Imagin Raytracer

User guide

Version 0.1.1

Olivier FERRAND - December, 2010

Table of contents

1. Introduction

This document is the user guide for the compilation, the installation and the use of Imagin Raytracer software and API.

2. Installation

For the users who got the archive with the source code files, they have to configure their compilation environment, compile the code and install the software.

For the others who got a pre-compiled version, they just have to unzip the archive in the installation directory.

2.1 Requirements for compilation

To compile Imagin Raytracer sources, the required environment includes :

- Posix compliant environment (Linux, Unix, Mac OS X for example).
- C/C++ compilers
- `make` tool.
- `autoconf` and `automake` tools.
- `lex` (or `flex`) lexical scanner.
- `yacc` (or `bison`) grammatical analyzer.

Optional requirements :

Without additional external library, Imagin manages only the BMP file format (for input and output images). If the PNG and Z libraries are installed, Imagin will also handle the PNG file format and it will be the default format.

- libpng 1.4.4 (library and header files)
- libz 1.2.5 (library and header files)

2.2 Compilation and installation

First, unpack the archive : copy the file in the target directory and unzip it :

```
tar xfz imagin-raytracer-0.1.1.tar.gz
```

Enter into the directory `imagin-raytracer-0.1.1` :

```
cd imagin-raytracer-0.1.1
```

Configure the compilation environment. For a default environment, type the command :

```
./configure
```

Read the `INSTALL` file in the `src` directory to see more details about the configuration of the environment. By default the installation directory is under `/usr/local` but you can specify another one with the option `--prefix`, for example :

```
./configure --prefix=/home/imagin/
```

It is also possible to configure the compilation environment with special optimizations depending on the model of microprocessor (see the documentation of both `autoconf` and compiler for more details).
For example, better performances for all microprocessors can be obtained with these parameters :

```
./configure CXXFLAGS="-O3"
```

If the configuration is ok, start the compilation :

```
make
```

If there is no error, install the software :

```
make install
```

It installs the executable file and the API which is composed by a library file and many header files. The API allows the use Imagin Raytracer's classes into another product.

For Windows users who compile under MinGW, it is necessary to use an external Posix threads library for win32, for example Pthreads-w32 at this address http://sourceware.org/pthreads-win32/index.html which provide header files and dynamic library.

2.3 Environment variables

These environment variables can be set :

- `IMAGIN_OUT` : output directory for image files. If the variable is not set, it is the current directory by default.
- `IMAGIN_PATH` : the inclusion directories list (for imported scene files, bitmaps and fonts files). Directories are separated with `";"` or `":"` character.

EXAMPLE

```
export IMAGIN_OUT="/usr/local/share/imagin-raytracer/scenes"

export IMAGIN_PATH="/usr/local/share/imagin-raytracer/scenes/include;/
usr/local/share/imagin-raytracer/scenes/maps"
```

By default, the linker uses the dynamic version of the optional libraries (libpng and libz). If necessary, you should add the path of those libraries in the LD_LIBRARY_PATH variable before running Imagin.

2.4 Directories

The installation creates these directories, under the root directory (by default `/usr/local/`, unless another is specified with the option `--prefix`) :

Directory	Description
`bin`	Location of the executable file `imagin` (or `imagin.exe` under Windows)
`lib`	Location of the API library file `libmagin.a`
`include`	Location of the API header files `img_*.h`
`share/imagin-raytracer/doc`	Documentation files

share/imagin-raytracer/scenes	Scenes sample files

3. Using Imagin Raytracer

3.1 Syntax

For rendering a scene, execute `imagin` command with the scene file as a parameter.

SYNTAX

```
imagin <scene_file> [options]
```

EXAMPLE

```
$ imagin scene.img

Scene:
  Scene script.............: scene.img
  Objects..................: 49
  Primitives...............: 49
  Lights...................: 1
  Medias...................: 1
  Background...............: Yes
Image:
  Output file(s)...........: scene.png
  Dimension................: 400x300 pixels
  Exposure adjustment......: No
Renderer:
  Threads number...........: 1
  Recursion max............: 5
  Anti aliasing............: No
  Shadow blur..............: No
  Depth of field blur......: No
  Motion blur..............: No
  Hierarchical bounding boxes.: No
  Photons mapping..........: No

Rendered "scene.img" : output file "scene.png"

Parse time...............: 0.01 s
Render time..............: 0.57 s
Pixels rendered/s........: 209146
Total user time..........: 0.60 s
```

3.2 Options

Option	Description	Default
`-w=<width>`	Image width in pixels	400
`-h=<height>`	Image height in pixels	300
`-area=<x1,y1:x2,y2>`	Rendered area coordinates in pixels	(0,0 : width-1, height-1)
`-pixel=<x,y>`	Render only one pixel	-
`-adjust_exposure`	Normalize the exposure of the image	-
`-image_format=<format>`	The format of the output image file (bmp, png24 or png48). png24 is a PNG format with 8 bits by color component and png48 is a PNG format with 16 bits by color component.	png24 (or bmp if pnglib not installed)

-t=<number>	Number of threads (1-16)	1
-r=<level>	Maximum level of recursion (0-16)	5
-aa=<level>	Anti-aliasing level (0-3)	0
-aa_threshold=<value>	Anti-aliasing threshold value (0.0-3.0)	0.3
-tile_width=<width>	Width of an elementary tile of image treated individually by a thread	Depends on image width
-hbb	Enable hierarchical bounding boxes to find objects intersections	-
-shadows=<samples>	Shadows blur samples (1-512)	1
-dof=<samples>	Depth of field blur samples (1-512)	1
-motion=<samples>	Motion blur samples (1-512)	1
-photons=<samples>	Number of thousands of photons casted	0
-r_photons=<level>	Maximum level of recursion for photons (0-16)	5
-photons_map	Generate photons map bitmap file	-
-photons_d=<none|r,h,w>	Distribution function characteristics for diffuse photons (experimental)	none
-photons_c=<none|r,h,w>	Distribution function characteristics for caustic photons (experimental)	none
-clock=<t|t1:t2>	Clock value or clock interval in second	0
-fps=<number>	Number of frame per second for animation	24
-exposure_time=<time>	Exposure time in seconds	0.02
-frame_offset=<offset>	Frame number offset for animation	0
-I<include>	Include directories	-
-o <output_file>	Output image filename	<input_file>.<png | bmp> or <input_file>.avi
-video	Generate a video file instead of a series of bitmap files for animation	-
-p	Parse only	-
-bbox	Draw bounding boxes on final image	-
--help | -h	Print help	-
--version | -v	Print version	-
--license | -l	Print license	-
--quiet | -q	Quiet mode	-

3.3 Output files

3.3.1 Size

The size of the image is set by the options `-w=<width>` and `-h=<height>`. In this image, it is possible to render only an area or a single pixel with the options `-area=<x1,y1:x2,y2>` and `-pixel=<x,y>`.

EXAMPLE

```
$ imagin scene.img -w=800 -h=600
```

3.3.2 Naming

Output files are in BMP (extension `.bmp`) or PNG (extension `.png`) format for bitmap files and AVI (extension `.avi`) for video files. The option `-video` creates an AVI file instead of one or many BMP or PNG file(s).

When a clock interval is specified and the option `-video` is not set, many BMP or PNG files are generated with the frame number added at the name. The numbering starts at 0, unless the option `-frame_offset=<offset>` is set.

By default, the name of the output file is the name of the scene file without extension, concatenated with the extension `.bmp` or `.png` or `.avi`. To change the name of the file, use the option `-o <filename>`.

EXAMPLE

```
$ imagin scene.img -q
Rendered "scene.img" : output file "scene.png"

$ imagin scene.img -q -o output
Rendered "scene.img" : output file "output.png"

$ imagin scene.img -clock=0:1 -fps=5 -q
Rendered "scene.img" : output file "scene_1.png"
Rendered "scene.img" : output file "scene_2.png"
Rendered "scene.img" : output file "scene_3.png"
Rendered "scene.img" : output file "scene_4.png"
Rendered "scene.img" : output file "scene_5.png"

$ imagin scene.img -clock=0:1 -fps=5 -q -video
Rendered "scene.img" : added to file "scene.avi"
Rendered "scene.img" : added to file "scene.avi"
Rendered "scene.img" : added to file "scene.avi"
Rendered "scene.img" : added to file "scene.avi"
Rendered "scene.img" : added to file "scene.avi"

$ imagin scene.img -clock=0:1 -fps=5 -q -video -o output
Rendered "scene.img" : added to file "output.avi"
Rendered "scene.img" : added to file "output.avi"
Rendered "scene.img" : added to file "output.avi"
Rendered "scene.img" : added to file "output.avi"
Rendered "scene.img" : added to file "output.avi"
```

3.3.3 Anti-aliasing

Anti-aliasing is a technique to reduce aliasing by smoothing the contours of the shapes in the image. The idea is to cast several rays per pixel and average their color. It increases the computing time

The anti-aliasing level is set by the option `-aa=<level>`. The level goes from 0 (none) to 3 (maximum).

A pixel is anti-aliased only if the difference of color between itself and his neighbors is greater to a specified threshold. The optional threshold varies from 0 (no difference) to 10 and is set by the option `-aa_threshold=<threshold>`. The default value is 0.3. When the threshold is set to 0, pixels are always anti-aliased.

EXAMPLE

```
$ imagin scene.img -aa=3 -aa_threshold=0.1
```

3.3.4 Illumination

When the scene is calculated, the illumination of the objects is computed as a numerical value in the range between zero and infinity. After that, when the bitmap file is generated, there is a loss of information because the illumination for each color component is coded with a single byte which has a value bounded in the interval [0-255]. To preserve the contrast between high and low illuminated areas in the scene, use the option `-adjust_exposure` to normalize the illumination of the scene before create the bitmap file.

EXAMPLE

```
$ imagin scene.img -adjust_exposure
```

3.4 Included directories

The scripts of the scenes can import other scripts, bitmap files or font files. Imagin Raytracer searches these files, first in the current directory, after in the directories specified by the option `-I` and at last in the directories in the `IMAGIN_PATH` environment variable.

EXAMPLE

```
$ imagin scene.img -I../fonts -I../maps
```

3.5 Recursion level

The recursion level is the maximum number of reflections or refractions allowed for a light ray or a photon. By default this level is limited to 5, but can be modified. It is necessary for some scenes with several reflections or refractions between objects, to increase this level, but it's increasing the computing time too.

With a level of 0, there's no reflection or refraction. With a level of 1, only one reflection or refraction is allowed for each ray or photon.

For the rays (raytracing), the level is set by the option `-r=<level>`. For the photons (photons mapping), this level is set by the option `-r_photons=<level>`.

EXAMPLE

```
$ imagin scene.img -r=8
```

3.6 Number of threads

On computers with multi-cores processors, it is advantageous to render a scene with several threads, each thread computing its own part of the image.

It's not useful to run more threads than the number of cores. It increases the time spent by the system to switch between threads, for lower performances.

EXAMPLE

```
$ imagin scene.img -t=2
```

3.7 Animation

An animation is computed and several frames are rendered when a clock interval is specified by the option `-clock=<t1:t2>` (With the option `-clock=<t>`, only one frame is rendered, for this clock value). One or many output files are generated, depending on the option `-video`.

The number of frames is calculated from the number of frames per second. This value is set by the option `-fps=<frames>`. The default value is 24.

EXAMPLE
```
$ imagin scene.img -clock=0:1 -fps=25
```

3.8 Motion blur

Motion blur is the result of the moving of objects during the exposure time of the camera. The blur effect is generated by rendering several frames. One per clock value in the exposure time interval. To have a blur effect, at least one object must be in motion during this interval, i.e. its position must depend on the clock value.

The number of frames which are rendered to obtain the blur effect is set by the option `-motion=<samples>`. The more the number of frames if high, the more the effect is realistic, but the computing time increases too.

The exposure time is set by the option `-exposure_time=<time>`. Note that contrary to the reality, the exposure time has no effect on the illumination of the scene. It's not possible to overexpose or to underexpose a scene.

EXAMPLE
```
$ imagin scene.img -clock=0 -expose_time=0.1 -motion=32
```

3.9 Shadows blur

Shadows blur is the result of the dimension of light sources. When a source light has a non-null size, a point of the scene can be partially lightened by this source light.

To simulate this effect, several rays must be casted to average the lightening of a point and obtain a blur shadow.

The number of rays casted by light source is set by the option `-shadows=<sample>`. The more the sample number is high, the more the effect is realistic. At least one source light must have a non-null dimension. This effect increases the computing time.

EXAMPLE
```
$ imagin scene.img -shadows=32
```

3.10 Depth of field blur

There is depth of field blur when at least one point of the scene is not in the three-dimensional area of sharp focus for the lens of the camera.

By default, the lens aperture is infinity, there's no depth of field blur. If the lens aperture of the camera is not infinity, a depth of field blur can be simulated by casting several rays and average them. The more the sample number is high, the more the effect is realistic. This effect increases the computing time.

The sample number is set by the option `-dof=<sample>`.

EXAMPLE
```
$ imagin scene.img -dof=32
```

3.11 Hierarchical bounding boxes

Hierarchical bounding boxes is a technique which stores the objects of the scene in a special tree structure called kd-tree, at a node depending on their bounding box. The result is a quick search of intersected objects if they are numerous.

To enable this feature, set the option `-hbb`. It's a very useful feature for the scenes with a large number of objects, but it can increase the computing time for others.

EXAMPLE

```
$ imagin scene.img -hbb
```

3.12 Photons mapping

Photons mapping is a technique that simulates effects that can't be obtained with the classical raytracing technique, including diffuse reflection and caustics.

These effects can be obtained only by computing the illumination starting from sources lights to the camera. With raytracing, the illumination is computed by casting rays from the camera to light sources.

During a first past, a large number of photons are casted from the source lights, their route are computed by application of the optic laws and their final position is stored in a tree structure.

In a second pass, the classical raytracing technique is applied, and the stored photons are used to compute the final illumination. The purpose is to add the contribution of the photons that are "near" the computed point. This notion of proximity is very important and relative. A configurable function of distribution of photons is used.

To enable the photons mapping, use the `-photons=<samples>` with samples number greater than zero. It represents the number of thousands of photons that are casted.

There are two distribution functions : one function for the photons which have a diffuse reflection, and one function for the others which create caustics.

Distribution functions are characterized by :

- Search radius for neighbors photons
- Height of the peak of the distribution function
- Width of the peak of the distribution function

These parameters are set with the two options `-photons_d=<none|r,h,w>` for diffuse photons and `-photons_c=<none|r,h,w>` for caustic photons. None value disable the type of photons.

EXAMPLE

```
$ imagin scene.img -photons_d=none -photons_c=0.2,5,0.1
```