNOPSIS

```
#include <Cg/cg.h>

CGstatecallback cgGetStateResetCallback( CGstate state );
```

PARAMETERS

state Specifies the state to retrieve the callback from.

DESCRIPTION

cgGetStateResetCallback returns the callback function used for resetting the state when the given state is encountered in a pass in a technique. See the cgSetStateCallbacks manpage for more information.

RETURN VALUES

cgStateResetCallback returns a pointer to the state resetting callback function. If **state** is not a valid state or if it has no callback, **NULL** is returned.

ERRORS

CG_INVALID_STATE_HANDLE_ERROR is generated if **state** does not refer to a valid state.

SEE ALSO

the cgSetStateCallbacks manpage, the cgCallStateResetCallback manpage, and the cgResetPassState manpage

NAME

cgGetStateSetCallback − get the state setting callback function for a state

SYNOPSIS

```
#include <Cg/cg.h>

CGstatecallback cgGetStateSetCallback( CGstate state );
```

PARAMETERS

state Specifies the state to retrieve the callback from.

DESCRIPTION

cgGetStateSetCallback returns the callback function used for setting the state when the given state is encountered in a pass in a technique. See the cgSetStateCallbacks manpage for more information.

RETURN VALUES

cgGetStateSetCallback returns a pointer to the state setting callback function. If **state** is not a valid state or if it has no callback, **NULL** is returned.

ERRORS

CG_INVALID_STATE_HANDLE_ERROR is generated if **state** does not refer to a valid state.

SEE ALSO

the cgSetStateCallbacks manpage, the cgCallStateSetCallback manpage, and the cgSetPassState manpage

NAME

cgGetStateType − returns the type of a given state

SYNOPSIS

```
#include <Cg/cg.h>

CGtype cgGetStateType(CGstate state);
```

PARAMETERS

state Specifies the state to return the type of.

DESCRIPTION

cgGetStateType returns the type of a state that was previously defined via the cgCreateState manpage, the cgCreateArrayState manpage, the cgCreateSamplerState manpage, or the cgCreateSamplerArrayState manpage.

RETURN VALUES

Returns the **CGtype** of the given state.

ERRORS

CG_INVALID_STATE_HANDLE_ERROR is generated if **state** does not refer to a valid state.

SEE ALSO

the cgCreateState manpage, the cgCreateArrayState manpage, the cgCreateSamplerState manpage, the cgCreateSamplerArrayState manpage, and the cgGetStateName manpage.

NAME

cgGetStateValidateCallback − get the state validatuib callback function for a state

SYNOPSIS

```
#include <Cg/cg.h>

CGstatecallback cgGetStateValidateCallback( CGstate state );
```

PARAMETERS

state Specifies the state to retrieve the callback from.

DESCRIPTION

cgGetStateValidateCallback returns the callback function used for validating the state when the given state is encountered in a pass in a technique. See the cgSetStateCallbacks manpage and the cgCallStateValidateCallback manpage for more information.

RETURN VALUES

cgStateValidateCallback returns a pointer to the state validateting callback function. If **state** is not a valid state or if it has no callback, **NULL** is returned.

ERRORS

CG_INVALID_STATE_HANDLE_ERROR is generated if **state** does not refer to a valid state.

SEE ALSO

the cgSetStateCallbacks manpage, the cgCallStateValidateCallback manpage, the cgValidateTechnique manpage, and the cgValidatePassState manpage

NAME

cgGetString − gets a special string

SYNOPSIS

```
#include <Cg/cg.h>

const char *cgGetString(CGenum sname);
```

PARAMETERS

sname An enumerant describing the string to be returned.

DESCRIPTION

cgGetString returns an informative string depending on the **sname**. Currently there is only one valid enumerant that may be passed in.

CG_VERSION

Returns the version string of the Cg runtime and compiler.

RETURN VALUES

Returns the string depending on **name**. Returns **NULL** in the event of an error.

ERRORS

CG_INVALID_ENUMERANT_ERROR if **sname** is an invalid enumerant.

SEE ALSO

NAME

cgGetStringAnnotationValue − get an string-valued annotation's value

SYNOPSIS

```
#include <Cg/cg.h>

const char *cgGetStringAnnotationValue(CGannotation ann);
```

PARAMETERS

ann Specifies the annotation.

DESCRIPTION

cgStringAnnotationValue allows the application to retrieve the value of a string typed annotation.

RETURN VALUES

Returns a pointer to a string.

If no value is available, **NULL** will be returned.

ERRORS

CG_INVALID_ANNOTATION_HANDLE_ERROR is generated if the handle **ann** is invalid.

SEE ALSO

the cgGetAnnotationType manpage, the cgGetFloatAnnotationValues manpage, the cgGetStringAnnotationValues manpage, and the cgGetBooleanAnnotationValues manpage,

NAME

cgGetStringParameterValue − get the value of a string parameter

SYNOPSIS

```
#include <Cg/cg.h>

const char *cgGetStringParameterValue(CGparameter param);
```

PARAMETERS

param Specifies the parameter whose value will be retrieved.

DESCRIPTION

cgGetStringParameterValue allows the application to get the value of a string parameter.

RETURN VALUES

A constant pointer to the parameter's string is returned.

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if the handle **param** is invalid.

CG_INVALID_PARAMETER_TYPE_ERROR is generated if the type of the given parameter is not **CG_STRING**.

SEE ALSO

the cgSetStringParameterValue manpage

NAME

cgGetStringStateAssignmentValue − get a string-valued state assignment's values

SYNOPSIS

```
#include <Cg/cg.h>

const char *cgGetStringStateAssignmentValue(CGstateassignment sa);
```

PARAMETERS

sa Specifies the state assignment.

DESCRIPTION

cgGetStringStateAssignmentValues allows the application to retrieve the *value*(s) of a string typed state assignment.

RETURN VALUES

Returns a pointer to a string.

ERRORS

CG_INVALID_STATE_ASSIGNMENT_HANDLE_ERROR is generated if the handle **sa** is invalid.

CG_INVALID_PARAMETER_TYPE_ERROR is generated if the state assignment is not string-typed.

SEE ALSO

the cgGetStateAssignmentState manpage, the cgGetStateType manpage, the cgGetFloatStateAssignmentValues manpage, the cgGetIntStateAssignmentValues manpage, the cgGetBoolStateAssignmentValue manpage, the cgGetProgramStateAssignmentValue manpage, the cgGetSamplerStateAssignmentValue manpage, and the cgGetTextureStateAssignmentValue manpage

NAME

cgGetTechniqueEffect − get a technique's effect

SYNOPSIS

```
#include <Cg/cg.h>

CGeffect cgGetTechniqueEffect( CGtechnique technique );
```

PARAMETERS

technique

Specifies the technique.

DESCRIPTION

cgGetTechniqueEffect allows the application to retrieve a handle to the effect a given technique belongs to.

RETURN VALUES

Returns a **CGeffect** handle to the effect. In the event of an error **NULL** is returned.

ERRORS

CG_INVALID_TECHNIQUE_HANDLE_ERROR is generated if **technique** does not refer to a valid technique.

SEE ALSO

the cgCreateEffect manpage, the cgCreateEffectFromFile manpage

NAME

cgGetTechniqueName − get a technique's name

SYNOPSIS

```
#include <Cg/cg.h>

const char * cgGetTechniqueName( CGtechnique tech );
```

PARAMETERS

tech Specifies the technique.

DESCRIPTION

cgGetTechniqueName allows the application to retrieve the name of a technique in a Cg effect. This name can be used later to retrieve the technique from the effect using **cgGetTechniqueByName**.

RETURN VALUES

Returns the null-terminated name string for the technique.

Returns null if **tech** is invalid.

ERRORS

CG_INVALID_TECHNIQUE_HANDLE_ERROR is generated if **tech** does not refer to a valid technique.

SEE ALSO

the cgGetNamedTechnique manpage, the cgGetFirst manpage, and the cgGetNextTechnique manpage,

NAME

cgGetTextureStateAssignmentValue − get a texture-valued state assignment's values

SYNOPSIS

```
#include <Cg/cg.h>

CGparameter cgGetTextureStateAssignmentValue(CGstateassignment sa);
```

PARAMETERS

sa Specifies the state assignment.

DESCRIPTION

cgGetTextureStateAssignmentValues allows the application to retrieve the *value*(s) of a state assignment that stores a texture parameter.

RETURN VALUES

Returns a **CGprogram** handle.

ERRORS

CG_INVALID_STATE_ASSIGNMENT_HANDLE_ERROR is generated if the handle **sa** is invalid.

CG_INVALID_PARAMETER_TYPE_ERROR is generated if the state assignment is not a texture.

SEE ALSO

the cgGetStateAssignmentState manpage, the cgGetStateType manpage, the cgGetFloatStateAssignmentValues manpage, the cgGetIntStateAssignmentValues manpage, the cgGetStringStateAssignmentValue manpage, the cgGetTextureStateAssignmentValue manpage, the cgGetSamplerStateAssignmentValue manpage, and the cgGetTextureStateAssignmentValue manpage

NAME

cgGetType − get the type enumerant assigned to a type name

SYNOPSIS

```
#include <Cg/cg.h>

CGtype cgGetType( const char *type_string );
```

PARAMETERS

type_string

A string containing the type name. The name is case-sensitive.

DESCRIPTION

cgGetType returns the enumerant assigned to a type name.

The following is an example of how cgGetType might be used.

```
CGtype Float4Type = cgGetType("float4");

if(cgGetParameterType(myparam) == Float4Type)
 {
  /* Do stuff */
 }
```

RETURN VALUES

Returns the type enumerant of **type_string**. If no such type exists **CG_UNKNOWN_TYPE** will be returned.

ERRORS

SEE ALSO

the cgGetTypeString manpage, the cgGetParameterType manpage

NAME

cgGetTypeString − get the type name associated with a type enumerant

SYNOPSIS

```
#include <Cg/cg.h>

const char * cgGetTypeString( CGtype type );
```

PARAMETERS

type The type enumerant.

DESCRIPTION

cgGetTypeString returns the type named associated with a type enumerant. =back

RETURN VALUES

Returns the type string of the enumerant **type**.

ERRORS

SEE ALSO

the cgGetType manpage, the cgGetParameterType manpage

NAME

cgGetUserType − get enumerant of user-defined type from a program or effect

SYNOPSIS

```
#include <Cg/cg.h>

CGtype cgGetUserType(CGhandle handle, int index);
```

PARAMETERS

handle The **CGprogram** or **CGeffect** in which the type is defined.

index The index of the user-defined type. **index** must be greater than or equal to **0** and less than the value returned by a corresponding call to cgGetNumUserTypes.

DESCRIPTION

cgGetUserType returns the enumerant associated with the user-defined type with the given **index** in the given **CGprogram** or **CGeffect**.

RETURN VALUES

Returns the type enumerant associated with the type with the given **index**.

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if **handle** is not a valid **CGprogram** or **CGeffect**.

CG_OUT_OF_ARRAY_BOUNDS_ERROR is generated if **index** is outside the proper range.

SEE ALSO

the cgGetNumUserTypes manpage, and the cgGetNamedUserType manpage

NAME

cgIsAnnotation − determine if a CGannotation references a valid Cg annotation

SYNOPSIS

```
#include <Cg/cg.h>

CGbool cgIsAnnotation(CGannotation ann);
```

PARAMETERS

ann Specifies the annotation handle to check.

DESCRIPTION

cgIsAnnotation returns **CG_TRUE** if **ann** references a valid annotation, **CG_FALSE** otherwise.

ERRORS

This function generates no errors.

NAME

cgIsContext − determine if a CGcontext references a Cg valid context

SYNOPSIS

```
#include <Cg/cg.h>

CGbool cgIsContext(CGcontext ctx);
```

PARAMETERS

ctx Specifies the context handle to check.

DESCRIPTION

cgIsContext returns **CG_TRUE** if **ctx** references a valid context, **CG_FALSE** otherwise.

ERRORS

SEE ALSO

the cgCreateContext manpage, the cgDestroyContext manpage

NAME

cgIsEffect − determine if a CGeffect references a valid Cg effect

SYNOPSIS

```
#include <Cg/cg.h>

CGbool cgIsEffect(CGeffect effect);
```

PARAMETERS

effect Specifies the effect handle to check.

DESCRIPTION

cgIsEffect returns **CG_TRUE** if **effect** references a valid effect, **CG_FALSE** otherwise.

ERRORS

This function generates no errors.

SEE ALSO

the cgCreateEffect manpage, the cgCreateEffectFromFile manpage

NAME

cgIsInterfaceType − determine if a type is an interface

SYNOPSIS

```
#include <Cg/cg.h>

CGbool cgIsInterfaceType(CGtype type);
```

PARAMETERS

type Specifies the type being evaluated.

DESCRIPTION

cgIsInterfaceType returns **CG_TRUE** if **type** is an interface (not just a struct), **CG_FALSE** otherwise.

ERRORS

SEE ALSO

the cgGetType manpage

NAME

cgIsParameter − determine if a CGparameter references a valid Cg parameter object

SYNOPSIS

```
#include <Cg/cg.h>

CGbool cgIsParameter( CGparameter param );
```

PARAMETERS

param Specifies the parameter handle to check.

DESCRIPTION

cgIsParameter returns **CG_TRUE** if **param** references a valid parameter object. This function is typically used for iterating through the parameters of an object. It can also be used as a consistency check when the application caches CGparameter handles. Certain program operations like deleting the program or context object that the parameter is contained in will cause a parameter object to become invalid.

RETURN VALUES

Returns CG_TRUE if **param** references a valid parameter object.

Returns CG_FALSE otherwise.

ERRORS

SEE ALSO

the cgGetNextParameter manpage

NAME

cgIsParameterGlobal − determine if a parameter is global

SYNOPSIS

```
#include <Cg/cg.h>

CGbool cgIsParameterGlobal(CGparameter param);
```

PARAMETERS

param Specifies the parameter handle to check.

DESCRIPTION

cgIsParameterGlobal returns **CG_TRUE** if **param** is a global parameter and **CG_FALSE** otherwise.

RETURN VALUES

Returns **CG_TRUE** if **param** is global.

Returns **CG_FALSE** otherwise.

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if the handle **param** is invalid.

SEE ALSO

NAME

cgIsParameterReferenced − determine if a program parameter is potentially referenced

SYNOPSIS

```
#include <Cg/cg.h>

CGbool cgIsParameterReferenced( CGparameter param );
```

PARAMETERS

param Specifies the handle of the parameter to check.

DESCRIPTION

cgIsParameterReferenced returns **CG_TRUE** if **param** is a program parameter, and is potentially referenced (used) within the program. It otherwise returns **CG_FALSE**.

Program parameters are those parameters associated directly with a CGprogram, whose handles are retrieved by calling, for example, cgGetNamedProgramParameter.

The value returned by **cgIsParameterReferenced** is conservative, but not always exact. A return value of **CG_TRUE** indicates that the parameter may be used by its associated program. A return value of **CG_FALSE** indicates that the parameter is definintely not referenced by the program.

If **param** is an aggregate program parameter (a struct or array), **CG_TRUE** is returned if any of **param**'s children are potentially referenced by the program.

If **param** is a leaf parameter and the return value is **CG_FALSE**, **cgGetParameterResource** may return **CG_INVALID_VALUE** for this parameter.

RETURN VALUES

Returns **CG_TRUE** if **param** is a program parameter, and is potentially referenced by the program.

Returns **CG_FALSE** otherwise.

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if the handle **param** is invalid.

SEE ALSO

the cgGetNamedProgramParameter manpage, the cgIsParameterUsed manpage, the cgGetParameterResource manpage

NAME

cgIsParameterUsed − determine if a CGparameter is potentially used

SYNOPSIS

```
#include <Cg/cg.h>

CGbool cgIsParameterUsed( CGparameter param, CGhandle container );
```

PARAMETERS

param Specifies the parameter to check.

container

Specifies the CGeffect, CGtechnique, CGpass, CGstateassignment, or CGprogram that may potentially use **param**.

DESCRIPTION

cgIsParameterUsed returns **CG_TRUE** if **param** is potentially used by the given **container**. If **param** is a struct or array, **CG_TRUE** is returned if any of its children are potentially used by **container**. It otherwise returns **CG_FALSE**.

The value returned by **cgIsParameterUsed** is conservative, but not always exact. A return value of **CG_TRUE** indicates that the parameter may be used by **container**. A return value of **CG_FALSE** indicates that the parameter is definintely not used by **container**.

The given **param** handle may reference a program parameter, an effect parameter, or a shared parameter.

The **container** handle may reference a CGeffect, CGtechnique, CGpass, CGstateassignment, or CGprogram.

If **container** is a CGprogram, **CG_TRUE** is returned if any of the program's referenced parameters inherit their values directly or indirectly (due to parameter connections) from **param**.

If **container** is a CGstateassignment, **CG_TRUE** is returned if the right-hand side of the state assignment may directly or indirectly depend on the value of **param**. If the state assignment involves a **CGprogram**, the program's parameters are also considered, as above.

If **container** is a CGpass, **CG_TRUE** is returned if any of the pass' state assignments potentially use **param**.

If **container** is a CGtechnique, **CG_TRUE** is returned if any of the technqiue's passes potentially use **param**.

If **container** is a CGeffect, **CG_TRUE** is returned if any of the effect's techniques potentially use **param**.

RETURN VALUES

CG_TRUE is returned if **param** is potentially used by **container**. **CG_FALSE** is returned otherwise.

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if the **param** handle is invalid, or if **container** is not the handle of a valid container.

SEE ALSO

the cgIsParameterReferenced manpage, the cgConnectParameter manpage

NAME

cgIsParentType − determine if a type is a parent of another type

SYNOPSIS

```
#include <Cg/cg.h>

CGbool cgIsParentType(CGtype parent, CGtype child);
```

PARAMETERS

parent Specifies the parent type.

child Specifies the child type.

DESCRIPTION

cgIsParentType returns **CG_TRUE** if **parent** is a parent type of **child**. Otherwise **CG_FALSE** is returned.

ERRORS

This function does not generate any errors.

SEE ALSO

the cgGetParentType manpage

NAME

cgIsPass − determine if a CGpass references a valid Cg pass

SYNOPSIS

```
#include <Cg/cg.h>

CGbool cgIsPass(CGpass pass);
```

PARAMETERS

pass Specifies the pass handle to check.

DESCRIPTION

cgIsPass returns **CG_TRUE** if **pass** references a valid pass, **CG_FALSE** otherwise.

ERRORS

This function generates no errors.

NAME

cgIsProgram − determine if a CGprogram handle references a Cg program object

SYNOPSIS

```
#include <Cg/cg.h>

CGbool cgIsProgram( CGprogram prog );
```

PARAMETERS

prog Specifies the program handle to check.

DESCRIPTION

cgIsProgram return CG_TRUE if **prog** references a valid program object. Note that this does not imply that the program has been successfully compiled.

RETURN VALUES

Returns CG_TRUE if **prog** references a valid program object.

Returns CG_FALSE otherwise.

ERRORS

SEE ALSO

the cgCreateProgram manpage, the cgDestroyProgram manpage, the cgGetNextProgram manpage

NAME

cgIsProgramCompiled − determine if a program has been compiled

SYNOPSIS

```
#include <Cg/cg.h>

CGbool cgIsProgramCompiled(CGprogram prog);
```

PARAMETERS

prog Specifies the program.

DESCRIPTION

cgIsProgramCompiled returns **CG_TRUE** if **prog** has been compiled and **CG_FALSE** otherwise.

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if **prog** is invalid.

SEE ALSO

the cgCompileProgram manpage, and the cgSetAutoCompile manpage

NAME

cgIsState − determine if a CGstate references a Cg valid state

SYNOPSIS

```
#include <Cg/cg.h>

CGbool cgIsState(CGstate state);
```

PARAMETERS

state Specifies the state handle to check.

DESCRIPTION

cgIsState returns **CG_TRUE** if **state** references a valid state, **CG_FALSE** otherwise.

ERRORS

This function generates no errors.

SEE ALSO

the cgCreateState manpage

NAME

cgIsStateAssignment − determine if a CGstate assignment references a valid Cg state assignment

SYNOPSIS

```
#include <Cg/cg.h>

CGbool cgIsStateAssignment(CGstateassignment sa);
```

PARAMETERS

sa Specifies the state assignment handle to check.

DESCRIPTION

cgIsStateAssignment returns **CG_TRUE** if **sa** references a valid state assignment, **CG_FALSE** otherwise.

ERRORS

This function generates no errors.

NAME

cgIsTechnique − determine if a CGtechnique references a valid Cg technique

SYNOPSIS

```
#include <Cg/cg.h>

CGbool cgIsTechnique(CGtechnique technique);
```

PARAMETERS

technique

Specifies the technique handle to check.

DESCRIPTION

cgIsTechnique returns **CG_TRUE** if **technique** references a valid technique, **CG_FALSE** otherwise.

ERRORS

This function generates no errors.

SEE ALSO

the cgCreateTechnique manpage, the cgDestroyTechnique manpage

NAME

cgIsTechniqueValidated − indicates whether the technique has passed validation

SYNOPSIS

```
#include <Cg/cg.h>

CGbool cgIsTechniqueValidated(CGtechnique technique);
```

PARAMETERS

technique

Specifies the technique handle.

DESCRIPTION

cgIsTechniqueValidated returns **CG_TRUE** if the technique has previously passes validation via a call to the cgValidateTechnique manpage. **CG_FALSE** is returned both if validation hasn't been attempted as well as if the technique has failed a validation attempt.

ERRORS

CG_INVALID_TECHNIQUE_HANDLE_ERROR is generated if **technique** does not refer to a valid technique.

SEE ALSO

the cgValidateTechnique manpage, the CallStateValidateCallback manpage

NAME

cgResetPassState − calls the state resetting callback functions for all of the state assignments in a pass.

SYNOPSIS

```
#include <Cg/cg.h>

void cgResetPassState(CGpass pass);
```

PARAMETERS

pass Specifies the pass handle.

DESCRIPTION

cgResetPassState resets all of the graphics state defined in a pass by calling the state resetting callbacks for all of the state assignments in the pass.

The semantics of ‘‘resetting state’’ will depend on the particular graphics state manager that defined the valid state assignments; it will generally either mean that graphics state is reset to what it was before the pass, or that it is reset to the default value. The OpenGL state manager in the OpenGL Cg runtime implements the latter approach.

ERRORS

CG_INVALID_PASS_ERROR is generated if **pass** does not refer to a valid pass.

CG_INVALID_TECHNIQUE_ERROR if the technique that the pass is a part of has failed validation.

SEE ALSO

the cgSetPassState manpage, the cgCallStateResetCallback manpage

NAME

cgSetArraySize − sets the size of a resizable array parameter

SYNOPSIS

```
#include <Cg/cg.h>

void cgSetArraySize(CGparameter param, int size);
```

PARAMETERS

param Specifies the array parameter handle.

size Specifies the new size of the array.

DESCRIPTION

cgSetArraySize sets the size of a resiable array parameter **param** to **size**.

EXAMPLE

If you have Cg program with a parameter like this :

```
// ...

float4 main(float4 myarray[])
 {
  // ...
 }
```

You can set the size of the **myarray** array parameter to **5** like so :

```
CGparameter MyArrayParam =
 cgGetNamedProgramParameter(Program, CG_PROGRAM, "myarray");

cgSetArraySize(MyArrayParam, 5);
```

RETURN VALUES

cgSetArraySize does not return any values.

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if **param** is an invalid parameter handle or not an array.

CG_ARRAY_PARAM_ERROR if **param** is not an array param.

CG_INVALID_DIMENSION_ERROR is generated if the dimension of the array parameter **param** is not 1.

CG_PARAMETER_IS_NOT_RESIZABLE_ARRAY_ERROR is generated if **param** is not a resizable array.

SEE ALSO

the cgGetArraySize manpage, the cgGetArrayDimension manpage, and the cgSetMultiDimArraySize manpage

NAME

cgSetAutoCompile − sets the auto-compile mode for a context

SYNOPSIS

```
#include <Cg/cg.h>

void cgSetAutoCompile(CGcontext ctx, CGenum flag);
```

PARAMETERS

ctx Specifies the context.

flag The auto-compile mode to set the **ctx** to. Must be one of the following :

- **CG_COMPILE_MANUAL**
- **CG_COMPILE_IMMEDIATE**
- **CG_COMPILE_LAZY**

DESCRIPTION

cgSetAutoCompile sets the auto compile mode for a given context. By default, programs are immediately recompiled when they enter an uncompiled state. This may happen for a variety of reasons including :

- Setting the value of a literal parameter.
- Resizing arrays.
- Binding structs to interface parameters.

flag may be one of the following three enumerants :

- **CG_COMPILE_IMMEDIATE**
  CG_COMPILE_IMMEDIATE will force recompilation automatically and immediately when a program enters an uncompiled state. This is the default mode.
- **CG_COMPILE_MANUAL**
  With this method the application is responsible for manually recompiling a program. It may check to see if a program requires recompilation with the entry point cgIsProgramCompiled. cgCompileProgram can then be used to force compilation.
- **CG_COMPILE_LAZY**
  This method is similar to **CG_COMPILE_IMMEDIATE** but will delay program recompilation until the program object code is needed. The advantage of this method is the reduction of extraneous recompilations. The disadvantage is that compile time errors will not be encountered when the program is enters the uncompiled state but will instead be encountered at some later time.

ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if **ctx** is invalid.

CG_INVALID_ENUMERANT_ERROR is generated if **flag** is invalid.

SEE ALSO

the cgCompileProgram manpage, and the cgIsProgramCompiled manpage

NAME

cgGetErrorCallback, **cgSetErrorCallback** − get and set the error callback

SYNOPSIS

```
#include <Cg/cg.h>

typedef void (*CGerrorCallbackFunc)(void);

void cgSetErrorCallback( CGerrorCallbackFunc func );
CGerrorCallbackFunc cgGetErrorCallback( void );
```

PARAMETERS

func A function pointer to the callback function.

DESCRIPTION

cgSetErrorCallback sets a callback function that will be called every time an error occurrs. The callback function is not passed any parameters. It is assumed that the callback function will call cgGetError to obtain the current error. To disable the callback function, cgSetErrorCallback may be called with **NULL**.

cgGetErrorCallback returns the currently set callback function. **NULL** will be returned if no callback function is set.

The following is an example of how to set and use an error callback :

```
void MyErrorCallback( void ) {
  int Error = cgGetError();
  fprintf(stderr, "ERROR : %s\n", cgGetErrorString(Error));
}

void main(int argc, char *argv[])
 {
  cgSetErrorCallback(MyErrorCallback);

  /* Do stuff */
 }
```

ERRORS

SEE ALSO

the cgGetError manpage, the cgGetErrorString manpage

NAME

cgGetErrorHandler, **cgSetErrorHandler** − get and set the error handler callback

SYNOPSIS

```
#include <Cg/cg.h>

typedef void (*CGerrorHandlerFunc)(CGcontext context, CGerror err, void *appdata);

void cgSetErrorHandler(CGerrorHandlerFunc func, void *appdata);
CGerrorHandlinerFunc cgGetErrorHandler(void **appdataptr);
```

PARAMETERS

func A pointer to the error handler callback function.

appdata A pointer to arbitrary application-provided data.

DESCRIPTION

cgSetErrorHandler specifies an error handler function that will be called every time a Cg runtime error occurrs. The callback function is passed:

context
The context in which the error occured. If the context cannot be determined, a context handle of 0 is used.

err The enumerant of the error triggering the callback.

appdata
The value of the pointer passed to **cgSetErrorHandler**. This pointer can be used to make arbitrary application-side information available to the error handler.

To disable the callback function, specify a **NULL** callback function pointer via **cgSetErrorHandler**.

cgGetErrorHandler returns the currently error handler callback function. **NULL** will be returned if no callback function is set. The value of the current **appdata** pointer will be copied into the location pointed to by **appdataptr** if **appdataptr** is not **NULL**.

The following is an example of how to set and use an error handler:

```
void MyErrorHandler(CGcontext ctx, CGerror err, void *data) {
  char *progname = (char *)data;
  fprintf(stderr, "%s: Error: %s\n", progname, cgGetErrorString(err));
}

void main(int argc, char *argv[])
{
  ...
  cgSetErrorHandler(MyErrorHandler, (void *)argv[0]);
  ...
}
```

ERRORS

SEE ALSO

the cgGetError manpage, the cgGetFirstError manpage, the cgGetErrorString manpage

NAME

cgSetMatrixParameter − sets the value of matrix parameters

SYNOPSIS

```
#include <Cg/cg.h>

/* type is int, float or double */
void cgSetMatrixParameter{ifd}{rc}(CGparameter param, const type *matrix);
```

PARAMETERS

param Specifies the parameter that will be set.

matrix An array of values to set the matrix parameter to. The array must be the number of rows times the number of columns in size.

DESCRIPTION

The cgSetMatrixParameter functions set the value of a given matrix parameter. The functions are available in various combinations.

There are versions of each function that take **int**, **float** or **double** values signified by the **i**, **f** or **d** in the function name.

There are versions of each function that assume the array of values are layed out in either row or column order signified by the **r** or **c** in the function name respectively.

The cgSetMatrixParameter functions may only be called with uniform parameters.

RETURN VALUES

The cgSetMatrixParameter functions do not return any values.

ERRORS

CG_NOT_MATRIX_PARAM_ERROR is generated if **param** is not a matrix parameter.

CG_INVALID_PARAM_HANDLE_ERROR is generated if **param** is not a valid parameter handle.

CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

SEE ALSO

the cgGetParameterRows manpage, the cgGetParameterColumns manpage, the cgGetMatrixParameterArray manpage, and the cgGetParameterValues manpage

NAME

cgSetMultiDimArraySize − sets the size of a resizable multi-dimensional array parameter

SYNOPSIS

```
#include <Cg/cg.h>

void cgSetMultiDimArraySize(CGparameter param, const int *sizes);
```

PARAMETERS

param Specifies the array parameter handle.

sizes An array of sizes for each dimension of the array.

DESCRIPTION

cgSetMultiDimArraySize sets the size of each dimension of resiable multi-dimensional array parameter **param**. **sizes** must be an array that has **N** number of elements where **N** is equal to the result of cgGetArrayDimension.

EXAMPLE

If you have Cg program with a parameter like this :

```
// ...

float4 main(float4 myarray[][][])
 {
  // ...
 }
```

You can set the sizes of each dimension of the **myarray** array parameter like so :

```
const int Sizes[] = { 3, 2, 4 };
CGparameter MyArrayParam =
 cgGetNamedProgramParameter(Program, CG_PROGRAM, "myarray");

cgSetMultiDimArraySize(MyArrayParam, Sizes);
```

RETURN VALUES

cgSetMultiArraySize does not return any values.

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if **param** is an invalid parameter handle or not an array.

CG_ARRAY_PARAM_ERROR if **param** is not an array param.

CG_PARAMETER_IS_NOT_RESIZABLE_ARRAY_ERROR is generated if **param** is not a resizable array.

SEE ALSO

the cgGetArraySize manpage, the cgGetArrayDimension manpage, and the cgSetArraySize manpage

NAME

cgSetParameter − sets the value of scalar and vector parameters

SYNOPSIS

```
#include <Cg/cg.h>

/* type is int, float or double */

void cgSetParameter1{ifd}(CGparameter param,
                          type x);

void cgSetParameter2{ifd}(CGparameter param,
                          type x,
                          type y);

void cgSetParameter3{ifd}(CGparameter param,
                          type x,
                          type y,
                          type z);

void cgSetParameter4{ifd}(CGparameter param,
                          type x,
                          type y,
                          type z,
                          type w);

void cgSetParameter{1234}{ifd}v(CGparameter param,
                                const type *v);
```

PARAMETERS

param Specifies the parameter that will be set.

x, y, z, and w
: The values to set the parameter to.

v The values to set the parameter to for the array versions of the set functions.

DESCRIPTION

The cgSetParameter functions set the value of a given scalar or vector parameter. The functions are available in various combinations.

Each function takes either 1, 2, 3, or 4 values depending on the function that is used. If more values are passed in than the parameter requires, the extra values will be ignored. If less values are passed in than the parameter requires, the last value will be smeared.

There are versions of each function that take **int**, **float** or **double** values signified by the **i**, **f** or **d** in the function name.

The functions with the **v** at the end of their names take an array of values instead of explicit parameters.

Once cgSetParameter has been used to set a parameter, the values may be retrieved from the parameter using the **CG_CURRENT** enumerant with cgGetParameterValues.

If an API-dependant layer of the Cg runtime (e.g. cgGL) is used, these entry points may end up making API (e.g. OpenGL) calls.

RETURN VALUES

The cgSetParameter functions do not return any values.

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if **param** is not a valid parameter handle.

SEE ALSO

the cgGetParameterValue manpage

NAME

cgSetParameterSemantic − set a program parameter's semantic

SYNOPSIS

```
#include <Cg/cg.h>

void cgGetParameterSemantic( CGparameter param, const char *semantic );
```

PARAMETERS

param Specifies the program parameter.

semantic
Specifies the semantic.

DESCRIPTION

cgSetParameterSemantic allows the application to set the semantic of a parameter in a Cg program.

RETURN VALUES

None.

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if the handle **param** is invalid.

CG_INVALID_PARAMETER_ERROR is generated if the parameter is not a leaf node or if the semantic string is NULL.

SEE ALSO

the cgGetParameterResource manpage, the cgGetParameterResourceIndex manpage, the cgGetParameterName manpage, and the cgGetParameterType manpage

NAME

cgSetParameterValue − set the value of any numeric parameter

SYNOPSIS

```
#include <Cg/cg.h>

/* type may be int, float, or double */
void cgSetParameterValue{i,f,d}{r,c}(CGparameter param,
                                     int nvals, type *v);
```

PARAMETERS

param Specifies the program parameter whose value will be set.

nvals The length of the **v** array, in number of elements.

v Source buffer from which the parameter values will be read.

DESCRIPTION

cgSetParameterValue allows the application to set the value of any numeric parameter or parameter array.

The given parameter must be a scalar, vector, matrix, or a (possibly-multidimensional) array of scalars, vectors, or matrices.

There are versions of each function that take **int**, **float** or **double** values signified by the **i**, **f** or **d** in the function name.

There are versions of each function that will cause any matrices referenced by **param** to be initialized in either row-major or column-major order, as signified by the **r** or **c** in the function name.

For example, **cgSetParameterValueic** sets the given parameter using the supplied array of integer data, and initializes matrices in column-major order.

If **v** is smaller than the total number of values in the given source parameter, **CG_NOT_ENOUGH_DATA_ERROR** is generated.

The total number of values in a parameter, ntotal, may be computed as follow:

```
int nrows = cgGetParameterRows(param);
int ncols = cgGetParameterColumns(param);
int asize = cgGetArrayTotalSize(param);
int ntotal = nrows*ncols;
if (asize > 0) ntotal *= asize;
```

RETURN VALUES

No value is returned.

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if the handle **param** is invalid.

CG_INVALID_POINTER_ERROR is generated if **v** is **NULL**.

CG_NOT_ENOUGH_DATA_ERROR is generated if **nvalues** is less than the total size of **param**.

CG_NON_NUMERIC_PARAMETER_ERROR is generated if **param** is of a non-numeric type.

SEE ALSO

the cgGetParameterRows manpage, the cgGetParameterColumns manpage, the cgGetArrayTotalSize manpage, and the cgGetParameterValue manpage

NAME

cgSetParameterVariability − set a parameter's variability

SYNOPSIS

```
#include <Cg/cg.h>

void cgSetParameterVariability(CGparameter param, CGenum vary);
```

PARAMETERS

param Specifies the parameter.

vary The variability the **param** will be set to.

DESCRIPTION

cgSetParameterVariability allows the application to change the variability of a parameter.

Currently parameters may not be changed to or from **CG_VARYING** variability. However parameters of **CG_UNIFORM** and **CG_LITERAL** variability may be changed.

Valid values for **vary** include :

CG_UNIFORM
A uniform parameter is one whose value does not change with each invocation of a program, but whose value can change between groups of program invocations.

CG_LITERAL
A literal parameter is folded out at compile time. Making a uniform parameter literal will often make a program more efficient at the expense of requiring a compile every time the value is set.

CG_DEFAULT
By default, the variability of a parameter will be overridden by the a source parameter connected to it unless it is changed with cgSetParameterVariability. If it is set to **CG_DEFAULT** it will restore the default state of assuming the source parameters variability.

RETURN VALUES

This function does not return a value.

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if the handle **param** is invalid.

CG_INVALID_ENUMERANT_ERROR is generated if **vary** is not a valid enumerant.

CG_INVALID_PARAMETER_VARIABILITY_ERROR is generated if the parameter could not be changed to the variability indicated by **vary**.

SEE ALSO

the cgGetParameterVariability manpage

NAME

cgSetPassProgramParameters − set uniform parameters specified via a compile statement

SYNOPSIS

```
#include <Cg/cg.h>

void cgSetPassProgramParameters(CGprogram prog);
```

PARAMETERS

prog Specifies the program

DESCRIPTION

Given the handle to a program specified in a pass in a CgFX file, **cgSetPassProgramParameters** sets the values of the program's uniform parameters given the expressions in the **compile** statement in the CgFX file.

(This entrypoint is normally only needed by state managers and doesn't need to be called by users.)

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if **prog** does not refer to a valid program.

NAME

cgSetPassState − calls the state setting callback functions for all of the state assignments in a pass.

SYNOPSIS

```
#include <Cg/cg.h>

void cgSetPassState(CGpass pass);
```

PARAMETERS

pass Specifies the pass handle.

DESCRIPTION

cgSetPassState sets all of the graphics state defined in a pass by calling the state setting callbacks for all of the state assignments in the pass.

ERRORS

CG_INVALID_PASS_ERROR is generated if **pass** does not refer to a valid pass.

CG_INVALID_TECHNIQUE_ERROR if the technique that the pass is a part of has failed validation.

SEE ALSO

the cgResetPassState manpage, the cgCallStateSetCallback manpage

NAME

cgSetSamplerState − initializes the state specified for a sampler parameter

SYNOPSIS

```
#include <Cg/cg.h>

void cgSetSamplerState(CGparameter param);
```

PARAMETERS

parameter

Specifies the parameter handle.

DESCRIPTION

cgSetSamplerState sets the sampler state for a sampler parameter that was specified via a **sampler_state** block in a CgFX file. The corresponding sampler should be bound via the graphics API before this call is made.

ERRORS

CG_INVALID_PARAMETER_HANDLE_ERROR is generated if **param** is not a valid parameter handle.

NAME

cgSetStateCallbacks − registers the set, reset, and validation callback functions for a state assignment.

SYNOPSIS

```
#include <Cg/cg.h>

void cgSetStateCallbacks(CGstate state, CGstatecallback set, CGstatecallback reset
                         CGstatecallback validate);
```

PARAMETERS

state Specifies the state handle.

set Specifices the pointer to the callback function to call for setting the state of state assignments based on **state**. This may be a **NULL** pointer.

reset Specifices the pointer to the callback function to call for resetting the state of state assignments based on **state**. This may be a **NULL** pointer.

validate Specifices the pointer to the callback function to call for validating the state of state assignments based on **state**. This may be a **NULL** pointer.

DESCRIPTION

cgSetStateCallbacks sets the three callback functions for a state definition. These functions are later called when the state a particular state assignment based on this state must be set, reset, or validated. Any of the callback functions may be specified as **NULL**.

ERRORS

CG_INVALID_STATE_ERROR is generated if **state** does not refer to a valid state definition.

SEE ALSO

the cgSetPassState manpage, the cgCallStateSetCallback manpage, the cgCallStateResetCallback manpage, the cgCallStateValidateCallback manpage, the cgValidateTechnique manpage.

NAME

cgSetStringParameterValue − get the value of a string parameter

SYNOPSIS

```
#include <Cg/cg.h>

void cgSetStringParameterValue(CGparameter param, const char *value);
```

PARAMETERS

param Specifies the parameter whose value will be set.

value Specifies the string to set the parameter's value as.

DESCRIPTION

cgSetStringParameterValue allows the application to set the value of a string parameter.

RETURN VALUES

None.

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if the handle **param** is invalid.

CG_INVALID_PARAMETER_TYPE_ERROR is generated if the type of the given parameter is not **CG_STRING**.

SEE ALSO

the cgSetStringParameterValue manpage

NAME

cgValidateTechnique − validate a technique from an effect

SYNOPSIS

```
#include <Cg/cg.h>

CGbool cgValidateTechnique(CGtechnique technique);
```

PARAMETERS

technique

Specifies the technique handle to validate.

DESCRIPTION

cgValidateTechnique returns **CG_TRUE** if all of the state assignments in all of the passes in **technique** are valid and can be used on the current hardware and **CG_FALSE** otherwise.

This function iterates over all state assignments in all passes and calls the cgCallStateValidateCallback manpage to test to see if the state assignment passes validation. If any state assignment fails validation, **CG_FALSE** is returned.

ERRORS

CG_INVALID_TECHNIQUE_HANDLE_ERROR is generated if **technique** does not refer to a valid technique.

SEE ALSO

the cgCallStateValidateCallback manpage, the cgSetStateCallbacks manpage

NAME

cgGLBindProgram − prepares a program for binding

SYNOPSIS

```
#include <Cg/cgGL.h>

void cgGLBindProgram(CGprogram prog);
```

PARAMETERS

prog Specifies the program.

DESCRIPTION

cgGLBindProgram binds a program to the current state. The program must have been loaded with cgGLLoadProgram before it can be bound. Also, the profile of the program must be enabled for the binding to work. This may be done with the cgGLEnableProfile function.

cgGLBindProgram will reset all uniform parameters that were set with the **cgGLSet*XXXXX*()** functions for profiles that do not support program local parameters (e.g. the vp20 profile).

RETURN VALUES

cgGLBindProgram does not return any values.

ERRORS

CG_INVALID_PROFILE_ERROR is generated if **prog**'s profile is not a supported OpenGL profile.

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if **prog** is not a valid program.

CG_PROGRAM_BIND_ERROR is generated if the program fails to bind for any reason.

SEE ALSO

the cgGLLoadProgram manpage, the cgGLSetParameter manpage, the cgGLSetMatrixParameter manpage, and the cgGLSetTextureParameter manpage,

NAME

cgGLDisableClientState − disables a vertex attribute in the OpenGL state

SYNOPSIS

```
#include <Cg/cgGL.h>

void cgGLDisableClientState(CGparameter param);
```

PARAMETERS

param Specifies the parameter.

DESCRIPTION

cgGLDisableClientState disables the vertex attribute associated with the given varying parameter.

RETURN VALUES

cgGLDisableClientState does not return any values.

ERRORS

CG_INVALID_PARAMETER_ERROR is generated if **param** is not a varying parameter.

SEE ALSO

the cgGLEnableClientState manpage

NAME

cgGLDisableProfile − disable a profile within OpenGL

SYNOPSIS

```
#include <Cg/cgGL.h>

void cgGLDisableProfile(CGprofile profile);
```

PARAMETERS

profile The profile enumerant.

DESCRIPTION

cgGLDisableProfile disables a given profile by making the appropriate OpenGL calls. For most profiles, this will simply make a call to **glDisable** with the approriate enumerant.

RETURN VALUES

cgGLDisableProfile does not return any values.

ERRORS

CG_INVALID_PROFILE_ERROR is generated if **profile** is invalid.

SEE ALSO

the cgGLEnableProfile manpage

NAME

cgGLDisableTextureParameter − disables the texture unit associated with the given texture parameter

SYNOPSIS

```
#include <Cg/cgGL.h>

void cgGLDisableTextureParameter(CGparameter param);
```

PARAMETERS

param Specifies the texture parameter that has a texture object associated with it.

DESCRIPTION

cgGLDisableTextureParameter unbinds and disables the texture object that was associated with the parameter **param**.

See the cgGLEnableTextureParameter manpage for more information.

RETURN VALUES

cgGLDisableTextureParameter does not return any values.

ERRORS

CG_INVALID_PROFILE_ERROR is generated if **param**'s profile is not a supported OpenGL profile.

CG_INVALID_PARAM_HANDLE_ERROR is generated if **param** is not a valid parameter handle.

CG_INVALID_PARAMETER_ERROR is generated **param** is not a texture parameter or if the parameter fails to set for any other reason.

SEE ALSO

the cgGLEnableTextureParameter manpage, and the cgGLSetTextureParameter manpage

NAME

cgGLEnableClientState − enables a vertex attribute in the OpenGL state

SYNOPSIS

```
#include <Cg/cgGL.h>

void cgGLEnableClientState(CGparameter param);
```

PARAMETERS

param Specifies the parameter.

DESCRIPTION

cgGLEnableClientState enables the vertex attribute associated with the given varying parameter.

RETURN VALUES

cgGLEnableClientState does not return any values.

ERRORS

CG_INVALID_PARAMETER_ERROR is generated if **param** is not a varying parameter.

SEE ALSO

the cgGLDisableClientState manpage

NAME

cgGLEnableProfile − enable a profile within OpenGL

SYNOPSIS

```
#include <Cg/cgGL.h>

void cgGLEnableProfile(CGprofile profile);
```

PARAMETERS

profile The profile enumerant.

DESCRIPTION

cgGLEnableProfile enables a given profile by making the appropriate OpenGL calls. For most profiles, this will simply make a call to **glEnable** with the approriate enumerant.

RETURN VALUES

cgGLEnableProfile does not return any values.

ERRORS

CG_INVALID_PROFILE_ERROR is generated if **profile** is invalid.

SEE ALSO

the cgGLDisableProfile manpage

NAME

cgGLEnableTextureParameter − enables the texture unit associated with the given texture parameter

SYNOPSIS

```
#include <Cg/cgGL.h>

void cgGLEnableTextureParameter(CGparameter param);
```

PARAMETERS

param Specifies the texture parameter that has a texture object associated with it.

DESCRIPTION

cgGLEnableTextureParameter binds and enables the texture object that was associated with the parameter **param**. It must be called after the cgGLSetTextureParameter manpage is called but before the geometry is drawn.

cgGLDisableTextureParameter should be called once all of the geometry is drawn to avoid applying the texture to the wrong geometry and shaders.

RETURN VALUES

cgGLEnableTextureParameter does not return any values.

ERRORS

CG_INVALID_PROFILE_ERROR is generated if **param**'s profile is not a supported OpenGL profile. In particular, if **param** is not a parameter handle retrieved from a **CGprogram** but was instead retrieved from a **CGeffect** or is a shared parameter created at runtime, this error will be generated since those parameters do not have a profile associated with them.

CG_INVALID_PARAM_HANDLE_ERROR is generated if **param** is not a valid parameter handle.

CG_INVALID_PARAMETER_ERROR is generated **param** is not a sampler parameter or if the parameter fails to set for any other reason.

SEE ALSO

the cgGLDisableTextureParameter manpage, and the cgGLSetTextureParameter manpage

NAME

cgGLGetLatestProfile − enable a profile within OpenGL

SYNOPSIS

```
#include <Cg/cgGL.h>

CGprofile cgGLGetLatestProfile(CGGLenum profile_type);
```

PARAMETERS

profile_type

The class of profile that will be returned. **profile_type** may be one of :

CG_GL_VERTEX
For the latest vertex profile.

CG_GL_FRAGMENT
For the latest fragment profile.

DESCRIPTION

cgGLGetLatestProfile returns the best available profile of a given class. It will check the available OpenGL extensions to see what the determine the best profile.

cgGLGetLatestProfile may be used in conjuction with cgCreateProgram to ensure that more optimal profiles are used as they are made available even though they might not be available at compile time or with a given version of the runtime.

RETURN VALUES

cgGLGetLatestProfile returns a profile enumerant for the latest profile of the given class. If no appropriate profile is available or an error occurs **CG_PROFILE_UNKNOWN** is returned.

ERRORS

CG_INVALID_ENUMERANT_ERROR is generated if **profile_type** is invalid.

SEE ALSO

the cgGLSetOptimalOptions manpage, and the cgCreateProgram manpage

NAME

cgGLGetManageTextureParameters − gets the manage texture parameters flag from a context

SYNOPSIS

```
#include <Cg/cgGL.h>

CGbool cgGLGetManageTextureParameters(CGcontext ctx);
```

PARAMETERS

ctx Specifies the context.

DESCRIPTION

Returns the manage texture management flag in a the context **ctx**. See cgGLManageTextureParameters for more information.

ERRORS

SEE ALSO

the cgGLSetManageTextureParameters manpage, the cgGLBindProgram manpage, and the cgGLUnbindProgram manpage

NAME

cgGLGetMatrixParameter − retrieves the value of matrix parameters

SYNOPSIS

```
#include <Cg/cgGL.h>

/* type is either float or double */
void cgGLGetMatrixParameter{fd}{rc}(CGparameter param, const type *matrix);
```

PARAMETERS

param Specifies the parameter that will be retrieved.

matrix An array to retrieve the matrix parameter to. The array must be the number of rows times the number of columns in size.

DESCRIPTION

The cgGLGetMatrixParameter functions retrieve the value of a given matrix parameter. The functions are available in various combinations.

There are versions of each function that take either **float** or **double** values signified by the **f** or **d** in the function name.

There are versions of each function that assume the array of values are layed out in either row or column order signified by the **r** or **c** in the function name respectively.

The cgGLGetMatrixParameter functions may only be called with uniform parameters.

RETURN VALUES

The cgGLGetMatrixParameter functions do not return any values.

ERRORS

CG_INVALID_PROFILE_ERROR is generated if **param**'s profile is not a supported OpenGL profile.

CG_NOT_MATRIX_PARAM_ERROR is generated if **param** is not a matrix parameter.

CG_INVALID_PARAM_HANDLE_ERROR is generated if **param** is not a valid parameter handle.

CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to retrieve for any other reason.

SEE ALSO

the cgGLGetMatrixParameterArray manpage, the cgGLSetMatrixParameterArray manpage and the cgGLSetParameter manpage

NAME

cgGLGetParameterArray − retrieaves an array matrix parameters

SYNOPSIS

```
#include <Cg/cgGL.h>

/* type is either float or double */

void cgGLGetMatrixParameterArray{fd}{rc}(CGparameter param,
                                         long offset,
                                         long nelements,
                                         const type *v);
```

PARAMETERS

param Specifies the array parameter.

offset An offset into the array parameter to start getting from. A value of **0** will start getting from the first element of the array.

nelements

The number of elements to get. A value of **0** will default to the number of elements in the array minus the **offset** value.

v The array retrieve the values into. The size of the array must be **nelements** times the number of elements in the matrix.

DESCRIPTION

The cgGLGetMatrixParameterArray functions retrieve an array of values from a give matrix array parameter. The functions are available in various combinations.

There are versions of each function that take either **float** or **double** values signified by the **f** or **d** in the function name.

There are versions of each function that assume the array of values are layed out in either row or column order signified by the **r** or **c** in the function name respectively.

There are versions of each function that assume the array of values are layed out in either row or column order signified by the **r** or **c** in the function name respectively.

RETURN VALUES

The cgGLGetParameter functions do not return any values.

ERRORS

CG_INVALID_PROFILE_ERROR is generated if **param**'s profile is not a supported OpenGL profile.

CG_ARRAY_PARAM_ERROR is generated if **param** is not an array parameter.

CG_NOT_MATRIX_PARAM_ERROR is generated if the elements of the array indicated by **param** are not matrix parameters.

CG_OUT_OF_ARRAY_BOUNDS_ERROR is generated if the **offset** and/or the **nelements** parameter are out of the array bounds.

CG_INVALID_PARAM_HANDLE_ERROR is generated if **param** is not a valid parameter handle.

CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to retrieve for any other reason.

SEE ALSO

the cgGLGetParameter manpage, the cgGLSetParameter manpage, and the cgGLSetParameterArray manpage

NAME

cgGLGetMatrixParameterArraydc − gets the value of matrix parameter arrays

DESCRIPTION

Please refer to the cgGLGetMatrixParameterArray manpage for more information.

SEE ALSO

the cgGetMatrixParameterArray manpage, the cgGLGetParameter manpage and the cgGetMatrixParameter manpage

NAME

cgGLGetMatrixParameterArraydcv − gets the value of matrix parameter arrays

DESCRIPTION

Please refer to the cgGLGetMatrixParameterArray manpage for more information.

SEE ALSO

the cgGetMatrixParameterArray manpage, the cgGLGetParameter manpage and the cgGetMatrixParameter manpage

NAME

cgGLGetMatrixParameterArraydr − gets the value of matrix parameter arrays

DESCRIPTION

Please refer to the cgGLGetMatrixParameterArray manpage for more information.

SEE ALSO

the cgGetMatrixParameterArray manpage, the cgGLGetParameter manpage and the cgGetMatrixParameter manpage

NAME

cgGLGetMatrixParameterArraydrv − gets the value of matrix parameter arrays

DESCRIPTION

Please refer to the cgGLGetMatrixParameterArray manpage for more information.

SEE ALSO

the cgGetMatrixParameterArray manpage, the cgGLGetParameter manpage and the cgGetMatrixParameter manpage

NAME

cgGLGetMatrixParameterArrayfc − gets the value of matrix parameter arrays

DESCRIPTION

Please refer to the cgGLGetMatrixParameterArray manpage for more information.

SEE ALSO

the cgGetMatrixParameterArray manpage, the cgGLGetParameter manpage and the cgGetMatrixParameter manpage

NAME

cgGLGetMatrixParameterArrayfcv − gets the value of matrix parameter arrays

DESCRIPTION

Please refer to the cgGLGetMatrixParameterArray manpage for more information.

SEE ALSO

the cgGetMatrixParameterArray manpage, the cgGLGetParameter manpage and the cgGetMatrixParameter manpage

NAME

cgGLGetMatrixParameterArrayfr − gets the value of matrix parameter arrays

DESCRIPTION

Please refer to the cgGLGetMatrixParameterArray manpage for more information.

SEE ALSO

the cgGetMatrixParameterArray manpage, the cgGLGetParameter manpage and the cgGetMatrixParameter manpage

NAME

cgGLGetMatrixParameterArrayfrv − gets the value of matrix parameter arrays

DESCRIPTION

Please refer to the cgGLGetMatrixParameterArray manpage for more information.

SEE ALSO

the cgGetMatrixParameterArray manpage, the cgGLGetParameter manpage and the cgGetMatrixParameter manpage

NAME

cgGLGetMatrixParameterdc − gets the value of matrix parameters

DESCRIPTION

Please refer to the cgGLGetMatrixParameter manpage for more information.

SEE ALSO

the cgGetMatrixParameter manpage, the cgGLGetParameter manpage and the cgGetMatrixParameter manpage

NAME

cgGLGetMatrixParameterdcv − gets the value of matrix parameters

DESCRIPTION

Please refer to the cgGLGetMatrixParameter manpage for more information.

SEE ALSO

the cgGetMatrixParameter manpage, the cgGLGetParameter manpage and the cgGetMatrixParameter manpage

NAME

cgGLGetMatrixParameterdr − gets the value of matrix parameters

DESCRIPTION

Please refer to the cgGLGetMatrixParameter manpage for more information.

SEE ALSO

the cgGetMatrixParameter manpage, the cgGLGetParameter manpage and the cgGetMatrixParameter manpage

NAME

cgGLGetMatrixParameterdrv − gets the value of matrix parameters

DESCRIPTION

Please refer to the cgGLGetMatrixParameter manpage for more information.

SEE ALSO

the cgGetMatrixParameter manpage, the cgGLGetParameter manpage and the cgGetMatrixParameter manpage

NAME

cgGLGetMatrixParameterfc − gets the value of matrix parameters

DESCRIPTION

Please refer to the cgGLGetMatrixParameter manpage for more information.

SEE ALSO

the cgGetMatrixParameter manpage, the cgGLGetParameter manpage and the cgGetMatrixParameter manpage

NAME

cgGLGetMatrixParameterfcv − gets the value of matrix parameters

DESCRIPTION

Please refer to the cgGLGetMatrixParameter manpage for more information.

SEE ALSO

the cgGetMatrixParameter manpage, the cgGLGetParameter manpage and the cgGetMatrixParameter manpage

NAME

cgGLGetMatrixParameterfr − gets the value of matrix parameters

DESCRIPTION

Please refer to the cgGLGetMatrixParameter manpage for more information.

SEE ALSO

the cgGetMatrixParameter manpage, the cgGLGetParameter manpage and the cgGetMatrixParameter manpage

NAME

cgGLGetMatrixParameterfrv − gets the value of matrix parameters

DESCRIPTION

Please refer to the cgGLGetMatrixParameter manpage for more information.

SEE ALSO

the cgGetMatrixParameter manpage, the cgGLGetParameter manpage and the cgGetMatrixParameter manpage

NAME

cgGLGetParameter − sets the value of scalar and vector parameters

SYNOPSIS

```
#include <Cg/cgGL.h>

/* type is either float or double */

cgGLGetParameter{1234}{fd}(CGparameter param, type *v);
```

PARAMETERS

param Specifies the parameter.

v A pointer to the buffer to return the values in.

DESCRIPTION

The cgGLGetParameter functions extract the values set by cgGLSetParameter functions. The functions are available in various combinations.

Each function may return either 1, 2, 3, or 4 values.

There are versions of each function that take either **float** or **double** values signified by the **f** or **d** in the function name.

The cgGLGetParameter functions may only be called with uniform parameters numeric parameters.

RETURN VALUES

The cgGLGetParameter functions do not return any values.

ERRORS

CG_INVALID_PROFILE_ERROR is generated if **param**'s profile is not a supported OpenGL profile.

CG_INVALID_PARAM_HANDLE_ERROR is generated if **param** is not a valid parameter handle.

CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to get for any other reason.

SEE ALSO

the cgGLSetParameter manpage, and the cgGLGetParameterArray manpage

NAME

cgGLGetParameter1d − gets the value of scalar or vector parameters

DESCRIPTION

Please refer to the cgGLGetParameter manpage for more information.

SEE ALSO

the cgGetParameter manpage, the cgGLGetParameter manpage and the cgGetParameter manpage

NAME

cgGLGetParameter1dv − gets the value of scalar or vector parameters

DESCRIPTION

Please refer to the cgGLGetParameter manpage for more information.

SEE ALSO

the cgGetParameter manpage, the cgGLGetParameter manpage and the cgGetParameter manpage

NAME

cgGLGetParameter1f − gets the value of scalar or vector parameters

DESCRIPTION

Please refer to the cgGLGetParameter manpage for more information.

SEE ALSO

the cgGetParameter manpage, the cgGLGetParameter manpage and the cgGetParameter manpage

NAME

cgGLGetParameter1fv − gets the value of scalar or vector parameters

DESCRIPTION

Please refer to the cgGLGetParameter manpage for more information.

SEE ALSO

the cgGetParameter manpage, the cgGLGetParameter manpage and the cgGetParameter manpage

NAME

cgGLGetParameter2d − gets the value of scalar or vector parameters

DESCRIPTION

Please refer to the cgGLGetParameter manpage for more information.

SEE ALSO

the cgGetParameter manpage, the cgGLGetParameter manpage and the cgGetParameter manpage

NAME

cgGLGetParameter2dv − gets the value of scalar or vector parameters

DESCRIPTION

Please refer to the cgGLGetParameter manpage for more information.

SEE ALSO

the cgGetParameter manpage, the cgGLGetParameter manpage and the cgGetParameter manpage

NAME

cgGLGetParameter2f − gets the value of scalar or vector parameters

DESCRIPTION

Please refer to the cgGLGetParameter manpage for more information.

SEE ALSO

the cgGetParameter manpage, the cgGLGetParameter manpage and the cgGetParameter manpage

NAME

cgGLGetParameter2fv − gets the value of scalar or vector parameters

DESCRIPTION

Please refer to the cgGLGetParameter manpage for more information.

SEE ALSO

the cgGetParameter manpage, the cgGLGetParameter manpage and the cgGetParameter manpage

NAME

cgGLGetParameter3d − gets the value of scalar or vector parameters

DESCRIPTION

Please refer to the cgGLGetParameter manpage for more information.

SEE ALSO

the cgGetParameter manpage, the cgGLGetParameter manpage and the cgGetParameter manpage

NAME

cgGLGetParameter3dv − gets the value of scalar or vector parameters

DESCRIPTION

Please refer to the cgGLGetParameter manpage for more information.

SEE ALSO

the cgGetParameter manpage, the cgGLGetParameter manpage and the cgGetParameter manpage

NAME

cgGLGetParameter3f − gets the value of scalar or vector parameters

DESCRIPTION

Please refer to the cgGLGetParameter manpage for more information.

SEE ALSO

the cgGetParameter manpage, the cgGLGetParameter manpage and the cgGetParameter manpage

NAME

cgGLGetParameter3fv − gets the value of scalar or vector parameters

DESCRIPTION

Please refer to the cgGLGetParameter manpage for more information.

SEE ALSO

the cgGetParameter manpage, the cgGLGetParameter manpage and the cgGetParameter manpage

NAME

cgGLGetParameter4d − gets the value of scalar or vector parameters

DESCRIPTION

Please refer to the cgGLGetParameter manpage for more information.

SEE ALSO

the cgGetParameter manpage, the cgGLGetParameter manpage and the cgGetParameter manpage

NAME

cgGLGetParameter4dv − gets the value of scalar or vector parameters

DESCRIPTION

Please refer to the cgGLGetParameter manpage for more information.

SEE ALSO

the cgGetParameter manpage, the cgGLGetParameter manpage and the cgGetParameter manpage

NAME

cgGLGetParameter4f − gets the value of scalar or vector parameters

DESCRIPTION

Please refer to the cgGLGetParameter manpage for more information.

SEE ALSO

the cgGetParameter manpage, the cgGLGetParameter manpage and the cgGetParameter manpage

NAME

cgGLGetParameter4fv − gets the value of scalar or vector parameters

DESCRIPTION

Please refer to the cgGLGetParameter manpage for more information.

SEE ALSO

the cgGetParameter manpage, the cgGLGetParameter manpage and the cgGetParameter manpage

NAME

cgGLGetParameterArray − sets an array scalar or vector parameters

SYNOPSIS

```
#include <Cg/cgGL.h>

/* type is either float or double */

void cgGLGetParameterArray{1234}{fd}(CGparameter param,
                                     long offset,
                                     long nelements,
                                     const type *v);
```

PARAMETERS

param Specifies the array parameter.

offset An offset into the array parameter to start getting from. A value of **0** will start getting from the first element of the array.

nelements

The number of elements to get. A value of **0** will default to the number of elements in the array minus the **offset** value.

v The array retrieve the values into. The size of the array must be **nelements** times the vector size indicated by the number in the function name.

DESCRIPTION

The cgGLGetArrayParameter functions retrieve an array of values from a give scalar or vector array parameter. The functions are available in various combinations.

Each function will retrieve either 1, 2, 3, or 4 values per array element depending on the function that is used.

There are versions of each function that take either **float** or **double** values signified by the **f** or **d** in the function name.

RETURN VALUES

The cgGLGetParameter functions do not return any values.

ERRORS

CG_INVALID_PROFILE_ERROR is generated if **param**'s profile is not a supported OpenGL profile.

CG_ARRAY_PARAM_ERROR is generated if **param** is not an array parameter.

CG_OUT_OF_ARRAY_BOUNDS_ERROR is generated if the **offset** and/or the **nelements** parameter are out of the array bounds.

CG_INVALID_PARAM_HANDLE_ERROR is generated if **param** is not a valid parameter handle.

CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

SEE ALSO

the cgGLGetParameter manpage, the cgGLSetParameter manpage, and the cgGLSetParameterArray manpage

NAME

cgGLGetParameterArray1d − gets the value of scalar or vector array parameters

DESCRIPTION

Please refer to the cgGLGetParameterArray manpage for more information.

SEE ALSO

the cgGLGetParameter manpage, and the cgGetParameterValues manpage

NAME

cgGLGetParameterArray1dv − gets the value of scalar or vector array parameters

DESCRIPTION

Please refer to the cgGLGetParameterArray manpage for more information.

SEE ALSO

the cgGLGetParameter manpage, and the cgGetParameterValues manpage

NAME

cgGLGetParameterArray1f − gets the value of scalar or vector array parameters

DESCRIPTION

Please refer to the cgGLGetParameterArray manpage for more information.

SEE ALSO

the cgGLGetParameter manpage, and the cgGetParameterValues manpage

NAME

cgGLGetParameterArray1fv − gets the value of scalar or vector array parameters

DESCRIPTION

Please refer to the cgGLGetParameterArray manpage for more information.

SEE ALSO

the cgGLGetParameter manpage, and the cgGetParameterValues manpage

NAME

cgGLGetParameterArray2d − gets the value of scalar or vector array parameters

DESCRIPTION

Please refer to the cgGLGetParameterArray manpage for more information.

SEE ALSO

the cgGLGetParameter manpage, and the cgGetParameterValues manpage

NAME

cgGLGetParameterArray2dv − gets the value of scalar or vector array parameters

DESCRIPTION

Please refer to the cgGLGetParameterArray manpage for more information.

SEE ALSO

the cgGLGetParameter manpage, and the cgGetParameterValues manpage

NAME

cgGLGetParameterArray2f − gets the value of scalar or vector array parameters

DESCRIPTION

Please refer to the cgGLGetParameterArray manpage for more information.

SEE ALSO

the cgGLGetParameter manpage, and the cgGetParameterValues manpage

NAME

cgGLGetParameterArray2fv − gets the value of scalar or vector array parameters

DESCRIPTION

Please refer to the cgGLGetParameterArray manpage for more information.

SEE ALSO

the cgGLGetParameter manpage, and the cgGetParameterValues manpage

NAME

cgGLGetParameterArray3d − gets the value of scalar or vector array parameters

DESCRIPTION

Please refer to the cgGLGetParameterArray manpage for more information.

SEE ALSO

the cgGLGetParameter manpage, and the cgGetParameterValues manpage

NAME

cgGLGetParameterArray3dv − gets the value of scalar or vector array parameters

DESCRIPTION

Please refer to the cgGLGetParameterArray manpage for more information.

SEE ALSO

the cgGLGetParameter manpage, and the cgGetParameterValues manpage

NAME

cgGLGetParameterArray3f − gets the value of scalar or vector array parameters

DESCRIPTION

Please refer to the cgGLGetParameterArray manpage for more information.

SEE ALSO

the cgGLGetParameter manpage, and the cgGetParameterValues manpage

NAME

cgGLGetParameterArray3fv − gets the value of scalar or vector array parameters

DESCRIPTION

Please refer to the cgGLGetParameterArray manpage for more information.

SEE ALSO

the cgGLGetParameter manpage, and the cgGetParameterValues manpage

NAME

cgGLGetParameterArray4d − gets the value of scalar or vector array parameters

DESCRIPTION

Please refer to the cgGLGetParameterArray manpage for more information.

SEE ALSO

the cgGLGetParameter manpage, and the cgGetParameterValues manpage

NAME

cgGLGetParameterArray4dv − gets the value of scalar or vector array parameters

DESCRIPTION

Please refer to the cgGLGetParameterArray manpage for more information.

SEE ALSO

the cgGLGetParameter manpage, and the cgGetParameterValues manpage

NAME

cgGLGetParameterArray4f − gets the value of scalar or vector array parameters

DESCRIPTION

Please refer to the cgGLGetParameterArray manpage for more information.

SEE ALSO

the cgGLGetParameter manpage, and the cgGetParameterValues manpage

NAME

cgGLGetParameterArray4fv − gets the value of scalar or vector array parameters

DESCRIPTION

Please refer to the cgGLGetParameterArray manpage for more information.

SEE ALSO

the cgGLGetParameter manpage, and the cgGetParameterValues manpage

NAME

cgGLGetProgramID − Returns the OpenGL program ID associated with the CGprogram object

SYNOPSIS

```
#include <Cg/cgGL.h>

GLuint cgGLGetProgramID(CGprogram prog);
```

PARAMETERS

prog Specifies the program.

DESCRIPTION

cgGLGetProgramID returns the identifier to the program GL object associated with the given Cg program. cgGLGetProgramID should not be called before cgGLLoadProgram is called. =head1 RETURN VALUES

cgGLGetProgramID returns a **GLuint** associate with the GL program object for profiles that use program object. Some profiles (e.g. fp20) do not have GL programs and will always return **0**.

ERRORS

CG_INVALID_PROFILE_ERROR is generated if **prog**'s profile is not a supported OpenGL profile.

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if **prog** is not a valid program.

SEE ALSO

the cgGLLoadProgram manpage, and the cgGLBindProgram manpage

NAME

cgGLGetTextureEnum − returns the GL enumerant for the texture unit associated with a parameter

SYNOPSIS

```
#include <Cg/cgGL.h>

GLenum cgGLGetTextureEnum(CGparameter param);
```

PARAMETERS

param Specifies the texture parameter.

DESCRIPTION

cgGLGetTextureEnum returns the OpenGL enumerant for the texture unit assigned to **param**. The enumerant has the form **GL_TEXTURE#_ARB** where **#** is the texture unit number.

RETURN VALUES

cgGLGetTextureEnum returns a **GLenum** of the form **GL_TEXTURE#_ARB**. If **param** is not a texture parameter **GL_INVALID_OPERATION** is returned.

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if **param** is not a valid parameter handle.

CG_INVALID_PARAMETER_ERROR is generated **param** is not a texture parameter or if the parameter fails to set for any other reason.

SEE ALSO

the cgGLSetTextureParameter manpage

NAME

cgGLGetTextureParameter − retrieves the value of texture parameters

SYNOPSIS

```
#include <Cg/cgGL.h>

GLuint cgGLGetTextureParameter(CGparameter param);
```

PARAMETERS

param Specifies the texture parameter.

DESCRIPTION

cgGLGetTextureParameter retrieves the value of a texture parameter.

RETURN VALUES

cgGLGetTextureParameter returns the name of the OpenGL object that the texture was set to. If the parameter has not been set, **0** will be returned.

ERRORS

CG_INVALID_PROFILE_ERROR is generated if **param**'s profile is not a supported OpenGL profile.

CG_INVALID_PARAM_HANDLE_ERROR is generated if **param** is not a valid parameter handle.

CG_INVALID_PARAMETER_ERROR is generated **param** is not a texture parameter or if the parameter fails to get for any other reason.

SEE ALSO

the cgGLSetTextureParameter manpage and the cgGLGetParameter manpage,

NAME

cgGLIsProfileSupported − determines if a given profile is supported by cgGL

SYNOPSIS

```
#include <Cg/cgGL.h>

CGbool cgGLIsProfileSupported(CGprofile profile);
```

PARAMETERS

profile The profile enumerant.

DESCRIPTION

cgIsProfile returns **CG_TRUE** if the profile indicated by the **profile** parameter is supported by the cgGL library. Otherwise it returns **CG_FALSE**.

A given profile may not be supported if OpenGL extensions it requires are not available.

ERRORS

SEE ALSO

the cgGLEnableProfile manpage, and the cgGLDisableProfile manpage

NAME

cgGLIsProgramLoaded − determines if a program is loaded

SYNOPSIS

```
#include <Cg/cgGL.h>

CGbool cgGLIsProgramLoaded(CGprogram program);
```

PARAMETERS

program The program handle.

DESCRIPTION

cgGLIsProgramLoaded returns **CG_TRUE** if **program** is loaded with cgGLLoadProgram and **CG_FALSE** otherwise.

ERRORS

SEE ALSO

the cgGLLoadProgram manpage

NAME

cgGLLoadProgram − prepares a program for binding

SYNOPSIS

```
#include <Cg/cgGL.h>

void cgGLLoadProgram(CGprogram prog);
```

PARAMETERS

prog Specifies the program.

DESCRIPTION

cgGLLoadProgram prepares a program for binding. All programs must be loaded before they can be bound to the current state. See cgGLBindProgram for more information about binding programs.

RETURN VALUES

cgGLLoadProgram does not return any values.

ERRORS

CG_INVALID_PROFILE_ERROR is generated if **prog**'s profile is not a supported OpenGL profile.

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if **prog** is not a valid program.

CG_PROGRAM_LOAD_ERROR is generated if the program fails to load for any reason.

SEE ALSO

the cgGLBindProgram manpage

NAME

cgGLRegisterStates − registers graphics pass states for CgFX files for on OpenGL

SYNOPSIS

```
#include <Cg/cgGL.h>

void cgGLRegisterStates(CGcontext ctx);
```

PARAMETERS

ctx The context in which to register the states.

DESCRIPTION

cgGLRegisterStates registers a set of states for passes in techniques in CgFX effect files. These states correspond to the set of OpenGL state that is relevant and/or useful to be setting in passes in effect files. See the Cg User's Guide for complete documentation of the states that are made available after calling **cgGLRegisterStates**.

ERRORS

CG_INVALID_CONTEXT_ERROR is generated if **context** is invalid.

SEE ALSO

the cgAddState manpage, the cgSetPassState manpage, the cgResetPassState manpage, the cgValidatePassState manpage.

NAME

cgGLSetManageTextureParameters − sets the manage texture parameters flag for a context

SYNOPSIS

```
#include <Cg/cgGL.h>

void cgGLSetManageTextureParameters(CGcontext ctx, CGbool flag);
```

PARAMETERS

ctx Specifies the context.

flag The flag to set the manage textures flag to.

DESCRIPTION

By default, cgGL does not manage any texture state in OpenGL. It is up to the user to enable and disable textures with cgGLEnableTexture and cgGLDisableTexture respectively. This behavior is the default in order to avoid conflicts with texture state on geometry that's rendered with the fixed function pipeline or without cgGL.

If automatic texture management is desired, cgGLSetManageTextureParameters may be called with **flag** set to **CG_TRUE** before cgGLBindProgram is called. Whenever cgGLBindProgram is called, the cgGL Runtime will make all the appropriate texture parameter calls on the application's behalf. To reset the texture state cgGLUnbindProgram may be used.

Calling cgGLSetManageTextureParameters with **flag** set to **CG_FALSE** will disable automatic texture management.

NOTE: When cgGLSetManageTextureParameters is set to **CG_TRUE**, applications should not make texture state change calls to OpenGL (such as glBindTexture, glActiveTexture, etc.) after calling cgGLBindProgram, unless the application is trying to override some parts of CgGL's texture management.

ERRORS

SEE ALSO

the cgGLGetManageTextureParameters manpage, the cgGLBindProgram manpage, and the cgGLUnbindProgram manpage

NAME

cgGLSetMatrixParameter − sets the value of matrix parameters

SYNOPSIS

```
#include <Cg/cgGL.h>

/* type is either float or double */
void cgGLSetMatrixParameter{fd}{rc}(CGparameter param, const type *matrix);
```

PARAMETERS

param Specifies the parameter that will be set.

matrix An array of values to set the matrix parameter to. The array must be the number of rows times the number of columns in size.

DESCRIPTION

The cgGLSetMatrixParameter functions set the value of a given matrix parameter. The functions are available in various combinations.

There are versions of each function that take either **float** or **double** values signified by the **f** or **d** in the function name.

There are versions of each function that assume the array of values are layed out in either row or column order signified by the **r** or **c** in the function name respectively.

The cgGLSetMatrixParameter functions may only be called with uniform parameters.

RETURN VALUES

The cgGLSetMatrixParameter functions do not return any values.

ERRORS

CG_INVALID_PROFILE_ERROR is generated if **param**'s profile is not a supported OpenGL profile.

CG_NOT_MATRIX_PARAM_ERROR is generated if **param** is not a matrix parameter.

CG_INVALID_PARAM_HANDLE_ERROR is generated if **param** is not a valid parameter handle.

CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

SEE ALSO

the cgGLGetMatrixParameter manpage, the cgGLSetMatrixParameterArray manpage and the cgGLSetParameter manpage

NAME

cgGLSetParameterArray − sets an array matrix parameters

SYNOPSIS

```
#include <Cg/cgGL.h>

/* type is either float or double */

void cgGLSetMatrixParameterArray{fd}{rc}(CGparameter param,
                                         long offset,
                                         long nelements,
                                         const type *v);
```

PARAMETERS

param Specifies the array parameter that will be set.

offset An offset into the array parameter to start setting from. A value of **0** will start setting from the first element of the array.

nelements

The number of elements to set. A value of **0** will default to the number of elements in the array minus the **offset** value.

v The array of values to set the parameter to. This must be a contiguous set of values that total **nelements** times the number of elements in the matrix.

DESCRIPTION

The cgGLSetMatrixParameterArray functions set the value of a given scalar or vector array parameter. The functions are available in various combinations.

There are versions of each function that assume the array of values are layed out in either row or column order signified by the **r** or **c** in the function name respectively.

There are versions of each function that take either **float** or **double** values signified by the **f** or **d** in the function name.

RETURN VALUES

The cgGLSetMatrixParameterArray functions do not return any values.

ERRORS

CG_INVALID_PROFILE_ERROR is generated if **param**'s profile is not a supported OpenGL profile.

CG_ARRAY_PARAM_ERROR is generated if **param** is not an array parameter.

CG_NOT_MATRIX_PARAM_ERROR is generated if the elements of the array indicated by **param** are not matrix parameters.

CG_OUT_OF_ARRAY_BOUNDS_ERROR is generated if the **offset** and/or the **nelements** parameter are out of the array bounds.

CG_INVALID_PARAM_HANDLE_ERROR is generated if **param** is not a valid parameter handle.

CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

SEE ALSO

the cgGLSetMatrixParameter manpage, and the cgGLGetMatrixParameterArray manpage

NAME

cgGLSetMatrixParameterArraydc − sets the value of matrix parameter arrays

DESCRIPTION

Please refer to the cgGLSetMatrixParameterArray manpage for more information.

SEE ALSO

the cgGetMatrixParameterArray manpage, the cgGLSetParameter manpage and the cgSetMatrixParameter manpage

NAME

cgGLSetMatrixParameterArraydcv − sets the value of matrix parameter arrays

DESCRIPTION

Please refer to the cgGLSetMatrixParameterArray manpage for more information.

SEE ALSO

the cgGetMatrixParameterArray manpage, the cgGLSetParameter manpage and the cgSetMatrixParameter manpage

NAME

cgGLSetMatrixParameterArraydr − sets the value of matrix parameter arrays

DESCRIPTION

Please refer to the cgGLSetMatrixParameterArray manpage for more information.

SEE ALSO

the cgGetMatrixParameterArray manpage, the cgGLSetParameter manpage and the cgSetMatrixParameter manpage

NAME

cgGLSetMatrixParameterArraydrv − sets the value of matrix parameter arrays

DESCRIPTION

Please refer to the cgGLSetMatrixParameterArray manpage for more information.

SEE ALSO

the cgGetMatrixParameterArray manpage, the cgGLSetParameter manpage and the cgSetMatrixParameter manpage

NAME

cgGLSetMatrixParameterArrayfc − sets the value of matrix parameter arrays

DESCRIPTION

Please refer to the cgGLSetMatrixParameterArray manpage for more information.

SEE ALSO

the cgGetMatrixParameterArray manpage, the cgGLSetParameter manpage and the cgSetMatrixParameter manpage

NAME

cgGLSetMatrixParameterArrayfcv − sets the value of matrix parameter arrays

DESCRIPTION

Please refer to the cgGLSetMatrixParameterArray manpage for more information.

SEE ALSO

the cgGetMatrixParameterArray manpage, the cgGLSetParameter manpage and the cgSetMatrixParameter manpage

NAME

cgGLSetMatrixParameterArrayfr − sets the value of matrix parameter arrays

DESCRIPTION

Please refer to the cgGLSetMatrixParameterArray manpage for more information.

SEE ALSO

the cgGetMatrixParameterArray manpage, the cgGLSetParameter manpage and the cgSetMatrixParameter manpage

NAME

cgGLSetMatrixParameterArrayfrv − sets the value of matrix parameter arrays

DESCRIPTION

Please refer to the cgGLSetMatrixParameterArray manpage for more information.

SEE ALSO

the cgGetMatrixParameterArray manpage, the cgGLSetParameter manpage and the cgSetMatrixParameter manpage

NAME

cgGLSetMatrixParameterdc − sets the value of matrix parameters

DESCRIPTION

Please refer to the cgGLSetMatrixParameter manpage for more information.

SEE ALSO

the cgGetMatrixParameter manpage, the cgGLSetParameter manpage and the cgSetMatrixParameter manpage

NAME

cgGLSetMatrixParameterdcv − sets the value of matrix parameters

DESCRIPTION

Please refer to the cgGLSetMatrixParameter manpage for more information.

SEE ALSO

the cgGetMatrixParameter manpage, the cgGLSetParameter manpage and the cgSetMatrixParameter manpage

NAME

cgGLSetMatrixParameterdr − sets the value of matrix parameters

DESCRIPTION

Please refer to the cgGLSetMatrixParameter manpage for more information.

SEE ALSO

the cgGetMatrixParameter manpage, the cgGLSetParameter manpage and the cgSetMatrixParameter manpage

NAME

cgGLSetMatrixParameterdrv − sets the value of matrix parameters

DESCRIPTION

Please refer to the cgGLSetMatrixParameter manpage for more information.

SEE ALSO

the cgGetMatrixParameter manpage, the cgGLSetParameter manpage and the cgSetMatrixParameter manpage

NAME

cgGLSetMatrixParameterfc − sets the value of matrix parameters

DESCRIPTION

Please refer to the cgGLSetMatrixParameter manpage for more information.

SEE ALSO

the cgGetMatrixParameter manpage, the cgGLSetParameter manpage and the cgSetMatrixParameter manpage

NAME

cgGLSetMatrixParameterfcv − sets the value of matrix parameters

DESCRIPTION

Please refer to the cgGLSetMatrixParameter manpage for more information.

SEE ALSO

the cgGetMatrixParameter manpage, the cgGLSetParameter manpage and the cgSetMatrixParameter manpage

NAME

cgGLSetMatrixParameterfr − sets the value of matrix parameters

DESCRIPTION

Please refer to the cgGLSetMatrixParameter manpage for more information.

SEE ALSO

the cgGetMatrixParameter manpage, the cgGLSetParameter manpage and the cgSetMatrixParameter manpage

NAME

cgGLSetMatrixParameterfrv − sets the value of matrix parameters

DESCRIPTION

Please refer to the cgGLSetMatrixParameter manpage for more information.

SEE ALSO

the cgGetMatrixParameter manpage, the cgGLSetParameter manpage and the cgSetMatrixParameter manpage

NAME

cgGLSetOptimalOptions − sets implicit optimization compiler options for a profile

SYNOPSIS

```
#include <Cg/cgGL.h>

void cgGLSetOptimalOptions(CGprofile profile);
```

PARAMETERS

profile The profile enumerant.

DESCRIPTION

cgGLSetOptimalOptions sets implicit compiler arguments that are appended to the argument list passed to cgCreateProgram. The arguments are chosen based on the the available compiler arguments, GPU, and driver.

The arguments will be appended to the argument list every time cgCreateProgram is called until the last **CGcontext** is destroyed.

RETURN VALUES

cgGLSetOptimalOptions does not return any values.

ERRORS

CG_INVALID_PROFILE_ERROR is generated if **profile** is invalid.

SEE ALSO

the cgGLCreateProgram manpage

NAME

cgGLSetParameter − sets the value of scalar and vector parameters

SYNOPSIS

```
#include <Cg/cgGL.h>

/* type is either float or double */

void cgGLSetParameter1{fd}(CGparameter param,
                           type x);

void cgGLSetParameter2{fd}(CGparameter param,
                           type x,
                           type y);

void cgGLSetParameter3{fd}(CGparameter param,
                           type x,
                           type y,
                           type z);

void cgGLSetParameter4{fd}(CGparameter param,
                           type x,
                           type y,
                           type z,
                           type w);

void cgGLSetParameter{1234}{fd}v(CGparameter param,
                                 const type *v);
```

PARAMETERS

param Specifies the parameter that will be set.

x, y, z, and w
: The values to set the parameter to.

v The values to set the parameter to for the array versions of the set functions.

DESCRIPTION

The cgGLSetParameter functions set the value of a given scalar or vector parameter. The functions are available in various combinations.

Each function takes either 1, 2, 3, or 4 values depending on the function that is used. If more values are passed in than the parameter requires, the extra values will be ignored. If less values are passed in than the parameter requires, the last value will be smeared.

There are versions of each function that take either **float** or **double** values signified by the **f** or **d** in the function name.

The functions with the **v** at the end of their names take an array of values instead of explicit parameters.

The cgGLSetParameter functions may be called with either uniform or varying parameters. When called with a varying parameter, the appropriate immediate mode OpenGL entry point will be called. However, cgGLGetParameter will only work with uniform parameters.

RETURN VALUES

The cgGLSetParameter functions do not return any values.

ERRORS

CG_INVALID_PROFILE_ERROR is generated if **param**'s profile is not a supported OpenGL profile.

CG_INVALID_PARAM_HANDLE_ERROR is generated if **param** is not a valid parameter handle.

CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

SEE ALSO

the cgGLGetParameter manpage, the cgGLSetParameterArray manpage, the cgGLSetMatrixParameter manpage, the cgGLSetMatrixParameterArray manpage, the cgGLSetTextureParameter manpage, the cgGLSetTextureParameterArray manpage, and the cgGLBindProgram manpage

NAME

cgGLSetParameter1d − sets the value of scalar or vector parameters

DESCRIPTION

Please refer to the cgGLSetParameter manpage for more information.

SEE ALSO

the cgGLSetParameter manpage, and the cgSetParameter manpage

NAME

cgGLSetParameter1dv − sets the value of scalar or vector parameters

DESCRIPTION

Please refer to the cgGLSetParameter manpage for more information.

SEE ALSO

the cgGLSetParameter manpage, and the cgSetParameter manpage

NAME

cgGLSetParameter1f − sets the value of scalar or vector parameters

DESCRIPTION

Please refer to the cgGLSetParameter manpage for more information.

SEE ALSO

the cgGLSetParameter manpage, and the cgSetParameter manpage

NAME

cgGLSetParameter1fv − sets the value of scalar or vector parameters

DESCRIPTION

Please refer to the cgGLSetParameter manpage for more information.

SEE ALSO

the cgGLSetParameter manpage, and the cgSetParameter manpage

NAME

cgGLSetParameter2d − sets the value of scalar or vector parameters

DESCRIPTION

Please refer to the cgGLSetParameter manpage for more information.

SEE ALSO

the cgGLSetParameter manpage, and the cgSetParameter manpage

NAME

cgGLSetParameter2dv − sets the value of scalar or vector parameters

DESCRIPTION

Please refer to the cgGLSetParameter manpage for more information.

SEE ALSO

the cgGLSetParameter manpage, and the cgSetParameter manpage

NAME

cgGLSetParameter2f − sets the value of scalar or vector parameters

DESCRIPTION

Please refer to the cgGLSetParameter manpage for more information.

SEE ALSO

the cgGLSetParameter manpage, and the cgSetParameter manpage

NAME

cgGLSetParameter2fv − sets the value of scalar or vector parameters

DESCRIPTION

Please refer to the cgGLSetParameter manpage for more information.

SEE ALSO

the cgGLSetParameter manpage, and the cgSetParameter manpage

NAME

cgGLSetParameter3d − sets the value of scalar or vector parameters

DESCRIPTION

Please refer to the cgGLSetParameter manpage for more information.

SEE ALSO

the cgGLSetParameter manpage, and the cgSetParameter manpage

NAME

cgGLSetParameter3dv − sets the value of scalar or vector parameters

DESCRIPTION

Please refer to the cgGLSetParameter manpage for more information.

SEE ALSO

the cgGLSetParameter manpage, and the cgSetParameter manpage

NAME

cgGLSetParameter3f − sets the value of scalar or vector parameters

DESCRIPTION

Please refer to the cgGLSetParameter manpage for more information.

SEE ALSO

the cgGLSetParameter manpage, and the cgSetParameter manpage

NAME

cgGLSetParameter3fv − sets the value of scalar or vector parameters

DESCRIPTION

Please refer to the cgGLSetParameter manpage for more information.

SEE ALSO

the cgGLSetParameter manpage, and the cgSetParameter manpage

NAME

cgGLSetParameter4d − sets the value of scalar or vector parameters

DESCRIPTION

Please refer to the cgGLSetParameter manpage for more information.

SEE ALSO

the cgGLSetParameter manpage, and the cgSetParameter manpage

NAME

cgGLSetParameter4dv − sets the value of scalar or vector parameters

DESCRIPTION

Please refer to the cgGLSetParameter manpage for more information.

SEE ALSO

the cgGLSetParameter manpage, and the cgSetParameter manpage

NAME

cgGLSetParameter4f − sets the value of scalar or vector parameters

DESCRIPTION

Please refer to the cgGLSetParameter manpage for more information.

SEE ALSO

the cgGLSetParameter manpage, and the cgSetParameter manpage

NAME

cgGLSetParameter4fv − sets the value of scalar or vector parameters

DESCRIPTION

Please refer to the cgGLSetParameter manpage for more information.

SEE ALSO

the cgGLSetParameter manpage, and the cgSetParameter manpage

NAME

cgGLSetParameterArray − sets an array scalar or vector parameters

SYNOPSIS

```
#include <Cg/cgGL.h>

/* type is either float or double */

void cgGLSetParameterArray{1234}{fd}(CGparameter param,
                                     long offset,
                                     long nelements,
                                     const type *v);
```

PARAMETERS

param Specifies the array parameter that will be set.

offset An offset into the array parameter to start setting from. A value of **0** will start setting from the first element of the array.

nelements

The number of elements to set. A value of **0** will default to the number of elements in the array minus the **offset** value.

v The array of values to set the parameter to. This must be a contiguous set of values that total **nelements** times the vector size indicated by the number in the function name.

DESCRIPTION

The cgGLSetParameterArray functions set the value of a given scalar or vector array parameter. The functions are available in various combinations.

Each function will set either 1, 2, 3, or 4 values per array element depending on the function that is used.

There are versions of each function that take either **float** or **double** values signified by the **f** or **d** in the function name.

RETURN VALUES

The cgGLSetParameterArray functions do not return any values.

ERRORS

CG_INVALID_PROFILE_ERROR is generated if **param**'s profile is not a supported OpenGL profile.

CG_ARRAY_PARAM_ERROR is generated if **param** is not an array parameter.

CG_OUT_OF_ARRAY_BOUNDS_ERROR is generated if the **offset** and/or the **nelements** parameter are out of the array bounds.

CG_INVALID_PARAM_HANDLE_ERROR is generated if **param** is not a valid parameter handle.

CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

SEE ALSO

the cgGLSetParameter manpage, and the cgGLGetParameterArray manpage

NAME

cgGLSetParameterArray1d − sets the value of scalar or vector array parameters

DESCRIPTION

Please refer to the cgGLSetParameterArray manpage for more information.

SEE ALSO

the cgGLSetParameter manpage, and the cgSetParameter manpage

NAME

cgGLSetParameterArray1dv − sets the value of scalar or vector array parameters

DESCRIPTION

Please refer to the cgGLSetParameterArray manpage for more information.

SEE ALSO

the cgGLSetParameter manpage, and the cgSetParameter manpage

NAME

cgGLSetParameterArray1f − sets the value of scalar or vector array parameters

DESCRIPTION

Please refer to the cgGLSetParameterArray manpage for more information.

SEE ALSO

the cgGLSetParameter manpage, and the cgSetParameter manpage

NAME

cgGLSetParameterArray1fv − sets the value of scalar or vector array parameters

DESCRIPTION

Please refer to the cgGLSetParameterArray manpage for more information.

SEE ALSO

the cgGLSetParameter manpage, and the cgSetParameter manpage

NAME

cgGLSetParameterArray2d − sets the value of scalar or vector array parameters

DESCRIPTION

Please refer to the cgGLSetParameterArray manpage for more information.

SEE ALSO

the cgGLSetParameter manpage, and the cgSetParameter manpage

NAME

cgGLSetParameterArray2dv − sets the value of scalar or vector array parameters

DESCRIPTION

Please refer to the cgGLSetParameterArray manpage for more information.

SEE ALSO

the cgGLSetParameter manpage, and the cgSetParameter manpage

NAME

cgGLSetParameterArray2f − sets the value of scalar or vector array parameters

DESCRIPTION

Please refer to the cgGLSetParameterArray manpage for more information.

SEE ALSO

the cgGLSetParameter manpage, and the cgSetParameter manpage

NAME

cgGLSetParameterArray2fv − sets the value of scalar or vector array parameters

DESCRIPTION

Please refer to the cgGLSetParameterArray manpage for more information.

SEE ALSO

the cgGLSetParameter manpage, and the cgSetParameter manpage

NAME

cgGLSetParameterArray3d − sets the value of scalar or vector array parameters

DESCRIPTION

Please refer to the cgGLSetParameterArray manpage for more information.

SEE ALSO

the cgGLSetParameter manpage, and the cgSetParameter manpage

NAME

cgGLSetParameterArray3dv − sets the value of scalar or vector array parameters

DESCRIPTION

Please refer to the cgGLSetParameterArray manpage for more information.

SEE ALSO

the cgGLSetParameter manpage, and the cgSetParameter manpage

NAME

cgGLSetParameterArray3f − sets the value of scalar or vector array parameters

DESCRIPTION

Please refer to the cgGLSetParameterArray manpage for more information.

SEE ALSO

the cgGLSetParameter manpage, and the cgSetParameter manpage

NAME

cgGLSetParameterArray3fv − sets the value of scalar or vector array parameters

DESCRIPTION

Please refer to the cgGLSetParameterArray manpage for more information.

SEE ALSO

the cgGLSetParameter manpage, and the cgSetParameter manpage

NAME

cgGLSetParameterArray4d − sets the value of scalar or vector array parameters

DESCRIPTION

Please refer to the cgGLSetParameterArray manpage for more information.

SEE ALSO

the cgGLSetParameter manpage, and the cgSetParameter manpage

NAME

cgGLSetParameterArray4dv − sets the value of scalar or vector array parameters

DESCRIPTION

Please refer to the cgGLSetParameterArray manpage for more information.

SEE ALSO

the cgGLSetParameter manpage, and the cgSetParameter manpage

NAME

cgGLSetParameterArray4f − sets the value of scalar or vector array parameters

DESCRIPTION

Please refer to the cgGLSetParameterArray manpage for more information.

SEE ALSO

the cgGLSetParameter manpage, and the cgSetParameter manpage

NAME

cgGLSetParameterArray4fv − sets the value of scalar or vector array parameters

DESCRIPTION

Please refer to the cgGLSetParameterArray manpage for more information.

SEE ALSO

the cgGLSetParameter manpage, and the cgSetParameter manpage

NAME

cgGLSetParameterPointer − sets a varying parameter with an attribute array

SYNOPSIS

```
#include <Cg/cgGL.h>

void cgGLSetParameterPointer(CGparameter param,
                             GLint fsize,
                             GLenum type,
                             GLsizei stride,
                             GLvoid *pointer);
```

PARAMETERS

param Specifies the parameter that will be set.

fsize The number of coordinates per vertex.

type The data type of each coordinate. Possible values are **GL_UNSIGNED_BYTE**, **GL_SHORT**, **GL_INT**, **GL_FLOAT**, and **GL_DOUBLE**.

stride Specifies the byte offset between consecutive vertices. If **stride** is **0** the array is assumed to be tightly packed.

pointer Specified the pointer to the first coordinate in the vertex array.

DESCRIPTION

cgGLSetParameterPointer sets a varying parameter to a given vertex array in the typical OpenGL style. See the OpenGL documentation on the various vertex array functions (e.g. glVertexPointer, glNormalPointer, etc...) for more information.

RETURN VALUES

The cgGLSetParameterPointer functions does not return any values.

ERRORS

CG_INVALID_PROFILE_ERROR is generated if **param**'s profile is not a supported OpenGL profile.

CG_INVALID_PARAM_HANDLE_ERROR is generated if **param** is not a valid parameter handle.

CG_UNSUPPORTED_GL_EXTENSION_ERROR is generated if **param** required a GL extension that's not available.

CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

SEE ALSO

the cgGLSetParameter manpage

NAME

cgGLSetStateMatrixParameter − sets the value of a matrix parameter to a matrix in the OpenGL state

SYNOPSIS

```
#include <Cg/cgGL.h>

void cgGLSetStateMatrixParameter(CGparameter param,
                                 CGGLenum matrix,
                                 CGGLenum transform);
```

PARAMETERS

param Specifies the parameter that will be set.

matrix An enumerant indicating which matrix should be retrieved from the OpenGL state. It may be any one of :

CG_GL_MODELVIEW_MATRIX
The current modelview matrix.

CG_GL_PROJECTION_MATRIX
The current projection matrix.

CG_GL_TEXTURE_MATRIX
The current texture matrix.

CG_GL_MODELVIEW_PROJECTION_MATRIX
The concatenated modelview and projection matrices.

transform
An optional transformation that may be applied to the OpenGL state matrix before it is set. The enumerants may be any one of :

CG_GL_MATRIX_IDENTITY
Doesn't apply any transform leaving the matrix as is.

CG_GL_MATRIX_TRANSPOSE
Transposes the matrix.

CG_GL_MATRIX_INVERSE
Inverts the matrix.

CG_GL_MATRIX_INVERSE_TRANSPOSE
Transforms and inverts the matrix.

DESCRIPTION

cgGLSetStateMatrixParameter retrieves matrices from the OpenGL state and sets a matrix parameter to the matrix. The matrix may optionally be transformed before it is set via the **transform** parameter.

cgGLSetStateMatrixParameter may only be called with any uniform matrix parameter. If the size of the matrix is less than 4x4, the upper left corner of the matrix that fits into the given matrix parameter will be returned.

RETURN VALUES

cgGLSetStateMatrixParameter does not return any values.

ERRORS

CG_INVALID_PROFILE_ERROR is generated if **param**'s profile is not a supported OpenGL profile.

CG_NOT_MATRIX_PARAM_ERROR is generated if **param** is not a matrix parameter.

CG_INVALID_ENUMERANT_ERROR is generated if any of the enumerant parameters to cgGLSetStateMatrixParameter are invalid.

CG_INVALID_PARAM_HANDLE_ERROR is generated if **param** is not a valid parameter handle.

CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

SEE ALSO

the cgGLSetMatrixParameter manpage, and the cgGLGetMatrixParameter manpage

NAME

cgGLSetTextureParameter − sets the value of texture parameters

SYNOPSIS

```
#include <Cg/cgGL.h>

void cgGLSetTextureParameter(CGparameter param, GLuint texobj);
```

PARAMETERS

param Specifies the parameter that will be set.

texobj An OpenGL texture object name. This is the value the parameter will be set to.

DESCRIPTION

cgGLSetTextureParameter sets the value of a given texture parameter to a OpenGL texture object.

Note that in order to use the texture, either the cgGLEnableTextureParameter manpage must be called after the cgGLSetTextureParameter manpage and before the geometry is drawn, or the cgGLSetManageTextureParameters manpage must be called with a value of **CG_TRUE**.

RETURN VALUES

cgGLSetTextureParameter does not return any values.

ERRORS

CG_INVALID_PROFILE_ERROR is generated if **param**'s profile is not a supported OpenGL profile.

CG_INVALID_PARAM_HANDLE_ERROR is generated if **param** is not a valid parameter handle.

CG_INVALID_PARAMETER_ERROR is generated **param** is not a texture parameter or if the parameter fails to set for any other reason.

SEE ALSO

the cgGLGetTextureParameter manpage, and the cgGLSetParameter manpage

NAME

cgGLSetupSampler − initializes a sampler's state and texture object handle

SYNOPSIS

```
#include <Cg/cgGL.h>

void cgGLSetupSampler(CGparameter param, GLuint texobj);
```

PARAMETERS

param Specifies the sampler that will be set.

texobj An OpenGL texture object name. This is the value the parameter will be set to.

DESCRIPTION

cgGLSetupSampler initializes a sampler; like the cgGLSetTextureParameter manpage, it informs the OpenGL Cg runtime which OpenGL texture object to associate with the sampler. Furthermore, if the sampler was defined in the source file with a **sampler_state** block that specifies sampler state, this sampler state is initialized for the given texture object.

Note that in order to use the texture, either the cgGLEnableTextureParameter manpage must be called after the cgGLSetTextureParameter manpage and before the geometry is drawn, or the cgGLSetManageTextureParameters manpage must be called with a value of **CG_TRUE**.

RETURN VALUES

cgGLSetupSampler does not return any values.

ERRORS

CG_INVALID_PROFILE_ERROR is generated if **param**'s profile is not a supported OpenGL profile.

CG_INVALID_PARAM_HANDLE_ERROR is generated if **param** is not a valid parameter handle.

CG_INVALID_PARAMETER_ERROR is generated **param** is not a texture parameter or if the parameter fails to set for any other reason.

SEE ALSO

the cgGLSetTextureParameter manpage, the cgGLGetTextureParameter manpage, and the cgGLSetManageTextureParameters manpage

NAME

cgGLUnbindProgram − unbinds the program bound in a profile

SYNOPSIS

```
#include <Cg/cgGL.h>

void cgGLUnbindProgram(CGprofile profile);
```

PARAMETERS

profile Specifies the profile.

DESCRIPTION

cgGLUnbindProgram unbinds the program bound in the profile specified by **profile**. It will also reset the texture state back to the state it was in at the point **cgGLBindProgram**|**cgGLBindProgram** was first called with a program in the given profile.

ERRORS

CG_INVALID_PROFILE_ERROR is generated if **prog**'s profile is not a supported OpenGL profile.

SEE ALSO

the cgSetManageTextureParameters manpage, the cgBindProgram manpage, and the cgUnbindProgram manpage

NAME

arbfp1 − OpenGL fragment profile for multi-vendor ARB_fragment_program extension

SYNPOSIS

```
arbfp1
```

DESCRIPTION

This OpenGL profile corresponds to the per-fragment functionality introduced by GeForce FX and other DirectX 9 GPUs. This profile is supported by any OpenGL implementation that conformantly implements ARB_fragment_program.

The compiler output for this profile conforms to the assembly format defined by **ARB_fragment_program**.

Data-dependent loops are not allowed; all loops must be unrollable.

Conditional expressions are supported without branching so both conditions must be evaluated.

Relative indexing of uniform arrays is not supported; use texture accesses instead.

3D API DEPENDENCIES

Requires OpenGL support for the multi-vendor **ARB_fragment_program** extension. This extension is supported by GeForce FX and later GPUS. ATI GPUs also support this extension.

PROFILE OPTIONS

NumTemps=n
: Number of temporaries to use (from 12 to 32).

MaxInstructionSlots=n
: Maximum allowable (static) instructions. Not an issue for NVIDIA GPUs.

NoDependentReadLimit=b
: Boolean for whether a read limit exists.

NumTexInstructions=n
: Maximum number of texture instructions to generate. Not an issue for NVIDIA GPUs, but important for ATI GPUs (set it to 32).

NumMathInstructions=n
: Maximum number of math instructions to generate. Not an issue for NVIDIA GPUs, but important for ATI GPUs (set it to 64).

MaxTexIndirections=n
: Maximum number of texture indirections. Not an issue for NVIDIA GPUs, but important for ATI GPUs (set it to 4).

MaxDrawBuffers=n
: Maximum draw buffers for use with **ARB_draw_buffers**. Set to 1 for NV3x GPUs. Use to 4 for NV4x or ATI GPUs.

MaxLocalParams=n
: Maximum allowable local parameters.

DATA TYPES

to-be-written

SEMANTICS

VARYING INPUT SEMANTICS

to-be-written

UNIFORM INPUT SEMANTICS

to-be-written

OUTPUT SEMANTICS

to-be-written

STANDARD LIBRARY ISSUES

to-be-written

NAME

arbvp1 − OpenGL vertex profile for multi-vendor ARB_vertex_program extension

SYNPOSIS

```
arbvp1
```

DESCRIPTION

This OpenGL profile corresponds to the per-vertex functionality introduced by GeForce3. This profile is supported by any OpenGL implementation that conformantly implements ARB_vertex_program.

The compiler output for this profile conforms to the assembly format defined by **ARB_vertex_program**.

Data-dependent loops are not allowed; all loops must be unrollable.

Conditional expressions are supported without branching so both conditions must be evaluated.

Relative indexing of uniform arrays *is* supported; but texture accesses are not supported.

3D API DEPENDENCIES

Requires OpenGL support for the multi-vendor **ARB_vertex_program** extension. These extensions were introduced by GeForce3 and Quadro DCC GPUs. ATI GPUs also support this extension.

PROFILE OPTIONS

NumTemps=n
: Number of temporaries to use (from 12 to 32).

MaxInstructions=n
: Maximum allowable (static) instructions.

MaxLocalParams=n
: Maximum allowable local parameters.

DATA TYPES

to-be-written

SEMANTICS

VARYING INPUT SEMANTICS

to-be-written

UNIFORM INPUT SEMANTICS

to-be-written

OUTPUT SEMANTICS

to-be-written

STANDARD LIBRARY ISSUES

to-be-written

NAME

fp20 − OpenGL fragment profile for NV2x (GeForce3, GeForce4 Ti, Quadro DCC, etc.)

SYNPOSIS

```
fp20
```

DESCRIPTION

This OpenGL profile corresponds to the per-fragment functionality introduced by GeForce3.

The capabilities of this profile are quite limited.

The compiler output for this profile conforms to the **nvparse** file format for describing **NV_register_combiners** and **NV_texture_shader** state configurations.

3D API DEPENDENCIES

Requires OpenGL support for **NV_texture_shader**, **NV_texture_shader2**, and **NV_register_combiners2** extensions. These extensions were introduced by GeForce3 and Quadro DCC GPUs.

Some standard library functions may require **NV_texture_shader3**. This extension was introduced by GeForce4 Ti and Quadro4 XGL GPUs.

PROFILE OPTIONS

None.

DATA TYPES

fixed — The **fixed** data type corresponds to a native signed 9−bit integers normalized to the [−1.0,+1.0] range.

float
half — In most cases, the **float** and **half** data types are mapped to **fixed** for math operations.

Certain built-in standard library functions corresponding to **NV_texture_shader** operations operate at 32−bit floating-point precision.

SEMANTICS

INPUT SEMANTICS

The varying input semantics in the **fp20** profile correspond to the respectively named varying output semantics of the **vp20** profile.

Binding Semantics Name	Corresponding Data
COLOR COLOR0 COL COL0	Input primary color
COLOR1 COL1	Input secondary color
TEX0 TEXCOORD0	Input texture coordinate sets 0
TEX1 TEXCOORD1	Input texture coordinate sets 1
TEX2 TEXCOORD2	Input texture coordinate sets 2

```
TEX3                            Input texture coordinate sets 3
TEXCOORD3

FOGP                            Input fog color (XYZ) and factor (W)
FOG
```

OUTPUT SEMANTICS

```
COLOR                           Output color (float4)
COLOR0
COL0
COL

DEPTH                           Output depth (float)
DEPR
```

STANDARD LIBRARY ISSUES

There are a lot of standard library issues with this profile.

Because the 'fp20' profile has limited capabilities, not all of the Cg standard library functions are supported. The list below presents the Cg standard library functions that are supported by this profile. See the standard library documentation for descriptions of these functions.

```
dot(floatN, floatN)
lerp(floatN, floatN, floatN)
lerp(floatN, floatN, float)
tex1D(sampler1D, float)
tex1D(sampler1D, float2)
tex1Dproj(sampler1D, float2)
tex1Dproj(sampler1D, float3)
tex2D(sampler2D, float2)
tex2D(sampler2D, float3)
tex2Dproj(sampler2D, float3)
tex2Dproj(sampler2D, float4)
texRECT(samplerRECT, float2)
texRECT(samplerRECT, float3)
texRECTproj(samplerRECT, float3)
texRECTproj(samplerRECT, float4)
tex3D(sampler3D, float3)
tex3Dproj(sampler3D, float4)
texCUBE(samplerCUBE, float3)
texCUBEproj(samplerCUBE, float4)
```

Note: The non-projective texture lookup functions are actually done as projective lookups on the underlying hardware. Because of this, the 'w' component of the texture coordinates passed to these functions from the application or vertex program must contain the value 1.

Texture coordinate parameters for projective texture lookup functions must have swizzles that match the swizzle done by the generated texture shader instruction. While this may seem burdensome, it is intended to allow 'fp20' profile programs to behave correctly under other pixel shader profiles. The list below shows the swizzles required on the texture coordinate parameter to the projective texture lookup functions.

```
Texture lookup function       Texture coordinate swizzle
```

```
tex1Dproj                    .xw/.ra
tex2Dproj                    .xyw/.rga
texRECTproj                  .xyw/.rga
tex3Dproj                    .xyzw/.rgba
texCUBEproj                  .xyzw/.rgba
```

TEXTURE SHADER OPERATIONS

In order to take advantage of the more complex hard-wired shader operations provided by **NV_texture_shader**, a collection of built-in functions implement the various shader operations.

offsettex2D
offsettexRECT

```
offsettex2D(uniform sampler2D tex,
            float2 st,
            float4 prevlookup,
            uniform float4 m)

offsettexRECT(uniform samplerRECT tex,
              float2 st,
              float4 prevlookup,
              uniform float4 m)
```

Performs the following

```
float2 newst = st + m.xy * prevlookup.xx + m.zw * prevlookup.yy;
return tex2D/RECT(tex, newst);
```

where ’st’ are texture coordinates associated with sampler ’tex’, ’prevlookup’ is the result of a previous texture operation, and ’m’ is the offset texture matrix. This function can be used to generate the ’offset_2d’ or ’offset_rectangle’ NV_texture_shader instructions.

offsettex2DScaleBias
offsettexRECTScaleBias

```
offsettex2DScaleBias(uniform sampler2D tex,
                     float2 st,
                     float4 prevlookup,
                     uniform float4 m,
                     uniform float scale,
                     uniform float bias)

offsettexRECTScaleBias(uniform samplerRECT tex,
                       float2 st,
                       float4 prevlookup,
                       uniform float4 m,
                       uniform float scale,
                       uniform float bias)
```

Performs the following

```
float2 newst = st + m.xy * prevlookup.xx + m.zw * prevlookup.yy;
float4 result = tex2D/RECT(tex, newst);
return result * saturate(prevlookup.z * scale + bias);
```

where ’st’ are texture coordinates associated with sampler ’tex’, ’prevlookup’ is the result of a previous texture operation, ’m’ is the offset texture matrix, ’scale’ is the offset texture scale and

'bias' is the offset texture bias. This function can be used to generate the 'offset_2d_scale' or 'offset_rectangle_scale' NV_texture_shader instructions.

tex1D_dp3(sampler1D tex, float3 str, float4 prevlookup

```
tex1D_dp3(sampler1D tex,
          float3 str,
          float4 prevlookup
```

Performs the following

```
return tex1D(tex, dot(str, prevlookup.xyz));
```

where 'str' are texture coordinates associated with sampler 'tex' and 'prevlookup' is the result of a previous texture operation. This function can be used to generate the 'dot_product_1d' NV_texture_shader instruction.

tex2D_dp3x2
texRECT_dp3x2

```
tex2D_dp3x2(uniform sampler2D tex,
            float3 str,
            float4 intermediate_coord,
            float4 prevlookup)

texRECT_dp3x2(uniform samplerRECT tex,
              float3 str,
              float4 intermediate_coord,
              float4 prevlookup)
```

Performs the following

```
float2 newst = float2(dot(intermediate_coord.xyz, prevlookup.xyz),
                      dot(str, prevlookup.xyz));
return tex2D/RECT(tex, newst);
```

where 'str' are texture coordinates associated with sampler 'tex', 'prevlookup' is the result of a previous texture operation and 'intermediate_coord' are texture coordinates associated with the previous texture unit. This function can be used to generate the 'dot_product_2d' or 'dot_product_rectangle' NV_texture_shader instruction combinations.

tex3D_dp3x3
texCUBE_dp3x3

```
tex3D_dp3x3(sampler3D tex,
            float3 str,
            float4 intermediate_coord1,
            float4 intermediate_coord2,
            float4 prevlookup)

texCUBE_dp3x3(samplerCUBE tex,
              float3 str,
              float4 intermediate_coord1,
              float4 intermediate_coord2,
              float4 prevlookup)
```

Performs the following

```
float3 newst = float3(dot(intermediate_coord1.xyz, prevlookup.xyz),
                      dot(intermediate_coord2.xyz, prevlookup.xyz),
                      dot(str, prevlookup.xyz));
return tex3D/CUBE(tex, newst);
```

where 'str' are texture coordinates associated with sampler 'tex', 'prevlookup' is the result of a previous texture operation, 'intermediate_coord1' are texture coordinates associated with the 'n-2' texture unit and 'intermediate_coord2' are texture coordinates associated with the 'n-1' texture unit. This function can be used to generate the 'dot_product_3d' or 'dot_product_cube_map' NV_texture_shader instruction combinations.

texCUBE_reflect_dp3x3

```
texCUBE_reflect_dp3x3(uniform samplerCUBE tex,
                      float4 strq,
                      float4 intermediate_coord1,
                      float4 intermediate_coord2,
                      float4 prevlookup)
```

Performs the following

```
float3 E = float3(intermediate_coord2.w, intermediate_coord1.w, strq.w);
float3 N = float3(dot(intermediate_coord1.xyz, prevlookup.xyz),
                  dot(intermediate_coord2.xyz, prevlookup.xyz),
                  dot(strq.xyz, prevlookup.xyz));
return texCUBE(tex, 2 * dot(N, E) / dot(N, N) * N - E);
```

where 'strq' are texture coordinates associated with sampler 'tex', 'prevlookup' is the result of a previous texture operation, 'intermediate_coord1' are texture coordinates associated with the 'n-2' texture unit and 'intermediate_coord2' are texture coordinates associated with the 'n-1' texture unit. This function can be used to generate the 'dot_product_reflect_cube_map_eye_from_qs' NV_texture_shader instruction combination.

texCUBE_reflect_eye_dp3x3

```
texCUBE_reflect_eye_dp3x3(uniform samplerCUBE tex,
                          float3 str,
                          float4 intermediate_coord1,
                          float4 intermediate_coord2,
                          float4 prevlookup,
                          uniform float3 eye)
```

Performs the following

```
float3 N = float3(dot(intermediate_coord1.xyz, prevlookup.xyz),
                  dot(intermediate_coord2.xyz, prevlookup.xyz),
                  dot(coords.xyz, prevlookup.xyz));
return texCUBE(tex, 2 * dot(N, E) / dot(N, N) * N - E);
```

where 'strq' are texture coordinates associated with sampler 'tex', 'prevlookup' is the result of a previous texture operation, 'intermediate_coord1' are texture coordinates associated with the 'n-2' texture unit, 'intermediate_coord2' are texture coordinates associated with the 'n-1' texture unit and 'eye' is the eye-ray vector. This function can be used generate the 'dot_product_reflect_cube_map_const_eye' NV_texture_shader instruction combination.

tex_dp3x2_depth

```
tex_dp3x2_depth(float3 str,
                float4 intermediate_coord,
                float4 prevlookup)
```

Performs the following

```
float z = dot(intermediate_coord.xyz, prevlookup.xyz);
float w = dot(str, prevlookup.xyz);
return z / w;
```

where 'str' are texture coordinates associated with the 'n'th texture unit, 'intermediate_coord' are texture coordinates associated with the 'n-1' texture unit and 'prevlookup' is the result of a previous texture operation. This function can be used in conjunction with the 'DEPTH' varying out semantic to generate the 'dot_product_depth_replace' NV_texture_shader instruction combination.

EXAMPLES

The following examples illustrate how a developer can use Cg to achieve NV_texture_shader/NV_register_combiners functionality.

Example 1

```
 struct VertexOut {
    float4 color     : COLOR0;
    float4 texCoord0 : TEXCOORD0;
    float4 texCoord1 : TEXCOORD1;
};

float4 main(VertexOut IN,
            uniform sampler2D diffuseMap,
            uniform sampler2D normalMap) : COLOR
{
    float4 diffuseTexColor = tex2D(diffuseMap, IN.texCoord0.xy);
    float4 normal = 2 * (tex2D(normalMap, IN.texCoord1.xy) - 0.5);
    float3 light_vector = 2 * (IN.color.rgb - 0.5);
    float4 dot_result = saturate(dot(light_vector, normal.xyz).xxxx);
    return dot_result * diffuseTexColor;
}
```

Example 2

```
struct VertexOut {
    float4 texCoord0 : TEXCOORD0;
    float4 texCoord1 : TEXCOORD1;
    float4 texCoord2 : TEXCOORD2;
    float4 texCoord3 : TEXCOORD3;
};
```

```
float4 main(VertexOut IN,
            uniform sampler2D normalMap,
            uniform sampler2D intensityMap,
            uniform sampler2D colorMap) : COLOR
{
    float4 normal = 2 * (tex2D(normalMap, IN.texCoord0.xy) - 0.5);
    float2 intensCoord = float2(dot(IN.texCoord1.xyz, normal.xyz),
                                dot(IN.texCoord2.xyz, normal.xyz));
    float4 intensity = tex2D(intensityMap, intensCoord);
    float4 color = tex2D(colorMap, IN.texCoord3.xy);
    return color * intensity;
}
```

NAME

fp30 − OpenGL fragment profile for NV3x (GeForce FX, Quadro FX, etc.)

SYNPOSIS

```
fp30
```

DESCRIPTION

This OpenGL profile corresponds to the per-fragment functionality introduced by the GeForce FX and Quadro FX line of NVIDIA GPUs.

The compiler output for this profile conforms to the assembly format defined by **NV_fragment_program**.

Data-dependent loops are not allowed; all loops must be unrollable.

Conditional expressions are supported without branching so both conditions must be evaluated.

Relative indexing of uniform arrays is not supported; use texture accesses instead.

3D API DEPENDENCIES

Requires OpenGL support for the **NV_fragment_program** extension. These extensions were introduced by the GeForce FX and Quadro FX GPUs.

PROFILE OPTIONS

None.

DATA TYPES

fixed — The **fixed** data type corresponds to a native signed fixed-point integers with the range [−2.0,+2.0), sometimes called *fx12*. This type provides 10 fractional bits of precision.

half — The **half** data type corresponds to a floating-point encoding with a sign bit, 10 mantissa bits, and 5 exponent bits (biased by 16), sometimes called *s10e5*.

float — The **half** data type corresponds to a standard IEEE 754 single-precision floating-point encoding with a sign bit, 23 mantissa bits, and 8 exponent bits (biased by 128), sometimes called *s10e5*.

SEMANTICS

VARYING INPUT SEMANTICS

The varying input semantics in the **fp30** profile correspond to the respectively named varying output semantics of the **vp30** profile.

```
Binding Semantics Name          Corresponding Data

COLOR                           Input primary color
COLOR0
COL
COL0

COLOR1                          Input secondary color
COL1

TEX0                            Input texture coordinate sets 0
TEXCOORD0

TEX1                            Input texture coordinate sets 1
TEXCOORD1

TEX2                            Input texture coordinate sets 2
TEXCOORD2
```

```
TEX3                            Input texture coordinate sets 3
TEXCOORD3

TEX4                            Input texture coordinate sets 4
TEXCOORD4

TEX5                            Input texture coordinate sets 5
TEXCOORD5

TEX6                            Input texture coordinate sets 6
TEXCOORD6

TEX7                            Input texture coordinate sets 7
TEXCOORD7

FOGP                            Input fog color (XYZ) and factor (W)
FOG
```

UNIFORM INPUT SEMANTICS

Sixteen texture units are supported:

```
Binding Semantic Name           Corresponding Data

TEXUNIT0                        Texture unit 0
TEXUNIT1                        Texture unit 1
...
TEXUNIT15                       Texture unit 15
```

OUTPUT SEMANTICS

```
COLOR                           Output color (float4)
COLOR0
COL0
COL

DEPTH                           Output depth (float)
DEPR
```

STANDARD LIBRARY ISSUES

Functions that compute partial derivatives *are* supported.

NAME

fp40 − OpenGL fragment profile for NV4x (GeForce 6xxx and 7xxx Series, NV4x-based Quadro FX, etc.)

SYNPOSIS

```
fp40
```

DESCRIPTION

This OpenGL profile corresponds to the per-fragment functionality introduced by the GeForce 6800 and other NV4x-based NVIDIA GPUs.

The compiler output for this profile conforms to the assembly format defined by **NV_fragment_program2**.

Data-dependent loops *are* allowed with a limit of 256 iterations maximum. Four levels of nesting are allowed.

Conditional expressions *can be* supported with data-dependent branching.

Relative indexing of uniform arrays is not supported; use texture accesses instead.

3D API DEPENDENCIES

Requires OpenGL support for the **NV_fragment_program2** extension. These extensions were introduced by the GeForce 6800 and other NV4x-based GPUs.

PROFILE OPTIONS

None.

DATA TYPES

fixed — The **fixed** data type corresponds to a native signed fixed-point integers with the range [−2.0,+2.0), sometimes called *fx12*. This type provides 10 fractional bits of precision.

half — The **half** data type corresponds to a floating-point encoding with a sign bit, 10 mantissa bits, and 5 exponent bits (biased by 16), sometimes called *s10e5*.

float — The **half** data type corresponds to a standard IEEE 754 single-precision floating-point encoding with a sign bit, 23 mantissa bits, and 8 exponent bits (biased by 128), sometimes called *s10e5*.

SEMANTICS

VARYING INPUT SEMANTICS

The varying input semantics in the **fp30** profile correspond to the respectively named varying output semantics of the **vp30** profile.

Binding Semantics Name	Corresponding Data
COLOR COLOR0 COL COL0	Input primary color
COLOR1 COL1	Input secondary color
TEX0 TEXCOORD0	Input texture coordinate sets 0
TEX1 TEXCOORD1	Input texture coordinate sets 1
TEX2 TEXCOORD2	Input texture coordinate sets 2

TEX3 TEXCOORD3	Input texture coordinate sets 3
TEX4 TEXCOORD4	Input texture coordinate sets 4
TEX5 TEXCOORD5	Input texture coordinate sets 5
TEX6 TEXCOORD6	Input texture coordinate sets 6
TEX7 TEXCOORD7	Input texture coordinate sets 7
FOGP FOG	Input fog color (XYZ) and factor (W)
FACE	Polygon facing. +1 for front-facing polygon or line or point -1 for back-facing polygon

UNIFORM INPUT SEMANTICS

Sixteen texture units are supported:

Binding Semantic Name	Corresponding Data
TEXUNIT0	Texture unit 0
TEXUNIT1	Texture unit 1
...	
TEXUNIT15	Texture unit 15

OUTPUT SEMANTICS

COLOR COLOR0 COL0 COL	Output color (float4)
DEPTH DEPR	Output depth (float)

STANDARD LIBRARY ISSUES

Functions that compute partial derivatives *are* supported.

NAME

vp20 − OpenGL fragment profile for NV2x (GeForce3, GeForce4 Ti, Quadro DCC, etc.)

SYNPOSIS

```
vp20
```

DESCRIPTION

This OpenGL profile corresponds to the per-vertex functionality introduced by GeForce3.

The compiler output for this profile conforms to the assembly format defined by **NV_vertex_program1_1** (which assumes **NV_vertex_program**).

Data-dependent loops are not allowed; all loops must be unrollable.

Conditional expressions are supported without branching so both conditions must be evaluated.

Relative indexing of uniform arrays *is* supported; but texture accesses are not supported.

3D API DEPENDENCIES

Requires OpenGL support for **NV_vertex_program** and **NV_vertex_program1_1** extensions. These extensions were introduced by GeForce3 and Quadro DCC GPUs.

PROFILE OPTIONS

None.

DATA TYPES

to-be-written

SEMANTICS

VARYING INPUT SEMANTICS

to-be-written

UNIFORM INPUT SEMANTICS

to-be-written

OUTPUT SEMANTICS

to-be-written

STANDARD LIBRARY ISSUES

to-be-written

NAME

vp30 − OpenGL fragment profile for NV3x (GeForce FX, Quadro FX, etc.)

SYNPOSIS

```
vp30
```

DESCRIPTION

This OpenGL profile corresponds to the per-vertex functionality introduced by the GeForce FX and Quadro FX line of NVIDIA GPUs.

The compiler output for this profile conforms to the assembly format defined by **NV_vertex_program2**.

Data-dependent loops and branching *are* allowed.

Relative indexing of uniform arrays *is* supported; but texture accesses are not supported.

3D API DEPENDENCIES

Requires OpenGL support for the **NV_vertex_program2** extension. These extensions were introduced by the GeForce FX and Quadro FX GPUs.

PROFILE OPTIONS

None.

DATA TYPES

to-be-written

SEMANTICS

VARYING INPUT SEMANTICS

to-be-written

UNIFORM INPUT SEMANTICS

to-be-written

OUTPUT SEMANTICS

to-be-written

STANDARD LIBRARY ISSUES

to-be-written

NAME

vp40 − OpenGL vertex profile for NV4x (GeForce 6xxx and 7xxx Series, NV4x-based Quadro FX, etc.)

SYNPOSIS

```
vp40
```

DESCRIPTION

This OpenGL profile corresponds to the per-vertex functionality introduced by the GeForce 6800 and other NV4x-based NVIDIA GPUs.

The compiler output for this profile conforms to the assembly format defined by **NV_vertex_program3** and **ARB_vertex_program**.

Data-dependent loops and branching *are* allowed.

Relative indexing of uniform arrays *is* supported.

Texture accesses are supported. However substantial limitations on vertex texturing exist for hardware acceleration by NV4x hardware.

NV4x hardware accelerates vertex fetches only for 1−, 3−, and 4−component floating-point textures. NV4x hardware does not accelerated vertex-texturing for cube maps or 3D textures. NV4x does allow non-power-of-two sizes (width and height).

3D API DEPENDENCIES

Requires OpenGL support for the **NV_fragment_program3** extension. These extensions were introduced by the GeForce 6800 and other NV4x-based GPUs.

PROFILE OPTIONS

None.

DATA TYPES

to-be-written

SEMANTICS

VARYING INPUT SEMANTICS

to-be-written

UNIFORM INPUT SEMANTICS

to-be-written

OUTPUT SEMANTICS

to-be-written

STANDARD LIBRARY ISSUES

to-be-written

NAME

abs − returns absolute value of scalars and vectors.

SYNPOSIS

```
float  abs(float  a);
float1 abs(float1 a);
float2 abs(float2 a);
float3 abs(float3 a);
float4 abs(float4 a);

half   abs(half  a);
half1  abs(half1 a);
half2  abs(half2 a);
half3  abs(half3 a);
half4  abs(half4 a);

fixed  abs(fixed  a);
fixed1 abs(fixed1 a);
fixed2 abs(fixed2 a);
fixed3 abs(fixed3 a);
fixed4 abs(fixed4 a);
```

PARAMETERS

a Vector or scalar of which to determine the absolute value.

DESCRIPTION

Returns the absolute value of a scalar or vector.

For vectors, the returned vector contains the absolute value of each element of the input vector.

REFERENCE IMPLEMENTATION

abs for a **float** scalar could be implemented like this.

```
float abs(float a)
{
  return max(-a, a);
}
```

PROFILE SUPPORT

abs is supported in all profiles.

Support in the fp20 is limited.

Consider **abs** to be free or extremely inexpensive.

SEE ALSO

the max manpage

NAME

acos − returns arccosine of scalars and vectors.

SYNPOSIS

```
float  acos(float a);
float1 acos(float2 a);
float2 acos(float2 a);
float3 acos(float3 a);
float4 acos(float4 a);

half   acos(half a);
half1  acos(half2 a);
half2  acos(half2 a);
half3  acos(half3 a);
half4  acos(half4 a);

fixed  acos(fixed a);
fixed1 acos(fixed2 a);
fixed2 acos(fixed2 a);
fixed3 acos(fixed3 a);
fixed4 acos(fixed4 a);
```

PARAMETERS

a Vector or scalar of which to determine the arccosine.

DESCRIPTION

Returns the arccosine of *a* in the range [0,pi], expecting *a* to be in the range [−1,+1].

For vectors, the returned vector contains the arccosine of each element of the input vector.

REFERENCE IMPLEMENTATION

acos for a **float** scalar could be implemented like this.

```
// Handbook of Mathematical Functions
// M. Abramowitz and I.A. Stegun, Ed.

// Absolute error <= 6.7e-5
float acos(float x) {
  float negate = float(x < 0);
  x = abs(x);
  float ret = -0.0187293;
  ret = ret * x;
  ret = ret + 0.0742610;
  ret = ret * x;
  ret = ret - 0.2121144;
  ret = ret * x;
  ret = ret + 1.5707288;
  ret = ret * sqrt(1.0-x);
  ret = ret - 2 * negate * ret;
  return negate * 3.14159265358979 + ret;
}
```

PROFILE SUPPORT

acos is supported in all profiles.

Support in the fp20 is limited.

SEE ALSO

the abs manpage, the asin manpage, the cos manpage, the sqrt manpage

NAME

all − returns true if a boolean scalar or all components of a boolean vector are true.

SYNPOSIS

```
bool all(bool a);
bool all(bool1 a);
bool all(bool2 a);
bool all(bool3 a);
bool all(bool4 a);
```

PARAMETERS

a Boolean vector or scalar of which to determine if any component is true.

DESCRIPTION

Returns true if a boolean scalar or all components of a boolean vector are true.

REFERENCE IMPLEMENTATION

all for a **bool4** vector could be implemented like this.

```
bool all(bool4 a)
{
  return a.x && a.y && a.z && a.w;
}
```

PROFILE SUPPORT

all is supported in all profiles.

Support in the fp20 is limited.

SEE ALSO

the any manpage

NAME

any − returns true if a boolean scalar or any component of a boolean vector is true.

SYNPOSIS

```
bool any(bool a);
bool any(bool1 a);
bool any(bool2 a);
bool any(bool3 a);
bool any(bool4 a);
```

PARAMETERS

a Boolean vector or scalar of which to determine if any component is true.

DESCRIPTION

Returns true if a boolean scalar or any component of a boolean vector is true.

REFERENCE IMPLEMENTATION

any for a **bool4** vector could be implemented like this.

```
bool any(bool4 a)
{
  return a.x || a.y || a.z || a.w;
}
```

PROFILE SUPPORT

any is supported in all profiles.

Support in the fp20 is limited.

SEE ALSO

the all manpage

NAME

asin − returns arcsine of scalars and vectors.

SYNPOSIS

```
float  asin(float a);
float1 asin(float2 a);
float2 asin(float2 a);
float3 asin(float3 a);
float4 asin(float4 a);

half   asin(half a);
half1  asin(half2 a);
half2  asin(half2 a);
half3  asin(half3 a);
half4  asin(half4 a);

fixed  asin(fixed a);
fixed1 asin(fixed2 a);
fixed2 asin(fixed2 a);
fixed3 asin(fixed3 a);
fixed4 asin(fixed4 a);
```

PARAMETERS

a Vector or scalar of which to determine the arcsine.

DESCRIPTION

Returns the arcsine of *a* in the range [−pi/2,+pi/2], expecting *a* to be in the range [−1,+1].

For vectors, the returned vector contains the arcsine of each element of the input vector.

REFERENCE IMPLEMENTATION

asin for a **float** scalar could be implemented like this.

```
// Handbook of Mathematical Functions
// M. Abramowitz and I.A. Stegun, Ed.

float asin(float x) {
  float negate = float(x < 0);
  x = abs(x);
  float ret = -0.0187293;
  ret *= x;
  ret += 0.0742610;
  ret *= x;
  ret -= 0.2121144;
  ret *= x;
  ret += 1.5707288;
  ret = 3.14159265358979*0.5 - sqrt(1.0 - x)*ret;
  return ret - 2 * negate * ret;
}
```

PROFILE SUPPORT

asin is supported in all profiles.

Support in the fp20 is limited.

SEE ALSO

the abs manpage, the acos manpage, the sin manpage, the sqrt manpage

NAME

atan − returns arctangent of scalars and vectors.

SYNPOSIS

```
float  atan(float a);
float1 atan(float2 a);
float2 atan(float2 a);
float3 atan(float3 a);
float4 atan(float4 a);

half   atan(half a);
half1  atan(half2 a);
half2  atan(half2 a);
half3  atan(half3 a);
half4  atan(half4 a);

fixed  atan(fixed a);
fixed1 atan(fixed2 a);
fixed2 atan(fixed2 a);
fixed3 atan(fixed3 a);
fixed4 atan(fixed4 a);
```

PARAMETERS

a Vector or scalar of which to determine the arctangent.

DESCRIPTION

Returns the arctangent of x in the range of −pi/2 to pi/2 radians.

For vectors, the returned vector contains the arctangent of each element of the input vector.

REFERENCE IMPLEMENTATION

atan for a **float** scalar could be implemented like this.

```
float atan(float x) {
    return atan2(x, float(1));
}
```

atan2 is typically implemented as an approximation.

PROFILE SUPPORT

atan is supported in all profiles but fp20.

SEE ALSO

the abs manpage, the acos manpage, the asin manpage, the atan2 manpage. the sqrt manpage, the tan manpage

NAME

atan2 − returns arctangent of scalars and vectors.

SYNPOSIS

```
float  atan2(float y, float x);
float1 atan2(float2 y, float1 x);
float2 atan2(float2 y, float2 x);
float3 atan2(float3 y, float3 x);
float4 atan2(float4 y, float4 x);

half   atan2(half y, half x);
half1  atan2(half2 y, half x);
half2  atan2(half2 y, half x);
half3  atan2(half3 y, half x);
half4  atan2(half4 y, half x);

fixed  atan2(fixed y, fixed x);
fixed1 atan2(fixed2 y, fixed x);
fixed2 atan2(fixed2 y, fixed x);
fixed3 atan2(fixed3 y, fixed x);
fixed4 atan2(fixed4 y, fixed x);
```

PARAMETERS

y Vector or scalar for numerator of ratio of which to determine the arctangent.

x Vector or scalar of denominator of ratio of which to determine the arctangent.

DESCRIPTION

atan2 calculates the arctangent of y/x. **atan2** is well defined for every point other than the origin, even if x equals 0 and y does not equal 0.

For vectors, the returned vector contains the arctangent of each element of the input vector.

REFERENCE IMPLEMENTATION

atan2 for a **float2** scalar could be implemented as an approximation like this.

```
float2 atan2(float2 y, float2 x)
{
  float2 t0, t1, t2, t3, t4;

  t3 = abs(x);
  t1 = abs(y);
  t0 = max(t3, t1);
  t1 = min(t3, t1);
  t3 = float(1) / t0;
  t3 = t1 * t3;

  t4 = t3 * t3;
  t0 =         - float(0.013480470);
  t0 = t0 * t4 + float(0.057477314);
  t0 = t0 * t4 - float(0.121239071);
  t0 = t0 * t4 + float(0.195635925);
  t0 = t0 * t4 - float(0.332994597);
  t0 = t0 * t4 + float(0.999995630);
  t3 = t0 * t3;
```

```
  t3 = (abs(y) > abs(x)) ? float(1.570796327) - t3 : t3;
  t3 = (x < 0) ?  float(3.141592654) - t3 : t3;
  t3 = (y < 0) ? -t3 : t3;

  return t3;
}
```

PROFILE SUPPORT

atan2 is supported in all profiles but fp20.

SEE ALSO

the abs manpage, the acos manpage, the asin manpage, the atan manpage. the sqrt manpage, the tan manpage

NAME

ceil − returns smallest integer not less than a scalar or each vector component.

SYNPOSIS

```
float  ceil(float a);
float1 ceil(float2 a);
float2 ceil(float2 a);
float3 ceil(float3 a);
float4 ceil(float4 a);

half   ceil(half a);
half1  ceil(half2 a);
half2  ceil(half2 a);
half3  ceil(half3 a);
half4  ceil(half4 a);

fixed  ceil(fixed a);
fixed1 ceil(fixed2 a);
fixed2 ceil(fixed2 a);
fixed3 ceil(fixed3 a);
fixed4 ceil(fixed4 a);
```

PARAMETERS

a Vector or scalar of which to determine the ceiling.

DESCRIPTION

Returns the ceiling or smallest integer not less than a scalar or each vector component.

REFERENCE IMPLEMENTATION

ceil for a **float** scalar could be implemented like this.

```
float ceil(float v)
{
  return -floor(-v);
}
```

PROFILE SUPPORT

ceil is supported in all profiles except fp20.

SEE ALSO

the floor manpage

NAME

clamp − returns smallest integer not less than a scalar or each vector component.

SYNPOSIS

```
float  clamp(float x, float a, float b);
float1 clamp(float2 x, float1 a, float1 b);
float2 clamp(float2 x, float2 a, float2 b);
float3 clamp(float3 x, float3 a, float3 b);
float4 clamp(float4 x, float4 a, float4 b);

half   clamp(half x, half a, half b);
half1  clamp(half2 x, half1 a, half1 b);
half2  clamp(half2 x, half2 a, half2 b);
half3  clamp(half3 x, half3 a, half3 b);
half4  clamp(half4 x, half4 a, half4 b);

fixed  clamp(fixed x, fixed a, fixed b);
fixed1 clamp(fixed2 x, fixed1 a, fixed1 b);
fixed2 clamp(fixed2 x, fixed2 a, fixed2 b);
fixed3 clamp(fixed3 x, fixed3 a, fixed3 b);
fixed4 clamp(fixed4 x, fixed4 a, fixed4 b);
```

PARAMETERS

x Vector or scalar to clamp.

a Vector or scalar for bottom of clamp range.

b Vector or scalar for top of clamp range.

DESCRIPTION

Returns *x* clamped to the range [*a*,*b*] as follows:

1) Returns *a* if *x* is less than *a*; else

2) Returns *b* if *x* is greater than *b*; else

3) Returns *x* otherwise.

For vectors, the returned vector contains the clamped result of each element of the vector *x* clamped using the respective element of vectors *a* and *b*.

REFERENCE IMPLEMENTATION

clamp for **float** scalars could be implemented like this.

```
float clamp(float x, float a, float b)
{
  return max(a, min(b, x));
}
```

PROFILE SUPPORT

clamp is supported in all profiles except fp20.

SEE ALSO

the max manpage, the min manpage, the saturate manpage

NAME

cos − returns cosine of scalars and vectors.

SYNPOSIS

```
float  cos(float a);
float1 cos(float2 a);
float2 cos(float2 a);
float3 cos(float3 a);
float4 cos(float4 a);

half   cos(half a);
half1  cos(half2 a);
half2  cos(half2 a);
half3  cos(half3 a);
half4  cos(half4 a);

fixed  cos(fixed a);
fixed1 cos(fixed2 a);
fixed2 cos(fixed2 a);
fixed3 cos(fixed3 a);
fixed4 cos(fixed4 a);
```

PARAMETERS

a Vector or scalar of which to determine the cosine.

DESCRIPTION

Returns the cosine of *a* in radians. The return value is in the range [−1,+1].

For vectors, the returned vector contains the cosine of each element of the input vector.

REFERENCE IMPLEMENTATION

cos is best implemented as a native cosine instruction, however **cos** for a **float** scalar could be implemented by an approximation like this.

```
cos(float a)
{
  /* C simulation gives a max absolute error of less than 1.8e-7 */
  const float4 c0 = float4( 0.0,            0.5,            1.0,           0.0
  const float4 c1 = float4( 0.25,          -9.0,            0.75,          0.15915
  const float4 c2 = float4( 24.9808039603, -24.9808039603, -60.1458091736, 60.1458
  const float4 c3 = float4( 85.4537887573, -85.4537887573, -64.9393539429, 64.9393
  const float4 c4 = float4( 19.7392082214, -19.7392082214, -1.0,           1.0

  /* r0.x = cos(a) */
  float3 r0, r1, r2;
```

```
r1.x  = c1.w * a;                       // normalize input
r1.y  = frac( r1.x );                   // and extract fraction
r2.x  = (float) ( r1.y < c1.x );        // range check: 0.0 to 0.25
r2.yz = (float2) ( r1.yy >= c1.yz );    // range check: 0.75 to 1.0
r2.y  = dot( r2, c4.zwz );              // range check: 0.25 to 0.75
r0    = c0.xyz - r1.yyy;                // range centering
r0    = r0 * r0;
r1    = c2.xyx * r0 + c2.zwz;           // start power series
r1    =     r1 * r0 + c3.xyx;
r1    =     r1 * r0 + c3.zwz;
r1    =     r1 * r0 + c4.xyx;
r1    =     r1 * r0 + c4.zwz;
r0.x  = dot( r1, -r2 );                 // range extract

return r0.x;
```

PROFILE SUPPORT

cos is fully supported in all profiles unless otherwise specified.

cos is supported via an approximation (shown above) in the vs_1, vp20, and arbvp1 profiles.

cos is unsupported in the fp20 and ps_1 profiles.

SEE ALSO

the acos manpage, the dot manpage, the frac manpage, the sin manpage, the tan manpage

NAME

cosh − returns hyperbolic cosine of scalars and vectors.

SYNPOSIS

```
float  cosh(float a);
float1 cosh(float2 a);
float2 cosh(float2 a);
float3 cosh(float3 a);
float4 cosh(float4 a);

half   cosh(half a);
half1  cosh(half2 a);
half2  cosh(half2 a);
half3  cosh(half3 a);
half4  cosh(half4 a);

fixed  cosh(fixed a);
fixed1 cosh(fixed2 a);
fixed2 cosh(fixed2 a);
fixed3 cosh(fixed3 a);
fixed4 cosh(fixed4 a);
```

PARAMETERS

a Vector or scalar of which to determine the hyperbolic cosine.

DESCRIPTION

Returns the hyperbolic cosine of *a*.

For vectors, the returned vector contains the hyperbolic cosine of each element of the input vector.

REFERENCE IMPLEMENTATION

cosh for a scalar **float** could be implemented like this.

```
float cosh(float x)
{
  return 0.5 * (exp(x)+exp(-x));
}
```

PROFILE SUPPORT

cosh is supported in all profiles except fp20.

SEE ALSO

the acos manpage, the cos manpage, the exp manpage, the sinh manpage, the tanh manpage

NAME

cross − returns the cross product of two three-component vectors

SYNPOSIS

```
float3 cross(float3 a, float3 b);

half3  cross(half3 a, half3 b);

fixed3 cross(fixed3 a, fixed3 b);
```

PARAMETERS

a Three-component vector.

b Three-component vector.

DESCRIPTION

Returns the cross product of three-component vectors *a* and *b*. The result is a three-component vector.

REFERENCE IMPLEMENTATION

cross for **float3** vectors could be implemented this way:

```
float3 cross(float3 a, float3 b)
{
  return a.yzx * b.zxy - a.zxy * b.yzx;
}
```

PROFILE SUPPORT

cross is supported in all profiles.

Support in the fp20 is limited.

SEE ALSO

the dot manpage

NAME

degrees − converts values of scalars and vectors from radians to degrees

SYNPOSIS

```
float  degrees(float  a);
float1 degrees(float1 a);
float2 degrees(float2 a);
float3 degrees(float3 a);
float4 degrees(float4 a);

half   degrees(half  a);
half1  degrees(half1 a);
half2  degrees(half2 a);
half3  degrees(half3 a);
half4  degrees(half4 a);

fixed  degrees(fixed  a);
fixed1 degrees(fixed1 a);
fixed2 degrees(fixed2 a);
fixed3 degrees(fixed3 a);
fixed4 degrees(fixed4 a);
```

PARAMETERS

a Vector or scalar of which to convert from radians to degrees.

DESCRIPTION

Returns the scalar or vector converted from radians to degrees.

For vectors, the returned vector contains each element of the input vector converted from radians to degrees.

REFERENCE IMPLEMENTATION

degrees for a **float** scalar could be implemented like this.

```
float degrees(float a)
{
  return 57.29577951 * a;
}
```

PROFILE SUPPORT

degrees is supported in all profiles except fp20.

SEE ALSO

the cos manpage, the radians manpage, the sin manpage, the tan manpage

NAME

determinant − returns the scalar determinant of a square matrix

SYNPOSIS

```
float determinant(float1x1 A);
float determinant(float2x2 A);
float determinant(float3x3 A);
float determinant(float4x4 A);
```

PARAMETERS

A Square matrix of which to compute the determinant.

DESCRIPTION

Returns the determinant of the square matrix *A*.

REFERENCE IMPLEMENTATION

The various **determinant** functions can be implemented like this:

```
float determinant(float1x1 A)
{
  return A._m00;
}

float determinant(float2x2 A)
{
  return A._m00*A._m11 - A._m01*A._m10;
}

float determinant(float3x3 A)
{
  return dot(A._m00_m01_m02,
             A._m11_m12_m10 * A._m22_m20_m21
           - A._m12_m10_m11 * A._m21_m22_m20);
}

float determinant(float4x4 A) {
  return dot(float4(1,-1,1,-1) * A._m00_m01_m02_m03,
               A._m11_m12_m13_m10*(  A._m22_m23_m20_m21*A._m33_m30_m31_m32
                                   - A._m23_m20_m21_m22*A._m32_m33_m30_m31)
             + A._m12_m13_m10_m11*(  A._m23_m20_m21_m22*A._m31_m32_m33_m30
                                   - A._m21_m22_m23_m20*A._m33_m30_m31_m32)
             + A._m13_m10_m11_m12*(  A._m21_m22_m23_m20*A._m32_m33_m30_m31
                                   - A._m22_m23_m20_m21*A._m31_m32_m33_m30));
}
```

PROFILE SUPPORT

determinant is supported in all profiles. However profiles such as fp20 and the ps_2 manpage without native floating-point will have problems computing the larger determinants and may have ranges issues computing even small determinants.

SEE ALSO

the mul manpage, the transpose manpage

NAME

dot − returns the scalar dot product of two vectors

SYNPOSIS

```
float  dot(float  a, float  b);
float1 dot(float1 a, float1 b);
float2 dot(float2 a, float2 b);
float3 dot(float3 a, float3 b);
float4 dot(float4 a, float4 b);

half   dot(half  a, half  b);
half1  dot(half1 a, half1 b);
half2  dot(half2 a, half2 b);
half3  dot(half3 a, half3 b);
half4  dot(half4 a, half4 b);

fixed  dot(fixed  a, fixed  b);
fixed1 dot(fixed1 a, fixed1 b);
fixed2 dot(fixed2 a, fixed2 b);
fixed3 dot(fixed3 a, fixed3 b);
fixed4 dot(fixed4 a, fixed4 b);
```

PARAMETERS

a First vector.

b Second vector.

DESCRIPTION

Returns the scalar dot product of two same-typed vectors *a* and *b*.

REFERENCE IMPLEMENTATION

dot for **float4** vectors could be implemented this way:

```
float dot(float4 a, float4 b)
{
  return a.x*b.x + a.y*b.y + a.z*b.z + a.w*b.w;
}
```

PROFILE SUPPORT

dot is supported in all profiles.

The **fixed3** dot product is very efficient in the fp20 and fp30 profiles.

The **float3** and **float4** dot products are very efficient in the vp20, vp30, vp40, arbvp1, fp30, fp40, and arbfp1 profiles.

The **float2** dot product is very efficient in the fp40 profile. In optimal circumstances, two two-component dot products can sometimes be performed at the four-component and three-component dot product rate.

SEE ALSO

the cross manpage, the mul manpage

NAME

length − return scalar Euclidean length of a vector

SYNPOSIS

```
float length(float v);
float length(float1 v);
float length(float2 v);
float length(float3 v);
float length(float4 v);

half length(half v);
half length(half1 v);
half length(half2 v);
half length(half3 v);
half length(half4 v);

fixed length(fixed v);
fixed length(fixed1 v);
fixed length(fixed2 v);
fixed length(fixed3 v);
fixed length(fixed4 v);
```

PARAMETERS

v Vector of which to determine the length.

DESCRIPTION

Returns the Euclidean length of a vector.

REFERENCE IMPLEMENTATION

length for a **float3** vector could be implemented like this.

```
float length(float3 v)
{
  return sqrt(dot(v,v));
}
```

PROFILE SUPPORT

length is supported in all profiles.

Support in the fp20 is limited.

SEE ALSO

the max manpage, the normalize manpage, the sqrt manpage, the dot manpage

NAME

max − returns the maximum of two scalars or each respective component of two vectors

SYNPOSIS

```
float  max(float  a, float  b);
float1 max(float1 a, float1 b);
float2 max(float2 a, float2 b);
float3 max(float3 a, float3 b);
float4 max(float4 a, float4 b);

half   max(half  a, half  b);
half1  max(half1 a, half1 b);
half2  max(half2 a, half2 b);
half3  max(half3 a, half3 b);
half4  max(half4 a, half4 b);

fixed  max(fixed  a, fixed  b);
fixed1 max(fixed1 a, fixed1 b);
fixed2 max(fixed2 a, fixed2 b);
fixed3 max(fixed3 a, fixed3 b);
fixed4 max(fixed4 a, fixed4 b);
```

PARAMETERS

a Scalar or vector.

b Scalar or vector.

DESCRIPTION

Returns the maximum of two same-typed scalars *a* and *b* or the respective components of two same-typed vectors *a* and *b*. The result is a three-component vector.

REFERENCE IMPLEMENTATION

max for **float3** vectors could be implemented this way:

```
float3 max(float3 a, float3 b)
{
  return float3(a.x > b.x ? a.x : b.x,
                a.y > b.y ? a.y : b.y,
                a.z > b.z ? a.z : b.z);
}
```

PROFILE SUPPORT

max is supported in all profiles. **max** is implemented as a compiler built-in.

Support in the fp20 is limited.

SEE ALSO

the clamp manpage, the min manpage

NAME

min − returns the minimum of two scalars or each respective component of two vectors

SYNPOSIS

```
float  min(float  a, float  b);
float1 min(float1 a, float1 b);
float2 min(float2 a, float2 b);
float3 min(float3 a, float3 b);
float4 min(float4 a, float4 b);

half   min(half  a, half  b);
half1  min(half1 a, half1 b);
half2  min(half2 a, half2 b);
half3  min(half3 a, half3 b);
half4  min(half4 a, half4 b);

fixed  min(fixed  a, fixed  b);
fixed1 min(fixed1 a, fixed1 b);
fixed2 min(fixed2 a, fixed2 b);
fixed3 min(fixed3 a, fixed3 b);
fixed4 min(fixed4 a, fixed4 b);
```

PARAMETERS

a Scalar or vector.

b Scalar or vector.

DESCRIPTION

Returns the minimum of two same-typed scalars *a* and *b* or the respective components of two same-typed vectors *a* and *b*. The result is a three-component vector.

REFERENCE IMPLEMENTATION

min for **float3** vectors could be implemented this way:

```
float3 min(float3 a, float3 b)
{
  return float3(a.x < b.x ? a.x : b.x,
                a.y < b.y ? a.y : b.y,
                a.z < b.z ? a.z : b.z);
}
```

PROFILE SUPPORT

min is supported in all profiles. **min** is implemented as a compiler built-in.

Support in the fp20 is limited.

SEE ALSO

the clamp manpage, the max manpage

NAME

radians − converts values of scalars and vectors from degrees to radians

SYNPOSIS

```
float  radians(float  a);
float1 radians(float1 a);
float2 radians(float2 a);
float3 radians(float3 a);
float4 radians(float4 a);

half   radians(half  a);
half1  radians(half1 a);
half2  radians(half2 a);
half3  radians(half3 a);
half4  radians(half4 a);

fixed  radians(fixed  a);
fixed1 radians(fixed1 a);
fixed2 radians(fixed2 a);
fixed3 radians(fixed3 a);
fixed4 radians(fixed4 a);
```

PARAMETERS

a Vector or scalar of which to convert from degrees to radians.

DESCRIPTION

Returns the scalar or vector converted from degrees to radians.

For vectors, the returned vector contains each element of the input vector converted from degrees to radians.

REFERENCE IMPLEMENTATION

radians for a **float** scalar could be implemented like this.

```
float radians(float a)
{
  return 0.017453292 * a;
}
```

PROFILE SUPPORT

radians is supported in all profiles except fp20.

SEE ALSO

the cos manpage, the degrees manpage, the sin manpage, the tan manpage

NAME

reflect − returns the reflectiton vector given an incidence vector and a normal vector.

SYNPOSIS

```
float  reflect(float  i, float  n);
float2 reflect(float2 i, float2 n);
float3 reflect(float3 i, float3 n);
float4 reflect(float4 i, float4 n);
```

PARAMETERS

i Incidence vector.

n Normal vector.

DESCRIPTION

Returns the reflectiton vector given an incidence vector *i* and a normal vector *n*. The resulting vector is the identical number of components as the two input vectors.

The normal vector *n* should be normalized. If *n* is normalized, the output vector will have the same length as the input incidence vector *i*.

REFERENCE IMPLEMENTATION

reflect for **float3** vectors could be implemented this way:

```
float3 reflect( float3 i, float3 n )
{
  return i - 2.0 * n * dot(n,i);
}
```

PROFILE SUPPORT

reflect is supported in all profiles.

Support in the fp20 is limited.

SEE ALSO

the dot manpage, the length manpage, the refract manpage

NAME

refract − computes a refraction vector.

SYNPOSIS

```
fixed3 refract(fixed3 i, fixed3 n, fixed eta);
half3  refract(half3  i, half3  n, half  eta);
float3 refract(float3 i, float3 n, float eta);
```

PARAMETERS

i Incidence vector.

n Normal vector.

eta Ratio of indices of refraction at the surface interface.

DESCRIPTION

Returns a refraction vector given an incidence vector, a normal vector for a surface, and a ratio of indices of refraction at the surface's interface.

The incidence vector *i* and normal vector *n* should be normalized.

REFERENCE IMPLEMENTATION

reflect for **float3** vectors could be implemented this way:

```
float3 refract( float3 i, float3 n, float eta )
{
  float cosi = dot(-i, n);
  float cost2 = 1.0f - eta * eta * (1.0f - cosi*cosi);
  float3 t = eta*i + ((eta*cosi - sqrt(abs(cost2))) * n);
  return t * (float3)(cost2 > 0);
}
```

PROFILE SUPPORT

refract is supported in all profiles.

Support in the fp20 is limited.

SEE ALSO

the abs manpage, the cos manpage, the dot manpage, the reflect manpage, the sqrt manpage

NAME

round − returns the rounded value of scalars or vectors

SYNOPSIS

```
float  round(float  a);
float1 round(float1 a);
float2 round(float2 a);
float3 round(float3 a);
float4 round(float4 a);

half   round(half  a);
half1  round(half1 a);
half2  round(half2 a);
half3  round(half3 a);
half4  round(half4 a);

fixed  round(fixed  a);
fixed1 round(fixed1 a);
fixed2 round(fixed2 a);
fixed3 round(fixed3 a);
fixed4 round(fixed4 a);
```

PARAMETERS

a Scalar or vector.

DESCRIPTION

Returns the rounded value of a scalar or vector.

For vectors, the returned vector contains the rounded value of each element of the input vector.

The round operation returns the nearest integer to the operand. The value returned by *round()* if the fractional portion of the operand is 0.5 is profile dependent. On older profiles without built-in *round()* support, round-to-nearest up rounding is used. On profiles newer than fp40/vp40, round-to-nearest even is used.

REFERENCE IMPLEMENTATION

round for **float** could be implemented this way:

```
// round-to-nearest even profiles
float round(float a)
{
  float x = a + 0.5;
  float f = floor(x);
  if (x == f) {
    if (a > 0)
      r = f - fmod(f, 2);
    else
      r = f + fmod(f, 2);
  }
}

// round-to-nearest up profiles
float round(float a)
{
  return floor(x + 0.5);
}
```

PROFILE SUPPORT

round is supported in all profiles except fp20.

SEE ALSO

the ceil manpage, the floor manpage, the fmod manpage

NAME

saturate − returns smallest integer not less than a scalar or each vector component.

SYNPOSIS

```
float  saturate(float x);
float1 saturate(float2 x);
float2 saturate(float2 x);
float3 saturate(float3 x);
float4 saturate(float4 x);

half   saturate(half x);
half1  saturate(half2 x);
half2  saturate(half2 x);
half3  saturate(half3 x);
half4  saturate(half4 x);

fixed  saturate(fixed x);
fixed1 saturate(fixed2 x);
fixed2 saturate(fixed2 x);
fixed3 saturate(fixed3 x);
fixed4 saturate(fixed4 x);
```

PARAMETERS

x Vector or scalar to saturate.

DESCRIPTION

Returns *x* saturated to the range [0,1] as follows:

1) Returns 0 if *x* is less than 0; else

2) Returns 1 if *x* is greater than 1; else

3) Returns *x* otherwise.

For vectors, the returned vector contains the saturated result of each element of the vector *x* saturated to [0,1].

REFERENCE IMPLEMENTATION

saturate for **float** scalars could be implemented like this.

```
float saturate(float x)
{
  return max(0, min(1, x));
}
```

PROFILE SUPPORT

saturate is supported in all profiles.

saturate is very efficient in the fp20, fp30, and fp40 profiles.

SEE ALSO

the clamp manpage, the max manpage, the min manpage

NAME

sin − returns sine of scalars and vectors.

SYNPOSIS

```
float  sin(float a);
float1 sin(float2 a);
float2 sin(float2 a);
float3 sin(float3 a);
float4 sin(float4 a);

half   sin(half a);
half1  sin(half2 a);
half2  sin(half2 a);
half3  sin(half3 a);
half4  sin(half4 a);

fixed  sin(fixed a);
fixed1 sin(fixed2 a);
fixed2 sin(fixed2 a);
fixed3 sin(fixed3 a);
fixed4 sin(fixed4 a);
```

PARAMETERS

a Vector or scalar of which to determine the sine.

DESCRIPTION

Returns the sine of *a* in radians. The return value is in the range [−1,+1].

For vectors, the returned vector contains the sine of each element of the input vector.

REFERENCE IMPLEMENTATION

sin is best implemented as a native sine instruction, however **sin** for a **float** scalar could be implemented by an approximation like this.

```
float sin(float a)
{
  /* C simulation gives a max absolute error of less than 1.8e-7 */
  float4 c0 = float4( 0.0,           0.5,           1.0,          0.0
  float4 c1 = float4( 0.25,         -9.0,           0.75,         0.15915494309
  float4 c2 = float4( 24.9808039603, -24.9808039603, -60.1458091736, 60.1458091736
  float4 c3 = float4( 85.4537887573, -85.4537887573, -64.9393539429, 64.9393539429
  float4 c4 = float4( 19.7392082214, -19.7392082214, -1.0,          1.0

  /* r0.x = sin(a) */
  float3 r0, r1, r2;
```

```
  r1.x  = c1.w * a - c1.x;               // only difference from cos!
  r1.y  = frac( r1.x );                  // and extract fraction
  r2.x  = (float) ( r1.y < c1.x );       // range check: 0.0 to 0.25
  r2.yz = (float2) ( r1.yy >= c1.yz );   // range check: 0.75 to 1.0
  r2.y  = dot( r2, c4.zwz );             // range check: 0.25 to 0.75
  r0    = c0.xyz - r1.yyy;               // range centering
  r0    = r0 * r0;
  r1    = c2.xyx * r0 + c2.zwz;          // start power series
  r1    =     r1 * r0 + c3.xyx;
  r1    =     r1 * r0 + c3.zwz;
  r1    =     r1 * r0 + c4.xyx;
  r1    =     r1 * r0 + c4.zwz;
  r0.x  = dot( r1, -r2 );                // range extract

  return r0.x;
}
```

PROFILE SUPPORT

sin is fully supported in all profiles unless otherwise specified.

sin is supported via an approximation (shown above) in the vs_1, vp20, and arbvp1 profiles.

sin is unsupported in the fp20 and ps_1 profiles.

SEE ALSO

the asin manpage, the cos manpage, the dot manpage, the frac manpage, the tan manpage

NAME

sinh − returns hyperbolic sine of scalars and vectors.

SYNPOSIS

```
float  sinh(float a);
float1 sinh(float2 a);
float2 sinh(float2 a);
float3 sinh(float3 a);
float4 sinh(float4 a);

half   sinh(half a);
half1  sinh(half2 a);
half2  sinh(half2 a);
half3  sinh(half3 a);
half4  sinh(half4 a);

fixed  sinh(fixed a);
fixed1 sinh(fixed2 a);
fixed2 sinh(fixed2 a);
fixed3 sinh(fixed3 a);
fixed4 sinh(fixed4 a);
```

PARAMETERS

a Vector or scalar of which to determine the hyperbolic sine.

DESCRIPTION

Returns the hyperbolic sine of *a*.

For vectors, the returned vector contains the hyperbolic sine of each element of the input vector.

REFERENCE IMPLEMENTATION

sinh for a scalar **float** could be implemented like this.

```
float sinh(float x)
{
  return 0.5 * (exp(x)-exp(-x));
}
```

PROFILE SUPPORT

sinh is supported in all profiles except fp20.

SEE ALSO

the acos manpage, the cos manpage, the cosh manpage, the exp manpage, the tanh manpage

NAME

tan − returns tangent of scalars and vectors.

SYNPOSIS

```
float  tan(float a);
float1 tan(float2 a);
float2 tan(float2 a);
float3 tan(float3 a);
float4 tan(float4 a);

half   tan(half a);
half1  tan(half2 a);
half2  tan(half2 a);
half3  tan(half3 a);
half4  tan(half4 a);

fixed  tan(fixed a);
fixed1 tan(fixed2 a);
fixed2 tan(fixed2 a);
fixed3 tan(fixed3 a);
fixed4 tan(fixed4 a);
```

PARAMETERS

a Vector or scalar of which to determine the tangent.

DESCRIPTION

Returns the tangent of *a* in radians.

For vectors, the returned vector contains the tangent of each element of the input vector.

REFERENCE IMPLEMENTATION

tan can be implemented in terms of the **sin** and **cos** functions like this:

```
float tan(float a) {
  float s, c;
  sincos(a, s, c);
  return s / c;
}
```

PROFILE SUPPORT

tan is fully supported in all profiles unless otherwise specified.

tan is supported via approximations of **sin** and **cos** functions (see the respective sin and cos manual pages for details) in the vs_1, vp20, and arbvp1 profiles.

tan is unsupported in the fp20 and ps_1 profiles.

SEE ALSO

the atan manpage, the atan2 manpage, the cos manpage, the dot manpage, the frac manpage, the sin manpage, the sincos manpage

NAME

tanh − returns hyperbolic tangent of scalars and vectors.

SYNPOSIS

```
float  tanh(float a);
float1 tanh(float2 a);
float2 tanh(float2 a);
float3 tanh(float3 a);
float4 tanh(float4 a);

half   tanh(half a);
half1  tanh(half2 a);
half2  tanh(half2 a);
half3  tanh(half3 a);
half4  tanh(half4 a);

fixed  tanh(fixed a);
fixed1 tanh(fixed2 a);
fixed2 tanh(fixed2 a);
fixed3 tanh(fixed3 a);
fixed4 tanh(fixed4 a);
```

PARAMETERS

a Vector or scalar of which to determine the hyperbolic tangent.

DESCRIPTION

Returns the hyperbolic tangent of *a*.

For vectors, the returned vector contains the hyperbolic tangent of each element of the input vector.

REFERENCE IMPLEMENTATION

tanh for a scalar **float** could be implemented like this.

```
float tanh(float x)
{
  float exp2x = exp(2*x);
  return (exp2x - 1) / (exp2x + 1);
}
```

PROFILE SUPPORT

tanh is supported in all profiles except fp20.

SEE ALSO

the atan manpage, the atan2 manpage, the cosh manpage, the exp manpage, the sinh manpage, the tan manpage

NAME

transpose − returns transpose matrix of a matrix

SYNPOSIS

```
float4x4 transpose(float4x4 A)
float3x4 transpose(float4x3 A)
float2x4 transpose(float4x2 A)
float1x4 transpose(float4x1 A)

float4x3 transpose(float3x4 A)
float3x3 transpose(float3x3 A)
float2x3 transpose(float3x2 A)
float1x3 transpose(float3x1 A)

float4x2 transpose(float2x4 A)
float3x2 transpose(float2x3 A)
float2x2 transpose(float2x2 A)
float1x2 transpose(float2x1 A)

float4x1 transpose(float1x4 A)
float3x1 transpose(float1x3 A)
float2x1 transpose(float1x2 A)
float1x1 transpose(float1x1 A)
```

PARAMETERS

A Matrix to tranpose.

DESCRIPTION

Returns the transpose of the matrix *A*.

REFERENCE IMPLEMENTATION

transpose for a **float4x3** matrix can be implemented like this:

```
float4x3 transpose(float3x4 A)
{
  float4x3 C;

  C[0] = A._m00_m10_m20;
  C[1] = A._m01_m11_m21;
  C[2] = A._m02_m12_m22;
  C[3] = A._m03_m13_m23;

  return C;
}
```

PROFILE SUPPORT

transpose is supported in all profiles.

SEE ALSO

the determinant manpage, the mul manpage