FlightGear Flight Simulator – Installation and Getting Started

Michael Basler (pmb@knUUt.de)
Bernhard Buckel (buckel@wmad95.mathematik.uni-wuerzburg.de)

June 4, 1999

Contents

Chapter 1

Want to have a free flight? Take *FlightGear* !

1.1 Yet another Flight Simulator?

Did you ever want to fly a plane yourself, but lacked the money or skills to do so? Do you belong to those real pilots, who want to improve their skills without having to take off? Do you want to try some dangerous maneuvers without risking your life? Or do you just want to have fun with a more serious game not killing any people? If any of these questions applies, PC flight simulators are just for you.

If you are reading this you might have got already some experience either using Microsoft's © FS98, Looking Glass' © Flight Unlimited II or any other of the commercially available PC flight simulators. As the price tag of those is usually within the 50$ range buying one of them should not be a serious problem given the fact, that running any serious PC flight simulator requires a hardware within the 1500$ range, despite dropping prices, at least.

Why then that effort of spending hundreds or thousands of hours of programming to build a free simulator? Obviously there must be good reason to do so:

- All of the commercial programs have a serious drawback: They are made by a small group of developers defining their properties - often quite inert and not listening too much to the customer. Anyone ever trying to contact Microsoft will immediately agree.

- Commercial PC flight simulators usually try to cover a market segment as broad as possible. For obvious reason, most of them want to serve the serious pilot as well as the beginner and the gamer. The result are compromises. As

FlightGear is free, there is no need for such compromises; it just can be given the properties its users want. It defines itself via building.

- Building a flight simulator is a challenge to the art of programming. Contributing to that project makes you belong to those being able to contribute to serious, ambitious and advanced software projects.

- It is fun. Not only is it fun to write the code (...or documentation...) but also to belong to that – temporarily changing – club of clever people on the net having discussed, struggled and finally succeeded in creating that project. Even reading the *FlightGear* mailing lists is informative and fun for itself.

The above-mentioned points make *FlightGear* different from its competitors in several respect. *FlightGear* aims to be a civilian, multi-platform, open, user-supported, user-extensible simulator.

- **Civilian:** The project is primarily aimed to civilian flight simulation. It should be appropriate for simulating general aviation as well as civilian aircraft. However, according to the open concept of development that sure does not exclude someone taking the code and integrating military components.

- **Multi-platform:** The developers are attempting to keep the code as platform-independent as possible. This is based on their observation that people interested in flight simulations run quite a variety of computer hardware and operating systems. The present code supports the following Operating Systems:

  - Linux (any platform),
  - Windows NT (i86 platform),
  - Windows 98(95),
  - BSD UNIX,
  - SGI IRIX,
  - SunOS,
  - MacIntosh (experimental).

  There is no known flight simulator, neither commercially nor free, supporting such a broad range of platforms.

- **Open:** The project is not restricted to a closed club of developers. Anyone who feels he or she being able to contribute is highly welcome. The code

(including documentation) is copyrighted under the terms of the Gnu Public License.

The Gnu Public License is often misunderstood. In simple terms it states that you can copy and freely distribute the program(s) licensed to it. You can modify them, if you like. You are even allowed to charge as much money for the distribution of the modified or original program as you want. However, you must distribute it complete with the entire source code and it must retain the original copyrights. In short:

"You can do anything with the software except making it non-free".

The full text of the Gnu Public License can be obtained from

http://www.gnu.org/copyleft/gpl.html.

- **User-supported, user-extensible:** Contrary to various commercial simulators available, scenery and aircraft format, internal variables, etc. are user accessible and documented from the beginning. Even without an explicit developmental documentation, which sure has to be written at some point, this is guaranteed by supplying the source code. It is the goal of the developers to build a basic engine to which scenery designers, panel engineers, maybe adventure or ATC routine writers, sound capturers and others can (and are asked to) add. It is our hope, that the project will finally gain from the creativeness and ideas of the hundreds of talented simmers across the world.

Without doubt, the success of the Linux project initiated by Linus Torvalds inspired several of the developers. Not only has it shown that distributed development of even highly sophisticated software projects over the Internet is possible. It led to a product which, in several respect, is better than its commercial competitors.

1.2 A short history of *FlightGear*

This project goes back to a discussion of a group of net-citizens in 1996 resulting in a proposal written by David Murr who, unfortunately, dropped out from the project (as well as the net) later. The original proposal is still available from the *FlightGear* web site and can be found under

http://www.flightgear.org/proposal-3.0.1

Although the names of the people and several of the details naturally changed in time, the spirit of that proposal was clearly retained up to the present status of the project.

Actual coding started in summer 1996 and by the end of that year essential graphics routines were completed. At that time, programming was mainly done and

coordinated by Eric Korpela from Berkeley University (korpela@ssl.Berkeley.EDU). Early code was running under Linux as well as under DOS, OS/2, Windows 95/NT, and Sun-OS. This was quite an ambitious project, as it involved, among others, writing all the graphics routines in a system-independent way just from scratch.

Development slowed down and finally stopped at the beginning of 1997 when Eric had to complete his thesis. At this point, the project seemed to be dead and traffic on the mailing list went down to nearly nothing.

It was Curt Olson from the University of Minnesota (curt@flightgear.org) who re-started the project in the middle of 1997. His idea was as simple as successful: Why invent the wheel a second time? There have been several free flight simulators available running on workstations under different flavors of UNIX. One of these, LaRCsim, having been developed by Bruce Jackson from NASA (jackson@larc.nasa.gov) seemed to be well-adapted for the present approach. Curt took this one apart and re-wrote several of the routines in a way making them buildable as well as run-able on the intended target platforms. The key idea in doing so was selecting a system-independent graphics platform, i. e. OpenGL, for the basic graphics routines.

Fig. 1: *The Navion flight model is one of the features FlightGear inherited from LaRCsim. Until now it is the only one plane being fully realized in FlightGear.*

In addition, a clever decision on the selection of the basic scenery data was already made in this very first version. *FlightGear* Scenery is created based on satellite data published by the U. S. Geological Survey. These terrain data are available for the whole world over the Internet for free from

http://edcwww.cr.usgs.gov/doc/edchome/ndcdb/ndcdb.html

for the US resp.

http://edcwww.cr.usgs.gov/landdaac/gtopo30/gtopo30.html
for other countries. Those freely accessible scenery data in conjunction with scenery building tools provided with *FlightGear* are an important prerequisite enabling anyone to create his or her own scenery, at least in principle.

This new *FlightGear* code - still largely being based on original LaRCsim code - was released in July 1997. From that moment the project gained momentum again. Here are some milestones from the more recent history of development:

- Sun, moon and stars are a field where PC flight simulators have been notoriously weak for ages. It is one of the great achievements of *FlightGear* that it includes accurate sun (watch, Microsoft!), moon, and planets, being moreover placed on their proper positions. The corresponding astronomy code was implemented in fall 1997 by Durk Talsma (pn_talsma@macmail.psy.uva.nl).

- Texture support was added by Curt Olson (curt@flightgear.org) in spring 1998. This marked a significant improvement in terms of reality. You may recall: MSFS had untextured scenery up to version 4.0. For this purpose, some high-quality textures were submitted by Eric Mitchell (mitchell@mars.ark.com.

- A HUD (head up display) was added based on code provided by Michele America (nomimarketing@mail.telepac.pt) and Charlie Hotchkiss (chotchkiss@namg.us.anritsu.com) in fall 1997 and continuously improved later, mainly by Norman Vine (nhv@laserplot.com). While being probably not a substitute for a panel and moreover possibly being a bit odd in that tiny Navion, this HUD has proven extremely useful in navigation until now.

- After improving scenery and texture support and adding some more features there was a disappointing side-effect in spring 1998: Frame rates dropped down to a point where *FlightGear* became inflyable. There were two main achievements overcoming this problem. First, with the advent of hardware OpenGL support and corresponding drivers for most of the graphics cards these features could be exploited in *FlightGear* as well, leading to a frame rate boost by a factor up to 10. Second, Curt Olson (curt@flightgear.org) implemented so-called view frustrum culling (a procedure to except part of the scenery not required from rendering) which gave another 20% or so of frame rate boost in May 1998.

  With these two achievements *FlightGear* became flyable again even on weaker machines as long as they included a 3D graphics board with hardware OpenGL support. With respect to this point one should keep in mind that code at

present is in no way optimized leaving a lot of room for further improvements of frame rate.

- A rudimentary autopilot implementing heading hold was contributed by Jeff Goeke-Smith (jgoeke@voyager.net) in April 1998. The autopilot was improved, included adding an altitude hold and a terrain follow switch, in October 1998.

- The basics for selectable menus were laid based on Steve Baker's (sjbaker@hti.com) portable library PLIB in June 1998. After having been idle for a long time, first working menu entries came to life in spring 1999.

- Friedemann Reinhard (mpt218@faupt212.physik.uni-erlangen.de) developed early panel code, including a working airspeed indicator, which was added in June 1998 and has been considerably improved until today.

- There was basic audio support, i. e. an audio library and some basic background engine sound, contributed by Steve Baker (sjbaker@hti.com) in Summer 1998. Today, the audio library is part of Steves's above-mentioned portable library PLIB. This same library was extended to support joystick /yoke/rudder later which brought *FlightGear* joystick support in October 1989, again marking a huge improvement in terms of realism.

- In September 1998 Curt Olson (curt@flightgear.org) succeeded in creating first complete terrain Scenery for the USA, which is available for download from

  ftp://ftp.kingmont.com/pub/kingmont/

  Scenery was further improved by Curt via adding features like lakes, rivers, coastlines and the like in spring 1999.

This is by no way a complete history and a lot of people making even important contributions were left out here. Besides the named achievements being more on the surface there was a lot of work done concerning the internal structure, by Steve Baker (sjbaker@hti.com), Norman Vine (nhv@laserplot.com), Gary R. Van Sickle (tiberius@braemarinc.com), and others. A more complete list of contributors to the project can be found in *Landing: Some further thoughts before leaving the plane*, Chapter 7, as well as in the file `Thanks` provided with the code. Moreover, the *FlightGear* Website contains a detailed history of all of the development under

http://www.flightgear.org/News/

1.3 System requirements

Compared to other recent flight simulators the system requirements for *FlightGear* are rather decent. A P100 is already sufficient, given you have a proper 3D graphics card, but of course for getting good performance we recommend a P200 or better, if you run it on a PC. On the other hand, any not too ancient UNIX workstation will run *FlightGear* as well.

While in principle you can run *FlightGear* on 3D boards without OpenGL support or even on systems without 3D graphics hardware at all, missing hardware OpenGL support can force even the fastest PIII to its knees (frame rates typically below 1 fps). Any cheap 3D graphics card will do as long as it features hardware OpenGL support. For Windows 98/NT drivers, you may contact the home page of the manufacturer. Moreover, you should have in mind that several OpenGL drivers are still marked as beta and moreover, and sometimes these drivers are provided by the makers of the graphics chip instead of the makers of the board. More detail on OpenGL drivers can be found under

http://www.x-plane.com/v4ibm.html

as well as under

http://www.flightgear.org/Hardware.

Next, you need around 16MB of free disk space for installing the executable including basic scenery. In case you want to compile the program yourself you need around 50MB for the source code and for temporary files created during compilation, independent of the operating system.

If you want to hear the sound effects any decent sound card should serve. Besides, *FlightGear* supports a joystick or yoke as well as rudder pedals under Linux as well as under Windows.

With respect to operating systems, *FlightGear* is being primarily developed under Linux, a free UNIX clone developed cooperatively over the net in much the same way as the *FlightGear* project itself. Moreover, *FlightGear* runs under Windows 95, Windows 98 and Windows NT and given you have a proper compiler installed can be build under all of these platforms as well. The primary compiler for all platforms is the free GNU C++ (i. e. the Cygnus compiler under Win32), however there is some support for MSVC as well. Moreover, *FlightGear* runs and can be build on several UNIX/X11 platforms with GNU C++ installed.

1.4 Whom this guide is addressed to and how it is organized

At first: There is not much of the material in this Guide being originally invented by ourself. You could even say with Montaigne that we ”merely gathered here a big bunch of other men’s flowers, having furnished nothing of my own but the strip to hold them together”. Most (but fortunately not all) of the information can as well be grabbed from the *FlightGear* home page being situated at

http://www.flightgear.org/

and its various sub pages. However, there still seems to be a small group of people preferring neatly printed manuals over loosely scattered Readmes and those may acknowledge our effort.

This *Installation and Getting Started* is intended as being a first step towards a more complete *FlightGear* documentation (with the other parts, supposedly, to be written by others). Its main addressee is the end-user who is not interested in the internal workings of OpenGL or in building his or her own scenery, for instance. It is our hope, that sometime there will be an accompanying *FlightGear Programmer’s Guide*, which could be based on some of the documentation under

http://www.flightgear.org/Docs,

a *FlightGear Scenery Design Guide*, and a *FlightGear Flight School*, at least.

This *Installation and Getting Started* is organized as follows:

The first Chapter 2, *Getting the engine: Installing OpenGL graphics drivers*, describes how to prepare the computer for handling *FlightGear* ’s graphics routines. *FlightGear* is based on a graphics library called OpenGL, thus you must install either hardware or software OpenGL support for your graphics board (except, you did so before).

Chapter 3, *Building the plane: Compiling the program*, explains how to build, i. e. compile the simulator. Depending on your platform this may or may not be required for you. There will at least be binaries available for those working on a Win32 (i. e. Windows 98 © or Windows NT ©) platform. For those on such systems, who want to take off immediately without going through the potentially troublesome process of compiling, we recommend just skipping that Chapter and going directly to the next one.

In Chapter 4, *Preflight: Installing FlightGear* , you find instructions for installing the binaries in case you did not so by building them in the previous Chapter. Moreover, you’ll have to install scenery and texture files, which will be described there, too.

The following Chapter 5, *Takeoff: How to start the program*, describes how to start the program including an overview on the command line options.

Chapter 6, *Flight: All about instruments, keystrokes and menus*, describes how to operate the program, i. e. to actually fly with *FlightGear*. This includes a (hopefully) complete list of key strokes, an overview on the menu entries, as well as a detailed description of the HUD (head up display) and the panel.

In Chapter 7, *Landing: Some further thoughts before leaving the plane*, we would like to give credits to those who did the hard work, and give an outlook on what remains to be done.

Finally: **We kindly ask others to help us improving this document by submitting corrections, improvements, and more. Notably, we invite others to contribute descriptions referring to alternative setups (graphics cards, operating systems, and compilers etc.). We will be more than happy to include those into forthcoming versions of this *Installation and Getting Started* (of course not without giving credit to the authors).**

We hope to continuously maintain this document at least for a foreseeable future, but probably will not be able to produce a new one for any single release of *FlightGear* . While we are both watching the mailing lists, it might help, if developers adding new functionality could send us a short note.

Chapter 2

Getting the engine: Installing OpenGL graphics drivers

FlightGear 's graphics engine is based on a graphics library called OpenGL. Its primary advantage is it's platform independence, i. e., programs written with OpenGL support can be compiled and executed on several platforms, given the proper drivers having been installed in advance. Thus, independent of if you want to run the binaries only or if you want to compile the program yourself you must install some sort of OpenGL support for your video card. Naturally, you can skip this Chapter in case you already did (maybe for Quake or some other game).

Unfortunately, there are so many graphics boards, graphics chips and drivers that we are unable to provide a complete description for all systems. To give beginners a hand, we just describe what we did to install drivers on our systems, which might be not too exotic.

By any means, try getting hardware OpenGL drivers for your system, which is exemplary described in Sections 2.1 to 2.4, resp. If you are unable to locate any such drivers you can try software support as detailed under 2.5.

2.1 3DFX under Linux

An excellent place to search for documentation about Linux and 3D accelerators is the *Linux 3Dfx HOWTO* at

http://www.gamers.org/dEngine/xf3D/howto/3Dfx-HOWTO.html.

It describes all the following steps in an in-depth fashion and should be your first aid in case something goes wrong with your 3D setup.

The 3DFX graphics card is a quite popular one (We tested the Voodoo1 to work). At first, you need the GLIDE library installed. Grab it at:

http://www.3dfx.com/software/download_glidel.html
and install it. Be careful, you need different Glide libraries for the different types of VooDoos (I, II, Banshee). There is even an install script included that will do things for you. The canonical place for GLIDE is `/usr/local/glide`, if you prefer another location, you'll have to edit the Makefile for *FlightGear* by hand. Be sure to read and understand the file `/usr/local/glide/README`. Next, you need to install the MESA library version 3.0 (or later). Grab it at

ftp://iris.ssec.wisc.edu/pub/Mesa,
unpack it and run

```
make linux-glide
```

in the Mesa directory. Follow the instructions in the `README` file, take a close look at `README.3DFX` and play with the demo programs.

Besides these, you need the GLUT library version 3.7 (or greater, aka GameGLUT) installed. Grab it at:

http://reality.sgi.com/opengl/glut3/glut3.html.
Note: Glut-3.7 is included with Mesa 3.0 so if you've already grabbed the latest version of mesa, you should have everything you need.

Finally, some more notes on the behavior of Voodoo boards:

Your card comes packaged with a loop-through-cable. If you have only one monitor, then the Voodoo will take it over when used. This means that all the applications on your desktop will continue running but you'll only see the *FlightGear* screen. If your window manager uses a focus-follows-mouse policy, don't move the mouse. If you lose the focus, there's no way to shut down *FlightGear* graciously! Better solution: Use two monitors, one for your desktop, connect the other one to your accelerator. You'll then get a window on your desktop which manages all keyboard events and you're still able to see your desktop.

Running *FlightGear* under Linux using a 3DFX accelerator board is somewhat tricky. Most of the boards behavior is controlled by environment variables. The two most important are:

- `MESA_GLX_FX`: When set to `f` rendering will be in fullscreen mode, `w` will perform rendering in a window at a significant speed penalty.
- `FX_GLIDE_NO_SPLASH`: When set to `1` the rotating 3DFX logo won't appear. For a description of all environment variables for VooDooI/II have a look at

  http://www.bahnhof.se/~engstrom/e_3dfxvars.htm.

This completes preparing your 3DFX equipped Linux PC for running *FlightGear*. Now proceed and install the support files as described later in this document.

2.2 Rendition Chipset under Windows 98/NT

This Section serves as an example for installing OpenGL drivers under Windows 98/NT. The Rendition 2100 chipset is, for instance, included in the Diamond Stealth II card performing especially well in somewhat weaker machines.

Diamond itself does not provide any OpenGL driver support for that board. However, Rendition, who make the graphics chip, do. Go to their Web site and grab the latest OpenGL Windows drivers from

http://www.rendition.com/download.html

Follow the description in `readme.txt`. We recommend making the drivers the default ones by copying them to `\windows\system` (which avoids the hassle of not being sure which driver actually runs).

With this step you're already done.

According to our experience, so-called mini-OpenGL drivers provided by some manufacturers for making Quake playable do not provide the level of OpenGL support required by *FlightGear* . At least, Rendition's mini-OpenGL driver definitely does not.

2.3 RIVA TNT Chipset under Windows 98/NT

Because of its high performance, the RIVA TNT is one of the most popular chipsets today. The Diamond Viper 550, ELSA Erazor-2, Creative Graphics Blaster, and more cards come equipped with this chip. At least the default Viper 550 drivers are known to us having native built-in OpenGL support making any add-on OpenGL drivers obsolete. Similar things should apply to the other RIVA TNT based boards. In any case, NVIDIA's reference drivers being available from

http://www.nvidia.com/

do the job as well.

2.4 3DFX chip based boards under Windows 98/NT

The 3DXF based 3D add-on or 2D/3D boards are perhaps the most popular ones today at all. 3DFX made Beta OpenGL Windows 98 drivers available on their Website at

http://www.3dfx.com.

From the main page go to `Develop 3DFX` and further to `SDKs and Demos` and grab them there.

First, make sure you have the file `glu32.dll` either under `\Windows\System` or elsewhere in your path. If not, install the MS OpenGL kit `opengl95` available

from Microsoft or elsewhere on the net (which by itself only provides software rendering).

Next, locate the file `3dfxopengl.dll`. in the 3DFX driver package, rename it to `opengl32.dll` and copy it into `\Windows\System` overwriting the file with the same name installed from the MS kit. This should get you going.

2.5 OpenGL software rendering under Windows 98/NT

If you have an accelerated 3D card, it is highly recommended you install hardware OpenGL drivers for your specific card.

However, in case you are really unable to find such drivers and want to try *FlightGear* despite this you can install SGI software OpenGL rendering. For this purpose, get the file `sgi-opengl2.exe` from

ftp://ftp.flightgear.org/pub/fgfs/Misc/.

This is a Windows 98/NT self extracting installation program. Install it by double-clicking in Windows explorer. The package includes some demo games you may wish to try by invoking them from the Start menu.

Chapter 3

Building the plane: Compiling the program

This central Chapter describes how to build *FlightGear* on several systems. In case you are on a Win32 (i. e. Windows 98 or Windows NT) platform you may not want to go though that potentially troublesome process but instead skip that Chapter and straightly go to the next one. (Not everyone wants to build his or her plane himself or herself, right?) However, there may be good reason at least to try building the simulator:

- In case you are on a UNIX/Linux platform there are supposedly no pre-compiled binaries available for your system. We do not see any reason why the distribution of pre-compiled binaries (with statically linked libraries) should not be possible for UNIX systems in principle as well, but in practice it is common to install programs like this one on UNIX systems by recompiling them.
- There are several options you can set only during compile time. One such option is the decision to compile with hardware or software OpenGL rendering enabled. A more complete list goes beyond this *Installation and Getting Started* and should be included in a future *FlightGear Programmer's Guide*.
- You may be proud you did.

On the other hand, compiling *FlightGear* is not a task for novice users. Thus, if you're a beginner (we all were once) we recommend postponing this and just starting with the binary distribution to get you flying.

As you will note, this Chapter is far from being complete. Basically, we describe compiling for two operating systems only, Windows 98/NT and Linux.

There is a simple explanation for this: These are just the systems we are working on. We hope to be able to provide descriptions for more systems based on contributions written by others.

3.1 Compiling under Linux

If you are running Linux you probably have to build your own binaries. The following is one way to do so.

1. Get the file `FlightGear-x.xx.tar.gz` from the `source` subdirectory under

   ftp://ftp.flightgear.org/pub/fgfs/Source/

2. Unpack it using :

   `tar xvfz FlightGear-x.xx.tar.gz`.

3. `cd` into `FlightGear-x.xx`. Run:

   `./configure`

   and wait a few minutes. configure knows about a lot of options. Have a look at the file `INSTALL` in the *FlightGear* source directory to learn about them. If run without options, configure assumes that you will install the data files under `/usr/local/lib/FlightGear`.

4. Assuming configure finished successfully, simply run

   `make`

   and wait for the make process to finish.

5. Now become root (for example by using the su command) and type

   `make install`.

   This will install the binaries in `/usr/local/bin`.

   There is a problem concerning permissions under Linux/Glide. All programs accessing the accelerator board need root permissions. The solution is either to play as root or make the `/usr/local/bin/fgfs` binary `setuid root`, i.e. when this binary is run root privileges are given. Do this by issuing (as root)

   `chmod +s /usr/local/bin/fgfs`.

   A solution for this problem is upcoming, keep an eye on the 3Dfx website if you run a 3Dfx board.

3.2 Compiling under Windows 98/NT

1. Windows, contrary to Linux which brings its own compiler, comes not equipped with developmental tools. Several compilers have been shown to work for compiling *FlightGear* , including the Cygnus Win32 port of GNU C++ and the MS Visual C compiler. Given that the project will be a free one we prefer the Cygnus Compiler as it provides a free development environment. However, we will be happy to include a proper description in case those who worked out how to compile with MSVC or other Compilers provide one.

2. Install and configure the Cygnus Gnu-Win32 development environment. The latest version is Beta 20. The main Cygnus Gnu-Win32 page is at:

   http://sourceware.cygnus.com/cygwin/.

   You can download the complete Cygnus Gnu-Win32 compiler from:

   ftp://go.cygnus.com/pub/sourceware.cygnus.com/cygwin/latest/full.exe.

   Be sure to read this package's README files to be found under the main page, first.

   To install the compiler, just run `full.exe` by double-clicking in Windows explorer. After doing so you'll find a program group called `Cygnus Solutions` in your Start menu. Do not forget making a copy of the shell under c:/bin, as detailed in the docs.

3. Open the Cygnus shell via its entry in the Start menu. Mount the drive where you want to build *FlightGear* as follows (assuming your *FlightGear* drive is `d:`):

   ```
   mkdir /mnt
   mount d:  /mnt
   ```

   You only have to do this once. The drive stays mounted (until you umount it) even through reboots and switching off the machine.

4. Before actually being able to compile *FlightGear* you have to install a hand full of support libraries required for building the simulator itself. Those go usually into `c:/usr/local` and it is highly recommended to choose just that place.

   First, you have to install the free win32 api library (the latest version being 0.1.5). Get the package `win32api-0.1.5.tar.gz` from:

   http://www.acc.umu.se/ anorland/gnu-win32/w32api.html

Conveniently you may unpack the package just onto you *FlightGear* drive. Copy the file to the named drive, open the Cygnus shell via the Start menu entry and change to the previously mounted drive with

```
cd /mnt
```

Now, you can unpack the distribution with

```
gzip -d win32api-0.1.5.tar.gz
tar xvf win32api-0.1.5.tar
```

This provides you with a directory containing the named libraries. For installing them, change to that directory with

```
cd win32api-0.1.5
```

and type

```
make
make install
```

This installs the libraries to their default locations under `c:/usr/local`

5. To proceed, you need the glut libraries. Get these from the same site named above

   http://www.acc.umu.se/ anorland/gnu-win32/w32api.html

   as `glutlibs-3.7beta.tar.gz`. Just copy the package to your *FlightGear* drive and unpack it in the same way as describes above. There is no need to run `make` here. Instead, just copy the two libraries `libglut.a` and `libglut32.a` to `c:/usr/local/lib`. There is no need for the two accompanying `*.def` files here.

6. Next, get the Glut header files, for instance, from

   ftp:://ftp.flightgear.org/pub/fgfs/Win32/Mesa-3.0-includes.zip

   Unpack these as usual with `unzip -d` and copy the contents of the resulting directory `/gl` to `c:/usr/local/include/gl`

7. Finally, you need Steve Backer's PLIB being one of the key libraries for *FlightGear*. Get the most recent version `plib-X.X.tar.gz` from

   http://www.woodsoup.org/projs/plib/

   (There are mirrors, but make sure they contain the most recent version!). Copy it to your *FlightGear* drive, open the Cygnus shell and unpack the library as described above.

Next, change into PLIB's directory. It is recommended to configure PLIB with the following command line (you can make a script as I did if it hurts)

```
CFLAGS="-O2 -Wall" CXXFLAGS="-O2 -Wall"
CPPFLAGS=-I/usr/local/include LDFLAGS=-L/usr/local/lib
./configure --prefix=/usr/local
--includedir=/usr/local/include/plib
```

You must write all this **on one line** without any line breaks in between!

Finally, build PLIB with

```
make
make install
```

8. Now, you're finally prepared to build *FlightGear* itself.

   Fetch the *FlightGear* code and special Win32 libraries. These can be found at:

   ftp://ftp.flightgear.org/pub/fgfs/Source/

   Grab the latest `FlightGear-X.XX.zip` and `win32-libs-X.XX.zip` files.

   (It you're really into adventures, you can try one of the recent snapshots instead.)

9. Unpack the *FlightGear* source code via

   `pkunzip -d FlightGear-X.XX.zip`.

10. Change to the newly created `FlightGear-X.XX directory` with e. g.

    `cd //D/FlightGear-X.XX`

    and unpack the Win32 libraries there:

    `pkunzip -d win32-libs-X.XX.zip`.

11. You will find a file called `install.exe` in the Win32 directory after unzipping `win32-libs-X.XX.zip`. This version of `install.exe` should replace the one in your `\H-i386-cygwin32\bin` directory – it's sole claim to fame is that it understands that when many calls to it say `install foo` they mean `install foo.exe`. If you skip this step and attempt an install with the older version present `make install` will fail.

    Side Note: We need to make a distinction between the `build tree` and the `install tree`. The `build tree` is what we've been talking about up until this point. This is where the source code lives and all the compiling

takes place. Once the executables are built, they need to be installed someplace. We shall call this install location the `install tree`. This is where the executables, the scenery, the textures, and any other run-time files will be located.

12. Configure the make system for your environment and your `install tree`. Tell the configure script where you would like to install the binaries and all the scenery and textures by using the `--prefix` option. In the following example the base of the `install tree` is `FlightGear`. Make sure you are within *FlightGear* 's `build tree` root directory.

13. Run:

    `./configure --prefix=/mnt/FlightGear`.

    Side note: The make procedure is designed to link against opengl32.dll, glu32.dll, and glut32.dll which most accelerated boards require. If this does not apply to yours or if you installed SGI's software rendering as mentioned in Subsection 2.5 you may have to change these to opengl.dll, glu.dll, and glut.dll. (In case you're in doubt check your `\windows\system` directory what you've got.)

    If this is the case for your video card, you can edit `.../Simulator/Main/Makefile` and rename these three libraries to their "non-32" counterparts. There is only one place in this `Makefile` where these files are listed.

14. Build the executable. Run:

    `make`.

    Assuming you have installed the updated version of `install.exe` (see earlier instructions) you can now create and populate the `install tree`. Run:

    `make install`.

    You can save a significant amount of space by stripping all the debugging symbols off of the executable. To do this, change to the directory in the `install tree` where your binary lives and run:

    `strip fgfs.exe` resp. `strip fgfs-sgi.exe`.

Chapter 4

Preflight: Installing *FlightGear*

4.1 Installing the Binaries on a Windows system

You can skip this Section and go to the installation of scenery in case you built *FlightGear* along the lines describes during the previous Chapter. If you did not and you're jumping in here your first step consists in installing the binaries. At present, there are only pre-compiled binaries available for Windows 98/NT while in principle it might be possible to create (statically linked) binaries for Linux as well.

The following supposes you are on a Windows 98 or Windows NT system. Installing the binaries is quite simple. Go to

ftp://ftp.flightgear.org/pub/fgfs/Win32/

get the latest binaries from that subdirectory named

`fgfs-win32-bin-X.XX.exe`

and unpack them via double clicking. This will create a directory `FlightGear` with several subdirectories. You are done.

4.2 Installing Support files

Independent on your operating system and independent on if you built the binaries yourself or installed the precompiled ones as described above you will need scenery, texture, sound, and some more support files. A basic package of all these is contained in the binaries directory mentioned above as

`fgfs-base-X.XX`.

Preferably, you may want to download the `.tar.gz` version if you are working under Linux/UNIX and the `.exe` version if you are under Windows 98/NT. Make sure you get the **most recent** version.

If you're working under Linux or UNIX, unpack the previously downloaded file with

`tar xvfz fgfs-base-X.XX.tar.gz`

while under Windows 98/NT just double click on the file (being situated in the root of your *FlightGear* drive.).

This already completes installing *FlightGear* and should you enable to invoke the program.

Some more scenery which, however, is not a substitute for the package mentioned above but rather is based on it can be found in the scenery subdirectory under

http://www.flightgear.org/Downloads/

These may be older versions which may or may not work with the most recent binaries.

In addition, there is a complete set of USA Scenery files available created by Curt Olson which can be downloaded from

ftp://ftp.kingmont.com/pub/kingmont/

The complete set covers several 100's of MBytes. Thus, Curt provides the complete set on CD-ROM for those who really would like to fly over all of the USA. For more detail, check the remarks in the downloads page above.

Finally, the binaries directory mentioned contains the complete *FlightGear* documentation as

`fgfs-manual-X.XX.exe`.

It includes a .pdf version of this *Installation and Getting Started* guide intended for pretty printing using Adobe's Acrobat reader being available from

http://www.adobe.com/acrobat

Moreover, if properly installed the .html version can be accessed via *FlightGear* 's `help` menu entry.

Chapter 5

Takeoff: How to start the program

5.1 Starting under Linux

Under Linux, *FlightGear* is invoked by

```
fgfs --option1 --option2...,
```

where the options are described in Section 5.3 below.

5.2 Starting under Windows 98/NT

In Windows explorer, change to the `\FlightGear\` directory. Call `runfgfs.bat` by double-clicking if you want to invoke the hardware accelerated version of *FlightGear* `fgfs.exe`, or `runfgfs-sgi.bat` if you installed SGI's software OpenGL support.

Alternatively, if for one or the other reason the batch does not work, you can open an MS-DOS shell, change to the directory where your binary resides (typically something like `d:\FlightGear\bin` where you might have to substitute `d:` in favor of your *FlightGear* directory), set the environment variable with

```
SET FG_ROOT=d:\FlightGear\bin
```

and invoke *FlightGear* (within the same shell – Windows environment settings are only valid locally within the same shell) via

```
fgfs --option1 --option2....
```

For getting maximum performance it is highly recommended to minimize (iconize) the non-graphics window while running *FlightGear*.

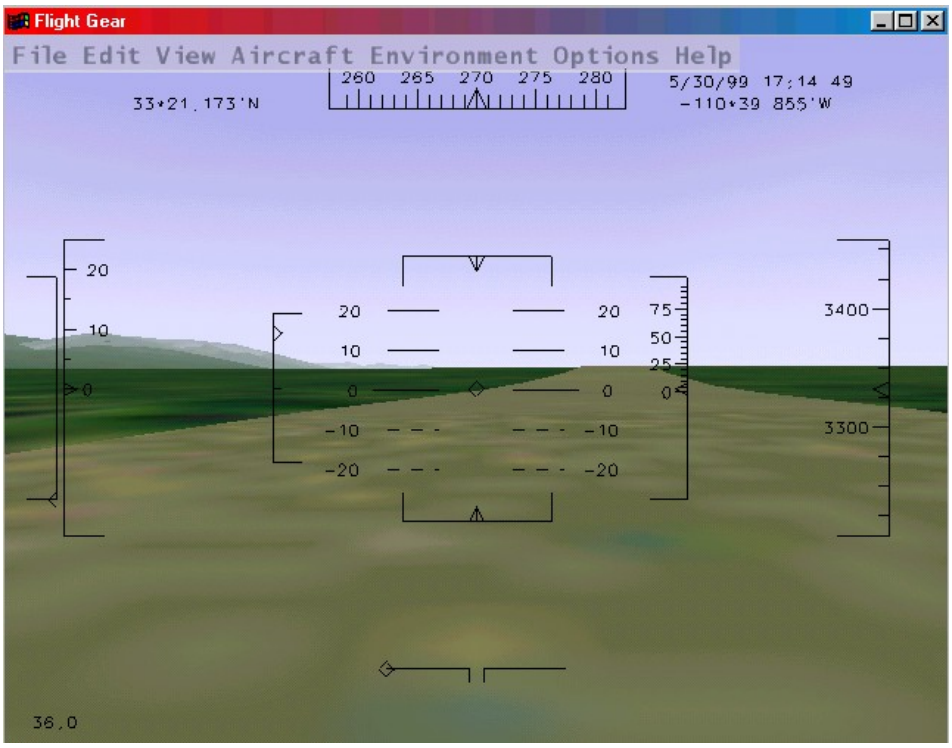

Fig. 2: *Ready for takeoff. We are at the default startup position in Arizona.*

5.3 Command line parameters

Following is a list and short description of the command line options available. In case of Windows 98/NT it is recommended to include these in `runfgfs.bat`.

5.3.1 General Options

- `--help`: gives a small help text, kind of a short version of this Section.
- `--fg-root=`*path*: tells *FlightGear* where to look for its data files if you didn't compile it with the default settings.
- `--disable-game-mode`: Disables fullscreen display.
- `--enable-game-mode`: Enables fullscreen rendering.
- `--disable-splash-screen`: Turns off the rotating 3DFX logo when the accelerator board gets initialized.

- `--enable-splash-screen`: If you like advertising, set this!
- `--disable-intro-music`: No MP3-sample is being played when *FlightGear* starts up.
- `--enable-intro-music`: If your machine is powerful enough, enjoy this setting.
- `--disable-mouse-pointer`: In the future, *FlightGear* will feature a mouse interface so that options can be set at runtime. As this feature is not implemented yet it seems wise to disable the mouse interface.
- `--enable-mouse-pointer`: Enables another mouse pointer in the *FlightGear* window. This is useful when running *FlightGear* in full screen mode and will allow access to the - yet to be implemented - mouse interface of *FlightGear.*
- `--disable-pause`: This will put you into *FlightGear* with the engine running, ready for Take-Off.
- `--enable-pause`: Starts *FlightGear* in pause mode.

5.3.2 Features

- `--disable-hud`: Switches off the HUD (**H**ead **U**p **D**isplay).
- `--enable-hud`: Turns the HUD on. This is the default.
- `--disable-panel`: Turns off the instrument panel. This is the default, as the instrument panel is not yet complete – but in our opinion should be given at least a try.
- `--enable-panel`: This will give you the look of a real cockpit.
- `--disable-sound`: Pretty self explaining, isn't it?
- `--enable-sound`:

5.3.3 Flight model

- `--fdm=abcd` There are four allowed values for abcd: `slew, jsb, larcsim, external`, which you might want to try. Default value is `larcsim`.

5.3.4 Initial Position and Orientation

- `--airport-id=ABCD`: If you want to start directly at an airport, enter its international code, i.e. KJFK for JFK airport in New York. A long/short list of the IDs of the airports being implemented can be found in `/Flight Gear/Airports`. You only have to unpack one of the files with gnuzip. Keep in mind, you need the terrain data for the relevant region!
- `--lon=degrees`: This is the starting longitude in degrees (west = -)
- `--lat=degrees`: This is the starting latitude in degrees (south = -)
- `--altitude=meters`: You may start in free flight at the given altitude. Watch for the next options to insert the plane with a given heading etc.
- `--heading=degrees`: Sets the initial heading.
- `--roll=degrees`: Initial roll angle.
- `--pitch=degrees`: Initial pitch angle.

5.3.5 Rendering Options

- `--fog-disable`: To cut down the rendering efforts, distant regions are vanishing in fog by default. If you disable fogging, you'll see farther but your frame rates will drop.
- `--fog-fastest`: The scenery will not look very nice but frame rates will increase.
- `--fog-nicest`: This option will give you a fairly realistic view of flying on a hazy day.
- `--fov=xx.x`: Sets the field of view in degrees. The value is displayed on the HUD. Default is 55.0.
- `--disable-fullscreen`: Self explaining, isn't it?
- `--enable-fullscreen`:
- `--shading-flat`: This is the fastest mode but the terrain will look ugly! This option might help if your video accelerator is really slow.
- `--shading-smooth`: This is the recommended (and default) setting - things will look really nice.

- `--disable-skyblend`: No fogging or haze, sky will be displayed using just one color. Fast but ugly!
- `--enable-skyblend`: Fogging/haze is enabled, sky and terrain look realistic. This is the default and recommended setting.
- `--disable-textures`: Terrain details will be disabled. Looks ugly, but might help if your video board is slow.
- `--enable-textures`: Default and recommended.
- `--enable-wireframe`: If you want to know how the world of *FlightGear* internally looks like, try this!
- `--geometry=WWWxHHH`: Defines the size of the window used, i.e. `WWWxHHH` can be `640x480`, `800x600`, or `1024x768`.

5.3.6 Scenery Options Options

- `--tile-radius=n`: Specifies the tiles radius; allowed values for `n` are 1 – 4.

5.3.7 HUD Options

- `--units-feed`: HUD displays units in feet.
- `--units-meters`: HUD displays units in meters.
- `--hud-tris`: HUD displays the number of triangles rendered.
- `--hud-culled`: HUD displays percentage of triangles culled.

5.3.8 Time options

- `--time-offset=[+-]hh:mm:ss`: Offset local time by this amount.
- `--start-date-gmt=yyyy:mm:dd:hh:mm:ss`: Specify a starting time and date. Time is Greenwich Mean Time.
- `--start-date-lst=yyyy:mm:dd:hh:mm:ss`: Specify a starting time and date. Uses local sidereal time.

Chapter 6

Flight: All about instruments, keystrokes and menus

This is a description of the main systems for controlling the program and piloting the plane: Historically, keyboard controls were developed first, and you can still control most of the simulator via the keyboard alone. Recently, they are becoming supplemented by several menu entries, making the interface more accessible, particularly for beginners, and providing additional functionality. A joysticks provides a more realistic alternative for actual piloting of the plane. Concerning instruments, there are again two alternatives: You can use the rather advanced HUD or the emerging panel.

6.1 Keyboard commands

While joysticks or yokes are supported as are rudder pedals, you can fly *FlightGear* using the keyboard alone. For proper controlling via keyboard (i) the `NumLock` key must be switched on (ii) the *FlightGear* window must have focus (if not, click with the mouse on the graphics window). Some of the keyboard controls might be helpful even in case you use a joystick.

After activating `NumLock` the following keyboard commands should work:

Tab. 1: *Main keyboard commands for FlightGear .*

Key	Action
Pg Up/Pg Dn	Throttle
Left Arrow/Right Arrow	Aileron
Up Arrow/Down Arrow	Elevator
Ins/Enter	Rudder
5	Center aileron/elevator/rudder
Home/End	Elevator trim

For changing views you have to de-activate `NumLock`. Now `Shift` + <`Numeric Keypad Key`> changes the view as follows:

Tab. 2: *View directions accessible after de-activating `NumLock`.*

Numeric Key	View direction
Shift-8	forward
Shift-7	left/forward
Shift-4	left
Shift-1	left/back
Shift-2	back
Shift-3	right/back
Shift-6	right
Shift-9	right/forward

The autopilot is controlled via the following controls:

Tab. 3: *Autopilot controls.*

Key	Action
Ctrl + A	Altitude hold toggle on/off
Ctrl + H	Heading hold toggle on/off
Ctrl + S	Autothrottle toggle on/off
Ctrl + T	Terrain follow toggle on/off
F11	Set target altitude
F12	Set target heading

The last one is especially interesting as it makes your Navion behave like a cruise missile.

Besides these basic keys there are some more special ones; most of these you'll probably not want to try during your first flight:

Tab. 4: *More control commands.*

Key	Action
H/h	Change color of HUD/toggle HUD off forward/backward
i/I	Minimize/maximize HUD
m/M	Change time offset (warp) used by t/T forward/backward
P	Toggles panel on/off
t/T	Time speed up/slow down forward/backward
x/X	Zoom in/out
z/Z	Change visibility (fog) forward/backward
b	Toggle brakes on/off
p	Toggle pause on/off
W	Toggle fullscreen mode on/off (Mesa/3dfx/Glide only)
F2	Refresh Scenery tile cache
F8	Toggle fog on/off
F9	Toggle texturing on/off
F10	Toggle menu on/off
F11	Sets heading in autopilot
F12	Sets altitude in autopilot
ESC	Exit program

6.2 Menu entries

Albeit the menu being not yet fully operational it provides several useful functions. At present, the following ones are implemented.

- **File**
  - **Reset** Resets you to the selected starting position. Comes handy in case you got lost or something went wrong.
  - **Save** Not yet operational.
  - **Print** Not yet operational.
  - **Close** Removes the menu. (Can be re-activated by hitting F10.)
  - **Exit** Exits the program.
- **Edit**
  - **Edit text** Not yet operational.
- **View**

  - **Toggle Panel** Toggles panel on/off.
  - **View** Not yet operational.
  - **Cockpit View** Not yet operational.

- **Aircraft**
  - **Communication** Not yet operational.
  - **Navigation** Not yet operational.
  - **Altitude** Not yet operational.
  - **Autopilot** Sliders for setting limiting values for the autopilot.

- **Environment**
  - **Weather** Not yet operational.
  - **Terrain** Not yet operational.
  - **Airport** Typing in an airport id beams you to that airport's position. *FlightGear* comes with an extended list of airport ids to be found under /FlightGear/Aircraft/apt_full.gz which you can unpack with gzip -d.

- **Options**
  - **Realism & Reliability** Not yet operational.
  - **Preferences** Not yet operational.

- **Help**
  - **Help** Should bring up this Getting Started Guide. At present not yet fully implemented. Under windows you can get it working by placing a file called **webrun.bat** like

    `c:\programme\netscape\communicator\program\netscape.exe`
    `d:\Flightgear\docs\installguide\html\getstart.html`

    (you may have to substitute your path/browser) somewhere in your path. Under UNIX a comparable shell script might do. Requires `fgfs-manual-X.XX.exe` being properly installed.
  - **About...** Not yet operational.

6.3 The head up display

At current, you have two options for reading off the main flight parameters of the plane: The HUD (**H**ead **U**p **D**isplay and the panel. Neither are HUDs used in usual general aviation planes nor in civilian ones. Rather they belong to the equipment of modern military jets. However, in view of the fact that the panel despite recent progress is not yet complete the HUD may well serve as a main instrument for controlling the plane. Besides, it might be easier to fly using this one than exploiting the panel and several of the real pilots might prefer it because of combining the readouts of critical parameters with an outside view onto the real world. (Several Cessna pilots might love to have one, but technology is simply too expensive for implementing HUDs in general aviation aircrafts.)

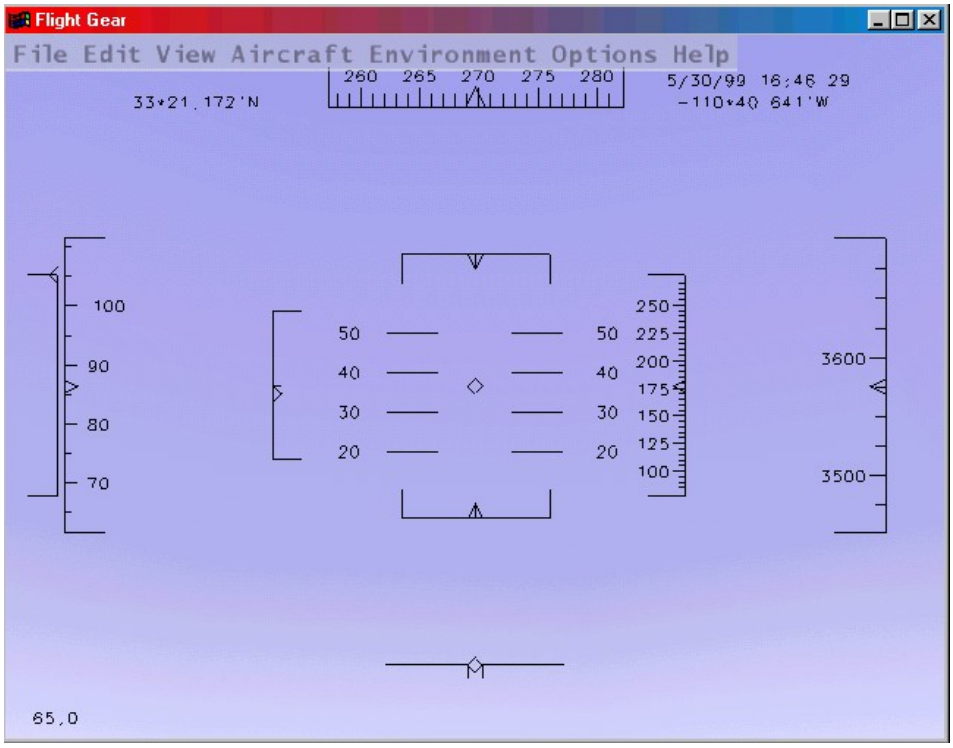

Fig. 3: *The HUD, or head up display.*

The HUD shown in Fig. 3 displays all main flight parameters of the plane. In the center you find the pitch indicator (in degrees) with the aileron indicator above and the rudder indicator below. A corresponding scale for the elevation can be found to the left of the pitch scale. On the bottom there is a simple turn indicator.

There are two scales at the extreme left: The inner one displays the speed

(in kts) while the outer one indicates position of the throttle. You may recall the Navion taking off at a speed of 100 kts. The two scales on the extreme r.h.s display your height, i. e. the left one shows the height above ground while the right of it gives that above zero, both being displayed in feet.

Besides this, the HUD displays some additions information. On the upper right you find date and time. Below, you see latitude and longitude of your current position on the l.h.s and r.h.s, resp. In the lower left corner there is a number indicating the frame rate, i.e. the number of times the picture being re-drawn each second.

You can change color of the **HUD** using the ”H” key. Pressing it several times minimizes the HUD.

6.4 The Panel

Besides the HUD, *FlightGear* has a panel which can be activated by pressing the ”P” key. (It is recommended disabling the HUD then by pressing ”H” several times.) While the panel is not yet fully complete the basic five flight instruments to scan are present and working.

Fig. 4: *The panel.*

In the center you find the artificial horizon (attitude indicator) displaying pitch and bank of your plane. It has pitch marks (hard to be seen in this version) as well as bank marks at 10, 20, 30, 60, and 90 degrees.

Left to the artificial horizon, you'll see the airspeed indicator. Not only does it have a speed indication in knots (recall: The Navion takes off at 100 kts) but also several arcs showing characteristic velocity rages you have to consider. At first, there is a green arc indicating the normal operating range of speed with the flaps (net yet being implemented in *FlightGear*) fully retracted. The white arc indicates the range of speed with flaps in action. The tiny yellow arc shows a range, which should only be used in smooth air. The upper end of it has a red radial indicating the speed never to be exceeded.

Below the airspeed indicator you can find the turn indicator. The airplane in the middle indicates the roll of your plane. If the left or right wing of the plane is aligned with one of the marks this indicates a standard turn, in which you make a full 360 degrees turn in exactly two minutes.

Below the plane, still in the turn indicator, is another instrument, called inclinometer. It indicates if rudder and ailerons are coordinated. During turns, you always have to operate aileron and rudder in such a way that the ball in the tube remains centered; otherwise the plane is skidding.

To the right of the artificial horizon you find the altimeter showing the height above sea level (not ground!). At present it is not yet working in *FlightGear* . Below the altimeter is the vertical speed indicator which, on the other hand, is operational. It indicates the rate of climbing or sinking of your plane in hundreds of feet per minute.

There is one more instrument working in the panel, i.e. the second one in the column on the r.h.s. indicating position of throttle.

This completes description of the present main *FlightGear* instruments. If you are looking for some interesting places to discover with *FlightGear* (which may or may not require downloading additional scenery) you may want to check

http://www.flightgear.org/Downloads/Places.

There is now a menu entry for entering directly the airport code of the airport you want to start from.

Finally, if you're done and are about to leave the plane, just hit the ESC key or use the corresponding menu entry to exit the program.

Chapter 7

Landing: Some further thoughts before leaving the plane

7.1 Those, who did the work

Did you enjoy the flight? In case you did, don't forget those who devoted hundreds of hours to that project. All of this work is done on a voluntary basis within spare time, thus bare with the programmers in case something does not work the way you want it to. Instead, sit down and write them a kind (!) letter proposing what to change. Alternatively, you can subscribe to the *FlightGear* mailing lists and contribute your thoughts there. Instructions to do so can be found under

http://www.flightgear.org/mail.html.

Essentially there are two lists, one of which being mainly for the developers and the other one for end users.

These are the people who did the job (This information was essentially taken from the file `Thanks` accompanying the code):

Raul Alonzo (amil@las.es)
Author of Ssystem and moon texture.

Michele America (nomimarketing@mail.telepac.pt)
Contributed to the HUD code.

Steve Baker (sjbaker@hti.com)
Author of PLIB, a graphics/audio/joystick interface written entirely on top of OpenGL/-GLUT used in *FlightGear* . An immense amount of coaching and tutelage, both on the subjects of flight simulation and OpenGL. It has been his comments and thoughts that have prompted the implementation of most of the more sophisticated features of *FlightGear.*

Michael Basler (pmb@knUUt.de)
Coauthor of Installation and Getting Started (together with Bernhard Buckel).

John S. Berndt (jsb@hal-pc.org)
Working on a complete C++rewrite/reimplimentation of the core FDM. Initially he is using X15 data to test his code, but once things are all in place we should be able to simulator arbitrary aircraft.

Paul Bleisch (pbleisch@acm.org)
Redid the debug system so that it would be much more flexible, so it could be easily disabled for production system, and so that messages for certain subsystems could be selectively enabled.

Also contributed a first stab at a config file/command line parsing system.

Jim Brennan (jjb@foothill.net)
Provided a big chunk of online space to store USA scenery for Flight Gear.

Bernie Bright (bbright@c031.aone.net.au)
Many C++ style, usage, and implementation improvements, STL portability and much, much more.

Bernhard H. Buckel (buckel@wmad95.mathematik.uni-wuerzburg.de)
Contributed the README.Linux. Coauthor of Installation and Getting Started (together with Michael Basler).

Gene Buckle (geneb@nwlink.com)
A lot of work getting *FlightGear* to compile with the MSVC++ compiler. Numerous hints on detailed improvements.

Oliver Delise (delise@rp-plus.de)
FAQ Maintainer.

Didier Chauveau (chauveau@math.univ-mlv.fr)
Provided some initial code to parse the 30 arcsec DEM files found at:
http://edcwww.cr.usgs.gov/landdaac/gtopo30/gtopo30.html.

Jean-Francois Doue
Vector 2D, 3D, 4D and Matrix 3D and 4D inlined C++ classes. (Based on Graphics Gems IV ed. Paul S. Heckbert)
http://www.animats.com/simpleppp/ftp/public_html/topics/developers.html.

Francine Evans (evans@cs.sunysb.edu)
http://www.cs.sunysb.edu/~evans/stripe.html
Wrote the GPL'd tri-striper.

Oscar Everitt (bigoc@premier.net)
Created single engine piston engine sounds as part of an F4U package for FS98. They are pretty cool and Oscar was happy to contribute them to our little project.

Jean-loup Gailly and Mark Adler (zlib@quest.jpl.nasa.gov)
Authors of the zlib library. Used for on-the-fly compression and decompression routines,
http://www.cdrom.com/pub/infozip/zlib/.

Thomas Gellekum (tg@ihf.rwth-aachen.de)
Changes and updates for compiling on FreeBSD.

Jeff Goeke-Smith (jgoeke@voyager.net)
Contributed our first autopilot (Heading Hold). Better autoconf check for external timezone/daylight variables.

Michael I. Gold (gold@puck.asd.sgi.com)
Patiently answered questions on OpenGL.

Charlie Hotchkiss (chotchkiss@namg.us.anritsu.com)
Worked on improving and enhancing the HUD code. Lots of code style tips and code tweaks. . .

Bruce Jackson (NASA) (e.b.jackson@larc.nasa.gov)
http://agcbwww.larc.nasa.gov/People/ebj.html
Developed the LaRCsim code under funding by NASA which we use to provide the flight model. Bruce has patiently answered many, many questions.

Tom Knienieder (knienieder@ms.netwing.at)
Ported Steve Bakers's audio library to Win32.

Reto Koradi (kor@mol.biol.ethz.ch)
http://www.mol.biol.ethz.ch/~kor
Helped with setting up fog effects.

Bob Kuehne (rpk@sgi.com)
Redid the Makefile system so it is simpler and more robust.

Vasily Lewis (vlewis@woodsoup.org)
http://www.woodsoup.org
Provided computing resources and services so that the Flight Gear project could have real home. This includes web services, ftp services, shell accounts, email lists, dns services, etc.

Christian Mayer (Vader@t-online.de)
Working on multi-lingual conversion tools for fgfs.
Contributed code to read msfs scenery textures.

Eric Mitchell (mitchell@mars.ark.com)
Contributed some topnotch scenery textures.

Anders Morken (amrken@online.no)
Maintains the European mirror of the *FlightGear* web pages.

Alan Murta (amurta@cs.man.ac.uk)
http://www.cs.man.ac.uk/aig/staff/alan/software/
Created the Generic Polygon Clipping library.

Curt Olson (curt@flightgear.org)
Primary organization of the project. First implementation and modifications based on LaRCsim. Besides putting together all the pieces provided by others mainly concentrating on the scenery engine as well as the graphics stuff.

Robin Peel (robinp@mindspring.com)
Maintains worldwide airport and runway database for *FlightGear* as we as X-Plane.

Friedemann Reinhard (mpt218@faupt212.physik.uni-erlangen.de)
Development of textured instrument panel.

Petter Reinholdtsen (pere@games.no)
Incorporated the Gnu automake/autoconf system (with libtool). This should streamline and standardize the build process for all UNIX-like platforms. It should have little effect on IDE type environments since they don't use the UNIX make system.

William Riley (riley@technologist.com)
Contributed code to add "brakes".

Paul Schlyter (pausch@saaf.se)
Provided Durk Talsma with all the information he needed to write the astro code.

Chris Schoeneman (crs@millpond.engr.sgi.com)
Contributed ideas on audio support.

Jonathan R Shewchuk (Jonathan_R_Shewchuk@ux4.sp.cs.cmu.edu)
Author of the Triangle program. Triangle is used to calculate the Delauney triangulation of our irregular terrain.

Gordan Sikic (gsikic@public.srce.hr)
Contributed a Cherokee flight model for LaRCsim. Currently is not working and needs to be debugged. Use configure `--with-flight-model=cherokee` to build the cherokee instead of the Navion.

Michael Smith (msmith99@flash.net)
Contributed cockpit graphics, 3d models, logos, and other images. Project Bonanza http://members.xoom.com/ConceptSim/index.html.

U. S. Geological Survey
http://edcwww.cr.usgs.gov/doc/edchome/ndcdb/ndcdb.html
Provided geographic data used by this project.

Durk Talsma (pn_talsma@macmail.psy.uva.nl)
Accurate Sun, Moon, and Planets. Sun changes color based on position in sky. Moon has correct phase and blends well into the sky. Planets are correctly positioned and have proper magnitude. help with time functions, GUI, and other things.

Gary R. Van Sickle (tiberius@braemarinc.com)
Contributed some initial GameGLUT support and other fixes.

Norman Vine (nhv@laserplot.com)
Many performance optimizations throughout the code. Many contributions and much advice for the scenery generation section. Lots of Windows related contributions. Improved HUD.

Roland Voegtli (webmaster@sanw.unibe.ch)
Contributed great photorealistic textures.

Carmelo Volpe (carmelo.volpe@csb.ki.se)
Porting *FlightGear* to the Metro Works development environment (PC/Mac).

Darrell Walisser (dwaliss1@purdue.edu)
Contributed a large number of changes to porting *FlightGear* to the Metro Works development environment (PC/Mac). Finally produced the first MacIntosh port.

Robert Allan Zeh (raz@cmg.FCNBD.COM)
Helped tremendously in figuring out the Cygnus Win32 compiler and how to link with .dll's. Without him the first run-able Win32 version of *FlightGear* would have been impossible.

7.2 What remains to be done

At first: If you read (and, maybe, followed) this guide until this point you may probably agree that *FlightGear*, even in its present state, is not at all for the birds. It is already a flight simulator which has a flight model, a plane, terrain scenery, texturing and simple controls.

Despite, *FlightGear* needs – and gets – further development. Except internal tweakings, there are several fields where *FlightGear* needs basics improvement and development.

A first direction is adding airports, streets, and more things bringing Scenery to real life.

Second, the panel needs further improvement including more working gauges.

Besides, there should be support for adding more planes and for implementing corresponding flight models differing from the Navion.

Another task is further implementation of the menu system, which should not be too hard with the basics being working now.

A main stream of active development concerns weather. At present there is simply none: no clouds, no rain, no wind. But there sure will be.

There are already people working in all of these directions. If you're a programmer and think you can contribute, you are invited to do so.

Achnowledgements

Obviously this document could not have been written without all those contributors mentioned above making *FlightGear* a reality.

Beyond this we would like to say special thanks to Curt Olson, whose numerous scattered Readmes, Thanks, Webpages, and personal eMails were of special help to us and were freely exploited in the making of this booklet.

Next, we gained a lot of help and support from Steve Baker and Norman Vine. Moreover, we would like to thank Steve Baker for a careful reading and for numerous hints on the first draft of this guide.

Further, we would like to thank Kai Troester for donating the solution of some of his compile problems to Chapter 8.

Chapter 8

Missed approach: If anything refuses to work

We tried to sort problems according to operating system to a certain extent , but if you encounter a problem it may be a wise idea to look beyond ”your” operating system – just in case. Besides, if anything fails, it is definitely a good idea to check the FAQ maintained by Oliver Delise (delise@rp-plus.de) being distributed along with the source code.

8.1 General problems

- *FlightGear* runs SOOO slow
  If the HUD indicates you are getting something like 1 fps (frame per second) or below you typically don't have working hardware OpenGL support. There may be several reasons for this. First, there may be no OpenGL hardware drivers available for older cards. In this case it is highly recommended to get a new board.

  Second, check if your drivers are properly installed. Several cards need additional OpenGL support drivers besides the ”native” windows ones. For more detail check Chapter 2.

  Third, check if your hardware driver is called `opengl32.dll` or just merely `opengl.dll`. By the default compilation, binaries are linked against `opengl32.dll`. If you require the non-32 version, consider rebuilding *FlightGear* with the libraries `opengl32.dll`, `glut32.dll`, and `glu32.dll` replaced by their non-32 counterparts. For more details check Chapter 3.

  If you installed the pre-compiled binaries `runfgfs.bat` invokes `fgfs.exe`

while `runfgfs.sgi.bat` invokes `fgfs.sgi.exe` with the first ones being linked against the 32-versions.

Usually, hardware accelerated drivers use the 32-libraries.

8.2 Potential problems under Linux

Since we don't have access to all possible flavors of Linux distributions, here are some thoughts on possible causes of problems. (This Section includes contributions by Kai Troester Kai.Troester@rz.tu-ilmenau.de.)

- Wrong library versions
  This is a rather common cause of grief especially when you prefer to install the libraries needed by *FlightGear* by hand. Be sure that especially the Mesa library contains support for the 3DFX board and that Glide libraries are installed and can be found. If a `ldd 'which fgfs'` complains about missing libraries you are in trouble.

- Missing permissions
  FlightGear needs to be setuid root in order to be capable of accessing the accelerator board. Be sure to issue a

  ```
  chown root.root /usr/local/bin/fgfs ;
  chmod 4755 /usr/local/bin/fgfs
  ```

  to give the *FlightGear* binary the proper rights. There is development of a device named `/dev/3dfx` underway, so this probably being remedied in the near future.

- Non-default install options
  FlightGear will display a lot of diagnostics when being started up. If it complains about bad looking or missing files, check that you installed them in the way they are supposed to be, i.e. latest version and proper location. The canonical location *FlightGear* wants its data files under `/usr/local/lib`. Be sure to grab the latest versions of everything that might be needed!

- Compile problems
  Check as far as you can, as a last resort (and a great information source, too) there are mailing lists for which information can be gotten at

  http://www.flightgear.org/mail.html.

  This will give you direct contact to the developers.

- Configure could not find Mesa and Glut though they are installed

  If the configure script could not find your Mesa and Glut libraries you should add the Mesa library-path (i.e. `/usr/local/Mesa`) to the EXTRA_DIRS variable in the file configure.in (i.e. `EXTRA_DIRS=''/usr/local/usr/X11R6/usr/local/Mesa''`). After this you have to run autoconf. (Please read README.autoconf for running autoconf)

- SuSE Distribution

  - If you have a SuSE distribution use the egcs compiler instead of the compiler delivered with SuSE. Grab it at

    http://egcs.cygnus.com
  - SuSE 6.0 users should also use the Glide, Mesa and Glut Libraries delivered with the distribution
  - A known problem of Flight Gear until version Version 0.57 with SuSE concerns `acconfig.h`. If 'make' stops and reports an error in relation with acconfig.h insert the following lines to `/usr/share/autoconf/acconfig.h`:

    ```
    /* needed to compile fgfs properly*/
    #undef FG_NDEBUG
    #undef PACKAGE
    #undef VERSION
    #undef WIN32a
    ```

    (a solution for this problem is coming soon)

  Additionally there are two versions of the GNU C compiler around: egcs and gcc (the classic one). gcc seems to have its own notion of some C++ constructs, so updating to egcs won't hurt and maybe help to compile the program.

8.3 Potential problems under Windows 98/NT

- The executable refuses to run.
  You may have tried to start the executable directly either by double-clicking `fgfs.exe` in Windows explorer or by invoking it in a MS-DOS shell. Double-clicking via explorer does never work (except you set the environment variable `FG_ROOT` in the autoexec.bat or otherwise). Rather double-click `runfgfs.bat` or `runfgfs-sgi.bat` For more detail, check Chapter 5.

Another potential problem might be you did not download the most recent versions of scenery and textures required by *FlightGear* , or you did not load any scenery or texture at all. Have a close look at this, as the scenery/texture format is still under development and may change frequently. For more detail, check Chapter 4.

A further potential source of trouble are so-called mini-OpenGL drivers provided by some manufacturers. In this case, *FlightGear* 's typically hangs while opening the graphics window. In this case, either replace the mini-OpenGL driver by a full OpenGL driver or or in case such is not available install software OpenGL support (see Section 2.5).

- *FlightGear* ignores the command line parameters.
  There is a problem with passing command line options containing a "=" to windows batch files. Instead, include the options into `runfgfs.bat`.

- While compiling with the Cygnus Compiler `Configure` complains not to find `glu32.dll`.

  Make sure you change to the Main FlightGear directory, e. g. with

  ```
  cd //D/FlightGear-X.XX
  ```

  before running `Configure` and `Make`. Do not forget the win32 library package.

- I am unable to build *FlightGear* under MSVC/MS DevStudio
  By default, *FlightGear* is build with GNU C++, i. e. the Cygnus compiler for Win32. For hints or Makefiles required for MSVC for MSC DevStudio have a look into

  http://www.flightgear.org/Downloads/Source.

  In principle, *FlightGear* should be buildable with the project files provided.

- Compilation of *FlightGear* dies not finding `gfc`.

  The library `gfc` cannot be build with the Cygnus compiler at present. It us supposed to be substituted by something else in the future.

  As the simulator is already built at this point, you simply can forget about that problem as long as you don't intend to build the scenery creation tools. Just go on with `make install`.

Index