# 3Delight User's Manual
# Version 1.0

# Short Contents

# Table of Contents

# 1 Welcome to 3Delight!

3Delight is a fast, high quality, RenderMan compliant renderer. 3Delight comes as a set of command line tools and libraries intended to help you render production quality images from 3D scene descriptions. At the heart of the rendering tools is the 3Delight rendering engine. This engine implements a fast scanline renderer, coupled with an on-demand ray-tracer. This combination allows 3Delight to render images quickly, yet giving you the ability to easily incorporate ray-tracing effects whenever you need them.

3Delight was first released publicly in August 2000. Since then, numerous releases, both public and internal, have contributed to make it a stable software with strong features. In the process, special care has been taken to make sure 3Delight keeps a performance edge.

Check regularly at http://www.3delight.com for the latest release of 3Delight.

## 1.1 What Is In This Manual ?

The remainder of this chapter will give an overview of the main features of the 3Delight rendering tools.

RenderMan compliance and 3Delight specific extensions are discussed in Chapter 4 [3Delight and RenderMan], page 20. The complete list of provided tools and libraries, along with operational details, is given in Chapter 3 [Using 3Delight], page 7. Shading Language (SL) support and details about shader compilation are given in Chapter 6 [Using shaders], page 55. Rendering tips and useful suggestions can be found in Chapter 5 [Rendering Guidelines], page 50. 3Delight's extensible display driver interface and descriptions of all provided display drivers can be found in Chapter 7 [Display Driver System], page 62.

The installation procedure and insights about 3Delight's working environment are given in Chapter 2 [Installation], page 4.

All bug reports are weclome! We will try to fix all bugs for the upcoming release of 3Delight. When sending a bug report, we would appreciate if you could provide:

**System Identification**
: This means the platform and the version of 3Delight you are using. If you do not have the latest version, please download it from http://www.3delight.com.

**The RIB**
: If you do not have a RIB file (because you use the C API), you still can output a RIB using the DL_RIB_OUTPUT environment variable. See Section 2.4 [Environment Variables], page 5.

**The shaders**
: Send them if you think they have something to do with the problem;

**The image**
: If the problem is a glitch in the image, please attach it with the email. Do not use heavy jpeg compression;

Everything should be sent to info@3delight.com. All material you send us will be kept confidential and destroyed afterward.

## 1.2 Features

Here is an overview of 3Delight features:

*RenderMan Compliant*
: The `renderdl` program can render any RenderMan Interface Bytestream (RIB) file (binary or text formats) or an application can link with the `lib3delight` library and directly use the RenderMan Application Programming Interface (API), refer to User's Manual Chapter 4 [3Delight and RenderMan], page 20 for details.

*RenderMan Shading Language Support*
: Programmable shading and lighting with an optimizing shader compiler. RenderMan shaders are fully supported (surface, displacement, light, volume and imager). Matrices, arrays, normals, vectors and all the standard shadeops are supported. DSO shadeops, message passing and output variables are also supported. Shaders can be either compiled or interpreted. See User's Manual Section 3.2 [Using the shader compiler], page 9.

*Rich Rendering Features*
: Depth of field, motion blur and surface displacement. Standard and deep shadow maps, as well as ray-traced shadows. Selective ray-tracing and global illumination. Atmospheric effects.

*Textures and Antialiasing*
: High quality filtered textures and selectable antialias filters including the quality `sinc` and `catmull-rom` filters;

*Rich Geometry Support*
: Subdivision Surfaces (catmull-clark), Polygons, patches (B-spline, Bezier, Catmull-Rom and others), NURBS (with trim-curves), Curves (Fur & Hair), quadrics, blobbies and procedural geometry. User defined variables, including vertex variables, attached to geometry are fully supported.

*Fast and Efficient Rendering*
: 3Delight can handle complex scenes, made of millions of primitives. From its initial design stage, rendering speed has been a TOP PRIORITY and it continues to be in its ongoing developpement.

*Extensible Display Drivers*
: 3Delight comes with the following display drivers: '`framebuffer`', '`tiff`', '`bmp`', '`zfile`', '`shadowmap`', '`dsm`', '`cineon`', '`radiance`', '`exr`' and '`eps`'. Since 3Delight's uses the "standard" RenderMan display driver interface, third parties display drivers are also supported. New extensions to display system are also supported, including multiple displays per render and display specific quantize parameters. See User's Manual Section 4.1 [options], page 20.

*Multi-platform Support with Specific Code Optimisation*
: 3Delight is available for Windows (Intel and AMD), Linux (Intel, AMD and PowerPC with AltiVec support), MacOS X and IRIX (MIPS4).

# 2 Installation

## 2.1 MacOS X

An auto-installing package is provided for MacOS X users.
Administrator privileges are needed to install 3Delight because everything will be copied to '`/Applications/Graphics/3Delight-1.0.6/`', which is not accessible to all users. The installation program will ask you for the appropriate password.

During the installation procedure, 3Delight will add a few lines to your '`~/.tcshrc`' file, the original file will be saved to '`~/.tcshrc.3delight.bck`'

To test your installation perform the following easy steps:

1. Open a Terminal and type the following commands:

   ```
   cd $$DELIGHT/examples/rtshadows
   renderdl shadtest.rib
   ```

2. A file named '`shadtest.tif`' should appear in the current directory. You can view it with any image viewer.

Other examples are available in the '`examples`' folder, we suggest you to run them.

If you have problems, write to info@3delight.com.

## 2.2 UNIX

For UNIX systems (IRIX, Linux), we provide an installation script that should do everything for you. Here is an example of an installation session (the 3Delight `.gz` file has been downloaded into the '`~/downloads`' directory and we will install it into the '`~/software`' directory).

```
% cd ~/downloads
% gunzip 3delight-1.0.6-Linux-i686-libstdc++-3.tar.gz
% tar xf 3delight-1.0.6-Linux-i686-libstdc++-3.tar
% cd 3delight-1.0.6-Linux-i686-libstdc++-3
% ./install --prefix ~/software/
% cd ~/software/3delight-1.0.6/
% unsetenv DELIGHT
% source .3delight_csh
% cd $DELIGHT/examples/opacity/
% shaderdl fonky.sl
% renderdl cubits.rib
% exit
```

After typing those commands, '`cubits.tiff`' should appear in your directory.

Note that if you do not specify the '`--prefix`' option to `install`, 3Delight will be installed in '`/usr/local`'[1].

Finally, you should add the following line to your '`.login`' file:

```
source ~/software/3delight-1.0.6/.3delight_csh
```

If you use `bash`, then you should add:

```
source ~/software/3delight-1.0.6/.3delight_bash
```

## 2.3 Windows

On Windows systems, simply run `setup-1.0.6.exe`. On some systems (namely Windows 2000), you'll have to open the Environment Variables dialog (found in "Start"->"Settings"->"Control Panel"->"System"->"Advanced"->"Environment Variables" tab), and hit "apply". This will set 3Delight's environment variables properly.

## 2.4 Environment Variables

If your installation succeeded, you'll only need to modify `DL_SHADERS_PATH` and `DL_TEXTURES_PATH` variables to suit your needs. If for some reason 3Delight does not run properly, check if the variables described below are set to correct values.

You may also contact us at info@3delight.com if you experience difficulties.

`DELIGHT` This variable must point to the root of your 3Delight installation.

`DL_SHADERS_PATH`
: This variable contains search paths for 3Delight shaders. It must contains a colon (or semi-colon on Windows systems) separated list of directories. A default value is '`.:$DELIGHT/shaders`' ('`$DELIGHT/shaders;.`' in Windows).

`DL_TEXTURES_PATH`
: Contains search paths for the textures used in the scenes rendered by 3Delight. The default is '`.`' (current directory).

`DL_DISPLAYS_PATH`
: Contains search paths for 3Delight display drivers. The default is '`$DELIGHT/displays`'.

`DL_ARCHIVES_PATH`
: Contains search paths for RIB archives. Used when loading RIBs using `ReadArchive` or `DelayedReadArchive`.

`DL_PROCEDURALS_PATH`
: Contains search paths for DSOs used by `DynamicLoad` as well as for programs used by `RunProgram`.

`DL_RESOURCE_PATH`
: Setting this environment variable is equivalent to set each of the '`shader`', '`display`', '`texture`' and '`archive`' paths individually.

[1] "root" persmissios are required when installing in '`/usr/local`'.

`DL_RIB_OUTPUT`
: This variable may contain the name of an output RIB file. You do not need to define this variable to use 3Delight. However, it may be useful if you want 3Delight to output the scene it renders. This variable is mainly used for debugging purposes.

`INFOPATH`
: If you use `info` to view the online documentation, add '`$DELIGHT/doc/info`' to this variable.

`PATH`
: You may also want to make the 3Delight executables accessibles by adding '`$DELIGHT/bin`' to the this environment variable.

`LD_LIBRARYN32_PATH`
: If you are installing 3Delight on an IRIX system and you decided to configure your environment by hand, you'll have to make the '`lib3delight.so`' dynamic library visible to the program loader. One way to do this is to add '`$DELIGHT/lib`' to the `LD_LIBRARYN32_PATH` environment variable.

`LD_LIBRARY_PATH`
: If you are instsalling 3Delight on a Linux sytem and you decided to configure your environment by hand, you have to make the '`lib3delight.so`' dynamic library visible to the program loader. One way to do this is to add '`$DELIGHT/lib`' to the `LD_LIBRARY_PATH` environment variable.

# 3 Using 3Delight

3Delights is a small collection of tools and a library. Here is an overview:

- `renderdl`, a RIB file reader. This programs reads a binary or ASCII-encoded RIB file and calls the renderer to produce an image.
- `shaderdl`, a shader compiler. The compiler can produce either object-code shaders (a DSO or a DLL) or byte-code shaders (commonly called interpreted shader).
- `tdlmake`, a texture optimizer which reads a variaty of inpout formats and produces a TIFF optimized for the renderer. The optimized file typically has a '`.tdl`' extension.
- `shaderinfo`, an utility used to gather information from compiled shaders, including: shader type, shader's parameters and parameters' default values.
- `dsm2tif`, an utility to convert deep shadow maps into TIFFs;
- `hdri2tif`, a dynamic range compression utility. Converts HDR images to low dynamic range TIFFs.
- `lib3delight`, a library that can be linked to other applications to render images and interrogate shaders. **The library complies to the RenderMan** API.
- 3Delighter, a very neat interface to 3Delight for MacOS X users that do not whish to play in an UNIX shell.

In the next sections, we will describe each of these tools in more details.

## 3.1 Using the RIB Renderer - `renderdl`

`renderdl` reads a file containing scene description commands and "executes" them. Such files are commonly called RIB files (RIB stands for RenderMan Interface Bytestream). There is two kinds of RIB files: ASCII encoded RIB files and binary encoded RIB files. A binary RIB file is smaller than its ASCII encoded equivalent, but an ASCII RIB file has the advantage of being editable in any text editor or word processor.

To render a RIB named '`file.rib`', just type:

```
% renderdl file.rib
```

It is possible to render more than one file:

```
% renderdl file1.rib file2.rib file3.rib
```

In this case, `renderdl` reads each file one after the other, and the graphic state is retained from one file to another (in other words, the graphic state at the end of one file is the starting graphic state for the next file). If a file cannot be found, it is simply skipped. This behaviour is useful to separate the actual scene description from rendering options. For example:

```
% renderdl fast.opt scene.rib
% renderdl slow.opt scene.rib
```

These will render the scene '`scene.rib`' twice but with different rendering optionts (note that '`fast.opt`' and '`slow.opt`' are normal RIB files). '`file.opt`' contains options for high quality rendering such as low `ShadingRate` and high `PixelSamples`, and '`slow.opt`' contains low quality (speedy) option settings.

If you do not specify a file name, `renderdl` will wait for you to enter scene description commands. This feature enables you to pipe commands directly in `renderdl`. For example, to enter scene description commands interactively (which is not really practical), do the following:

```
% renderdl
Reading (stdin)
<enter commands here>
```

If you wish to pipe the content of '`file.rib`' in `renderdl`, type:

```
% cat file.rib | renderdl
```

`renderdl` options are described in the following sub-section.

### 3.1.1 Command Line Options

'`-v`' Prints 3Delight's version number and name;

'`-h`' Prints a short help screen;

'`-beep`' Beep when all RIBs are rendered;

'`-beeps`' Beep after *each* rendered RIBs;

'`-d`' Will force a display to the '`framebuffer`' display driver. Note that a '`framebuffer`' display driver will be added to the displays declared inside the RIB so those will still be called. If there is already a '`framebuffer`' display driver declared in the RIB then this options has no effect;

'`-D`' Has the same effect as '`-d`' but will automatically close the framebuffer display driver when finished;

'`-frames f1 f2`'
Only render the frames between f1 and f2, inclusivly. This options enables you to render some specific frames inside one RIB file. Frames outside the specified interval are skipped;

'`-crop l r t b`'
Sets a crop window defined by <l r t b> (left right top bottom). The values should be given in screen coordinates, which means that all value are between 0.0 and 1.0 inclusively. This command line option will override any `CropWindow` command present in the RIB file.

'`-noinit`' Do not read the '`.renderdl`' file. See Section 3.1.2 [the .renderdl file], page 9.

'`-stats`' Prints "end of frame statistics". Has the same effect as putting the following line in the RIB file: `Option "statistics" "endofframe" 3`

'`-progress`'
Prints a progress status after each bucket. Can also be enabled from inside the RIB. See [progress option], page 25.

### 3.1.2 The '.renderdl' File

When started, renderdl immediatly looks for an initialization file named '.renderdl'. This file is a normal RIB that may contain any standard RIB command, enabling the user to put whatever default options are needed for rendering, such as variable declarations, standard screen format, performance options, etc. . . .

The locations in which 3Delight will look for this file are (in order):

1. In the current working directory;
2. In user's home directory[1];
3. In the directory pointed to by the DELIGHT environment variable.

The loading of '.renderdl' can be bypassed using the '-noinit' option (see Section 3.1.1 [renderdl options], page 8).

## 3.2 Using the Shader Compiler - shaderdl

3Delight shaders are written in RenderMan Shading Language (SL). 3Delight also supports most of the common extensions to this language.

The shader compiler can produce either an object-code shader (a DSO on IRIX and Linux, a DLL on Windows) or a byte-code shader (commonly called interpreted shader). By default, the compiler produces byte-code shaders. If you have a C++ compiler installed on your computer, you can ask shaderdl to produce object-code shaders by setting the '--dso' command line switch.

The main advantage of byte-code (interpreted) shaders is that they are platform independent: they can be used on IRIX, MacOSX, Linux or Windows computers. This is not the case of object-code shaders. Object-code shaders, however, can be faster than byte-code shaders. We suggest you to produce object-code for those shaders that are computationally intensive, such as volume shaders.

To compile a shader, give the command:

```
% shaderdl shader.sl
```

As you may have noticed, you do not have to specify an output file name to shaderdl. The output file name is the name of the shader (the identifier following the keyword surface, displacement, light, or volume in the SL program) followed by the suffix '.sdl'.

Some renderers' shader compiler append the name of the platform at the end of object-code shader's name. 3Delight shader compiler does not use this convention. An object-code shader will always have the same name, regardless of the platform for which it is compiled. We recommend that you keep shaders compiled for different platforms in separate directories (this of course is not necessary for interpreted shaders).

The shaderdl program is a script that call a suite of compilation tools. We explain the compilation process in the next sub-section. Then, we list the command line options accepted by shaderdl and explain how you can use them to modify the behaviour of shaderdl. Finally, we explain how you can customize shaderdl to fit some particuliar needs.

[1] As specified by the HOME environment variable.

### 3.2.1 Compilation Process

This section details the steps that occur when a shader is compiled. The compilation process is started by the `shaderdl` script. This script defines some variables and then gives control to `slcdl`, which is responsible for calling other tools.

The first step is the preprocessing step. 3Delight has its own preprocessor (`cppdl`) which does a work very similar to a standard C preprocessor. The preprocessor takes a SL program with #-directives, processes the #-directives and outputs a SL program without #-directives.

Then comes the shading language translation step. If you elected to output byte-code shaders, `slcdl` parses the SL file generated by the preprocessor and outputs byte-code in a text file. The file name is the name of the shader found in the SL file, followed by the '`.sdl`' extension.

If elected to output object-code shaders, `slcdl` parse the SL file and output a C++ program, instead of byte-code. This C++ program is then compiled by a C++ compiler (that you must provide) and linked into a dynamic shared object (DSO) or a dynamic link library (DLL). The DSO or DLL file name is the name of the shader found in the SL file, followed by the '`.sdl`' extension.

When using a shader, there is no need to worry about whether the shader is in byte-code or object-code. 3Delight will guess this by itself.

### 3.2.2 Command Line Options

As every compiler, `shaderdl` understends a set of command-line options that control the compilation process, those are specified before the input file name:

```
% shaderdl [options] shader.sl
```

There is no need to specify an output file name to `shaderdl` since it is automatically set to the name of the shader (the identifier following the keyword `surface`, `displacement`, `light`, `volume` or `imager` in the SL code) followed by the suffix '`.sdl`'.

Valid command line options are:

'`-d <directory>`'
: Specifies destination directory for compiled shaders. The default is the current working directory.

'`--dso`'
: Generate object-code shaders. To use this option, you must have a C++ compiler installed. The compilation script included with 3Delight is configured to work with `CC` on IRIX, with `g++` on Linux and with `Visual C++` on Windows. If you have a different compiler installed, you will have to modify the compilation script. For more information, see Section 3.2.3 [Customizing the compile script], page 11.

'`--int`'
: Generate byte-code shaders (default).

'`-O<n>`'
: Specifies the optimisation level, from 0 to 3. '`-O0`' turns off all optimisations, '`-O1`' optimises a little bit, '`-O3`' optimises aggressively. The default is '`-O3`'.

'`-w<n>`'
: Sepcifies warning level:

0. Disable all warnings (not recommanded);
1. Log important warnings only (default);
2. Log all warnings.

'`-I<directory>`'
Specifies a directory to search for `#include`'d files.

'`-D<symbol>`'
Defines a preprocessor symbol.

'`-E`' Stops after the preprocessing step. See Section 3.2.1 [Compilation process], page 10.

'`-c`' Stops after the shading language to C++ translation pass. See Section 3.2.1 [Compilation process], page 10.

'`--keep-cpp-file`'
'`--dont-keep-cpp-file`'
Specifies whether or not to keep the intermediate files generated by the preprocessor. They are not kept by default. See Section 3.2.1 [Compilation process], page 10.

'`--keep-c++-file`'
'`--dont-keep-c++-file`'
Specifies whether or not to keep the C++ files generated by the shading language translation pass. They are kept by default. See Section 3.2.1 [Compilation process], page 10.

'`--no-array-check`'
Turns off array run time bound checking. Enabled by defaut.

'`--use-shadeops`'
'`--dont-use-shadeops`'
Enable [disable] use of shadeops. Disabled by default.

'`-v`'

'`--version`'
Print the compiler version number and exit.

'`-h`'

'`--help`' Print a help message and exit.

### 3.2.3 Customizing the Compilation Script

The `shaderdl` script sets some environment variables and then calls `slcdl`. This program uses those environment variables to find the C compiler and call it with the right options.

You can change the default settings of `shaderdl` by modifiying the configuration script '`shaderdl.init`' ('`shaderdl.init.bat`' under Windows), which should be placed in your '`$HOME/etc`' directory.

An example '`shaderdl.init`' would be

```
#! /bin/sh

SLC_CC_OPTIONS="-w -g"
```

This configuration file simply tells the C++ compiler to generate debug information in the compiled DSO (or DLL).

Under Windows, the default compiler is Microsoft Visual C++. However, we provides alternate `shaderdl` scripts for use with other popular compilers, like gcc or Borland. To use them, just rename the one that you need to '`shaderdl.bat`'.

## 3.3 Using the Texture Optimizer - `tdlmake`

`tdlmake` preprocesses TIFF, JPEG, RADIANCE and OpenEXR[2] files in order to save them into an efficient texture format suitable to 3Delight. It can also convert '`zfile`'s into shadow maps. We recommend running `tdlmake` on all textures, before rendering, for two reasons:

- `tdlmake` will create a mipmapped version of the original texture, allowing 3Delight to produce nicer images;
- 3Delight employs a caching mechanism for texture data which works well with tiled images, using raw (striped) non-converted TIFFs may degrade overall performance.

Note that a converted file is a normal TIFF that can be viewed with any image viewer. We suggest using a '`.tdl`' extension for 3Delight texture files.

### 3.3.1 Command Line Options

`tdlmake` is invoqued by specifying at least two file names and an optional set of command-line switches:

```
% tdlmake [options] input.tif [input2.tif ... input6.tif] output.tif
```

Valid options are:

'`-envlatl`'
: Generate a latitude-longitude environement map;

'`-envcube`'
: Generate a cubic environment map. Needs six TIFFs in input, ordered as follow: +x, -x, +y, -y, +z, -z;

'`-shadow`'
: Generate a shadowmap from a zfile. When generating a shadowmap, only the '`-c-`' option is functional.

'`-lzw`'
: Compress output texture using LZW algorithm. This option is enabled by default since compressed textures take much less space and there is no noticeable speed penality when accessing them;

'`-deflate`'
: Compress output texture using the Deflate algorithm. Has a better compression ration than LZW;

[2] OpenEXR format is only availabe on Linux and IRIX platforms.

‘`-packbits`’
Compress output texture using Apple’s PackBits algorithm. Compression ratio is not as good as with LZW or Deflate but decompression is very fast;

‘`-c-`’ Do not compress output texture;

‘`-fov n`’ Specifies a field of view, in degrees, for cubic environment maps. Default is ‘`90`’ degrees;

‘`-mode <black|clamp|periodic>`’
‘`-smode <black|clamp|periodic>`’
‘`-tmode <black|clamp|periodic>`’
Specifies what action should be taken when accessing a texture (using `texture()`) outside its defined parametric range ($s$, $t = [0..1]$):

‘`black`’ Texture is black outside its paramtric range;

‘`clamp`’ Texture’s borders extend to infinity;

‘`periodic`’
Texture is tiled infinitely;

‘`-smode`’ and ‘`-tmode`’ specifiy the wrapping modes of the texture in $s$ or $t$ only. Default mode is ‘`black`’ for normal textures and ‘`periodic`’ for latitude-longitude environment maps. Note that this option does not affect the look of the ‘`.tdl`’, its effects will only be noticeable when using `texture()` from inside a shader.

‘`-filter <box|triangle|gaussian|catmull-rom|bessel|sinc>`’
Specifies a downsampling filter to use when creating mipmap levels. The default filter is sinc. Here is a table showing the complete list of supported filters, along with their ‘`filterwidth`’ and window defaults:

| filter | filterwidth | window | Comment |
|---|---|---|---|
| **box** | 1.0 | – | This filter tends to blur textures, use only if texture generation speed is an issue; |
| **triangle** | 2.0 | – | filterwidth larger than 2.0 is unnecessary; |
| **gaussian** | 2.50 | – | A good filter that might produce slightly blurry results, not as much as the box filter though; |
| **catmull-rom** | 4.0 | – | A better filter (producing sharper textures); |
| **bessel** | 6.47660 | '`lanczos`' | Filter width choosen as to include 2 roots; |
| **sinc** | 8.0 | '`lanczos`' | We recommend this high quality filter as the best choice. Although it can produce some ringing artifacts on some textures. Using a filterwidth smaller than 4 is not recommended. |

'`-window <lanczos|hamming|hann|blackman>`'
: A windowing function can be applied to bessel and sync filters (which are infinite in width) to achieve a softer cut at the filter's support boundaries. Possible windowing schemes are '`lanczos`', '`hamming`', '`hann`' and '`blackman`'.

'`-filterwidth n`'
: Overrides the default filter width in $s$ and $t$. **Important** : filter width is the *diameter* of the filter and *not* the radius;

'`-sfilterwidth n`'
'`-tfilterwidth n`'
: Overrides the default filter width in $s$ or $t$;

'`-blur n`'
: Blurs or sharpens the output image. Values larger than one will make the image more blurry and values smaller than one will produce sharper results. This function works by scaling the filter function by the specified value. This is *not* the same thing as scaling '`sfilterwidth`' and '`tfilterwidth`'. Default is '`1`'.

'`-quality <low|medium|high>`'
: Controls mipmap downsampling strategy: when using '`low`', each mipmap level is created from the previous one. At '`medium`' quality, each level is created from the 2nd previous level. At '`high`' quality, each level is created from up to the

4th previous level. The default '`medium`' setting is more than enough for most applications.

'`-flips`'
'`-flipt`' Flip the image horizontally or vertically;

'`-flipst`' Flip the image in both directions;

'`-progress`'
Shows texture creation progress, only useful for really large textures (or really slow computers!);

'`-v`' Prints version and copyright informations;

'`-h`' Shows help text.

### 3.3.2 Supported Input Formats

`tdlmake` supports most of the common formats used in a production environment:

TIFF `tdlmake` supports stripped or tiled TIFFs with 1 to 6 channels. Suppported data types are:

- 1, 4, 8, 16 or 32 bits of unsigned integer data;
- 8, 16 or 32 bits of signed integer data;
- 32 bits of floating point data.

Unsigned single channel images (grayscale or b&w) can have either MINISBLACK or MINISWHITE photometric. Either way, the output file ('`.tdl`') will be MINISBLACK. Signed files are required to have either RGB or MINISBLACK photometric. Files with less than 8 bits per sample are promoted to 8 bits per sample in the output. Otherwise, the output format is the same as the input. `LogLuv` encoded TIFFs are supported and are kept in their `LogLuv` form when processed. TIFFs with separate planes are not supported.

JPEG `tdlmake` supports 8 bit (grayscale) and 24 bit JPEGs.

*Radiance Picture Format*
Files produced by Radiance[3] are supported. Both XYZE and RGBE encoding modes are recognized by `tdlmake`. Run-lenght encoded files are also supported. Note that there is a restriction on image orientation: only -Y +X orientation is correctly recognized, any other orientation will be reverted to -Y +X. This is not a major problem though because it can be corrected by using the '`-flips`' and '`-flipt`' options (see Section 3.3.1 [tdlmake options], page 12).

*OpenEXR* ILM's OpenEXR format is supported by `tdlmake` with the restriction that images should be in RGB or RGBA format[4].

*zfile* zfiles are only used to produce shadow maps.

For cube environment maps ('`-envcube`'), all six images are required to be of the same file format.

[3] Refer to http://floyd.lbl.gov/radiance/.
[4] Refer to http://www.openexr.org.

### 3.3.3 Quality and Performance

As all other 3Delight tools, `tdlmake` was designed to accomodate the tough demands of a production environment. In this case, special attention was given to filtering quality and memory usage, without penalising execution speed. Among other things:

- All internal filtering algorithms are executed in floating point to preserve as much accuracy as possible;
- An efficient caching system is used to keep memory usage footprint *very* low. `tdlmake` was tested on textures as large as 16K x 16K (1 Gigabyte of disk space) and accomplished its work by using less than 12Megs of memory.
- The filtering process has been designed to have a linear cost increase relative to '`-filterwidth`', unlike some software in which the cost is quadratic. For example, '`-filterwidth 8`' will run twice as slow as '`-filterwidth 4`'.

### 3.3.4 Examples

Here are some examples using `tdlmake` on the command line:

To create a 3Delight texture named '`grid.tdl`' from a TIFF named '`grid.tif`' using a gaussian downsampling filter of width 4:

```
% tdlmake -filter gaussian -filterwidth 4 grid.tif grid.tdl
```

To create a cubic environment map in which all cube sides were rendered using 90 degrees field of view:

```
% tdlmake -fov 90 -envcube \
  in1.tif in2.tif in3.tif in4.tif in5.tif in6.tif \
  envmap.tdl
```

or (won't work in a DOS shell) :

```
% tdlmake -fov 90 -envcube in?.tif envmap.tdl
```

To create a texture using the high quality downsampling mode and show progress while doing so:

```
% tdlmake -progress -quality high grid.tif grid.tdl
```

To create a shadow map from a zfile (Section 7.3 [dspyzfile], page 64):

```
% tdlmake -shadow data.z shadowmap.tdl
```

## 3.4 Using `dsm2tif` to Visualize DSMs

`dsm2tif` is an utility to convert 3Delight's propriatary deep shadow map files into viewable TIFF files. A DSM contains visibility informations for any given depth in a scene, so one single 2D image (eg. a TIFF) cannot describe all the information provided by a DSM; that is why command line parameters are provided to specify at which depth the evaluation will occur.

'`-z depth`' Specifies a relative depth at which the deep shadow map will be evaluated. "Relative" meaning that each pixel in the image will be evaluated at its own

depth, computed as follows: $(pZmax - pZmin) * depth + pZmin$. $pZmin$ and $pZmax$ being the depths of the closest and the furtherst features present in the **pixel**. '`-z 1`' is the default, which shows the amount of light that passes through to infinity at each pixel;

'`-Z depth`' Specifies an absolute depth (range: [0..1]) to evaluate the deep shadow map. "Absolute" meaning that all pixels in the output image will be evaluated at the same depth, computed as follows: $(Zmax - Zmin) * depth + Zmin$. $Zmin$ and $Zmax$ being the depths of the closest and the furtherst features in the DSM;

'`-bias n`' Specifies a shadow bias for deep shadow map evaluation. This is needed to avoid self-occlusion problems. If the produced TIFF contains noisy areas, consider increasing this parameter. Default is 0.015;

'`-opacity`'
By default, `dsm2tif` will assign a "visibility" value to each pixel in the output TIFF, specifying this option will assign "opacity" values to each pixel ($opacity = 1 - visiblity$). This means that dark areas in the image indicate that light passes through unnocluded;

'`-mipmap n`'
This options specifies which mipmap level will be converted. Note that mipmapping can be diseabled by an option given to the display driver (see Section 7.5 [dspy_dsm], page 64);

'`-8`'
'`-16`'
'`-32`'
'`-float`' Choose data format for the output TIFF. Default is 8 bits per channel;

'`-lzw`'
'`-deflate`'
'`-packbits`'
'`-logluv`' Selects a compression scheme. Default is '`lzw`'. '`-logluv`' is only supported with '`-float`' data type;

'`-v`' Prints version and copyright informations;

'`-h`' Shows help text.

## 3.5 Using `hdri2tif` on High Dynamic Range Images

`hdri2tif` emplois a range compression algorithm[5] to convert high dynamic range images into displayable, low dynamic range, TIFFs. All the important HDRI formats are recognized in input:

1. Floating point TIFFs as well as LogLuv encoded TIFFs;
2. Radiance files, uncompressed or RLE packed;

[5] *Erik Reinhard, Mike Stark, Peter Shirley and Jim Ferwerda, 'Photographic Tone Reproduction for Digital Images', ACM Transactions on Graphics, 21(3), pp 267–276, July 2002 (Proceedings of SIGGRAPH 2002).*

3. ILM's OpenEXR files.

Note that all formats must have 3 or 4 channels. The fourth channel will be considered as alpha information and will stay untouched in the output image.

`hdri2tif` accepts the following options:

'`-middle-gray n`'
'`-key n`' Sets the value of the middle-gray in the image, default is 0.18. Range is [0..1]. Specifying a value of 0 will enable an histogram based automatic estimation (one can use '`-verbose`' to display the computed key value);

'`-simple`' By default, `hdri2tif` uses a "dodging-and-burning" algorithm which is relatively costly[6]. Specifying this option will force the use of a simpler algorithm that is well suited for images with a medium dynamic range (lower than 10 "zones"). Note that this method may produce bad results on very high dynamic range images;

'`-white n`' Sets the luminance value in the image that will be mapped to pure white (1.0 in the low dynamic range output image). Setting this parameter to smaller values will produce more "burning". The default action is to set the white point to the maximum luminance in the image, which removes burning. This option is only meaningful when paired with the '`-simple`' option because the default algorithm automatically computes the white point. Specifying a value of 0 will enable an histogram based automatic estimation (one can use '`-verbose`' to display the computed value);

'`-gamma n`' Specifies a gamma correction to be applied on the produced image. Gamma correction is performed *after* the range compression algorithm.

'`-nthreads n`'
`hdri2tif` can run in a multi-threaded mode for increased performance. This options specifies how many threads to use. This option has *no* effect when '`-simple-operator`' is used.

'`-verbose`'
Displays information while processing the image;

'`-help`' Displays a brief help text.

The following options can be used to further control the "dodging-and-burning" algorithm. Modifying these options is not recommanded.

'`-numscales n`'
Specifies the number of gaussian convolutions to use when computing luminance information for each pixel. More gaussians mean a more precise result but also a slower computation. This option is only meaningfull with the "dodging-and-burning" algorithm (no '`-simple`'). Default is 8;

'`-sharpness n`'
Sets the sharpness parameter. Higher values mean sharper images. Default is 8;

[6] Implies frequency domain computations.

‘-lzw’
‘-deflate’
‘-packbits’
‘-none’ Selects compression method to use for output TIFF. ‘-lzw’ is enabled by default.

## 3.6 Using shaderinfo to Interrogate Shaders

shaderinfo is a utility to interrogate compiled shader. This can prove usefull when you do not have the shader's source code and you want to know what are its parameters.

To get information about a shader, use shaderinfo like this:

```
% shaderinfo matte
```

shaderinfo generate the following output:

```
surface "matte"
    "Ka" "uniform float"
                Default value: 1
    "Kd" "uniform float"
                Default value: 1
```

To find a shader, shaderinfo will use the search paths specified by the DL_SHADERS_PATH (or DL_RESOURCE_PATH) environment variable.

shaderinfo has a special switch that helps declaring parameter types in a RIB file:

```
% shaderinfo -d matte
```

will generate the following output:

```
surface "matte"
Declare "Ka" "uniform float"
Declare "Kd" "uniform float"
```

Declarations can be pasted directly into the RIB.

## 3.7 Using the 3Delight GUI (Mac only)

3Delight includes a simple GUI for MacOS X users. This is good news for those of you who are not familiar with a UNIX shell. Shader compilation, RIB rendering and editing are among the services provided by this software.

3Delighter may be updated after a 3Delight release, we suggest you to check for the latest version at http://icoldwell.com/3delighter/.

3Delighter can be lauched from the ‘/Applications/Graphics/3Delight-1.0.6/bin’ directory.

Many thanks to Robert Coldwell for this software.

# 4 3Delight and RenderMan

There are two ways to describe a scene to 3Delight: the first is by using the RenderMan Application Programming interface (API) and the second is by using the RenderMan Interface Bytestream (RIB) files. The RenderMan API (as well as the RIB format) is well described in *RenderMan Interface Version 3.2*. This document is available electronically at https://renderman.pixar.com/products/rispec/index.htm.

A RIB file is a set of commands meant to be interpreted by a RIB reader such as `renderdl`. There is almost a one to one correspondance between the API calls and RIB commands.

For an in depth documentation about RenderMan and the RenderMan API, consult the following two classics.

- *The RenderMan Companion*. Steve UPSTILL. Addison Wesley.
- *Advanced RenderMan: Creating CGI for Motion Pictures*. Larry GRITZ and Anthony A. APODACA. Morgan Kauffman.

Check www.3delight.com/links.htm for a direct link to those references.

## 4.1 Options

Options are parameters of the graphic state that apply to the entire scene. All options are saved at each `FrameBegin` and restored at the corresponding `FrameEnd` call. It is important to set these options *before* the `WorldBegin`...`WorldEnd` block since it is a RenderMan assumption that rendering may begin any time after `WorldBegin`; all options must be fixed by then. Also, and contrary to transforms, the order in which options appear is not important.

The RenderMan API declares a standard set of options (which are used to control camera and image parameters) and opens a door to implementation specific options through the `RiOption` call. 3Delight supports all of the standard options and has some specific calls; both groups are described in the following sections.

### 4.1.1 Image and Camera Options

All the standard options are supported and are listed in the table below along with their default values. Some of those options have an extended behaviour in 3Delight:

`Display` You can open more than one display at once by giving `Display` a file name that begins with a `"+"`. For example, the following two commands will write the image to '`image.tif`' *and* to the framebuffer.

```
Display "image.tif" "file" "rgb"
Display "+window" "framebuffer" "rgb"
```

Since the renderer uses floating point numbers internally, the colors are exposed and quantized using the values specified with the `Exposure` and `Quantize` commands. However, you may want to expose and quantize values differently for each `Display`. This is made possible by specifying exposure, quantization and dithering values directly on the `Display` line. An extended `Display` command would look like:

```
Display "file" "driver" "variable"
    "exposure" [gain gamma]
    "quantize" [zero one min max]
    "dither" [amplitude]
```

Note that there are four values given to the `quantize` token, compared to three for the `Quantize` command. This extended syntax allows to specify a value for *zero* (black), whereas the `Quantize` command assumes this value to be '`0`'. Using this fourth parameter, colors are quantized using the following formula:

```
value = round(zero + value * (one - zero) + dithervalue)
value = clamp(value, min, max)
```

`Hider` `Hider` only supports the '`hidden`' hider. It takes one optional parameter, '`jitter`', to tell 3Delight what sampling pattern to use. Valid values for this parameter are '`0`' (regular sampling grid), '`1`' (jittered grid) or '`2`' (quasi Monte-Carlo sampling). For example:

```
Hider "hidden" "jitter" 0
```

tells 3Delight to sample the scene using a regular grid pattern. The default sampling pattern is the jittered grid.

`Hider "depthfilter" "`*filter name*`"`

Sets a filter type for depth values filtering. The following filters are recognized:

'`min`' The renderer will take the minimum $z$ value of all the subsamples in a given pixel;

'`max`' The renderer will take the maximum $z$ value of all the subsamples in a given pixel;

'`average`' The renderer will average all subsamples $z$ values in a given pixel;

'`midpoint`'
For each *subsample* in a pixel, the renderer takes the average $z$ value of the two closest surfaces. Depth associated with the pixel is then computed using a *min* filter of these averages. Note that midpoint filtering may slowdown your renders slightly.

| **Option** | **Default Value** | **Comments** |
|---|---|---|
| Format | 640 480 1 | – |
| FrameAspectRatio | 1.0 | Square pixels |
| ScreenWindow | -1.3333 1.3333 -1 1 | Computed using `Format` (640/480 = 1.333...) |
| CropWindow | 0 1 0 1 | No croping, the entire screen is visible; |
| Projection | "orthographic" | – |
| Clipping | 0 1e30 | – |
| PixelSamples | 2 2 | Enough for a preview. If motion-blur or depth of field is used increase those values to at least 4 4; |
| PixelFilter | "box" 2 2 | Enough for a preview. A high quality setting would be `"sinc" 6 6`; |
| Exposure | 1.0 1.0 | Gain = 1, Gamma = 1; |
| Display | "default.tif" "tiff" "rgb" | Create a RGB TIFF named '`default.tif`'. |

Standard options and their default values.

### 4.1.2 Implementation Specific Options

**Rendering Options**

`Option "shadow" "float bias0" [0.225]`
`Option "shadow" "float bias1" [0.3]`
`Option "shadow" "float bias" [0.225]`

When using *shadow maps*, a surface may exhibit self-shadowing. This problem is caused by precision errors and appears as gray spots on the surface. This problem occurs when the distance of an object to a light source computed while rendering a shadow map is slightly less than the distance computed while rendering the image. To prevent self shadowing, you can specify a bias, which is a value that will be added to the shadow map value. If '`bias0`' is different from '`bias1`', the renderer will choose a random value between them for each sample. Note that those parameters do not affect ray-traced shadows. The value of "bias0" and "bias1" should be set to '`0`' if the "midpoint" algorithm is used to compute the shadow map, see Section 4.1 [options], page 20 for more details on how to use the "midpoint" algorithm.

`Option "render" "string bucketorder" "horizontal"`

In order to save memory, the image is rendered progressivly in small group of pixels refered to as "buckets". This option specifies the rendering order of the buckets. Valid values are:

'`horizontal`'
: Left to right, top to bottom (this is the default);

'`vertical`'
: Buckets are rendered from top to bottom, left to right;

'`spiral`' In a clockwise spiral starting at the center of the image;

'`circle`' In concentric circles starting at the center of the image;

'`random`' In no particular order. Should not be used since it is not memory efficient.

**Quality vs. Performance Options**

`Option "limits" "integer bucketsize[2]" [16 16]`

This option specifies the dimension of a bucket, in pixels.

`Option "limits" "integer gridsize" [256]`[1]

During the rendering process, geometry is split into small grids of micro-polygons. Shading is performed on one grid at a time. This option specifies the maximum number of such micro-polygons per grid.

[1] Grid size set to cover approximately one bucket when `ShadingRate`=1.

`Option "limits" "integer eyesplits" [6]`[2]
: Specifies the number of times a primitive crossing the eye plane will be split before being discarded.

`Option "limits" "integer texturememory" [8192]`
: The memory needed to hold the texture for some scene may exceed the amount of physical (and logical!) memory available. To render such scenes efficiently, 3Delight uses a memory caching system to keep texture memory usage below some predefined threshold. This option specifies the amount of memory, in kilobytes, dedicated to the textures. Increasing this amount may improve texture map, shadow map and environment map access performances. Note that 3Delight also offers a texture file caching mechanism over networks, see Section 5.4 [Network Cache], page 52.

`Option "limits" "integer texturesample" [1]`
: Specifies the default oversampling value for texture/environment lookups.

`Option "shadow" "integer sample" [16]`
: Specifies the default oversampling value for shadow map and deep shadow map lookups. This only applies if the `shadow` call does not contain an oversampling value (see Section 4.5.5 [texture mapping shadeops], page 43). This option does not affect ray-traced shadows.

`Option "trace" "integer maxdepth" [4]`[3]
: This option sets the maximum recursion level for the ray tracer. Valid values are between 0 and 16, inclusively. Any value outside of this range will be interpreted as 16. A value of 0 turns off ray tracing.

**Search Paths**

A search path is a colon or semi-colon separated list of directories. The directories should be separated using a slash (‘`/`’) character. Under Windows, it is also possible to use `cygwin`’s convention: ‘`//c/dir1/...`’. When an environment variable is encountered in a path, it is replaced by its value. The environment variable should be specified using the UNIX convention (e.g. `$HOME`). The tilde (‘`~`’) character, when placed at the beginning of a path, has the same signification as the `HOME` environment variable. If the path specified path is ‘`@`’, it is replaced by the default path (see below); if a ‘`&`’ is specified, it is replaced by the previous path list.

Search paths are specified using the following commands:

`Option "searchpath" "string shader" "`*`path-list`*`"`
: Sets ‘`shader`’ and DSO shadeops search path;

[2] See *Advanced RenderMan: Creating CGI for Motion Pictures*, p.151, for a complete discussion about eye splits.

[3] Was `Option "render" "max_raylevel"` in versions prior to 1.0 .

`Option "searchpath" "string` **texture**`" "`*path-list*`"`
Sets '`texture`' search path;

`Option "searchpath" "string` **display**`" "`*path-list*`"`
Sets '`display`' search path;

`Option "searchpath" "string` **archive**`" "`*path-list*`"`
Sets '`archive`' search path.

`Option "searchpath" "string` **procedural**`" "`*path-list*`"`
Sets '`procedural`' search path.

`Option "searchpath" "string` **resource**`" "`*path-list*`"`
Sets '`resource`' search path. This is equivalent to adding this path to all the above search paths.

The default values for these options are taken from the environment variables `DL_SHADERS_PATH`, `DL_TEXTURES_PATH`, `DL_DISPLAYS_PATH`, `DL_ARCHIVES_PATH` and `DL_RESOURCE_PATH` respectively. Default values are shown in the table below.

**Statistics**

Statistics can prove useful when finetuning renderer's behaviour (for quality or performance reason).

`Option "statistics" "integer endofframe" [0]`
Specifies the desired statistics level. The level ranges from 0 to 3, 0 meaning no statistics at all and 3 meaning all possible statistics. A level of '2' is enough for most usages.

`Option "statistics" "string filename" [""]`
By default, statistics are dumped to the console (more specifically, to '`stdout`'); this option can be used to specify an output file name instead.

`Option "statistics" "string progress" ["off"]`
Turning this option '`on`' will force 3Delight to output a progress status[4] to the console. The outputted text will look like this:

```
3DL INFO: Progress: 0.00 %
3DL INFO: Progress: 0.08 %
3DL INFO: Progress: 0.17 %
3DL INFO: Progress: 0.25 %
3DL INFO: Progress: 0.33 %
3DL INFO: Progress: 0.42 %
3DL INFO: Progress: 0.50 %
....
```

Each line is printed after a bucket is done rendering.

**Network Cache**

3Delight has a network caching system for texture files located on "slow access" media (such as NFS and CD-ROM drives). Refer to Section 5.4 [Network Cache], page 52 for a detailed description.

[4] Approximate, since it is computed as follow : $\frac{numberofbucketsrendered}{totalnumberofbuckets}$.

`Option "netcache" "string cachedir" [""]`

Specifies a directory for data caching. The directory should be locally mounted, such as '`/tmp/3delight_cache`' and should be dedicated solely to 3Delight. Do not use '`/tmp`' as a cache directory ('`/tmp/3delight_cache/`' is a better choice).

`Option "netcache" "integer cachesize" [1000]`

Specifies cache size in megabytes. One megabyte being 1024 Kbytes. Make sure the specified size is enough to hold many textures if you want to fully take advantage of this performance feature. Idealy big enough to hold all the textures used in a given scene.

| Option | Default Value | Comments |
|---|---|---|
| "shadow" "bias0" | 0.225 | – |
| "shadow" "bias1" | 0.300 | – |
| "render" "bucketorder" | ["horizontal"] | – |
| "limits" "bucketsize" | [16 16] | – |
| "limits" "gridsize" | 256 | One grid is approximatly as large as a bucket when `ShadingRate`=1; |
| "limits" "eyesplits" | 6 | Enough for most cases, larger values may slow down rendering; |
| "limits" "texturememory" | 8192 | 8 megabytes; |
| "limits" "texturesample" | 1 | – |
| "shadow" "sample" | 16 | – |
| "trace" "maxdepth" | 4 | – |
| "searchpath" "shader" | '`$DELIGHT/shaders`' | – |
| "searchpath" "texture" | '`$DELIGHT/textures`' | – |
| "searchpath" "display" | '`$DELIGHT/displays`' | – |
| "searchpath" "archive" | '`$DELIGHT/archive`' | – |
| "searchpath" "procedural" | "" | – |
| "searchpath" "resource" | "" | – |
| "netcache" "cachedir" | [""] | No caching; |
| "netcache" "cachesize" | 1000 | 1 Gig of disk space. Has no effect if caching is disabled; |
| "statistics" "endofframe" | 0 | No stats; |
| "statistics" "file" | "" | Output to console ('`stdout`'). |

Implementation specific options and their default values

## 4.2 Attributes

Attributes are components of the graphic state that are associated with elements of the scene, such as geometric primitives and light sources. As with options, there are two kinds of attributes: the standard and the implementation specific. Standard attributes, along with their default values, are listed in the table below. Implementation specific attributes are passed through the `Attribute` command. All attributes are saved at `AttributeBegin` and restored at `AttributeEnd`.

| **Attribute Name** | **Default values** | **Comments** |
|---|---|---|
| Color | [1 1 1] | White; |
| Opacity | [1 1 1] | Opaque; |
| TextureCoordinates | [0 0 1 0 0 1 1 1] | Only applies to paramtric surfaces such as `Patch` and `NuPatch`; |
| Surface | '`defaultsurface`' | A surface shader that does not need a lightsource to produce visible results; |
| Atmosphere | – | – |
| Displacement | – | – |
| LightSource | – | – |
| ShadingRate | 1 | Approximatly one shading computation at every pixel; |
| GeometricApproximation "motionfactor" | 1 | ShadingRate is highered by one for each 16 pixels of motion, this also affects depth of field; |
| Matte | 0 | – |
| Sides | 2 | Objects visible from both sides; |
| Orientation | "outside" | Do not inverse transform handedness; |
| TrimCurve | – | No trim curves defined. |

Standard attributes and their default values.

3Delight has a set of implementation specific attributes, they are listed below in their respective category.

**Primitives Identifiaction**

`Attribute "identifier" "name" "none"`
: Subsequent objects will be named using the string given by this attribute. This is useful when 3Delight reports warning or errors about some primitive.

### Primitives Visiblity and Ray Tracing

`Attribute "visibility" "integer camera" [1]`
`Attribute "visibility" "integer trace" [0]`[1]
`Attribute "visibility" "string transmission" ["transparent"]`[2]
Specifies to wich specific ray an object is visible. For '`transmission`' rays, controls how a specific surface cast shadows on other surfaces. Possible values for this attribute are:

`"transparent"`
: The object will not cast shadows on any other object since it is completly transparent;

`"opaque"`
: The object will cast a shadow as if it was a completely opaque object;

`"Os"`
: The object will cast shadow according to the opacity value given by the `Opacity` attribute;

`"shader"`
: The object will cast shadows according to the opacity value computed by the surface shader.

`Attribute "trace" "float bias" [.01]`
This bias affects all traced rays. The bias specifies an offset added to ray's starting point (in ray's direction) so to avoid intersection with the emitting surface.

`Attribute "light" "string shadows" ["off"]`
Turns automatic shadow calculation ["on"] or ["off"] for `LightSource`s specified after this directive.

### Global Illumination

`Attribute "irradiance" "nsamples" [64]`
Specifies the default number of samples to use when calling `occlusion()` or `indirectdiffuse()` shadeops. This default value can be overriden by passing a parameter to those shadeops. See Section 4.5.4 [Lighting and Ray Tracing], page 39.

`Attribute "irradiance" "shadingrate" [1]`
3Delight can use a different shading rate for irradiance computations to speed up `occlusion()` and `indirectdiffuse()` computations. Using higher shading rates will enable irradiance interpolation across shaded grids which will speed up rendering;

`Attribute "irradiance" "maxerror" [1]`
Controls a maximum tolerable error when using interpolation (for irradiance shading rate higher than 1). Parameter range is [0..1].

### Displacement

[1] Was `Attribute "visibility" "integer reflection"` in versions prior to 1.0 .
[2] Was `Attribute "visibility" "integer shadow"` in versions prior to 1.0 .

`Attribute "displacementbound" "float sphere" [`*radius*`] "string coordinatesystem" "`*ss-name*`"`
`Attribute "displacementbound" "float sphere" [`*radius*`] "matrix transform" [`*matrix*`]`
`Attribute "bound" "float displacements" [`*radius*`]`
Whenever a displacement shader is applied to a primitive, one must specify a displacement bound to inform the renderer about maximum displacement amplitude. There are three ways to specify a displacement bound: the first way is to tell the renderer that a displaced point will not move outside a sphere of a given radius, in a given coordinate system. The second way, very similar to the first one, uses a matrix instead of a name to describe the space. The third way, provided for backward compatibility only, specifies a sphere of a given radius living in the camera coordinate system.

`Attribute "trace" "int displacements" [0]`[3]
If true, objects will be displaced prior to ray/primitive intersection tests. If false, displaced object will only be bump-mapped when viewied from traced rays. Turning on this option may degrade performance (in terms of memory usage and speed).

**Other Attributes**

`Attribute "trimcurve" "string sense" ["inside"]`
Specifies wether the "interior" or the "exterior" of a trim curve should be trimmed (cut) on a NURB surface. The default value is "inside". Specifying "outside" will reverse the behaviour.

`Attribute "dice" "rasterorient" [1]`
When raster oriented dicing is off (`[0]`), surfaces are diced in camera space instead of being adaptivly diced to meet a certain shading rate when rasterized[4]. The default is on.

[3] This is the equivalent of the deprecated `Attribute "render" "int truedisplacment"`.
[4] Useful for "ambient occlusion" renderings.

| **Attribute** | **Default value** | **Comment** |
|---|---|---|
| "identifier" "name" | "" | Primitives are not named by default; |
| "visibility" "camera" | [1] | Everything is visible to the camera by default; |
| "visibility" "trace" | [0] | Invisible to `trace()`; |
| "visibility" "transmission" | "transparent" | Invisible to `shadow()`, `transmission()` and `occlusion()`; |
| "trace" "bias" | [0.01] | – |
| "trace" "displacements" | [0] | No true dispalements when tracing rays; |
| "irradiance" "shadingrate" | [1] | `indirect()` and `occlusion()` evaluation occurs at every shading sample; |
| "irradiance" "maxerror" | [0] | Interpolation disabled; |
| "displacementbound" | – | No displacement bound by default. Do not forget to specify one when using `Displacement`; |
| "light" "shadows" | ["off"] | – |
| "limits" "eyesplits" | [6] | Large values can lead to *long* render times; |
| "trimcurve" "sense" | ["inside"] | By default, trim curves cut holes in surfaces. Specifying '`outside`' reverses the behaviour (trim curves cut what is outside the curve); |
| "dice" "rasterorient" | [1] | – |

Implementation specific attributes and their default values

Here is a few examples showing how to set some attributes:

1. Primitives visible from camera and shadow rays (as opaque objects) but not reflection/refraction rays

```
Attribute "visibility" "camera" [1]
Attribute "visibility" "transmission" ["opaque"]
Attribute "visibility" "trace" [0]
```

   or in a more compact way,

```
Attribute "visibility" "camera" 1 "transmission" ["opaque"] "trace" 0
```

2. Specify a displacement bound of 0.1 in object space

```
Attribute "displacementbound" "sphere" [0.1]
          "coordinatesystem" ["object"]
```

## 4.3 Geometric Primitives

3Delight supports the entire set of geometric primitives defined by RenderMan. Since the primitives are explained in great detail in the RenderMan specification, the following chapters will only give a brief description and some useful implementation details.

### 4.3.1 Subdivision Surfaces

Catmull-Clark subdivision surfaces are supported by 3Delight[1]. Also, all the standard tags are recognized:

`"hole"` A per-face value, tagging faces that are considered as holes. The parameter is an array of integers listing the indices of the hole faces;

`"corner"` A per-vertex value, tagging vertices that are considered semi-sharp. The tag should be specified with an array of integers, giving the indices of the vertices as well as an array of floating point values giving their sharpness. A sharpness of 0.0 indicates a smooth corner and a sharpness of 10.0 indicating an infinitely sharp one;

`"crease"` A per-edge value, tagging edges that are considered as semi-sharp. This tag is specified with an array of integers, indicating the list of vertices that form an edge chain. An array of floating point values specifies the sharpness value for each edge in the chain;

`"interpolateboundary"`
A per-surface value, specifying that all boundary edges and vertices are infinitly sharp. There is no parameters to this tag.

Unlike other rendering packages, 3Delight does not attempt to tesselate the entire subdivision surface into many small polygons (thus taking a large amount of memory); instead, a lazy and adaptive process is used to generate only those portions of the surface that are needed for some specific bucket. Also, 3Delight generates smooth b-spline surfaces (instead of polygons) whenever possible.

[1] Other subdivision schemes (such as Loop and Butterfly) will be rendered as polygonal meshes.

All standard variable types are supported: `constant`, `uniform`, `varying`, `vertex`, `facevarying` and `facevertex`. `facevertex` is used exactly as `facevarying` but interpolates the variables according to surface's subdivision rules. This eliminates many distortion artifacts due to bilinear interpolation in `facevarying`s.

### 4.3.2 Parametric Patches

All parametric patches are supported, this includes:

`Patch` Specifies a single bicubic or bilinear patch. All the standard basis matrices are supported: `"bezier"`, `"bspline"`, `"catmull-rom"` and `"hermite"`.

`PatchMesh` Specifies a rectangular mesh of bilinear or bicubic patches. As with `Patch`, all the basis matrices are supported. It is recommanded to use `PatchMesh` instead of `Patch` when specifying connected surfaces.

`NuPatch` Specfies a NURBS surface. Trimming is supported.

The following variable types are allowed for parametric patches: `constant`, `uniform`, `varying`, `vertex` and `facevarying`.

### 4.3.3 Curves

Both linear and cubic curves are supported. The following variables are implemented: `constant`, `uniform`, `varying`, `vertex` and `facevarying`. Note that *uniform*s are specified per curve segement (and not per curve), this may change in a futur version.

### 4.3.4 Polygons

All polygonal primitives are supported, this includes: `Polygon`, `GeneralPolygon`, `PointsPolygons` and `PointsGeneralPolygons`. All polygonal primitives support the following variable types: `constant`, `uniform`, `varying`, `vertex` and `facevarying`.

### 4.3.5 Points

Points are supported in 3Delight through the standard RenderMan `Points` primitive. Points' size is controlled using either `"width"` or `"constantwidth"` variable (`RI_WIDTH` or `RI_CONSTANTWIDTH` in the API). The way 3Delight renders points is selectable during primitive declaration using the `"uniform string type"` variable which can take two values: '`sphere`' or '`disk`'. Default is '`sphere`'.

EXAMPLE

```
WorldBegin
  Translate 0 0 10
  Color 1 1 1
  TransformBegin
    Rotate -30 0 1 0
    Rotate 60 1 0 0
    Scale 0.3 0.3 0.3
    # Render some points using the 'sphere' primitive ...
    Points
      "P" [ -3 2 1 3 2 1 -3 0 1 -1 0 1 1 0 1 3 0 1
            -1 -4 1 1 -4 1 -3 2 -1
             3 2 -1 -3 0 -1 -1 0 -1 1 0 -1 3 0 -1 -1 -4 -1 1 -4 -1 ]
      "constantwidth" [0.2]
      "uniform string type" "sphere"
  TransformEnd
WorldEnd
```

### 4.3.6 Implicit Surfaces (Blobbies)

Implicit surfaces are implemented and can be specified using blobs, segments and planes. All the standard operators are supported, including: add, substract, min, max, multiply, divide and negate. Depth files specified to repulsion planes are not supported yet. Variables specified to implicit surfaces will be correctly interpolated over the surface.

### 4.3.7 Quadrics

All six quadrics (plus the torus) are supported. This includes: `Sphere`, `Disk`, `Hyperboloid`, `Paraboloid`, `Cone`, `Cylinder` and `Torus`. All those primitives accept the following variable types: `constant`, `uniform`, `varying` and `vertex`. Note that `vertex` variables behave *exactly* as `varying`s on quadrics.

### 4.3.8 Procedural Primitives

Procedural primitives from the C binding using `RiProcedural` are supported. Built-in procedures for the RIB binding (`RiProcDelayedReadArchive`, `RiProcRunProgram` and `RiProcDynamicLoad`) are also fully supported on all platforms.

### Limitations

Procedural primitives cannot be used inside a `RiMotionBegin`/`RiMotionEnd` block. They can still be subject to transformation motion blur as well as contain motion blocks themselves. When a procedural primitive contains a motion block, it is the user's responsability to take it into account when providing the renderer with a bounding box for the procedural primitive.

Procedural primitives cannot be used inside a `RiObjectBegin`/`RiObjectEnd` block. They can however contain such a block and use objects declared before themselves (with the caveats mentionned below).

The following parts of the graphics state are not saved and restored for procedural primitives:

1. Token declarations made with `RiDeclare`.
2. Named coordinate systems created with `RiCoordinateSystem`.
3. Retained geometry objects (`RiObjectBegin`/`RiObjectEnd`).

This means that creating/declaring one of the above inside a procedural primitive may affect the graphics state for later procedural primitives. Likewise, a declaration made after the procedural primitive was declared (but before it was needed by the renderer) may affect it. This means it is necessary to declare the required elements before the procedural primitives that require them and not replace them until after the end of world block, when rendering is complete. It also wise to avoid using these declarations inside the procedural primitive itself. This behavior is subject to change in future releases depending on user feedback about it. Contact us at info@3delight.com if these features are important to you.

## 4.4 Optional Capabilities and Extensions

3Delight is RenderMan compliant. It supports all RenderMan required capabilities and a great deal of optional ones. Here is a list of only the optional capabilities that are either partially or not yet supported:

*MotionBegin*
*MotionEnd*
: Partially implemented. You can only specifies two motion steps.

*PixelVariance*
: Adaptive oversampling is not supported. Use `PixelSamples` to specify oversampling.

*ColorSamples*
: Colors samples other than RGB are not supported by 3Delight.

The following list of features are also not yet implemented.

```
SolidBegin
SolidEnd
Deformation
Interior
Exterior
AreaLightSource
Detail
DetailRange
RelativeDetail
```

You are welcome to contact us at info@3delight.com in the event you need any of the above functionalities.

## 4.5 Shading Language

3Delight supports all the standard Shading Language (SL) built-in shadeops and constructs. The complete set of standard shadeops is listed below. Descriptions are kept brief, if any, since shadeops are already described in great details in the RenderMan specifications. Shadeops marked with (*) contain specific extensions and those marked with (**) are not part of the current standard. It is also possible to link shaders with C or C++ code to add new shadeops, see Section 8.3 [DSO Shadeops], page 78.

### 4.5.1 Mathematics

`float` **radians** ( `float` *degrees* )
`float` **degrees** ( `float` *radians* )
`float` **sin** ( `float` *radians* )
`float` **asin** ( `float` *a* )
`float` **cos** ( `float` *radians* )
`float` **acos** ( `float` *a* )
`float` **tan** ( `float` *radians* )
`float` **atan** ( `float` *a* )

`float` **pow** ( `float` *x*, *y* )
`float` **exp** ( `float` *x* )
`float` **sqrt** ( `float` *x* )
`float` **inversesqrt** ( `float` *x* )
`float` **log** ( `float` *x* [, *base*] )
`float` **mod** ( `float` *x*, *y* )
`float` **abs** ( `float` *x* )
`float` **sign** ( `float` *x* )
`float` **floor** ( `float` *x* )
`float` **ceil** ( `float` *x* )
`float` **round** ( `float` *x* )

`type` **min** ( `type` *x*, *y* )
`type` **max** ( `type` *x*, *y* )
`type` **clamp** ( `type` *x*, *min*, *max* )

`float` **step** ( `float` *min*, *value* )
`float` **smoothstep** ( `float` *min*, *max*, *value* )

`type` **mix** ( `type` *x*, *y*, *alpha* )
Returns x*(1-*alpha*) + y**alpha*. For multi-component types (color, point, ...), the operation is performed for each component.

`float` **filteredstep** ( `float` *edge*, *value*, ... )
`float` **filteredstep** ( `float` *edge*, *value1*, *value2*, ... )
Simliar to `step()` but the return value is filtered over the area of the micro-polygon being shaded. Useful for shader anti-aliasing. Filtering kernel is selected using the "*filter*" optional parameter. Recognized filters are "gaussian", "box", "triangle" and "catmull-rom". Default is "catmull-rom". If two values are provided, return value is filtered in the range [value1..value2].

`type` **Du** ( `type` *x* )
`type` **Dv** ( `type` *x* )
`type` **Deriv** ( `type` *num*; `float` *denom* )
`Du()` and `Dv()` compute the parametric derivative of the given expressions with respect to the *u* and the *v* parameters of the underlying surface[1].

### 4.5.2 Noise and Random

`type` **noise** ( `float` *x* )
`type` **noise** ( `float` *x*, *y* )
`type` **noise** ( `point` *Pt* )
`type` **noise** ( `point` *Pt*; `float` *w* )
1D, 2D, 3D and 4D noise function. *type* can be `float`, `color`, `point` or `vector`.

`type` **pnoise** ( `float` *x*, *period* )
`type` **pnoise** ( `float` *x*, *y*, *xperiod*, *yperiod* )
`type` **pnoise** ( `point` *Pt*, *Ptperiod* )
`type` **pnoise** ( `point` *Pt*; `float` *w*; `point` *Ptperiod*; `float` *wperiod* )
Same as noise but has periodicity *period*. Maximum period is 256.

`type` **cellnoise** ( `float` *x* )
`type` **cellnoise** ( `float` *x*, *y* )
`type` **cellnoise** ( `point` *Pt* )
`type` **cellnoise** ( `point` *Pt*, `float` *w* )
Cellular noise functions (1D, 2D, 3D and 4D).

`type` **random** ( )
Returns a random `float`, `color` or `point`. Returned range is [0..1]. Can return `uniform` or `varying` values. Here is a trick to put a random color in each grid of micro-polygons:

```
uniform color red = random();
uniform color green = random();
uniform color blue = random();

Ci = color( red, green, blue );
```

### 4.5.3 Geometry, Matrices and Colors

`float` **xcomp** ( `point` *Pt* )
`float` **ycomp** ( `point` *Pt* )
`float` **zcomp** ( `point` *Pt* )
`void` **setxcomp** ( `output point` *Pt*; `float` *x* )
`void` **setycomp** ( `output point` *Pt*; `float` *y* )
`void` **setzcomp** ( `output point` *Pt*; `float` *z* )

[1] `Deriv()` computes the following expression: $\frac{Du(num)}{Du(denum)} + \frac{Dv(num)}{Dv(denum)}$

`float` **comp** ( `matrix` *M*; `float` *row*, *col* )
Returns *M*[*row*,*col*].

`void` **setcomp** ( `output matrix` *M*; `float` *row*, *col*, *x* )
*M*[*row*,*col*] = *x*.

`float` **comp** ( `color` *c*; `float` *i* )

`void` **setcomp** ( `output color` *c*, `float` *i*, *x* )

`point` **transform** ( `string` [*fromspace*,] *tospace*; `point` *Pt* )
`point` **transform** ( [`string` *fromspace*;] `matrix` *M*; `point` *Pt* )
`vector` **vtransform** ( `string` [*fromspace*,] *tospace*; `vector` *V* )
`vector` **vtransform** ( [`string` *fromspace*;] `matrix` *M*; `vector` *V* )
`normal` **ntransform** ( `string` [*fromspace*,] *tospace*; `normal` *Nr* )
`normal` **ntransform** ( [`string` *fromspace*;] `matrix` *M*; `normal` *Nr* )
Transform a `point`, `vector` or `normal` from some given space (*fromspace*) to another space (*tospace*). If the optional *fromspace* is not given, it is assumed to be the '`current`' space.

`color` **ctransform** ( `string` [*fromspace*,] *tospace*; `color` *src_color* )
Transforms color *src_color* from color space *fromspace* to color space *tospace*. If the optional *fromspace* is not specified, it is assumed to be '`rgb`'. 3Delight knows about the following color spaces: RGB, HSV, HSL, YIQ and XYZ[2]. If an unknown color space is given, 3Delight will return *src_color*.

`float` **distance** ( `point` *Pt1*, *Pt2* )
`float` **length** ( `vector` *V* )
`vector` **normalize** ( `vector` *V* )

`float` **ptlined** ( `point` *Pt1*, *Pt2*, *Q* )
Returns minimum distance between a point *Q* and a line *segment* defined by *Pt1*, *Pt2*.

`point` **rotate** ( `point` *Q*; `float` *angle*; `point` *Pt1*, *Pt2* )
Rotates a point *Q* around the line defined by *Pt1*, *Pt2* by a given *angle*. New point position is returned. Note that *angle* is assumed to be in radians.

`float` **area** ( `point` *Pt* )
Returns `length(Du(Pt)^Dv(Pt))`, which is approximatly the area of one micro-polygon on the surface defined by *Pt*.

`vector` **faceforward** ( `vector` *N*, *I*[, *Nref*] )
Flip N, if needed, so it faces in the direction opposite to I. *Nref* gives the element surface normal; if not provided, *NRef* will be set to *Ng*.

`vector` **reflect** ( `vector` *I*, *N* )

[2] When converting to and from XYZ color space, 3Delight considers that RGB tristimulus values conform to Rec. 709 (with a white point of $D_{65}$ ).

`vector` **refract** ( `vector` *I*, *Nr*; `float` *eta* )
Returns the refracted vector for the incoming vector *I*, surface normal *Nr* and index of refraction ratio *eta*.

`float` **depth** ( `point` *Pt* )
Returns the normalized *z* coordinate of *Pt* in camera space. Return value is in the range [0..1] (0=near clipping plane, 1=far clipping plane). *Pt* is assumed to be defined in '`current`' space.

`normal` **calculatenormal** ( `point` *Pt* )
Use this function to compute the normal of a surface defined by *Pt*. Often used after a displacement operation. Equivalent to `Du(Pt)^Dv(Pt)`, but faster.

`float` **determinent** ( `matrix` *M* )
`matrix` **translate** ( `matrix` *M*; `point` *Tr* )
`matrix` **rotate** ( `matrix` *M*; `float` *angle*; `vector` *axis* )
`matrix` **scale** ( `matrix` *M*; `point` *Sc* )
Basic matrix operations. The *angle* parameter passed to `rotate()` is assumed to be in radians.

### 4.5.4 Lighting and Ray Tracing

`color` **ambient** ( )
Returns the contribution from ambient lights. A light is considered ambient if it does not contain an `illuminate()` or `solar()` statement.

`color` **diffuse** ( `vector` *Nr* )
Computes the diffuse light contribution. Lights placed behind the surface element being shaded are not considered. *Nr* is assumed to be of unit length. Light shaders that contain a parameter named `uniform float __nondiffuse` will ony be evaluated if the parameter is set to 0.

`color` **specular** ( `vector` *Nr*, *V*; `float` *roughness* ) *
Computes the specular light contribution. Lights placed behind the object are not considered. *Nr* and *V* are assumed to be of unit length. Light shaders that contain a parameter named `uniform float __nonspecular` will ony be evaluated if the parameter is set to 0.

`color` **specularbrdf** ( `vector` *L*, *Nr*, *V*; `float` *roughness* )
Computes the specular light contribution. Similar to `specular()` but receives a *L* variable (incoming light vector) enabling it to run in custom `illuminance()` loops.

`color` **specularstd** ( `normal` *N*; `vector` *V*; `float` *roughness*)
Since 3Delight implements its own specular model in `specular`, we provide this function in case you need the standard specular model described in all graphic books. `specularstd()` is implemented as follows:

```
color specularstd( normal N; vector V; float roughness )
{
    extern point P;
    color C = 0;
    point Nn = normalize(N);
    point Vn = normalize(V);

    illuminance(P, Nn, PI/2)
    {
        extern vector L;
        extern color Cl;

        vector H = normalize(normalize(L)+Vn);
        C += Cl * pow(max(0.0, Nn.H), 1/roughness);
    }

    return C;
}
```

`color` **phong** ( `vector` *Nr*, *V*; `float` *size* )
: Computes specular light contribution using the Phong illumination model. *Nr* and *V* are assumed to be of unit length. As in `specular()`, this function is also sensitive to the `__nonspecular` light shader parameter.

`color` **trace** ( `point` *Pt*; `vector` *R* [; `output float` *dist*] ) *
: Returns color of light arriving to *Pt* from direction *R*. This is accomplished by tracing a ray from position *Pt* in direction specified *R*. Only objects tagged as visible to reflection rays will be considered during the operation (using `Attribute "visibility" "trace"`, Section 4.2 [attributes], page 28). If the optional *dist* parameter is specified, it will contain the distance to the intersection point or a very large number ($> 1e30$) if no intersections found. Note that *Pt* and *R* must lie in '`current`' space.

`float` **trace** ( `point` *Pt*; `vector` *R* )
: Returns distance to the nearest object, when looking from *Pt* in the direction specified by the unit vector *R*. This function may be substancially faster than the color version of `trace()` since it might avoid costly shading operations. *Pt* and *R* must lie in '`current`' space.

`color` **transmission** ( `point` *Pt1*, *Pt2* )[3]
: Determines the visibility between *Pt1* and *Pt2* using ray tracing. Returns `color 1` if unoccluded and `color 0` if totally occluded. Inbetween values indicate the presence of a translucent surface between *Pt1* and *Pt2*. Only objects tagged as visible to transmission/shadow rays will be considered during the operation (using `Attribute "visibility" "transmission"`, Section 4.2 [attributes], page 28). *Pt1* and *Pt2* must lie in '`current`' space.

[3] Was `visibility()` in versions prior to 1.0.

`float` **occlusion** ( `point` *Pt* ; `vector` *R* ; [`float` *samples* ;] ... ) *

Computes the amount of occlusion, using ray tracing, as seen from *Pt* in direction *R* and solid angle 2*PI (hemisphere). Returns 1.0 if all rays hit an object (totally occluded) and 0.0 if no hits (totally unoccluded). The optional *samples* parameter specifies the number of rays to trace to compute occlusion; if absent or set to 0, 3Delight will use `Attribute "irradiance" "nsamples"`, see Section 4.2 [attributes], page 28. `occlusion()` accepts optional token/value pairs, those are explained in the table below. *Pt* and *R* must lie in '`current`' space.

EXAMPLE

```
/* Returns the amount of occlusion using default number of
   samples */
float hemi_occ = occlusion( P, Ng );

/* Returns the amount of occlusion for the hemisphere surrounding P,
   uses a rough approximation with 8 samples */
hemi_occ = occlusion( P, Ng, 8 );

/* Same as above, but only consider objects closer than 10 units and
   in a solid angle of Pi/2
*/
hemi_occ = occlusion( P, Ng, 8, "maxdist", 10, "angle", PI/4 );

/* Same as above, but only consider light coming from an hemisphere
   oriented toward (0,1,0)
*/
uniform vector sky = vector (0, 1, 0);
hemi_occ =
    occlusion( P, Ng, 8, "maxdist", 10, "angle", PI/4, "axis", sky );
```

`float` **indirectdiffuse** ( `point` *Pt* ; `vector` *R* ; [`float` *samples* ;] ... ) *

Computes diffuse illumination arising from diffuse-to-diffuse indirect light transport by sampling an hemisphere around some point *Pt* and direction *R*[4]. Use this shadeop to render "color bleeding" effects. Also, this function makes it possible to lookup into an HDR image when sampled rays do not hit any geometry; the map is specified using the `"environmentmap"` parameter as shown in the table below. Computing the occlusion while calling this function is also possible (through the `occlusion` parameter), with the following two restrictions:

- It is not possible to specify an `axis` parameter to `indirectdiffuse()` as with `occlusion()`;
- `indirectdiffuse()` only sees geometry tagged as visible to reflections, as opposed to `occlusion()` which sees geometry visible to shadows. For more informations about visibility attributes refer to Section 4.2 [attributes], page 28.

*Pt* and *R* must lie in '`current`' space.

[4] Ray directions are sampled from a cosine distribution. Also applies to `occlusion()`.

| Name | Type | Default | Description |
|---|---|---|---|
| "angle" | `uniform float` | $PI/2$ | Controls the solid angle considered, default covers the entire hemisphere; |
| "axis" | `uniform vector` | – | If specified, and different from `vector 0`, indicates the direction of the light casting hemisphere. Rays that are not directed toward this axis will not be considered. This is useful for specifying skylights. This has not effect on indirect diffusion computations, only `occlusion()` shadeop uses this parameter. *axis* doesn't have to be of unit length; |
| "bias" | `uniform float` | 0.01 | Bias to add when intersecting surfaces to avoid precision related self-intersections. Default value taken from `Attribute "trace" "bias"`, see Section 4.2 [attributes], page 28; |
| "maxdist" | `uniform float` | 1e38 | Only consider intersections closer than this distance; |
| "environmentmap" | `uniform string` | `""` | Specifies an environment map to lookup when a sampled ray doesn't hit any geometry. Only available in the `indirectdiffuse()` shadeop; |
| "environmentdir" | `output varying vector` | – | If specified, will be set to the averange un-occluded direction, which is the average of all sampled directions that did not hit any geometry. Note that this vector is defined in '`current`' space, so it is necessary to transform it to '`world`' space if an `environment()` lookup is intended. |

`occlusion()` and `indirectdiffuse()` optional parameters.

### 4.5.5 Texture Mapping

`type` **texture** ( `string` *texturename*[`float` *channel*]; ... )
`type` **texture** ( `string` *texturename*[`float` *channel*]; `float` *s*, *t*; ... )
`type` **texture** ( `string` *texturename*[`float` *channel*]; `float` *s1*, *t1*, *s2*, *t2*, *s3*, *t3*, *s4*, *t4*; ... )

Returns a filtered texture value (`type` can be either a `float` or a `color`), at the specified texture coordinates. If no texture coordinates are provided, *s* and *t* will be used. An optional *channel* can be specified to select a starting channel in the texture, this can be useful when a texture contains more that three channels. Use `tdlmake` to prepare your textures for improved performance and memory usage (see Section 3.3 [Using the texture optimizer], page 12). `texture()` accepts a list of optional parameters as summerized in the table below.

EXAMPLE

```
/* Sharper result (width<1) */
color c = texture( "grid.tdl", s, t, "width", 0.8 );

/* Returns the green component */
float green = texture( "grid.tdl"[1] );

/* Returns the alpha channel, or 1.0 (opaque) if no
   alpha channel is present */
float alpha = texture( "grid.tdl"[3], "fill", 1.0 );
```

`type` **environment** ( `string` *texturename*[*channel*]; `vector` *V*; ... )
`type` **environment** ( `string` *texturename*[*channel*]; `vector` *V1*, *V2*, *V3*, *V4*; ... )

Returns a filtered texture value from an environment map, for some specified direction. As in `texture()`, an optional *channel* can be specified to select a starting channel when performing texture lookups. Use `tdlmake` to prepare cubic and long-lat envmaps. If an unprepared TIFF is given to `environment()`, it will be considered as a lat-long environment map. `environement()` recognizes the same list of optional parameters as `texture()`.

EXAMPLE

```
/* Do an env lookup */
normal Nf = faceforward( N, I );
color c = environment( "env.tdl", Nf );

/* Only fetch the alpha channel, if no alpha present, returns 1 */
float red_comp = environment( "env.tdl"[3], Nf, "fill", 1 );
```

| Name | Type | Default | Description |
|---|---|---|---|
| "blur" | `varying float` | 0 | Specifies an additional area to be added to the texture lookup area in both $s$ and $t$, expressed in units of texture coordinates (range = [0..1]). A value of 1.0 would request that the entire texture be blurred in the result; |
| "sblur" | `varying float` | 0 | Specifies "blur" in $s$ only; |
| "tblur" | `varying float` | 0 | Specifies "blur" in $t$ only; |
| "width" | `uniform float` | 1 | Multiplies the width of the filtered area in both $s$ and $t$; |
| "swidth" | `uniform float` | 1 | Specifies "width" in $s$ only; |
| "twidth" | `uniform float` | 1 | Specifies "width" in $t$ only; |
| "samples" | `uniform float` | 4 | Specifies the sampling rate. Only useful for `environment()`, `texture()` uses an addaptative sampling algorithm; |
| "fill" | `uniform float` | 0 | If a channel is not present in the texture, use this value; |
| "filter" | `uniform string` | "gaussian" | Specifies the reconstruction filter to use when accessing the texture map. Supported filters are: '`gaussian`', '`triangle`' and '`box`'. |

`texture()` and `environment()` optional parameters.

`type` **shadow** ( `string` *shadowmap*[`float` *channel*]; `point` `Pt; ...` ) *
`type` **shadow** ( `string` *shadowmap*[`float` *channel*]; `point` *Pt1*, *Pt2*, *Pt3*, *Pt4*;
`...` ) *

Computes occlusion at some point in space using a shadow map or a deep shadow map. Shadow lookups are automatically antialiased. When using deep shadow maps, colored shadows and motion blur will be correclty computed. It is possible to perform a ray traced shadow test by passing a `"!"` or `"shadow"`. Note that if ray tracing is used, only objects tagged as visible to shadows will be considered (using `Attribute "visibility" "transmission"`, Section 4.2 [attributes], page 28). Optional parameters to `shadow()` are described in the table below. For additional information about shadow maps and deep shadow maps refer to Section 5.1 [Shadows], page 50.

| Name | Type | Default | Description |
|---|---|---|---|
| "blur" | `varying float` | 0 | Specifies an additional area to be added to the texture lookup area in both *s* and *t*, expressed in units of texture coordinates (range = [0..1]). A value of 1.0 would request that the entire texture be blurred in the result; |
| "sblur" | `varying float` | 0 | Specifies "blur" in *s* only; |
| "tblur" | `varying float` | 0 | Specifies "blur" in *t* only; |
| "width" | `uniform float` | 1.0 | Multiplies the width of the filtered area in both *s* and *t*; |
| "swidth" | `uniform float` | 1.0 | Specifies "width" in *s* only; |
| "twidth" | `uniform float` | 1.0 | Specifies "width" in *t* only; |
| "samples" | `uniform float` | 16 | Specifies the number of samples to use for shadow lookups. This influances the antialis quality. A value of 16 is recommended for shadow maps and, 4 for deep shadow maps; |
| "bias" | `varying float` | 0.225 | Used to prevent self-shadowing. If set to 0, the global bias will be used, as specified by `Option "shadow" "bias"` (see Section 4.1 [options], page 20); |

`shadow()` optional parameters.

### 4.5.6 String Manipulation

`string` **concat** ( `string` *str1*, `...`, *strn* )

Concatenates one or more strings into one string.

`string` **format** ( `string` *pattern*; *val1*, `...`, *valn* ) *

Similar to the C `sprintf` function. *pattern* is a string contening conversion characters. Recognized conversion characters are :

`%f` Formats a float using the style [-]ddd.ddd. Number of fractional digits depends on the precision used (see example);

`%e` Formats a float using the style [-]d.ddde dd (that is, exponential notation). This is the recommended conversion for floats when precision matters;

`%g` The floating point is converted to style `%f` or `%e`. Here is a snapshot from the `info` pages on `printf()`:

> The style used depends on the value convered; style `%e` will be used only if the exponent resulting from the conversion is less than -4 or greater than or equal to the precision. Trailing zeros are removed from the fractional part of the result; a decimal-point character appears only if it is followed by a digit...

`%d` Equivalent to `%.0f`, useful to format integers;

`%p` Formats a point-like type (`point`, `vector`, `normal`) using the style `[%f %f %f]`;

`%c` Same as `%p`, but for colors;

`%m` Formats a matrix using the style `[%f %f %f %f, %f %f %f %f, %f %f %f %f, %f %f %f %f]`;

`%s` Formats a string.

Note that all conversion characters recognise the precision specifier.

EXAMPLE

```
/* Formats a float using exponential notation */
string expo = format( "%e", sqrt(27) );

/* Formats a float, with 5 decimals in the fractional part */
point p = sqrt(5);
string precision5 = format( "p = %.5p", p );

/* Aligns text */
string aligned = format( "%20s", "align me please" );
```

`void` **printf** ( `string` *pattern*; *val1*, ..., *valn* ) *

Same as `format()` but prints the formatted string to '`stdout`' instead of returning a string.

`float` **match** ( `string` *pattern*, *subject* )

Does a string pattern match on *subject*. Returns 1 if pattern exists anywhere within sibjuct, 0 otherwhise. The *pattern* can be any standard `regex`[1] expression.

### 4.5.7 Message Passing and Information

`float` **textureinfo** ( `string texturename, fieldname; output uniform type` *variable* )

Returns information about a particular texture, environment or shadow map. *fieldname* specifies the name of the information as listed in the table below. If *fieldname*

[1] See '`man regex`' for details (`man` pages available on UNIX-like platforms only).

is known, and *variable* is of the correct type, `textureinfo()` will return 1.0. In case of failure, 0.0 is returned.

EXAMPLE

```
/* mapres[0] will contain map resolution in x,
   mapres[1] will contain map resolution in y */

uniform float mapres[2];
textureinfo( "grid.tdl", "resolution", mapres );

/* Get current to camera matrix used to create the shadow map */

uniform matrix Nl;
if( textureinfo( "main-spot.tdl", "viewingmatrix", Nl )!= 1.0 )
{
    Nl = 1;
}
```

| **Field** | **Type** | **Description** |
|---|---|---|
| "resolution" | `uniform float[2]` | Returns texture map resolution; |
| "type" | `uniform string` | Returns type of texture map. Can be one of the following: "texture", "shadow" or "environment"; |
| "channels" | `uniform float` | Returns the total number of channels in the map; |
| "viewingmatrix" | `uniform matrix` | Returns a matrix representing the transform from "current" space to "camera" space in which the map was created; |
| "projectionmatrix" | `uniform matrix` | Returns a matrix representing the transform from "current" space to map's raster space in which $x$ varies from -1 to 1 and $y$ from 1 to -1 ($x$ increases from left to right and $y$ increases from top to bottom). |

`textureinfo()` possible field names.

`float` **atmosphere** ( `string` *paramname* ; `output type` *variable* )
`float` **displacement** ( `string` *paramname* ; `output type` *variable* )
`float` **lightsource** ( `string` *paramname* ; `output type` *variable* )
`float` **surface** ( `string` *paramname* ; `output type` *variable* )

Functions to access a parameter in one of the shaders attached to the geometric primitive being shaded. The operation will succeed if the shader exists, the parameter is present and the type is compatible, in which case 1.0 is returned. In case of failure, 0.0 is returned and *variable* is unchanged. Note that assigning a `varying` shader

parameter to an `uniform` *variable* will fail. Also, `lightsource()` is only available inside an `illuminance()` block and refers to the light source being examined.

`float` **attribute** ( `string` *`dataname`* ; `output type` *`varaible`* )

Returns the value of the data that is part of the primitive's attribute state. The operation will succeed if *dataname* is known and the type is correct, in which case 1.0 will be returned. In case of failure, 0.0 is returned and *variable* is unchanged. The supported data names are listed in the table below.

| **Name** | **Type** |
|---|---|
| "ShadingRate" | `uniform float` |
| "Sides" | `uniform float` |
| "Matte" | `uniform float` |
| "GeometricApproximation:motionfactor" | `uniform float` |
| "displacementbound:sphere" | `uniform float` |
| "displacementbound:coordinatesystem" | `uniform string` |
| "identifer:name" | `uniform string` |

Data fields known to `attribute()`.

`float` **option** ( `string` *`dataname`* ; `output type` *`varaible`* )

Returns the data that is part of the renderer's global option state. The operation will succeed if *dataname* is known and the type is correct, in which case 1.0 will be returned. In case of failure, 0.0 is returned and *variable* is unchanged. The supported data names are listed in the table below.

| **Name** | **Type** | **Description** |
|---|---|---|
| "Format" | `uniform float[3]` | = [ x res, y res, aspect ratio ]; |
| "FrameAspectRatio" | `uniform float` | Frame aspect ratio; |
| "CropWindow" | `uniform float[4]` | Crop window coordinates; as specified by `RiCropWindow`; |
| "DepthOfField" | `uniform float[3]` | = [ fstop, focal length, focal distance ]; as specified by `RiDepthOfField`; |
| "Shutter" | `uniform float[2]` | = [ shutter open, shutter close ]; as specified by `RiShutter`; |
| "Clipping" | `uniform float[2]` | = [near, far]; as specified by `RiClipping`. |

Data fields known to `option()`.

`float` **rendererinfo** ( `string` *`dataname`*`;` `output type` *`varaible`* )

Returns information about the renderer. The operation will succeed if *dataname* is known and the type is correct, in which case 1.0 will be returned. In case of failure, 0.0 is returned and *variable* is unchanged. The supported data names are listed in the table below.

| Name | Type | Description |
|---|---|---|
| "renderer" | `uniform string` | = "3Delight"; |
| "version" | `uniform float[4]` | = [Major, Minor, release, 0] (eg [1,0,6,0] ); |
| "versionstring" | `uniform string` | version expressed as a string, (e.g., "1.0.6.0"). |

data fields known to `rendererinfo()`.

`float` **raylevel** ( ) `**`

Returns ray tracer recursion level. 0 is returned for camera rays.

`float` **isshadowray** ( ) `**`

Returns 1.0 if the shader is being run to evalute the opacity of an object.

`float` **isindirectray** ( ) `**`

Returns 1.0 if the shader is being run to evalute indirect illumination.

### 4.5.8 Limitations

Some features are not currently supported by the 3Delight shader compiler:

- Smooth derivatives.
- Light categories.
- Broad `solar` lights.

You are welcome to contact us at info@3delight.com in the event you need any of the above functionalities.

# 5 Rendering Guidelines

## 5.1 Shadows

3Delight has an extensive support of shadow rendering methods, it is up to you to choose the algorithm that suits you the best.

### 5.1.1 Standard Shadow Maps

Those are normal shadow maps that are widely used in the industry. Generating such shadow maps implies placing the camera at the position of the light source and rendering the scene from that view point ('`zfile`' or '`shadowmap`' display driver has to be selected, see Section 7.3 [dspyzfile], page 64 and Section 7.4 [dspyshadowmap], page 64). The shadow map is then used from inside a light source shader to cast shadows on objects. Shadow maps have a number of advantages:

- *They are fast to generate.* Indeed, shadow maps can be generated much faster than a normal "color" image, mainly because only depth informations are needed. When rendering a shadow map, one could remove all surface and light shaders, even displacement shaders can be removed if they do not affect the geometry too much. Also, filtering (using `PixelFilter` command) can be lowered (even to 1x1) and `ShadingRate` increased (up to 10 or more).
- *They can be reused in more than one render.* If the scene is static and only the camera moves, a generated shadow map can be used for all subsequent renders. This often happens in the lighting stage of a production pipeline;
- *They can be used to generate low cost penumbra.* Specifying an appropriate blur to the `shadow()` call, one can simulate penumbra effects;
- *They provide fairly good results when used carefully.* Many of the recent CG productions use normal shadow maps and obtain excellent results.

Now, the drawbacks:

- *Self shadowing.* The most common problem encountoured when using shadow maps. It appears as dark artifacts in areas that should appear completly lit;
- *Nearly impossible to generate high quality area shadows.* Even if tweeking with shadow blur can give a nice penumbra effect, it is impossible to generate a true area shadow;
- *Expensive to generate really sharp shadows.* High resolution shadow maps are needed which often leads to higher render times and memory/disk usage;
- *No motion blur in shadows.* Moving geometry will cast still shadows, it is wise to remove motion blur when rendering shadow maps;
- *No coloured shadows.* Translucent surfaces will cast opaque shadows;
- *Only objects that are in the shadow map can cast shadows.* That is why shadow maps work so well with spot lights: they can only light a limited field of view. Point lights are more tricky to handle (need six shadow maps) and distant lights are difficult to setup with shadow maps.

When creating shadow maps, make sure that shadow casting objects are framed correctly (and tightly) in the camera view which is used to render the shadow map. If objects are too far (small in shadow map view), precision problems may arise and high resolutions shadow maps will be needed. If objects are too close and parts of them are clipped, shadows might be missed.

3Delight supports the "midpoint" (Section 4.1 [options], page 20) algorithm for normal shadow maps. This should help you get rid of shadow bias problems in most cases.

### 5.1.2 Deep Shadow Maps

Deep shadow maps retain many of the features found in normal shadow maps and provide some more goodies:

- *Translucent shadows.* Translucent surfaces will cast correct shadows;
- *Shadows in participating media.* It is possible to render volumetric shadows using DSMs (the display driver has a flag to account for this feature, see Section 7.5 [dspy_dsm], page 64);
- *Supports mip-mapping.* DSMs are mip-mapped, which means that they will exhibit much less aliasing artifacts than normal shadow maps;
- *Need lower resolutions.* Instead of generating large shadow maps to boost quality, one can generate a smaller DSM by using higher `PixelSample` values when creating the DSM;
- *Cope well with fine geometric details.* Fine details such as hair and particles can be handled without increasing shadow maps resolution too much;
- *Render time shadow lookups are faster.* Since DSMs are prefiltered, render time lookups will be faster (in general) than with normal shadow maps since the signal reconstruction step is more efficient. Also, DSMs are more texture cache friendly than shadow maps.
- *Easily integrateable in a rendering pipeline.* No modifications are necessary in shaders when changing shadow map format (normal or deep), the `shadow()` shadeop will work with both formats.

Before throwing DSMs into your production pipeline, consider the following facts:

- *More expensive to compute.* Generating good DSMs is generally slower than shadow maps. This is often the case when many `PixelSample`s are used during the generation step;
- *Need more disk space.* DSMs can grow quite large, mainly for two reasons:
  1. DSMs are mipmapped by default;
  2. more informations are stored per texel.

*Using the "midpoint" option to produce deep shadow maps can lead to unpredictable results!*

For deep shadow map creation please refer to Section 7.5 [dspy_dsm], page 64.

### 5.1.3 Raytraced Shadows

Tracing shadow rays can be used as an alternative to shadow maps. This has the advantage of being simple to use, but it can be slow compared to shadow maps, especially with complex scenes. To enable tracing of shadow rays for a luminaire, use this directive in the RIB file (before the `LightSource` directive):

```
Attribute "light" "shadows" ["on"]
```

You can control how each objects will cast shadows by using `Attribute "visiblity" "transmission"`. Refer to Section 4.2 [attributes], page 28.

Also, the `shadow()` shadeop has a support for ray-traced shadows: by passing `"!"` or `"shadow"` as a prameter instead of a shadow map name, a shadow ray will be traced!

## 5.2 Ray Tracing

The `trace()` function of the shading language is fully implemented in 3Delight, with the exception that moving object will not be motion blurred.

3Delight also implements other functions related to ray-tracing, they are listed below and described in more detail in Section 4.5 [shading language], page 36.

```
color trace ( point from; vector dir [; output float dist] )
float trace ( point from; vector dir )
color transmission ( point p1, p2 )
float isshadowray()
float raylevel()
```

Make sure that visibility attributes are properly set: by default, 3Delight marks objects as being visible to camera *only*.

## 5.3 Ray Tracing

3Delight is capable of exporting and importing 3Delight has an extensive support for HDRI I/O.

## 5.4 Network Cache

3Delight offers a special extension for more efficient rendering in networked environments in case of sustained texture access[1]: a file cache to minimize network traffic and file server load. This proves particulary powerfull when using a large quantity of rendering servers.

The working principle of 3Delight's network cache is to copy files locally and reuse them when needed. Cache size and location are controlled using `RiOption`. If a texture is needed and the file cache is full, one or more textures will be removed from the cache to make space for the new one; a LRU (Least Recently Used) strategy is used to choose which texture(s) to remove.

The design of the network cache was justified by the following observations:

[1] Not avilable on Windows platforms.

- Rendering a sequence on a render farm puts a great amount of pressure on the network and file server(s): starting a heavy job simultaneously on many machines can make the network inusable and the file server non responsive, especially on slow networks or weakly configured environment;
- Accessing a texture locally is faster than accessing it on a network mounted system such as a NFS. Clearly, texture access can have a significant impact on rendering speed;
- File server(s) load would be greatly minimized since file access requests happen less often;
- The nature of the rendering process makes it very probable that a texture used during a rendering of a given image will be used again in forthcoming renderings of more images, so keeping it handy is a good investment;
- Storage is cheaper, and simpler, than bandwidth!

### 5.4.1 Activating the Network Cache

To enable the network cache, use the following `RiOption`:

```
Option "netcache" "string cachedir" ["/tmp/3delight_cache/"]
```

This informs 3Delight to use a directory named '`/tmp/3delight_cache/`' to cache textures files. If the directory is already created, 3Delight will use whatever cached files it contains. It is important to use a *locally mounted* directory since caching network files on a network volume is useless. Also, the choosen directory should be dedicated to 3Delight: *do not put any of your files in that directory since they can disapear without notice.*

It is possible to disable the cache again by calling the same `RiOption` with an null file name. This can be useful in multi-frame RIBs.

```
Option "netcache" "string cachedir" [""]
```

Cache size is controlled using another `RiOption`. For example, to specify 1000 megabytes of network cache, use:

```
Option "netcache" "integer cachesize" 1000
```

Specifiying a size which is smaller than the actual cache size will cause files to be removed from the cache until the specified size is reached.

There is no need to specify which files to cache, 3Delight will automatically detect slow access files and cache them. Slow access files are files mounted on a NFS disk or a CD-ROM. Also, caching is perfomed in a lazy fashion: if an object is not visible, its textures will not be cached.

### 5.4.2 Purging the Network Cache

The network cache can be purged manually at any moment *if no renderings are running on the machine*. Simply erase the directory and all its contained files:

```
rm -rf /tmp/3delight_cache
```

### 5.4.3 Safety

Many precautions have been taken to ensure the proper operation of the network cache:

1. 3delight will **not** access the original textures in any dangerous way, only **reading** is performed on those textures;
2. The cache is kept synchronized with the files it mirrors: if an original texture is newer than the cached one, the cache will be updated;
3. A texture is identified by its *full path*: texture files that have the same name in different directories will not collide;
4. UNIX file links are resolved prior to caching, this ensures that a given texture will be cached only once even if many links point to it;
5. The network cache is multi-process safe. Even if many renderings are running on the same machine, the cache is kept in a consistant state: one 3Delight instance will not remove a texture used by another instance!
6. Cache directory is created with full access permissions to '`user`' and '`group`', but only read access to '`other`'[2];
7. If, for any reason, 3Delight is unable to cache a texture, it will revert to use the original, and this is *the worst case scenario*.

[2] Permissions mask : `0775`.

# 6 Using Shaders

You can control precisely the look of the image produced by 3Delight by associating shaders with elements of your scene. There are many types of shaders; each one is used to control a particular aspect:

- `surface` shaders control the way an object reflects light;
- `displacement` shaders modify the geometry of underlying surfaces;
- `lightsource` shaders control how a luminaire emits light in the scene;
- `atmosphere` shaders are used to create atmospherical effects, such as fog and smoke;
- `imager` shaders are applied on the computed image and can perform simple image processing tasks.

Shaders can be bought from third parties or found on various web sites. You can also write your own shaders, see Section 6.1 [Writing shaders], page 55 for more information. If you wish to write your own shaders then *The RenderMan Interface Version 3.2* is a good document to read. Is is available electronically at https://renderman.pixar.com/products/rispec/index.htm.

The next sections explains how to write your own shaders, how to use specific features of the 3Delight shading language and how to install your shaders properly, so that 3Delight will be able to find them.

## 6.1 Writing Shaders

Some shader examples can be found in the '`shaders/src`' directory. If you need a tutorial on how to write shaders you should read the following classics:

- Steve UPSTILL. *The RenderMan Companion*. Addison Wesley.
- Larry GRITZ and Anthony A. APODACA. *Advanced RenderMan: Creating CGI for Motion Pictures*. Morgan Kauffman.

Since each renderer has its own implementation of the shading language, shaders writers may want to isolate compiler specific code. `shaderdl` predefines the preprocessor symbol `DELIGHT` in order to make this possbile.

```
#if defined(DELIGHT)
print("Compiled with 3Delight\n");
#elif defined(RDC)
print("Compiled with RenderDotC\n");
#elif defined(BMRT)
print("Compiled with Blue Moon Rendering Tools\n");
#else
print("yet another rman renderer\n");
#endif
```

Do not forget that before using a shader in 3Delight, one must compile it using 3Delight shader compiler, `shaderdl`. See Section 3.2 [Using the shader compiler], page 9 for details on this.

## 6.2 Installing Shaders

Once you have compiled a shader, you must install it at the appropriate place in your file system. The "appropriate place" is a place where 3Delight can find them!

When 3Delight receives a `Surface`, `Displacement`, `LightSource` or `Volume` command, it searches immediately for the shader given as the first argument to the command. The shader can be specified using a full path or a relative path to the shader file. On Unix, a full path is a path that starts with a '`/`'. On Windows, a full path starts with a drive (e.g. '`C:`'). All other paths are interpreted as relative paths. If you want to write cross-platform RIB files, then you should not specify your shaders using full paths.

If you specify a shader using its full path, 3Delight only looks for the shader at the location specified by that path.

If you specify a shader using a relative path, 3Delight tries to find the shader in the directories specified by the shader search paths list. Each directory from the list is tried in turn, starting with the first one. As soon as a matching shader is found, the search stops. For more information, see Section 4.1 [options], page 20.

When specifying a shader, you should use the slash ('`/`') character to separate directories, even under Windows. If 3Delight encounters an environment variable, it will replace it by its value. The environment variable has to be specified using the Unix convention (like this: `$HOME`).

A tilde ('`~`') character at the beginning of a path has the same signification as the `$HOME` environment variable.

If the specified shader cannot be found, 3Delight signals an error message and replaces it by the default shader. The default shader is '`defaultsurface`' for surfaces, '`spotlight`' for lightsources, '`bumpy`' for displacements and '`null`' for atmosphere.

## 6.3 Interrogating Shaders

The easiest way to get informations about a specific shader is to run `shaderinfo`, see Section 3.6 [Using shaderinfo], page 19. Alternatively, it is possible to write a program that makes calls to the '`lib3delight`' library.

### 6.3.1 Using '`lib3delight`' to Interrogate Shaders

You can link your program with '`lib3delight`' (see Section 8.1 [linking with 3delight], page 67 to query informations about shaders. Include the file '`slo.h`' to get the prototypes of the following functions and the type definitions they use.

`void` **Slo_SetPath** ( `char*` *`i_path`* )
: Set the paths where the library will look for shaders.

`void` **Slo_SetShader** ( `char* i_name` )
: Find and open the shader named *i_name*. Close any shader that was previously opened by `Slo_SetShader`.

`char*` **Slo_GetName** ( `void` )
: Returns the name of the currently opened shader.

`SLO_TYPE` **Slo_GetType** ( `void` )
: Return the type of the currently opened shader. See the file '`slo.h`' for details about `SLO_TYPE`.

`int` **Slo_GetNArgs** ( `void` )
: Return the number of parameters of this shader.

`SLO_VISSYMDEF*` **Slo_GetArgById** ( `int i` )
: Return information about the *i*-th parameter of the shader. The first parameter has an id of 1. See the file '`slo.h`' for details about `SLO_VISSYMDEF`.

`SLO_VISSYMDEF*` **Slo_GetArgByName** ( `char *i_name` )
: Return information about the parameter named *i_name*. See the file '`slo.h`' for details about `SLO_VISSYMDEF`.

`SLO_VISSYMDEF` ***Slo_GetArrayArgElement** ( `SLO_VISSYMDEF *i_array, int i_index` )
: If a parameter is an array (as specified by `Slo_GetArgById()` or `Slo_GetArgByName`), each of its element should be accessed using this function.

`void` **Slo_EndShader** ( `void` )
: Close the current shader.

`char*` **Slo_TypetoStr** ( `SLO_TYPE i_type` )
: Get a string representation of type *i_type*.

`char*` **Slo_StortoStr** ( `SLO_STORAGE i_storage` )
: Get a string representation of storage class *i_storage*.

`char*` **Slo_DetailtoStr** ( `SLO_DETAIL i_detail` )
: Get a string representation of variable detail *i_detail*.

Next is the complete source code of the `shaderinfo` utility, it is compilable on all supported platforms, see Section 8.1 [linking with 3delight], page 67. Note that string parameters are represented using a `const char * const *` and not a `const char *` like in some implementations.

```
/*****************************************************************************/
/*                                                                           */
/*    Copyright (c)The 3Delight Team.                                        */
/*    All Rights Reserved.                                                   */
/*                                                                           */
/*****************************************************************************/

// ===========================================================================
// = VERSION
//     $Revision: 1.49 $
// = AUTHOR
//     Aghiles Kheffache
// = DATE RELEASED
//     $Date: 2003/07/30 20:21:54 $
// = RCSID
//     $Id: 3delight.texinfo,v 1.49 2003/07/30 20:21:54 olivier Exp $
// ===========================================================================

#include "slo.h"

#include <string.h>
#include <stdlib.h>
#include <stdio.h>

#include <assert.h>

int main( int argc, char ** argv )
{
    unsigned i, j, k, kk;

    int exitCode = 0;

    bool RIBDeclare = false;
    int start = 1;

    /* DlDebug::InitErrorSignalsHandling(); */

    if( argc<=1 || !strcmp(argv[1], "-h") || !strcmp(argv[1], "--help") )
    {
        printf( "Usage: shaderinfo file1 [file2 ... fileN]\n" );
        printf( "  -d : output declarations in RIB format\n" );
        printf( "  -v : show version to console\n" );
        printf( "  -h : show this help\n\n" );

        return 0;
    }

    if( argc==2 && (!strcmp(argv[1], "-v") || !strcmp(argv[1], "--version")) )
    {
        printf( "shaderinfo version 2.0.\n" );
        printf( "Copyright (c) 1999-2003 The 3Delight Team.\n" );

        return 0;
    }

    if( (argc == 1) || !strcmp(argv[1], "-d") )
    {
        RIBDeclare = true;
```

```
    start++;
}

const char* path = getenv( "DL_SHADERS_PATH" );

Slo_SetPath( path ? path : "." );

for( i = start; i < argc; i++ )
{
    int err = Slo_SetShader( argv[i] );

    if( err != 0 )
    {
        fprintf(
            stderr,
            "shaderinfo: unable to open shader \"%s\" (error %d).\n",
            argv[i], err );

        exitCode = 1;

        continue;
    }

    printf( "\n%s \"%s\"\n", Slo_TypetoStr(Slo_GetType()), Slo_GetName() );

    for( j = 0; j < Slo_GetNArgs(); j++ )
    {
        SLO_VISSYMDEF *parameter = Slo_GetArgById( j+1 );

        if( !parameter || !parameter->svd_valisvalid )
        {
            fprintf( stderr, "shaderinfor: parameter %d is invalid\n", j );
            continue;
        }

        const char *name = parameter->svd_name;
        const char *detail = Slo_DetailtoStr( parameter->svd_detail );
        const char *type = Slo_TypetoStr( parameter->svd_type );
        unsigned arraylen = parameter->svd_arraylen;

        if( RIBDeclare )
        {
            /* Output a string suitable for RIB syntax */

            if( arraylen == 1 )
                printf( "Declare \"%s\" \"%s %s\"\n", name, detail, type );
            else
                printf( "Declare \"%s\" \"%s %s[%d]\"\n",
                    name, detail, type, arraylen );

            continue;
        }

        if( parameter->svd_storage == SLO_STOR_OUTPUTPARAMETER )
            printf( "    \"%s\" \"output %s %s", name, detail, type );
        else
            printf( "    \"%s\" \"%s %s", name, detail, type );
```

```
if( arraylen > 1 )
    printf( "[%d]", arraylen );

printf( "\"\n" );
printf( "\t\tDefault value: " );

switch( parameter->svd_type )
{
case SLO_TYPE_COLOR:
case SLO_TYPE_POINT:
case SLO_TYPE_VECTOR:
case SLO_TYPE_NORMAL:
case SLO_TYPE_MATRIX:

    if( parameter->svd_spacename[0] != (char)0 )
        printf( "\"%s\" ", parameter->svd_spacename );

    break;

default:
    break;
}

if( arraylen > 1 )
    printf( "{" );

for( k=0; k<arraylen; k++ )
{
    SLO_VISSYMDEF *elem = Slo_GetArrayArgElement( parameter, k+1 );

    if( !elem )
    {
        printf( "<error>" );
        continue;
    }

    switch( parameter->svd_type )
    {
    case SLO_TYPE_SCALAR:
        printf( "%g", *elem->svd_default.scalarval );
        break;

    case SLO_TYPE_POINT:
    case SLO_TYPE_VECTOR:
    case SLO_TYPE_NORMAL:
    case SLO_TYPE_COLOR:
        printf( "[%g %g %g]",
            elem->svd_default.pointval->xval,
            elem->svd_default.pointval->yval,
            elem->svd_default.pointval->zval );
        break;

    case SLO_TYPE_MATRIX:
        printf( "[%g ",  elem->svd_default.matrixval[0] );

        for( kk = 1; kk < 15; kk++ )
            printf( "%g ", elem->svd_default.matrixval[kk] );
```

```
                    printf( "%g]",  elem->svd_default.matrixval[15] );
                    break;

                case SLO_TYPE_STRING:
                    printf( "\"%s\"", *elem->svd_default.stringval );
                    break;

                default:
                    break;
                }

                if( k != (arraylen-1) )
                {
                    printf( ", " );
                }
            }

            if( arraylen > 1 )
                printf( "}" );

            printf( "\n" );
        }

        Slo_EndShader();

        printf( "\n" );
    }

    return exitCode;
}
```

### 6.3.2 Caveats

There are some cases where the default value information is not accurate. The information below applies both to `shaderinfo` and to '`libSloInfo`'.

If the default value is the result of a function call, the default value is undefined. The library will do its best to evaluate the function, but the results are not guaranteed to be exact or logical.

If the default value is the result of an arithmetic operation between two points that are not in the same space, for example:

```
surface bad(
        point A = point "object" (3, 2, 1) + point "camera" (1, 2, 3))
{
        ...
}
```

The default returned value will be the result of the arithmetic operation as if the two point were in the '`current`' space, and the space of the default value will be set to '`current`':

```
    "A" "uniform point"
                Default value: "current" [4 4 4]
```

# 7 Display Driver System

3Delight comes with a set of useful display drivers. Since 3Delight uses the "standard" RenderMan display driver interface, it is possible to use third parties display drivers also. Each display driver is discribed with more detail in the following sections.

## 7.1 The framebuffer display driver

This display driver opens a window and displays the rendered image in it. It supports the following parameter:

`"string autoclose" "off"`
: If this parameter is set to '`on`', the window is automatically closed at the end of the render. The default behavior is to leave the window open until the user closes it.

Note that when 3Delight finishes the render, it will exit without waiting for the user to close the window opened by this display driver, letting the user start multiple renders while keeping the windows open. Of course, if `autoclose` is enabled, the window will be closed automatically.

## 7.2 The TIFF display driver

This display driver lets you write the rendered image into a TIFF file. Compressed TIFFs and cropping are supported. Only RI_RGB, RI_RGBA, RI_A are supported for now. This display driver understands the following parameters.

`"string fullimage" ["on"]`
: When using `CropWindow`, this parameter enables you to create an image of the original size, with the cropped region placed inside. Areas that are outside the crop window will be set to black (and alpha will be 0). This option is useful when compositing a cropped image with a matching complete image. The default value is '`off`'.

`"string compression" ["lzw"]`

`"lzw"` LZW compression, default;

`"none"` no compression;

`"packbits"`
: run length encoding;

`"deflate"`
: Deflate compression (zip);

`"logluv"`[1] LogLuv compression, perfect for high dynamic range images. Expects floating point data as input. A limitation of this compression

[1] Visit `http://positron.cs.berkeley.edu/~gwlarson/pixformat/tiffluv.html` for more informations about this encoding scheme.

scheme is that the alpha channel cannot be stored in the same TIFF directory as the image, so it is written separatly in a 2nd directory. The white point is also stored in the TIFF when using this compression.

`"string description" "s"`
Specifies a description string which will be stored in the TIFF.

`"string extratags" ["on"]`
By default, this display driver will write some useful informations to the TIFF by using the tagging mechanism[2]; however, some tags are not supported by all TIFF software (See table below), this option can turn those tags to '`off`';

| **Tag name** | **Description** |
|---|---|
| TIFFTAG_SOFTWARE | Will be set to something like "3Delight 0.9.6 (Sep 26 2002) "Fidelio"""; |
| TIFFTAG_IMAGEDESCRIPTION | Will be set to a user provided description, if any; |
| TIFFTAG_XPOSITION | x cropping position in pixels, only active when cropping and "fullimage" is "off" (default); |
| TIFFTAG_YPOSITION | y cropping position in pixels, only active when cropping and "fullimage" is "off" (default); |
| TIFFTAG_WHITEPOINT | White point when using LOGLUV compression; |
| TIFFTAG_PIXAR_IMAGEFULLWIDTH* | Set when an image has been cropped out of a larger image; reflects the width of the original uncropped image; |
| TIFFTAG_PIXAR_IMAGEFULLLENGTH* | Set when an image has been cropped out of a larger image; reflects the height of the original uncropped image; |
| TIFFTAG_PIXAR_TEXTUREFORMAT* | Set to "Image" |
| TIFFTAG_PIXAR_MATRIX_WORLDTOSCREEN* | World to Normalized Device Coordinates transformation matrix; |
| TIFFTAG_PIXAR_MATRIX_WORLDTOCAMERA* | Wrold to Camera transformation matrix. |

Supported Tags. Those marked with an '*' can be disabled using '`extratags`' option

[2] For a detailed specification of the TIFF format, refer to http://www.libtiff.org.

## 7.3 The zfile display driver

A '`zfile`' is a file that contains enough informations to build a shadow map. The format of a zfile is as follows:

| Offset | Type | Name | Description |
|---|---|---|---|
| 0 | int32 | magic | 0x2f0867ab on big indian machines, and 0xab67082f on little indian machines; |
| 4 | short | xres | x resolution in pixels; |
| 6 | short | yres | y resolution in pixels; |
| 8 | float[16] | Np | World to Normalized Device Coordinate transformation matrix, in row major order; |
| 72 | float[16] | Nl | World to Camera matrix, in row major order; |
| 136 | float[xres*yres] | z data | top to bottom, left to right ordering. |

To build a shadow map using a zfile, use "`tdlmake` '`-shadow`'" as described in Section 3.3 [Using the texture optimizer], page 12 or issue a `MakeShadow` command in a RIB file.

## 7.4 The shadowmap display driver

Use this display driver to create shadow maps without passing by a zfile. Shadow maps in 3Delight are normal TIFFs. The following parameter can be passed to this display driver.

"`uniform string zcompression`" "`lzw|zip|none`"
: Enables or disables compression. Disabled by default.

*Do not run* `tdlmake` *on files produced by this display driver*, since they are already in the right format.

## 7.5 The DSM display driver

This display driver produces deep shadow maps. As explained in Section 5.1 [Shadows], page 50, DSMs support opacity, prefiltering and motion blur. It is suggested to use DSMs whenever possible.

"`float tolerance`" `[0.01]`
: DSMs tend to grow quite large, unfortunatly. The display driver will try to compress them using an algorithm that will shrink file size without exceeding some specified error threshold. A value of '`0`' (that is, no error allowed) will produce huge DSMs and should **never** be used and a value of '`1`' will compress DSMs too much and results will be unsatisfactory. A typical good value is 0.05. If this parameter is not specified or set to -1, the display driver will choose an appropriate tolerance, which is the recommended action.

"`string compressionlevel`" `["1"]`
: This option turns on an additional compression method which makes DSMs even smaller **without** sacrificing quality or performance. The default is "`1`" which enables some additional compression. "`0`" can be used to disable the additional compression.

`"string mipmaping" ["on"]`
: Disables or enables mipmapping for this deep shadow map. Mipmapping is enabled by default but one could save more than 25% of storage space by using raw DSMs. Be careful though, since mipmapped DSMs usually produce nicer results (especially when viewed from far away).

`"string volumeinterpretation" ["discrete"]`
: Specifies whatever the deep shadow map will be used to compute shadows cast by solid objects or those cast by participating media (fog, clouds, smoke, etc). When rendering participating media shadows, one should specify `"continuous"` to this parameter. Default is '`discrete`'.

EXAMPLE

```
# Write out a deep shadow map for partictipating media
Display "shadow.dsm" "dsm" "rgbaz"
    "string volumeinterpretation" ["continuous"]
```

*Do not run* `tdlmake` *on files produced by this display driver*, since they are already in the right format.

## 7.6 Encapsulated Postsript display driver

Use this display driver to produce '`.eps`' files. Those are Poscript files that can be used inside '`.pdf`' and '`.ps`' files. The alpha channel is ignored in an '`.eps`' file. Only one option is supported:

`"uniform string fullimage" "off"`
: When using a crop window, this parameter enables the user to create an image of the original size, with the rendered crop window placed inside. Areas that are outside the crop window will be set to black. The default value is '`off`'.

## 7.7 Kodak Cineon display driver

This display driver allows you to write Kodak's Cineon files. Each channel is encoded in 10bits using a logarithmic scale (regardless of `Exposure` and `Quantize`). Such an encoding is appropriate for film recording since it is designed to cover film's recordable color range. A color encoded using the Cineon format can be much brighter than white, therefore it is recommended to use unquantized values for this display driver to avoid clipping the colors to (1,1,1).

The alpha channel is supported by this display driver but it is not guaranteed to be readable by other software. We suggest to use RGB (no alpha) for maximum portabilty.

Three parameters are recognised by this display driver:

`"uniform integer setpoints[2]" [ref_white ref_black]`
: This sets the reference black and the reference white values. The default values are 180 for reference black and 685 for reference white;

`"uniform integer ref_white"` *value*
: Only sets the reference white;

`"uniform integer ref_black"` *value*
: Only sets the reference black;

An explanation of those parameters can be found in the Cineon documentations available on the web. The standard extension for a Cineon file is '`.cin`'.

## 7.8 Radiance display driver

Use this display driver to write files compatible with Radiance. Floating point values are required for this display driver since it is intended as a HDRI output. The following parameter is recognised by this display driver:

`"uniform string colorspace" ["rgb"]`
: Specifies in which color space to encode the values. The two possible values are `"rgb"` and `"xyz"` and the default is `"rgb"`. Be careful when using `"xyz"` color space since it implies a RGB to XYZ conversion inside the display driver and RGB values are assumed to be **linear**, so setting `Exposure` to 1 1 is essential. Note that run-length encoding of Radiance files is not supported.

EXAMPLE

```
# Write out a radiance file, using XYZ colorspace.
Exposure 1 1
Quantize "rgb" 0 0 0 0  # use floating point data
Display "hdri.pic" "radiance" "rgb" "string colorspace" "xyz"
```

## 7.9 OpenEXR display driver

This display driver[3] writes out ILM's OpenEXR files using the '`half`' floating point format. Floating point input is requested. Only one parameter is supported:

`"uniform string compression" ["zip"]`
: Specifies which compression to use. Valid compression schemes are:

  `"none"` Disable compression;

  `"zip"` Enable '`deflate`' compression using the '`zlib`' library. This is the default compression scheme;

  `"rle"` Enable run-lenght encoding. Compression ratio is good only on images with large constant color areas. Use only if compression or decompression speed matters;

  `"piz"` Data is compressed using a huffman algorith applied to wavelet transformed data. Apparently provides good compression ratios on photographic data.

[3] Only supported on Linux and IRIX platforms.

# 8 Developer's Corner

## 8.1 Linking with 3Delight

3Delight comes with a library which implements the RenderMan API interface. Here is how to link with it on different platforms.

'IRIX' `CC -o main main.cpp -I$DELIGHT/include -L$DELIGHT/lib/ -L/lib/i686 -l3delight -lm -ldl -lc`

'Linux' `g++ -o main main.cpp -I$DELIGHT/include -L$DELIGHT/lib/ -L/lib/i686 -l3delight -lm -ldl -lc`

'Windows' `cl /I%DELIGHT%/include %DELIGHT%/lib/3delight.lib main.cpp`

'MacOS X' `g++ -o main main.cpp -I$DELIGHT/include -L$DELIGHT/lib -framework CoreFoundation -lstdc++ -l3delight`

## 8.2 Writing Display Drivers

3Delight comes with a set of standard display drivers that are suitable for most applications (see Chapter 7 [Display Driver System], page 62). However, it is possible to write custom display drivers if some specific functionality is needed. Basically, a display driver is a DSO (DLL under Windows) which implements an interface that 3Delight understends. This interface, along with all the data types used, is described in the '`$DELIGHT/include/ndspy.h`' header file and is further investigated in the following sections.

### 8.2.1 Required Entry Points

A display driver must implement four mandatory entry points:

`PtDspyError` **DspyImageQuery** ( `PtDspyImageHandle` *image*, `PtDspyQueryType` *type*, `size_t` *size*, `void *`*data* )
Queries the display driver about format informations.

`PtDspyError` **DspyImageOpen** ( `PtDspyImageHandle *` *image*, `const char *`*drivername*, `const char *`*filename*, `int` *width*, `int` *height*, `int` *paramcount*, `const UserParameter *`*parameters*, `int` *formatcount*, `PtDspyDevFormat *`*format*, `PtFlagStuff *`*flagsstuff*)
Opens a display driver.

`PtDspyError` **DspyImageData** ( `PtDspyImageHandle` *image*, `int` *xmin*, `int` *xmax_plus_one*, `int` *ymin*, `int` *ymax_plus_one*, `int` *entrysize*, `const unsigned char *`*data* )
Sends data to display driver.

`PtDspyError` **DspyImageClose** ( `PtDspyImageHandle` *image* )
Close the display driver.

An optional entry point is also defined:

`PtDspyError` **DspyImageDelayClose** ( `PtDspyImageHandle` *image* )
Close the display driver in a separate process.

Every function is detailed in the following sections.

## DspyImageQuery

This function is called for two reasons:

1. 3Delight needs to know the default resolution of the display driver. This may happen if the user did not call `Format`;
2. 3Delight needs to know whether the display driver overwrites or not the specified file (not used at the moment).

Parameters are:

*type* Can take two values: `PkOverwriteQuery` and `PkSizeQuery`. For each query, a different data structure needs to be filled. The structures are declared in '`$DELIGHT/include/ndspy.h`'.

*size* Maximum size of the structure to fill;

*data* A pointer to the data to fill. Copy the appropriate structure here.

See Section 8.2.2 [display driver example], page 71.

## DspyImageOpen

Called before rendering starts. It is time for the display driver to initialize data, open file(s), ....

Here is a description of all the parameters passed to this function.

*image* This is an opaque pointer that is not used in any way by 3Delight. It should be allocated and used by the display driver to pass informations to `DspyImageData` and `DspyImageClose`. For instance, a TIFF display driver would put some useful informations about the TIFF during `DspyImageOpen` so that `DspyImageData` could access the opened file.

*drivername*
Gives the device driver name as specified by `Display`. For example:

```
Display "super_render" "framebuffer" "rgb"
```

will provide '`framebuffer`' in *drivername*;

*filename* Gives the filename provided in the `Display` command. For example:

```
Display "render.tif" "tiff" "rgb"
```

will provide '`render.tif`' in *filename*;

*width*
*height* Give the resolution of the image, in pixels. If the image is cropped, *width* and *height* will reflect the size of the *cropped* window;

*paramcount*
Total number of user parameters provided in this call;

*UserParameter*
An array of user parameters, of size *paramcount*. `UserParameter` is defined as:

```
typedef struct
{
const char *name;
char valueType, valueCount;
const void *value;
int nbytes;
} UserParameter;
```

*name* is the name of the parameter, *valueType* is its type, which can be one of the following: '`i`' for an integer type, '`f`' for an IEEE floating point type and '`s`' for a string type. *valueCount* is used for parameters that have more than one value, such as matrices and arrays. *value* is the pointer to the actual data. A set of standard parameters is alway provided, those are described in the table below.

*formatcount*
Number of output channels.

*formats* An array of channel descriptions of size *formatcount*. A channel description contains a name and a type:

```
typedef struct
{
char *name;
unsigned type;
} PtDspyDevFormat;
```

Parameters can be passed to a display driver when issuing the `Display` command:

```
Display "render" "my_display" "rgb" "string compression" "zip"
```

In this case, '`my_display`' driver will receive the parameter "compression".

| Name | Type | Count | Comments |
|---|---|---|---|
| NP | 'f' | 16 | World to Normalized Device Coordinates (NDC) transfom |
| Nl | 'f' | 16 | World => Camera transform |
| near | 'f' | 1 | Near clipping plane, as declared by `Clipping` |
| far | 'f' | 1 | Far clipping plane, as declared by `Clipping` |
| origin | 'i' | 2 | Crop window origin in the image, in pixels |
| OriginalSize | 'i' | 2 | Since *width* and *height* only provide to `DspyImageOpen` reflect the size of the croped window, this variable gives the original, uncropped window size |
| PixelAspectRatio | 'f' | 1 | Pixel aspect ratio as given by `Format` |
| Software | 's' | 1 | Name of the rendering software: "3Delight" |

Default user parameters passed to `DspyImageOpen()`.

## DspyImageData

3Delight will call this function when enough pixels are available for output. Most of the time this will happen after each rendered bucket. However, if `DspyImageOpen` asks for scanline data ordering, a call to this function will be issued when a *row* of buckets is rendered.

*image* Opaque data allocated in `DspyImageOpen`

*xmin*
*xmax_plus_one*
*ymin_plus_one*
*ymax_plus_one*
Screen coordinates containing provided pixels data;

*entrysize* Size, in bytes, of one pixel. For example, if 8 bit RGBA data was asked, entrysize will be set to 4.

*data* Pointer to the actual data, organized in row major order.

## DspyImageClose

Called at the end of each rendered frame. It is time to free all resources that were used and deallocate *image* (which was allocated by `DspyImageOpen`).

## DspyImageDelayClose

If this entry point is defined in the display driver, it will be called instead of `DspyImageClose()` with the difference being that the call will occur in a separate process so that 3Delight can exit without waiting for the display driver. The '`framebuffer`' display driver uses this functionality (see Section 7.1 [framebuffer], page 62).

### 8.2.2 A Complete Example

```
/*
    Copyright (c)The 3Delight Team.
    All Rights Reserved.
*/

//
// = LIBRARY
//     3delight
// = AUTHOR(S)
//     Aghiles Kheffache
// = VERSION
//     $Revision: 1.4 $
// = DATE RELEASED
//     $Date: 2003/06/30 19:30:17 $
// = RCSID
//     $Id: zfile.cpp,v 1.4 2003/06/30 19:30:17 aghiles Exp $
//

#include <ndspy.h>
#include <uparam.h>

#include <assert.h>
#include <stdio.h>
#include <float.h>
#include <string.h>
#include <limits.h>

/* ZFile Display Driver Implementation */

const unsigned kDefaultZFileSize = 512;

/*
zfile format:
    zFile format is (matrices and image are row-major):
    magic # (0x2f0867ab)                                    (4 bytes)
    width (short)                                           (2 bytes)
    height (short)                                          (2 bytes)
    shadow matrices (32 floats, 16 for NP and 16 for Nl)  (128 bytes)
    image data (floats)                        (width*height*4 bytes)

    NOTE
    Matrices are stored in row major format.
*/
class zFile
{
```

```
public:
    zFile(
        const char* fileName,
        const float* np, const float* nl,
        unsigned short width, unsigned short height )

    : m_file(0x0), m_width(width), m_height(height),
    m_currentLine(0), m_pixelsLeftOnLine(width)
    {
        m_file = fopen( fileName, "wb" );

        if( m_file )
        {
            unsigned long magic = 0x2f0867ab;

            assert( sizeof(long) == 4 );

            fwrite( &magic, 4, 1, m_file );
            fwrite( &m_width, sizeof(m_width), 1, m_file );
            fwrite( &m_height, sizeof(m_height), 1, m_file );
            fwrite( np, sizeof(float), 16, m_file );
            fwrite( nl, sizeof(float), 16, m_file );
        }
    }

    ~zFile()
    {
        if( m_file )
        {
            fclose(m_file);
        }
    }

    bool Valid() const { return m_file != 0x0; }

    unsigned GetWidth() const {return m_width;}
    unsigned GetHeight() const {return m_height;}

    bool WriteScanline(
        unsigned short y, unsigned short size, const float* data )
    {
        if( y != m_currentLine || size > m_pixelsLeftOnLine )
        {
            return false;
        }

        m_pixelsLeftOnLine -= size;
```

```
        if( m_pixelsLeftOnLine == 0 )
        {
            ++m_currentLine;
            m_pixelsLeftOnLine = m_width;
        }

        return fwrite( data, sizeof(float), size, m_file ) == size;
    }

private:
    FILE* m_file;
    unsigned short m_width;
    unsigned short m_height;

    unsigned short m_currentLine;
    unsigned short m_pixelsLeftOnLine;
};

/*
    A utility function to get user parameters ...
*/
const void* GetParameter(
    const char *name,
    unsigned n,
    const UserParameter parms[] )
{
    for( unsigned i=0; i<n; i++ )
    {
        if(0 == strcmp(name, parms[i].name))
        {
            return parms[i].value;
        }
    }
    return 0x0;
}

/*
    Open
*/
PtDspyError DspyImageOpen(
    PtDspyImageHandle *i_phImage,
    const char *i_drivername,
    const char *i_filename,
    int i_width,  int i_height,
    int i_parametercount,
    const UserParameter i_parameters[],
```

```
    int i_numFormat,
    PtDspyDevFormat i_format[],
    PtFlagStuff *flagstuff )
{
    int i;
    bool zfound = false;

    const float* nl =
        (float*)GetParameter( "Nl", i_parametercount, i_parameters );

    const float* np =
        (float*)GetParameter( "NP", i_parametercount, i_parameters );

    /* Loop through all provided data channels and only ask for the 'z'
       channel. */

    for( i=0; i<i_numFormat; i++ )
    {
        if( strcmp(i_format[i].name, "z") != 0 )
        {
            i_format[i].type = PkDspyNone;
        }
        else
        {
            i_format[i].type = PkDspyFloat32;
            zfound = true;
        }
    }

    if( !zfound )
    {
        fprintf( stderr, "dspy_z : need 'z' in order to proceed.\n" );
        return PkDspyErrorUnsupported;
    }

    if( !nl || !np )
    {
        fprintf(
            stderr,
            "dspy_z : need Nl & Np matrices in order to proceed. bug.\n" );
        return PkDspyErrorBadParams;
    }

    if (i_width > USHRT_MAX || i_height > USHRT_MAX)
    {
        fprintf(
            stderr,
```

```
            "dspy_z : image too large for zfile format" \
            " (use shadowmap ddriver).\n" );
        return PkDspyErrorUndefined;
    }

    zFile* aZFile = new zFile( i_filename, np, nl, i_width, i_height );

    if( !aZFile || !aZFile->Valid() )
    {
        fprintf(
            stderr,
            "dspy_z : cannot create file" \
            "(permissions ? free disk space ?).\n" );

        delete aZFile;
        return PkDspyErrorNoResource;
    }

    *i_phImage = (void*) aZFile;

    /* Ask display manager to provide data scanline by scanline
    */
    flagstuff->flags |= PkDspyFlagsWantsScanLineOrder;

    return PkDspyErrorNone;
}

/*
    DspyImageQuery
*/
PtDspyError DspyImageQuery(
    PtDspyImageHandle i_hImage,
    PtDspyQueryType i_type,
    size_t i_datalen,
    void *i_data )
{
    zFile *aZFile = (zFile*) i_hImage;

    if( !i_data )
    {
        return PkDspyErrorBadParams;
    }

    switch( i_type )
    {
        case PkSizeQuery:
        {
```

```
            PtDspySizeInfo sizeQ;

            if( aZFile )
            {
                sizeQ.width = aZFile->GetWidth();
                sizeQ.height = aZFile->GetHeight();
                sizeQ.aspectRatio = 1;
            }
            else
            {
                sizeQ.width = kDefaultZFileSize;
                sizeQ.height = kDefaultZFileSize;
                sizeQ.aspectRatio = 1;
            }

            memcpy(
                i_data, &sizeQ,
                i_datalen>sizeof(sizeQ) ? sizeof(sizeQ) : i_datalen );

            break;
        }

        case PkOverwriteQuery:
        {
            PtDspyOverwriteInfo overwQ;

            overwQ.overwrite = 1;

            memcpy(
                i_data, &overwQ,
                i_datalen>sizeof(overwQ) ? sizeof(overwQ) : i_datalen );

            break;
        }

        default:
            return PkDspyErrorUnsupported;
    }

    return PkDspyErrorNone;
}


/*
    DspyImageData

    Data is expected in scanline order (as asked in DspyImageOpen()).
```

```
*/
PtDspyError DspyImageData(
    PtDspyImageHandle i_hImage,
    int i_xmin, int i_xmax_plusone,
    int i_ymin, int i_ymax_plusone,
    int i_entrySize,
    const unsigned char* i_data )
{
    zFile* aZFile = (zFile*) i_hImage;
    const float* fdata = (const float*) i_data;

    if( !aZFile || !fdata )
    {
        return PkDspyErrorBadParams;
    }

    /* Perform some sanity checks but everything should be fine really ...
       :> */

    if( i_ymax_plusone - i_ymin > 1 ||
        i_xmin != 0 ||
        i_xmax_plusone != aZFile->GetWidth() ||
        i_entrySize != sizeof(float) )
    {
        return PkDspyErrorBadParams;
    }

    if( !aZFile->WriteScanline(i_ymin, i_xmax_plusone - i_xmin, fdata) )
    {
        return PkDspyErrorNoResource;
    }

    return PkDspyErrorNone;
}

/*
    DspyImageDelayClose

    Not used by 3Delight yet.
*/
PtDspyError DspyImageDelayClose( PtDspyImageHandle i_hImage )
{
    return DspyImageClose( i_hImage );
}

/*
    DspyImageClose
```

```
    delete our object.
*/
PtDspyError DspyImageClose( PtDspyImageHandle i_hImage )
{
    zFile* aZFile = (zFile*) i_hImage;

    if( !aZFile )
    {
        return PkDspyErrorUndefined;
    }

    delete aZFile;

    return PkDspyErrorNone;
}
```

### 8.2.3 Compilation Directives

Here is the compilation command line for the given example ('`zfile.cpp`'):

**Linux** '`g++ -shared -o zfile.so -I$DELIGHT/include zfile.cpp`'

**IRIX** '`CC -shared -o zfile.so -I$DELIGHT/include zfile.cpp`'

**MacOS X** '`g++ -o zfile.so -I$DELIGHT/include -arch "ppc" -bundle zfile.cpp`'

**Windows** '`cl -I%DELIGHT%/include -LD zfile.cpp`'

## 8.3 DSO Shadeops

It is possible to extend the capabilities of the shading language by calling C or C++ functions from inside shaders. When compiling a shader, if the compiler encounters a function it doesn't now, it will automatically search all the directories specfied by the '`-I`' command line option[1] looking for a DSO containing a definition of the unknown function.

A DSO must contain a data table for each shadeop it implements. One such table simply describes the possible return values of the shadeops and its init and cleanup functions.

All this is better explained by an example:

```
#include "shadeop.h"
#include <stdio.h>

/*
  A simple DSO shadeops.
```

[1] All compiler's command line options are specified in Section 3.2 [Using the shader compiler], page 9.

```
  Notes that 'extern "C"' is not necessary for '.c' files.
  Only c++ files need that.
*/

extern "C" {
        SHADEOP_TABLE(sqr) =
        {
                {"float sqr(float)", "sqr_init", "sqr_cleanup"},
                {"point sqr_p(point)", "sqr_init", "sqr_cleanup"},
                {""}
        };
}

extern "C" SHADEOP_INIT(sqr_init)
{
        return 0x0; /* No init data */
}

/*
        returns the given float, squared.

        NOTES
        - argv[0] contains a pointer to the result, in this case a float.
        -
*/
extern "C" SHADEOP( sqr )
{
        float *result = (float *)argv[0];
        float f = *((float*)argv[1]);
        *result = f * f;

        return 0;
}

/*
        returns the given point, squared.
*/
extern "C" SHADEOP(sqr_p)
{
        float *result = (float *)argv[0];
        float *f = ((float*)argv[1]);

        result[0] = f[0] * f[0];
        result[1] = f[1] * f[1];
        result[2] = f[2] * f[2];
```

```
        return 0;
}

extern "C" SHADEOP_CLEANUP( sqr_cleanup )
{
        /* Nothing to do */
}
```

Here is how to compile a DSO under differnt environments:

**Linux** g++ -shared -o sqr.dso $DELIGHT/include sqr.cpp

**IRIX** CC -shared -o sqr.dso $DELIGHT/include sqr.cpp

**Windows** cl -I%DELIGHT%/include -LD sqr.cpp

To avoid memory leaks, use the macro `ASSIGN_STRING` (declared in '`shadeop.h`' code to copy strings. This macro will delete (if needed) the old string before assigning the new one. Here is an example that illustrates how to use the macro:

```
SHADEOP_TABLE(test) =
{
    {"string teststring(string)", "", ""},
    {""}
};

extern "C" SHADEOP(teststring)
{
    const char **res = (const char **)(argv[0]);
    const char **arg1 = (const char **)(argv[1]);

    ASSIGN_STRING(*res, "Hello");
    ASSIGN_STRING(*arg1, "World");
}
```

## 8.4 Writing Procedural Primitives

This section is meant to provide examples on how to write simple procedural primitives for 3Delight. For more complete documentation about procedural primitives and the interface to them, refer to the RenderMan Interface Specification available at https://renderman.pixar.com/products/rispec/index.htm.

### 8.4.1 The RunProgram Procedural Primitive

This example shows how to write a procedural primitive in the form of an external program which generates RIB. It can be invoked in a RIB as such: `Procedural "RunProgram" ["sphere" "0 0.5 1"] [-1 1 -1 1 -1 1]` The 3 floats it receives as parameters represent a color. The program itself outputs a sphere of that color. This example also shows how to generate RIB using lib3delight. Note that on Windows 3Delight will search for `sphere.exe` if it doesn't find `sphere` in its procedural search path.

Note that a poorly written program (especially one which fails to output the \377 delimiter) may easily hang the renderer. Great care was taken in 3Delight to check for the

most common errors but there are still some which are not caught. It also important that your program be written to accept multiple requests.

```
#include <stdio.h>
#include <stdlib.h>

#include "ri.h"

int main(int argc, char **argv)
{
    char buf[256];

    /*
      You can still use the standard error output to diagnose your program.
      This allows you to see that your program is started only once even if it
      is invoked several times in a frame.
    */
    fprintf(stderr, "diagnostic: sphere program started.\n");

    /*
      Requests from the renderer to the program are passed on the standard
      input. Each request is written on a single line. The detail level required
      is written first, followed by the arguments passed by the user in the RIB.
      The two are separated by a single space.
    */
    while (fgets(buf, 256, stdin) != NULL)
    {
        RtFloat detail;
        RtColor color = {1.0f, 1.0f, 1.0f};
        sscanf(buf, "%g %g %g %g", &detail, &color[0], &color[1], &color[2]);

        /*
          Calling RiBegin with a file name as a parameter causes the commands
          to be output as RIB to that file when using lib3delight. Using
                  "stdout" or "stderr" will output the RIB to the standard output or
                  error respectively.

          It is important to call RiBegin()/RiEnd() inside the loop (for each
          request) to ensure that the output is received properly by the
          renderer.
        */
        RiBegin("stdout");

        RiColor(&color[0]);
        RiSphere(1.0f, -1.0f, 1.0f, 360.0f, RI_NULL);

        /*
```

```
          Outputting a single 0xFF character (377 in octal) is the method to
          signal the renderer that the program has finished processing this
          request. This can also be done manually if you choose not to use the
          library to output your RIB.
        */
        RiArchiveRecord(RI_VERBATIM, "\377");

        RiEnd();
    }

    fprintf(stderr, "diagnostic: sphere program is about to end.\n");
    return 0;
}
```

This example can be compiled with the following commands:

**Linux** g++ -o spher -O3 -I$DELIGHT/include/ -L$DELIGHT/lib/ sphere.cpp -l3delight

**Mac OS X** g++ -o sphere -O3 -I$DELIGHT/include/ -L$DELIGHT/lib/ sphere.cpp -l3delight

**IRIX** CC -o sphere -O3 -n32 -TARG:isa=mips4 -TARG:processor=r10k -I$DELIGHT/include/ -L$DELIGHT/lib/ sphere.cpp -l3delight

**Windows** CL /Ox /I%DELIGHT%\include\ sphere.cpp %DELIGHT%\lib\3Delight.lib

### 8.4.2 The DynamicLoad Procedural Primitive

This example shows how to write a procedural primitive in the form of a DSO. It is special in that it calls itself a number of times to create a Menger's Sponge. It can be invoked in a RIB as such: `Procedural "DynamicLoad" ["sponge" "3"] [-1 1 -1 1 -1 1]` The single parameter it takes is the maximal recursion depth. It also makes use of the `detailsize` parameter of the subdivision routine to avoid outputting too much detail. Note that 3Delight will attempt to find `sponge.so` (or `sponge.dll` on Windows) and then `sponge` in the procedural search path. This allows you not to specify the extension.

```
#include <stdlib.h>

#include "ri.h"

#if defined(_WIN32)
#define DLLEXPORT __declspec(dllexport)
#else
#define DLLEXPORT
#endif

#ifdef __cplusplus
extern "C" {
#endif
```

```
/* Declarations */
RtPointer DLLEXPORT ConvertParameters(RtString paramstr);
RtVoid DLLEXPORT Subdivide(RtPointer data, float detail);
RtVoid DLLEXPORT Free(RtPointer data);

RtPointer DLLEXPORT ConvertParameters(RtString paramstr)
{
    int* depth = (int*) malloc(sizeof(int));
    *depth = 3;                 /* decent default value */
    sscanf(paramstr, "%d", depth);
    return depth;
}

RtVoid DLLEXPORT Subdivide(RtPointer blinddata, RtFloat detailsize)
{
    int depth = *(int*) blinddata;

    /* Simple usage of detailsize to avoid drawing too much detail */
    if (depth <= 0 || detailsize <= 5.0f)
    {
        /* Draw a cube */
        RtInt nverts[] = {4, 4, 4, 4, 4, 4};
        RtInt verts[] = {
            3, 7, 6, 2,         /* top face    */
            5, 1, 0, 4,         /* bottom face */
            7, 3, 1, 5,         /* back face   */
            3, 2, 0, 1,         /* left face   */
            6, 7, 5, 4,         /* right face  */
            2, 6, 4, 0};        /* front face  */

        RtFloat points[] = {
            -1, -1, -1,      -1, -1, 1,
            -1, 1, -1,       -1, 1, 1,
            1, -1, -1,       1, -1, 1,
            1, 1, -1,        1, 1, 1};
        RiPointsPolygons(
            (RtInt)6, nverts, verts, RI_P, (RtPointer)points, RI_NULL);
    } else {
        /* Recursive call, reduce depth and scale the object by 1/3 */
        RtBound bound = {-1, 1, -1, 1, -1, 1};
        int* newDepth;
        unsigned x,y,z;
        RiScale(1.0/3.0, 1.0/3.0, 1.0/3.0);

        for (x = 0; x < 3; ++x)
            for (y = 0; y < 3; ++y)
```

```
                for (z = 0; z < 3; ++z)
                    if (x % 2 + y % 2 + z % 2 < 2)
                    {
                        RiTransformBegin();
                        RiTranslate(
                            x * 2.0 - 2.0,
                            y * 2.0 - 2.0,
                            z * 2.0 - 2.0);
                        newDepth = (int*) malloc(sizeof(int));
                        *newDepth = depth - 1;
                        /* We could make the recursive call using
                           RiProcDynamicLoad but that would be more complex and
                           slightly less efficient */
                        RiProcedural(newDepth, bound, Subdivide, Free);
                        RiTransformEnd();
                    }
    }
}

RtVoid DLLEXPORT Free(RtPointer blinddata)
{
    free(blinddata);
}

#ifdef __cplusplus
}
#endif
```

This example can be compiled with the following commands:

**Linux** gcc -o sponge.so -O3 -I$DELIGHT/include/ -shared sponge.c

**Mac OS X** mkdir -p sponge.so/Contents/MacOS ; gcc -o sponge.so/Contents/MacOS/sponge -O3 -I$DELIGHT/include/ -L$DELIGHT/lib/ -arch "ppc" -bundle sponge.c -l3delight

**IRIX** CC -o sponge.so -O3 -n32 -TARG:isa=mips4 -TARG:processor=r10k -I$DELIGHT/include/ -shared sponge.c

**Windows** CL /Ox /LD /I%DELIGHT%\include\ sponge.c %DELIGHT%\lib\3Delight.lib

# 9 Acknowledgement

We would like to thank Moritz Moeller, Graeme Nattress, Jason Belec, Frederick Gustafsson, Yu Umebayashi, Daniel Moreno, Goran Kocov, Brian Perry and John McDonald for their very helpfull suggestions and bug reports. Also, thanks to Robert Coldwell, the creator of the wonderfull 3Delighter and Ming Mah for the MacOS X framebuffer. And last, but not least, Jean-Jacques Maine, always in the front line when it comes to 3Delight testing.

# 10 Copyrights and Trademarks

3Delight is RenderMan compliant in that it reads RenderMan RIB files and implements the RenderMan API.

```
The RenderMan (R) Interface Procedures and Protocol are:
Copyright @copyright{} 1988, 1989, Pixar. All Rights Reserved.
RenderMan (R) is a registered trademark of Pixar.
```

3Delight uses the libtiff library to read and write TIFF files. The libtiff license requires the following statements to appear in the documentation of the software:

```
The libtiff library is:
Copyright @copyright{} 1988-1997 Sam Leffler
Copyright @copyright{} 1991-1997 Silicon Graphics, Inc.

Permission to use, copy, modify, distribute, and sell the libtiff library
and its documentation for any purpose is hereby granted without fee,
provided that (i) the above copyright notices and this permission notice
appear in all copies of the software and related documentation, and (ii)
the names of Sam Leffler and Silicon Graphics may not be used in any
advertising or publicity relating to the software without the specific,
prior written permission of Sam Leffler and Silicon Graphics.
```

The Windows version of 3Delight uses the regex library. The regex license requires the following statements to appear in the documentation of the software:

```
The regex library is:
Copyright @copyright{} 1992 Henry Spencer.
Copyright @copyright{} 1992, 1993 The Regents of the University of California.
All rights reserved.

The regex library code is derived from software contributed to Berkeley by
Henry Spencer of the University of Toronto. Redistribution and use in source
and binary forms, with or without modification, are permitted provided that
the following conditions are met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software
must display the following acknowledgement:
```

3Delights uses ILM's OpenEXR library to read OpenEXR files. The license requires the following to appear in the software:

```
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

# Concept Index

## 3



## A



## B



## C



## D



## E

## F



## G



## H



## I



## J



## L



## M



## N



## O



## P

## Q



## R



## S



## Z





## T



## U



## V



## W

# Function Index

## *



## A



## C



## D



## E



## F



## I



## L



## M



## N



## O



## P



## R

## S



## T



## V



## X



## Y



## Z