Slartibartfast - user manual

Martin Kahoun

May 14, 2010

Contents

List of Figures

1 Introduction

Slartibartfast is cross-platform procedural planetary body generator written in `C++` using `OpenGL`. It's a demonstration of what can be done with several numbers, Perlin noise and simple fractal function. It basically operates in a way that you supply planet parameters such as radius, maximum height, random seed plus several others that influence output of the Perlin noise and the program will generate a planetary body. You can then rotate with it, zoom it and change the way how it's rendered. Rendering ranges from simple wireframe up to texture blending provided by `GLSL` shaders.

Slartibartfast allows you to save any generated planet as a simple text file containing parameters used for its generation, so you can re-generate the planet later. Apart from this, *Slartibartfast* also provides a functionality for exporting the planet into .obj model format for further usage outside this demo.

1.1 Before you start

Slartibartfast has several prerequisites in order to run smoothly. First make sure that you have a decent graphics card that supports at least `OpenGL 2.0` and thus `GLSL 1.1` shading language, because the shaders are written in it. If your card doesn't support shaders, you will still be able to see what came out of the generator, but won't be able to see the terrain texture.

Second, the program depends on Allegro library supplied with the binary distribution. However the distribution is Windows 32bit only, so if you decide to build it from the sources, you will have to build Allegro and AllegroGL first (both are included in binary and source distribution).

2 Installation

If you are running Windows and don't mind the 32 bit binary that is supplied, feel free to skip this section, because there's no need to install anything. Everything sufficient should be in the `sfast-x.x.x-win32` directory. Just place it somewhere your account has write privileges, because the program creates two more text files upon start. The rest of the section focuses on building the program from sources.

2.1 Prerequisites

In order to successfully compile this program you need to have three libraries present in your system:

- `OpenGL SDK` - should be provided by your gfx card vendor, `MESA3D` is strongly discouraged (however it is shipped along)

- `Allegro 4.2.2` - supplied in the libs directory, do not use `Allegro 5` branch (it uses different API)
- `AllegroGL 0.4.3` - also supplied in the libs directory

Unzip both `Allegro` libraries and build them using instructions provided for your platform, then proceed. On Linux you can find `Allegro` library in your package manager, however `AllegroGL` will seldomly be there and you will have to build it from the source.

Should any problems occur during build, please refer to the section 2.5.

2.2 Linux instructions

Slartibartfast is supplied with makefile. Open it in your favorite editor, make sure that `PLATFORM` variable is set to "`linux`" and then type "`make`" into your shell and you should be fine. If you have `Code::Blocks` installed, you can open up supplied project. Once opened, you'd better check linker options. It should read following (the order is important):

```
-lagl
‘allegro-config --libs‘
-lGL
-lGLU
```

After that, just hit compile and you're done.

2.3 Windows instructions

There should be no pitfalls on Windows hopefully. However build process isn't that straightforward. You have several options. The best bet is to have `Code::Blocks` or `MinGW` compiler suite. However any other compiler or IDE will do as well.

2.3.1 MinGW32

Slartibartfast is supplied with makefile and `MinGW` comes with Windows port of `GNU Make`. So, just make sure that `PLATFORM` variable is set to "`windows`" and then run `make` or `mingw32-make` if you have newer version of the suite and that's it.

2.3.2 Code::Blocks

Open the supplied project, open "Project –> Build options –> Linker settings" and make sure that it reads following:

```
-lagl
-lalleg
-luser32
-lgdi32
-lopengl32
-lglu32
```

Clicking compile button should do the work. Depending on the compiler you are using it's possible that you'll have to change the linker settings a bit, but it's important to have same order. On the other hand, `C::B` comes with `MinGW` suite, so you should have no problem.

2.3.3 Other IDE's

Open up your favorite IDE and create new project. Add all source files in the `src` directory to the project. All header files are contained within `include` directory so make sure you have your project set up in a way, that compiler knows where to find them. When it's all ready, open up build options and add link in following libraries (with the exact order):

- AllegroGL (agl/agld)
- Allegro (alleg/allegd)
- User32 (or 64)
- GDI
- OpenGL
- GLU

Now you can build the project. If you did everything right you should be awarded with a freshly compiled executable. Depending on whether you linked `Allegro` statically or dynamically you need to supply `alleg42.dll` with the binary or copy it into `C:\Windows\System`.

2.4 Other operating systems

Officially they aren't supported (since I don't have any means to test on them). However the code should be portable, which means that if there's port of `OpenGL` and `Allegro` libraries and there's a `C++` compiler you should have no problem with building this project on that platform. If you do, please let me know on my e-mail adress.

2.5 Troubleshooting

Generally if you follow the instructions given inside both `Allegro` libraries you should not run into any problems. Though some circumstances may complicate the build process. Bellow are known issues that may arise during build of `Allegro` libraries or the project itself.

2.5.1 No root privileges

It could happen that you are on a computer without administrator privileges, but you need the compiler to know where the libraries are. Both `Allegro` libraries offer an option to install them locally into your home directory. When doing so, you need to adjust your `PATH` variable to include your home – specific instructions are in build document supplied with the library.

However be sure to build `AllegroGL` library and subsequently the project from the same shell in which you have set up the `PATH` variable, otherwise the compiler won't know about the libraries.

2.5.2 AllegroGL related error

On some compilers you can get cryptic message along the lines:

```
allegrogl/GLext/gl_ext_api.h:1827: error: '<anonymous>' has incomplete type
allegrogl/GLext/gl_ext_api.h:1827: error: invalid use of 'GLvoid'
```

This is known issue on some versions of `g++` compiler, but there's a simple workaround. Just open up the mentioned file and change the line 1827 from:

```
AGL_API(void, EndTransformFeedbackNV,     (GLvoid))
```

to this:

```
GL_API(void, EndTransformFeedbackNV,     (void))
```

This should do the trick.

2.5.3 Outdated OpenGL headers

You can get compiler error message that `glUseProgram()` or any other shader related function wasn't declared. This indicates that your `OpenGL` headers are of version bellow 2.0. Optimally you should get newer version of `OpenGL` distribution, but if that's not an option for you, *Slartibartfast* comes with `OpenGL2.0` headers included inside `GL` directory. On Windows you'd probably need some headers from the `Mesa` library that is included in the `libs` directory.

Just make sure that the compiler looks for the headers into that directory first and includes them at compile time and you should be fine. (If you are using makefile you'll need to add `"-IGL"` to `CCFLAGS` variable.)

3 Usage

Upon the first run `Slartibartfast` will use default settings – running in 640x480x32 windowed mode – and will generate `.ini` file. You can edit this file in order to change the settings. All options should be self-explanatory or explained in the comment above them or in appendix C. One of the important settings is the texture size. The bigger, the longer will it take to generate a planet but the better it will look. You can also set Level Of Detail settings to improve performance.

If anything bad happens or something isn't working as expected you can inspect the `log.txt` that is kept next to the executable and is appended on every run.

Slartibartfast is provided with a sample set of terrain textures contained within `Textures` directory. Do not delete or rename the directory itself or its contents. However you are encouraged to swap default textures with your own as you wish. Replacement texture must be named exactly as the default texture and must be in a `.bmp` format.

3.1 Basics

After the initial planet is generated you will be presented with wireframe view of it with various information printed in top-left and bottom left corners (as seen in figure 1. The default camera (called "homeworld") will be always focused to the center of the planet. You can zoom in and out using your mouse wheel and rotate by holding right mouse button and moving the mouse. By pressing `F1` you can get in-program key overview. You can close the program by pressing `Esc` key.

By pressing `'C'` key you can switch to the "spectator" camera which allows you to look freely around using the mouse (again holding right mouse button). Spectator camera movement is controlled by `'W'`, `'S'`, `'A'`, `'D'` and mouse. You can switch back to "homeworld" camera by pressing `'C'` again.

Key `'L'` toggles dynamic Level Of Detail on/off. Key `'R'` toggles on/off the rotation of the light (visible in non-wireframe view). Keys `'1'`, `'2'`, `'3'`, `'4'` (not the numpad keys, but the numbered keys bellow function keys) switch between wireframe, culled wireframe, cube-mapped gray scale texture and terrain texture rendering. Terrain texture is done via shaders, so it might not be working depending on your graphics hardware.

For review of all useful keys see appendix A.

3.2 Console

Pressing the `'~'` (tilde) key will bring you to simple console. This is the heart of the generator. You can get overview of all commands by typing in

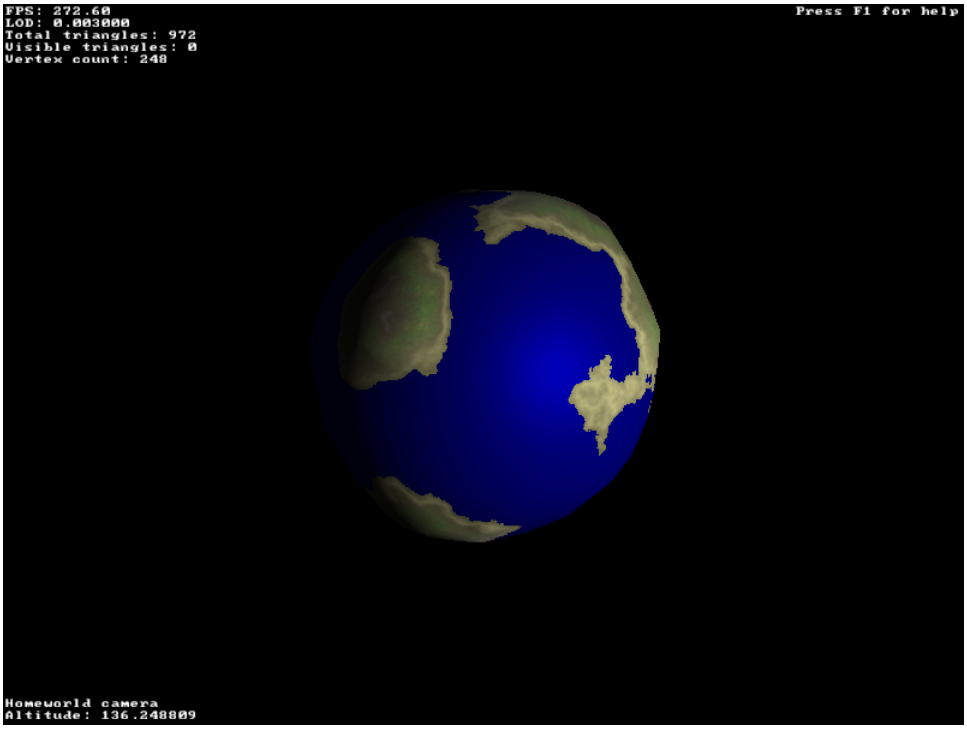

Figure 1: GLSL rendered planet

`cmdlist` and pressing `Enter`. Another helpful command is `help`. If you want to show parameters of the currently displayed planet type in `params`. For all commands see appendix B.

All commands are composed of one word – a combination of alphanumeric characters and other characters except `'~'` (tilde). All arguments are space separated – this disallows filenames or paths with spaces.

3.3 Generating new planet

Typing in `generate` will produce new planet - don't worry if the program becomes unresponsive for a moment (the lag will depend on specified texture size). If you want to review with which parameters will be the new planet generated just type in `nparams`. Changing parameters is quite intuitive. Just type in the name of the parameter and then its new value separated by space. All parameters accept floating point numbers, except `seed` which is integer.

All parameters are listed and explained in appendix B. Most of them should be pretty clear except `w_factor` and `exponent`. They are set by default to 0.5 and 2.0 respectively. Both parameters are tied with the world generator. While the `seed` influences what will come out of Perlin noise, these two define how the values from the noise function will be transformed for more interesting results. This is done with function called fractal Brownian

motion. It is a fractal function that looks like this:

$$w = 1.0$$

$$n = 1.0$$

$$fBm = w \cdot noise(x \cdot n, y \cdot n, z \cdot n)$$

Where w is weight value and n is something called exponent or lacunarity. The function is executed several times to combine results of more than one octave (each value of this function is called octave). Between the runs w and n values are modified by these expressions:

$$w = w \cdot w_{factor}$$

$$n = n \cdot exponent$$

So in a nutshell `w_factor` influences how much from larger octaves will used. And `exponent` influences how smooth he noise will be or how big differences will occur between two close points. Try experimenting with some values and observe the results.

3.4 Saving and loading a planet

If you wish to save currently displayed planet, type `save filename` into the console, where filename is name of the file (including path) in which you want the data stored. Extension is purely up to you. *Slartibartfast* will try to create a file as it is. However it won't give you warning when you're overwriting a file, so be careful! Planets are saved in plain-text, but hand editing them isn't recommended. If you would like to do it, please refer to the development documentation for more info.

Loading is done by typing in `load` followed by a path to the file you want to load. For the time being the program doesn't support file listing. However it will tell you if a failure occurs. You can get more info by inspecting `log.txt`.

3.5 Exporting data into .obj format

You can export any planet into the `.obj` format by typing `export` followed by integer (N) and a filename into console. N is level of subdivision. Level 0 will dump only the basic cube. Further levels will dump recursive subdivisions of this cube. It is not recommended though to input N larger than 12, otherwise you will observe noticeable holes in the mesh. This is due the limitation imposed on the vertex buffer – which can hold only 65535 verticies.

Exporting will create `.obj` file and dump into it vertex positions, normals and cube-map coordinates along with polygons. It will also create six `.bmp` files containing appropriate faces of the cube-map.

4 Other operating systems

Officially they aren't supported (since I don't have any means to test on them). However the code should be portable with little to no effort. That means that as long as there's a port of `OpenGL` and `Allegro` library and there's a `C++` compiler you should have no problem building this project on that platform. If you do, please let me know on my e-mail address.

5 Contact info

This program is distributed as is. That means without any warranty or support. The provided contact info should be used for feature requests, announcements of source code changes (if you feel brave enough to fiddle with the code) or just anything you have on your mind. Don't expect I'll answer any questions how to build it. All the help is provided within this document. Help with building prerequisites is contained within their documentation or on-line on:

`www.allegro.cc`

My e-mail address is:

`firstname(dot)surname(at)centrum(dot)cz`

A List of control keys

General:	
'Esc'	exit program
'F1'	show/hide help
'~'	open/close console
'R'	toggle light rotation
'L'	toggle dynamic level of detail
'='	increase level of detail
'-'	decrease level of detail
View port control:	
'C'	switch camera type
'W'	move forward (spectator camera)
'S'	move backward (spectator camera)
'A'	strafe left (spectator camera)
'D'	strafe right (spectator camera)
Render mode switching:	
'1'	wireframe mode
'2'	culled wireframe mode – all faces not facing the camera are culled
'3'	gray scale mode – solid rendering using height-map texture
'4'	texture mode – solid rendering using custom rendering pipeline (shaders)

B List of console commands

command	parameters	description
`cmdlist`		prints out list of available commands
`exit`		immediately quits the program
`export`	*N filename*	exports the planet in `.obj` format as N-th subdivision of basic cube into specified file
`generate`		generates planet according to parameters (see below)
`help`	[*command*]	prints basic help. If supplied with command name it prints help for that command
`load`	*filename*	tries to load specified planet
`save`	*filename*	saves current planet into specified file
`nparams`		prints out parameters for the new planet
`params`		prints out parameters of the current planet
`radius`	*(float)r*	sets radius of the new planet to r
`ocean_level`	*(float)l*	sets ocean level of the new planet to l
`max_height`	*(float)h*	sets maximal height of the new planet to h
`seed`	*(int)s*	sets seed for the new planet to s
`omega`	*(float)o*	sets angular velocity of the new planet to o
`exponent`	*(float)e*	sets exponent used in fBm algorithm of the new planet to e
`w_factor`	*(float)w*	sets weight factor used in fBm algorithm of the new planet to w

C Program settings - ini file options

Format of the configuration file is quite simple. Any line beginning with `#` will be treated as a comment. All other lines are either blank or in the format of `option = value;`

option	value	description
`width`	*(int)w*	sets graphics window width to w
`height`	*(int)h*	sets graphics window width to h
`color_depth`	*(int)d*	sets colour depth to d bits. $d$should be either 8, 15, 16, 24 or 32
`field_of_view`	*(int)fov*	sets graphics window width to fov. Should be between 0 and 180 degrees
`enable_fullscreen`	*(int)*	toggles full screen mode on/off (1/0)
`enable_max_performance`	*(int)*	toggles the so-called laptop unfriendly mode on/off (1/0). If set to 1 it eats as many system resources it can get.
`texture_size`	*(int)sz*	sets cube map texture size to $sz*sz$. Not that there are totally 6 textures generated of this size. The bigger the size the longer will it take to generate the planet.
`enable_dynamic_LOD`	*(int)*	toggles dynamic level of detail on/off (1/0)
`default_threshold`	*(float)t*	sets default triangle split threshold (level of detail) to t. Anything between 0.0 and 0.1
`min_threshold`	*(float)min*	sets lower bound for the split threshold to min. Anything between 0.0 and 0.1
`max_threshold`	*(float)max*	sets upper bound for the split threshold to min. Anything between 0.0 and 0.1