FlightGear Flight Simulator – Installation and Getting Started

Michael Basler (pmb@epost.de) and Martin Spott (Martin.Spott@uni-duisburg.de)

including contributions by
Jon Berndt (jsb@hal-pc.org),
Bernhard Buckel (buckel@mail.uni-wuerzburg.de),
Cameron Moore (cameron@unbeatenpath.net),
Curt Olson (curt@flightgear.org),
Dave Perry (skidavem@mindspring.com),
Darrell Walisser dwaliss1@purdue.edu,
and others

Getting Started version 0.52
Date XXX

This documentation was written for ***FlightGear*** version 0.8.0.

Contents

Preface

FlightGear is a free Flight Simulator developed cooperatively over the Internet by a group of Flight Simulation and Programming Enthusiasts. This ”Installation and Getting Started” is meant to give beginners a guide in getting ***FlightGear*** up and running. It is not intended to provide complete documentation of all the features and add-ons of ***FlightGear*** but, instead, focuses on those aspects necessary to get into the air.

This guide is split into two parts. The first part describes how to install the program while the second part details on how to actually fly with ***FlightGear***.

The chapters concentrate on the following aspects:

Part I: Installation

Chapter 1, *Want to have a free flight? Take* ***FlightGear***, introduces the concept, describes the system requirements, and classifies the different versions available.

Chapter **??**, *Building the plane: Compiling the program*, explains how to build (compile and link) the simulator. Depending on your platform this may or may not be required. Generally, there will be executable programs (binaries) available for several platforms. Those on such systems who want to take off immediately, without going through the potentially troublesome process of compiling, may skip this Chapter.

In Chapter 2, *Preflight: Installing* ***FlightGear***, you will find instructions for installing the binaries in case you did not build them yourself as specified in the previous Chapter. You will need to install scenery, textures, and other support files collected in the base package.

Part II: Flying with *FlightGear*

The following Chapter 3, *Takeoff: How to start the program*, describes how to actually start the installed program. It includes an overview on the numerous command line options as well as configuration files.

Chapter 4, *In-flight: All about instruments, keystrokes and menus*, describes how to operate the program, i. e. how to actually fly with ***FlightGear***. This includes a (hopefully) complete list of pre-defined keyboard commands, an overview on the menu entries, detailed descriptions on the instrument panel and HUD (head up display), as well as hints on using the mouse functions.

In Appendix A, *Missed approach: If anything refuses to work*, we try to help you work through some common problems faced when using ***FlightGear***.

The Appendix B, *OpenGL graphics drivers*, describes some special problems you may encounter in case your system lacks support for the OpenGL graphics API OpenGL which ***FlightGear*** is based on.

In the final Appendix C, *Landing: Some further thoughts before leaving the plane*, we would like to give credit to those who deserve it, sketch an overview on the development of ***FlightGear***and point out what remains to be done.

Accordingly, we suggest reading the Chapters as follows:

Installation	
Users of binary distributions (notably under Windows):	2
Installation under Linux/UNIX:	**??**, 2
Installation under MacIntosh:	2
Operation	
Program start (all users):	3
Keycodes, Panel, Mouse. . . (all users):	4
Troubleshooting	
General issues:	A
Graphics problems:	B
Optionally	1, C

While this introductory guide is meant to be self contained, we strongly suggest having a look into further documentation, especially in case of trouble:

- For additional hints on troubleshooting and more, **please read the FAQ**

  http://www.flightgear.org/Docs/FlightGear-FAQ.html,

  The FAQ contains a host of valuable information, especially on rapidly changing flaws and additional reading, thus we strongly suggest consulting it in conjunction with our guide.

- A handy **leaflet** on operation for printout is available at

  http://www.flightgear.org/Docs/InstallGuide/FGShortRef.html,

- Additional user documentation on special aspects is available within the base package under the directory `/FlightGear/Docs`.

Finally:

We know, most people hate reading manuals. If you are sure the graphics driver for your card supports OpenGL (check documentation; for instance all NVIDIA Windows and Linux drivers for TNT/TNT2/Geforce/Geforce2/Geforce3 do) and if you are using one of the following operating systems:

- Windows 95/98/ME/NT/2000/XP,
- Macintosh Mac OSX
- Linux
- SGI Irix

you can possibly skip at least Part I of this manual and exploit the pre-compiled binaries. These as well as instructions on how to set them up, can be found at

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

In case you are running ***FlightGear*** on Linux, you may also be able to get binaries bundled with your distribution. Several vendors already include ***FlightGear*** binaries into their distributions.

Just download them, install them according to the description and run them via the installed `runfgfs` script or the batch file `runfgfs.bat`, respectively.

There is no guarantee for this approach to work, though. If it doesn't, don't give up! Have a closer look through this guide notably Section 2 and be sure to check out the FAQ.

Part I

Installation

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 ability to do so? Are you a real pilot looking to improve your 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 without any violence? If any of these questions apply to you, PC flight simulators are just for you.

You may already have some experience using Microsoft's © Flight Simulator 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 that running any serious PC flight simulator requires PC hardware within the $1500 range, despite dropping prices.

With so many commercially available flight simulators, why would we spend thousands of hours of programming and design work to build a free flight simulator? Well, there are many reasons, but here are the major ones:

- All of the commercial simulators have a serious drawback: they are made by a small group of developers defining their properties according to what is important to them and providing limited interfaces to end users. Anyone who has ever tried to contact a commercial developer would agree that getting your voice heard in that environment is a major challenge. In contrast, ***FlightGear*** is designed by the people and for the people with everything out in the open.
- Commercial simulators are usually a compromise of features and usability.

Most commercial developers want to be able to serve a broad segment of the population, including serious pilots, beginners, and even casual gamers. In reality the result is always a compromise due to deadlines and funding. As ***FlightGear*** is free and open, there is no need for such a compromise. We have no publisher breathing down our necks, and we're all volunteers that make our own deadlines. We are also at liberty to support markets that no commercial developer would consider viable, like the scientific research community.

- Due to their closed-source nature, commercial simulators keep developers with excellent ideas and skills from contributing to the products. With ***FlightGear***, developers of all skill levels and ideas have the potential to make a huge impact on the project. Contributing to a project as large and complex as ***FlightGear*** is very rewarding and provides the developers with a great deal of pride in knowing that we are shaping the future of a great simulator.

- Beyond everything else, it's just plain fun! I suppose you could compare us to real pilots that build kit-planes or scratch-builts. Sure, we can go out a buy a prebuilt aircraft, but there's just something special about building one yourself.

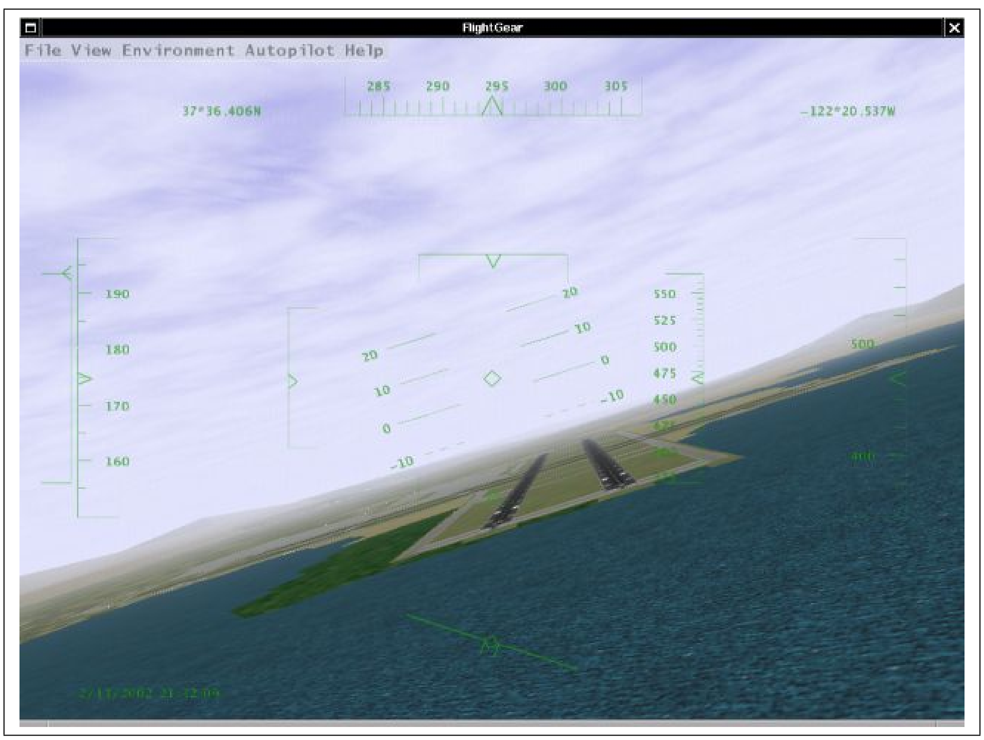

Fig. 1: ***FlightGear*** *under UNIX: Bad approach to San Francisco International* - *by one of the authors of this manual...*

The points mentioned above form the basis of why we created ***FlightGear***. With those motivations in mind, we have set out to create a high-quality flight simulator that aims to be a civilian, multi-platform, open, user-supported, and user-extensible platform. Let us take a closer look at each of these characteristics:

- **Civilian:** The project is primarily aimed at civilian flight simulation. It should be appropriate for simulating general aviation as well as civilian aircraft. Our long-term goal is to have ***FlightGear*** FAA-approved as a flight training device. To the disappointment of some users, it is currently not a combat simulator; however, these features are not explicitly excluded. We just have not had a developer that was seriously interested in systems necessary for combat simulation.

- **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 distribution and platform),
  - Windows NT/2000/XP (Intel/AMD platform),
  - Windows 95/98/ME,
  - BSD UNIX,
  - SGI IRIX,
  - Sun-OS,
  - Macintosh.

  At present, there is no known flight simulator – commercial or free – supporting such a broad range of platforms.

- **Open:** The project is not restricted to a static or elite cadre of developers. Anyone who feels they are able to contribute is most welcome. The code (including documentation) is copyrighted under the terms of the GNU General Public License (GPL).

  The GPL is often misunderstood. In simple terms it states that you can copy and freely distribute the program(s) so licensed. You can modify them if you like and even charge as much money as want to for the distribution of the modified or original program. However, you must freely provide the entire source code to anyone who wants it, and it must retain the original copyrights. In short:

”You can do anything with the software except make it non-free”.

The full text of the GPL can be obtained from the ***FlightGear*** source code or from

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

- **User-supported and user-extensible:** Unlike most commercial simulators, ***FlightGear***”s scenery and aircraft formats, internal variables, APIs, and everything else are user accessible and documented from the beginning. Even without any explicit development documentation (which naturally has to be written at some point), one can always go to the source code to see how something works. 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 artists, and others can build upon. It is our hope that the project, including the developers and end users, will benefit from the creativity and ideas of the hundreds of talented ”simmers” around the world.

Without doubt, the success of the Linux project, initiated by Linus Torvalds, inspired several of the developers. Not only has Linux shown that distributed development of highly sophisticated software projects over the Internet is possible, it has also proven that such an effort can surpass the level of quality of competing commercial products.

1.2 System Requirements

In comparison to other recent flight simulators, the system requirements for ***FlightGear*** are not extravagant. A decent PII/400, or something in that range, should be sufficient given you have a proper 3-D graphics card. Additionally, any modern UNIX-type workstation with a 3-D graphics card will handle ***FlightGear*** as well.

One important prerequisite for running ***FlightGear*** is a graphics card whose driver supports OpenGL. If you don’t know what OpenGL is, the overview given at the OpenGL website

http://www.opengl.org

says it best: “Since its introduction in 1992, OpenGL has become the industry’s most widely used and supported 2-D and 3-D graphics application programming interface (API)...”.

FlightGear does not run (and will never run) on a graphics board which only supports Direct3D. Contrary to OpenGL, Direct3D is a proprietary interface, being restricted to the Windows operating system.

You may be able to run ***FlightGear*** on a computer that features a 3-D video card not supporting hardware accelerated OpenGL – and even on systems without 3-D graphics hardware at all. However, the absence of hardware accelerated OpenGL support can bring even the fastest machine to its knees. The typical signal for missing hardware acceleration are frame rates below 1 frame per second.

Any modern 3-D graphics featuring OpenGL support will do. For Windows video card drivers that support OpenGL, visit the home page of your video card manufacturer. You should note that sometimes OpenGL drivers are provided by the manufacturers of the graphics chip instead of by the makers of the board. If you are going to buy a graphics card for running ***FlightGear***, one based on a NVIDIA chip (TNT X/Geforce X) might be a good choice.

To install the executable and basic scenery, you will need around 50 MB of free disk space. In case you want/have to to compile the program yourself you will need about an additional 500 MB for the source code and for temporary files created during compilation. This does not include the development environment, which will vary in size depending on the operating system and environment being used. Windows users can expect to need approximately 300 MB of additional disk space for the development environment. Linux and other UNIX machines should have most of the development tools already installed, so there is likely to be little additional space needed on those platforms.

For the sound effects, any capable sound card should suffice. Due to its flexible design, ***FlightGear*** supports a wide range of joysticks and yokes as well as rudder pedals under Linux and Windows. ***FlightGear*** can also provide interfaces to full-motion flight chairs.

FlightGear is being developed primarily under Linux, a free UNIX clone (together with lots of GNU utilities) developed cooperatively over the Internet in much the same spirit as ***FlightGear*** itself. ***FlightGear*** also runs and is partly developed under several flavors of Windows. Building ***FlightGear*** is also possible on a Macintosh OSX and several different UNIX/X11 workstations. Given you have a proper compiler installed, ***FlightGear*** can be built under all of these platforms. The primary compiler for all platforms is the free GNU C++ compiler (the Cygnus Cygwin compiler under Win32).

If you want to run ***FlightGear*** under Mac OSX we suggest a Power PC G3 300 MHz or better. As a graphics card we would suggest an ATI Rage 128 based card as a minimum. Joysticks are supported under Mac OS 9.x only; there is no joystick support under Max OSX at this time.

1.3 Choosing A Version

Concerning the ***FlightGear*** source code there exist two branches, a stable branch and a developmental branch. Even version numbers like 0.6, 0.8, and (someday hopefully) 1.0 refer to stable releases, while odd numbers like 0.7, 0.9, and so on refer to developmental releases. The policy is to only do bug fixes in the even versions, while new features are generally added to odd-numbered versions which, after all things have stabilized, will become the next stable release with a version number calculated by adding 0.1.

To add to the confusion, there usually are several versions of the "unstable" branch. First, there is a "latest official release" which the pre-compiled binaries are based on. It is available from

ftp://ftp.flightgear.org/pub/fgfs/Source/FlightGear-X.Y.Z.tar.gz

For developers there exist CVS snapshots of the source code, available from

ftp://www.flightgear.org/pub/flightgear/Devel/Snapshots/.

While theses are quite recent, they may still be sometimes a few days back behind development. Thus, if you really want to get the very latest and greatest (and, at times, buggiest) code, you can use a tool called anonymous cvs available from

http://www.cvshome.org/

to get the recent code. A detailed description of how to set this up for ***FlightGear*** can be found at

http://www.flightgear.org/cvsResources/.

Unfortunately, the system implemented above does not really work as it should. As a matter of fact, the stable version is usually so much outdated, that it does not at all reflect the state of development ***FlightGear*** has reached. Given that the recent developmental versions on the other hands may contain bugs (…undocumented features), we recommend using the "latest official (unstable) release" for the average user. This is the latest version named at

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

usually this is also the version which the binary distributions available at

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

are based on. If not otherwise stated, all procedures in this "Installation and Getting Started" will be based on these packages.

1.4 Flight Dynamics Models

Historically, ***FlightGear*** has been based on a flight model it inherited (together with the Navion airplane) from LaRCsim. As this had several limitations (most important, many characteristics were hard wired in contrast to using configuration files), there were several attempts to develop or include alternative flight models. As a result, ***FlightGear*** supports several different flight models, to be chosen from at runtime.

The most important one is the JSB flight model developed by Jon Berndt. Actually, the JSB flight model is part of a stand-alone project called ***JSBSim***, having its home at

http://jsbsim.sourceforge.net/.

Concerning airplanes, the JSB flight model at present provides support for a Cessna 172, a Cessna 182, a Cessna 310, and for an experimental plane called X15. Jon and his group are gearing towards a very accurate flight model, and the JSB model has become ***FlightGear***'s default flight model.

As an interesting alternative, Christian Mayer developed a flight model of a hot air balloon. Moreover, Curt Olson integrated a special slew mode called Magic Carpet, which helps you to quickly fly from point A to point B.

Recently, Andrew Ross contributed another flight model called ***YASim*** for *Yet Another Simulator*. At present, it sports another Cessna 172, a Turbo 310, a fairly good DC-3 model, along with a Boeing 747, Harrier, and A4. ***YASim*** takes a fundamentally different approach since it's based on geometry information rather than aerodynamic coefficients. Where JSBSim will be exact for every situation that is known and flight tested, but may have odd and/or unrealistic behavior outside normal flight, YASim will be sensible and consistent in almost every flight situation, but is likely to differ in performance numbers.

As a further alternative, there is the UIUC flight model. It was developed by a team from the University of Illinois at Urbana-Champaign to research aircraft icing. The project uses ***FlightGear*** as it's frontend for doing visual simulations. Learn more about this project at its homepage

http://amber.aae.uiuc.edu/ m-selig/apasim.html.

The UIUC provides a host of different aircraft including several Cessna C172, a Learjet 24, a Twin Otter and much more.

Please note, that the UIUC models do **not** have a working gear model. You might experience some difficulties when starting on the ground. The nose gear will be too weak, and the airplane will tend to fall on it's nose. This can be circumvented by starting in the air or pulling back on the stick more than usual.

It is even possible to drive FlightGear's scene display using an external FDM running on a different computer – although this might not be a setup recommended to people just getting in touch with ***FlightGear***.

1.5 About This Guide

There is little, if any, material in this Guide that is presented here exclusively. 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 herein can also be obtained from the ***FlightGear*** web site located at

http://www.flightgear.org/

Please, keep in mind that there are several mirrors of the ***FlightGear*** web sites, all of which are linked to from the ***FlightGear*** homepage listed above. You may prefer to download ***FlightGear*** from a mirror closer to you than from the main site.

This ***FlightGear*** *Installation and Getting Started* manual is intended to be a first step towards a complete ***FlightGear*** documentation. The target audience is the end-user who is not interested in the internal workings of OpenGL or in building his or her own scenery. It is our hope, that someday there will be an accompanying ***FlightGear*** *Programmer's Guide* (which could be based on some of the documentation found at

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

a ***FlightGear*** *Scenery Design Guide*, describing the Scenery tools now packaged as ***TerraGear***; and a ***FlightGear*** *Flight School* package.

As a supplement, we recommend reading the ***FlightGear*** FAQ to be found at
http://www.flightgear.org/Docs/FlightGear-FAQ.html
which has a lot of supplementary information that may not be included in this manual.

We kindly ask you to help us refine this document by submitting corrections, improvements, and suggestions. All users is invited to contribute descriptions of alternative setups (graphics cards, operating systems etc.). We will be more than happy to include those into future versions of this *Installation and Getting Started* (of course not without giving credit to the authors).

While we intend to continuously update this document, we may not be able to produce a new version for every single release of ***FlightGear***. To do so would require more manpower that we have now, so please feel free to jump in and help out. We hope to produce documentation that measures up to the quality of ***FlightGear*** itself.

Chapter 2

Preflight: Installing *FlightGear*

You can skip this Section if you built ***FlightGear*** along the lines described in the previous Chapter. If you did not and you're jumping in here, your first step will consist in installing the binaries. At present, there are pre-compiled binaries available for

- Windows (any flavor),
- Macintosh OSX,
- Linux,
- SGI Irix.

2.1 Installing the binary distribution on a Windows system

The following supposes you are on a Windows (95/98/Me/NT/2000/XP) system. Installing the binaries is quite simple. Go to

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

and download the three files `fgfs-base-X.X.X.zip`, `fgfs-manual-X.X.X.zip`, and `fgfs-win32-bin-X.X.X.zip` from

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

to a drive of your choice. Windows XP includes a program for unpacking *.zip files. If you are working under an older version of Windows, we suggest getting Winzip from http://www.winzip.com/.

For a free alternative, you may consider `unzip` from Info-ZIP,

http://www.info-zip.org/pub/infozip/

Extract the files named above. If you choose drive `c:` you should find a file `runfgfs.bat` under `c:/Flightgear` now. Double-clicking it should invoke the simulator.

In case of doubt about the correct directory structure, see the summary at the end of chapter **??**.

2.2 Installing the binary distribution on a Macintosh system

If your Macintosh is running the conventional Mac OS 9 or earlier, there are versions up to ***FlightGear*** 0.7.6 available being provided courtesy Darrell Walisser). Download the file `FlightGear_Installer_0.X.X.sit` from the corresponding subdirectory under

http://icdweb.cc.purdue.edu/ walisser/fg/.

This file contains the program as well as the required base package files (scenery etc.). For unpacking, use `Stuffit Expander 5.0` or later.

The latest build available for Mac OS 9.x is 0.7.6, located in the same place. The base package is part of the download for Mac OS 9.x, but not for Mac OSX.

Alternatively, if you are running Mac OS X, download `fgfs-0.X.X.gz` from the same site named above. The Mac OS X builds are in a gzip file in the same directory. There is a readme file in the directory to help people identify what to download.

Mac OS X requires that you first download the base package. Then extract it with

```
tar -zxvf fgfs-base-X.X.X.tar.gz
gunzip fgfs-X.X.X.-date.gz
```

Note that there is no `runfgfs` script for Mac OS X yet.

2.3 Installing the binary distribution on a Debian Linux system

Download the file `flightgear_0.7.6-6_i386.deb` (being provided courtesy Ove Kaaven) from any of the Debian mirror sites listed at http://packages.debian.org/unstable/games/flightgear.html.

Like any Debian package, this can be installed via

`dpkg --install flightgear_0.7.6-6_i386.deb`.

After installation, you will find the directory `/usr/local/Flightgear` containing the script `runfgfs` to start the program.

2.4 Installing the binary distribution on a SGI IRIX system

If there are binaries available for SGI IRIX systems, download all the required files (being provided courtesy Erik Hofman) from

http://www.a1.nl/ ehofman/fgfs/

and install them. Now you can start ***FlightGear*** via running the script `/opt/bin/fgfs`.

2.5 Installing add-on scenery

There is a complete set of scenery files worldwide available created by Curt Olson which can be downloaded via a clickable map at

http://www.flightgear.org/Downloads/world-scenery.html

Moreover, Curt provides the complete set of US Scenery on CD-ROM for those who really would like to fly over all of the USA. For more detail, check the remarks on the downloads page above.

For installing these files, you have to unpack them under `/Flightgear/Scenery`. Do not de-compress the numbered scenery files like 958402.gz! This will be done by ***FlightGear*** on the fly.

As an example, consider installation of the scenery package w120n30 containing the Grand Canyon Scenery.

After having installed the base package, you should have ended up with the following directory structure:

```
/usr/local/FlightGear/Scenery
/usr/local/FlightGear/w130n30
/usr/local/FlightGear/w130n30/w122n37
/usr/local/FlightGear/Scenery/w130n30/w123n37
```

with the directories w122n37 and w123n37m, resp. containing numerous *.gz files. Installation of the Grand Canyon scenery adds to this the directories

```
/usr/local/FlightGear/w120n30/w112n30
/usr/local/FlightGear/w120n30/w112n31
...
/usr/local/FlightGear/w120n30/w120n39.
```

2.6 Installing documentation

Most of the packages named above include the complete ***FlightGear*** documentation including 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.

Besides, the source code contains a directory `docs-mini` containing numerous ideas on and solutions to special problems. This is also a good place for further reading.

Part II

Flying with *FlightGear*

Chapter 3

Takeoff: How to start the program

3.1 Launching the simulator under Unix/Linux

Under Linux (or any other flavor of Unix), ***FlightGear*** will be invoked by

`runfgfs --option1 --option2...`,

where the options will be described in Section 3.4 below.

If something strange happens while using this shell script, if you want to do some debugging (i.e. using ”strace”) or if you just feel nice to be ”keen”, then you can start ***FlightGear*** directly by executing the ”fgfs” binary. In this case you should at least add one variable to your environment, which is needed to locate the (mostly) shared library built from the sources of the ***SimGear*** package. Please add the respective directory to your `LD_LIBRARY_PATH`. You can do so with the following on Bourne shell (compatibles):

```
LD_LIBRARY_PATH=/usr/local/FlightGear/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH/
```

or on C shell (compatibles):

```
setenv LD_LIBRARY_PATH
/usr/local/FlightGear/lib:$LD_LIBRARY_PATH
```

Besides this (used by the dynamic linker) ”fgfs” knows about the following environment variable:

`FG_ROOT`: root directory for the FlightGear base package,
which corresponds to the `--fg-root=`*path* option as described in Sec. 3.4.1

3.2 Launching the simulator under Windows

For launching ***FlightGear*** from Windows explorer, change to the directory `/FlightGear` and double-click the file `runfgfs.bat`.

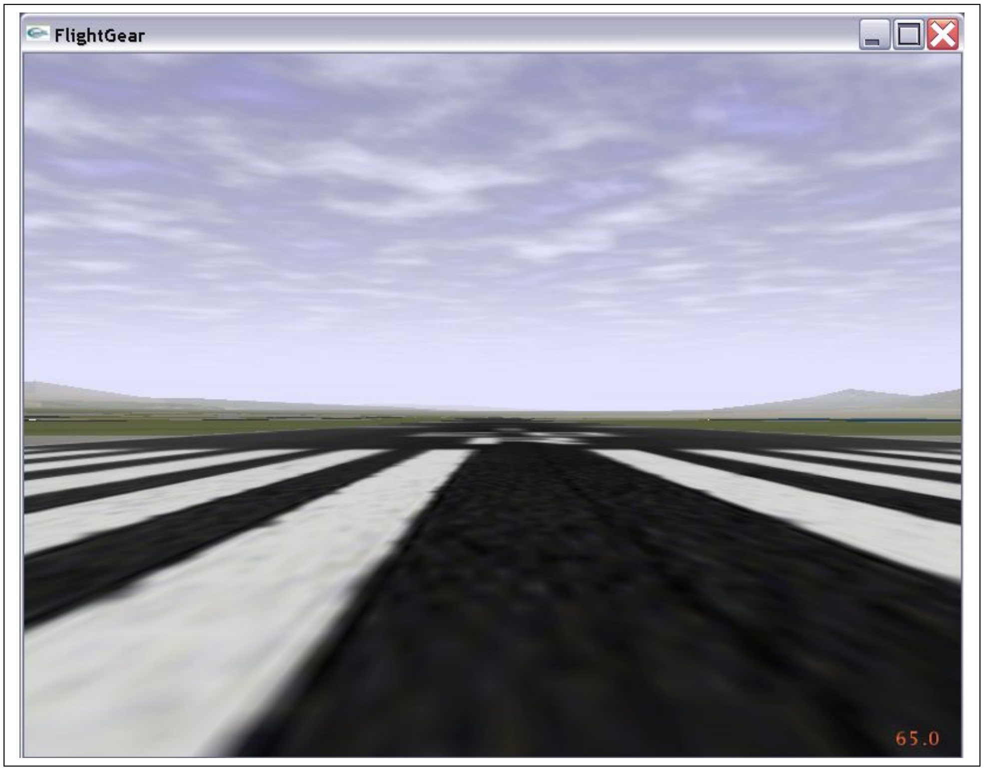

Fig. 3: *Ready for takeoff. Waiting at the default startup position at San Francisco Itl., KSFO.*

Alternatively, if for one or the other reason the batch file above does not work or is missing, you can open a command shell, change to the directory where your binary resides (typically something like `c:/FlightGear/bin` where you might have to substitute `c:` in favor of your ***FlightGear*** directory), set the environment variable via (note the backslashes!)

```
SET FG_ROOT=c:\FlightGear\
```

and invoke ***FlightGear*** (within the same MS-DOS shell, as environment settings are only valid locally within the same shell) via

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

Of course, you can create your own `runfgfs.bat` with Windows `Editor` using the two lines above.

For getting maximum performance it is recommended to minimize (iconize) the text output window while running ***FlightGear***.

3.3 Launching the simulator under Mac OS X

Say, you downloaded the base package and binary to your home directory. Then you can open `Terminal.app` and execute the following sequence:

`setenv FG_ROOT  /fgfs-base-X.X.X ./fgfs-X.X.X.-date --option1 -- option 2` (one line)

or

`./fgfs-X.X.X-version-date --fg-root=~/fgfs-base-X.X.X --option1 --option2`. (one line)

3.4 Command line parameters

Following is a complete list and short description of the numerous command line options available for ***FlightGear***. If you are running ***FlightGear*** under Windows you can include these into `runfgfs.bat`.

However, in case of options you want to re-use continually it is recommended to include them into a file called `.fgfsrc` under Unix systems and `system.fgfsrc`, resp. under Windows. This file has to be in the top ***FlightGear*** directory (for instance /usr/local/Flightgear). As it depends on your preferences, it is not delivered with ***FlightGear***, but can be created with any text editor (notepad, emacs, vi, if you like).

3.4.1 General Options

- `--help`: Shows the most relevant command line options only.
- `--help -verbose`: Shows all command line options.
- `--fg-root=`*path*: Tells ***FlightGear*** where to look for its root data files if you didn't compile it with the default settings.
- `--fg-scenery=`*path*: Allows specification of a path to the base scenery path , in case scenery is not at the default position under `$FG_ROOT/Scenery`; this might be especially useful in case you have scenery on a CD-ROM.

- `--disable-game-mode`: Disables full screen display.
- `--enable-game-mode`: Enables full screen display.
- `--disable-splash-screen`: Turns off the rotating 3DFX logo when the accelerator board gets initialized (3DFX only).
- `--enable-splash-screen`: If you like advertising, set this!
- `--disable-intro-music`: No audio sample is being played when ***FlightGear*** starts up. Suggested in case of trouble with playing the intro.
- `--enable-intro-music`: If your machine is powerful enough, enjoy this setting.
- `--disable-mouse-pointer`: Disables extra mouse pointer.
- `--enable-mouse-pointer`: Enables extra mouse pointer. Useful in full screen mode for old Voodoo based cards.
- `--disable-random-objects`: Exclude random scenery objects (buildings).
- `--enable-random-objects`: Include random scenery objects (buildings). This is the default.
- `--disable-freeze`: This will put you into ***FlightGear*** with the engine running, ready for Take-Off.
- `--enable-freeze`: Starts ***FlightGear*** in frozen state.
- `--disable-fuel-freeze`: Fuel is consumed normally.
- `--enable-fuel-freeze`: Fuel tank quantity is forced to remain constant.
- `--disable-clock-freeze`: Time of day advances normally.
- `--enable-clock-freeze`: Do not advance time of day.
- `--control-mode`: Specify your control device (joystick, keyboard, mouse) Defaults to joystick (yoke).
- `--disable-auto-coordination`: Switches auto coordination between aileron/rudder off (default).

- `--enable-auto-coordination`: Switches auto coordination between aileron/rudder on (recommended without pedals).

- `--browser-app=/path/to/app`: specify location of your web browser. Example: `--browser-app=`
`''C:\Programme\Internet Explorer\iexplore.exe''` (Note the " " because of the broken word Internet Explorer!).

- `--prop:name=value:` set property `name` to `value`
Example: `--prop:/engines/engine0/running=true` for starting with running engines. Another example:
`--aircraft=c172`
`--prop:/consumables/fuels/tank[0]/level-gal=10`
`--prop:/consumables/fuels/tank[1]/level-gal=10`
filles the Cessna for a short flight.

- `--config=path:` Load additional properties from the given path. Example: `runfgfs --config=./Aircraft/X15-set.xml`

- `--units-feed`: Use feet for distances.

- `--units-meters`: Use meters for distances.

3.4.2 Features

- `--disable-hud`: Switches off the HUD (**H**ead **U**p **D**isplay).

- `--enable-hud`: Turns the HUD on.

- `--enable-anti-aliased-hud`: Turns on anti-aliaseded HUD lines for better quality, if hardware supports this.

- `--disable-anti-aliased-hud`: Turns off anti-aliaseded HUD lines.

- `--enable-panel`: Turns the instrument panel on (default).

- `--disable-panel`: Turns the instrument panel off.

- `--disable-sound`: Self explaining.

- `--enable-sound`: See above.

3.4.3 Aircraft

- `--aircraft=`*name of aircraft definition file* Example: `--aircraft=c310`. For possible choices check the directory `/FlightGear/Aircraft`. Do not include the extension `''-set.xml''` into the aircraft name but use the remaining beginning of the respective file names for choosing an aircraft. This way flight model, panel etc. are all loaded in a consistent way. For a full list, see Sec. 3.6 below.

- `--show-aircraft`: Print a list of the currently available aircraft types.

3.4.4 Flight model

- `--fdm=`*abcd* Select the core flight model. Options are `jsb, larcsim, yasim, magic, balloon, external, ada, null`. Default value is `jsb` (***JSBSim***). larcsim is the flight model which ***FlightGear*** inherited from the LaRCSim simulator. yasim is Any Ross' Yet Another Flight Dynamics Simulator. Magic is a slew mode. Balloon is a hot air balloon. External refers to remote control of the simulator. Null selects no flight dynamics model at all. The UIUC flight model is not chosen this way but via the next option! For further information on flight models cf. Section 1.4 and below.

- `--aero=`*abcd* Specifies the aircraft model to load. Default is a Cessna c172. Alternatives available depend on the flight model chosen.

- `--model-hz=`*n* Run the Flight Dynamics Model with this rate (iterations per second).

- `--speed=`*n* Run the Flight Dynamics Model this much faster than real time.

- `--notrim` Do NOT attempt to trim the model when initializing JSBSim.

- `--on-ground`: Start up at ground level (default).

- `--in-air`: Start up in the air. Naturally, you have to specify an initial altitude as below for this to make sense. This is a must for the X15.

- `--wind=`*DIR@SPEED*: Specify wind coming from the direction DIR (in degrees) at speed SPEED (knots).

3.4.5 Aircraft model directory (Only for the UIUC Flight Dynamics Model)

- `--aircraft-dir=`*path*: Aircraft directory relative to the root-path, defined via `$FG_ROOT` or `--fg-root`.

Remark: The difference in the handling of UIUC models has historic reasons. These models use the LaRCsim FDM. As this FDM isn't the default FDM any more you have to specify it manually. Also the airplane description needs manual interaction as you have to specify the directory by hand where the specific aircraft data resides. So you have to use the following for flying the 'TwinOtter':

```
fgfs --fdm=larcsim --aero=uiuc
--aircraft-dir=Aircraft-uiuc/TwinOtter
```

Fortunately work has been done to simplificate this. At least those airplanes can be flown easily by using an appropriate '--aircraft'-string. These are the following:

```
--aircraft=747-uiuc, --aircraft=beech99-uiuc,
--aircraft=c172-uiuc, --aircraft=c310-uiuc
```

If time permits the remaining aircrafts will be adjusted soon. Please have a look at `$FG_ROOT/Aircraft-uiuc` for the avaliable aircrafts provided by the UIUC model collection. Also please read the notes in Section 1.4 on UIUC.

3.4.6 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 etc. A long/short list of the IDs of the airports being implemented can be found in `/FlightGear/Airports`. You only have to unpack one of the files with `gunzip`. Keep in mind, you need the terrain data for the relevant region, though!

- `--offset-distance=`*nm*: Here you can specify the distance to threshold in nm.

- `--offset-azimuth=`*deg*: Here you can specify the heading to threshold in degrees.

- `--lon=`*degrees*: This is the startup longitude in degrees (west = -).

- `--lat=`*degrees*: This is the startup latitude in degrees (south = -).

- `--altitude=`*feet*: This is useful if you want to start in free flight in connection with `--in-air`. Altitude specified in feet unless you choose `--units-meters`.

- `--heading=`*degrees*: Sets the initial heading (yaw angle) in degrees.
- `--roll=`*degrees*: Sets the startup roll angle (roll angle) in degrees.
- `--pitch=`*degrees*: Sets the startup pitch angle (pitch angle) in degrees.
- `--uBody=`*feet per second*: Speed along the body X axis in feet per second, unless you choose `--units-meters`.
- `--vBody=`*feet per second*: Speed along the body Y axis in feet per second, unless you choose `--units-meters`.
- `--wBody=`*feet per second*: Speed along the body Z axis in feet per second, unless you choose `--units-meters`.
- `--vc=`*knots*: Allows specifying the initial airspeed in knots (only in connection with `--fdm=jsb`).
- `--mach=`*num*: Allows specifying the initial airspeed as Mach number (only in connection with `--fdm=jsb`).
- `--glideslope=`*degrees*: Allows specifying the flight path angle (can be positive).
- `--roc=`*fpm*: Allows specifying the initial climb rate (can be negative).

3.4.7 Rendering Options

- `--bpp=`*depth*: Specify the bits per pixel.
- `--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 rate will increase.
- `--fog-nicest`: This option will give you a fairly realistic view of flying on a hazy day.
- `--enable-clouds`: Enable cloud layer (default).
- `--disable-clouds`: Disable cloud layer.
- `--fov=`*defrees*: Sets the field of view in degrees. Default is 55.0.

- `--disable-fullscreen`: Disable full screen mode (default).
- `--enable-fullscreen`: Enable full screen mode.
- `--shading-flat`: This is the fastest mode but the terrain will look ugly! This option might help if your video processor 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*** looks like internally, try this!
- `--disable-wireframe`: No wireframe. Default.
- `--geometry=`*WWWxHHH*: Defines the size of the window used, i.e. `WWWxHHH` can be `640x480`, `800x600`, or `1024x768`.
- `--view-offset=`*xxx*: Allows setting the default forward view direction as an offset from straight ahead. Possible values are `LEFT, RIGHT, CENTER`, or a specific number of degrees. Useful for multi-window display.
- `--visibility=`*meters*: You can specify the initial visibility in meters here.
- `--visibility-miles=`*miles*: You can specify the initial visibility in miles here.

3.4.8 HUD Options

- `--hud-tris`: HUD displays the number of triangles rendered.
- `--hud-culled`: HUD displays percentage of triangles culled.

3.4.9 Time Options

- `--time-offset=`*[+-]hh:mm:ss*: Offset local time by this amount.
- `--time-match-real`: Synchronize real-world and ***FlightGear*** time.
- `--time-match-local`: Synchronize local real-world and ***FlightGear*** time.
- `--start-date-sys=`*yyyy:mm:dd:hh:mm:ss*: Specify a starting time and date. Uses your system time.
- `--start-date-gmt=`*yyyy:mm:dd:hh:mm:ss*: Specify a starting time and date. Time is Greenwich Mean Time.
- `--start-date-lat=`*yyyy:mm:dd:hh:mm:ss*: Specify a starting time and date. Uses local aircraft time.

3.4.10 Network Options

- `--httpd=`*port*: Enable http server on the specified port.
- `--telnet=`*port*: Enable telnet server on the specified port.
- `--jpg-httpd=`*port*: Enable screenshot http server on the specified port.
- `--enable-network-olk`: Enables Oliver Delises's Multipilot mode.
- `--disable-network-olk`: Disables Oliver Delises's Multipilot mode (default).
- `--net-hud`: HUD displays network info.
- `--net-id=`*name*: Specify your own callsign

3.4.11 Route/Waypoint Options

- `--wp=`*ID[@alt]*: Allows specifying a waypoint for the GC autopilot; it is possible to specify multiple waypoints (i.e. a route) via multiple instances of this command.
- `--flight-plan=`*[file]*: This is more comfortable if you have several waypoints. You can specify a file to read them from.

Note: These options are rather geared to the advanced user who knows what he is doing.

3.4.12 IO Options

- `--garmin=`*params*: Open connection using the Garmin GPS protocol.
- `--joyclient=`*params*: Open connection to an Agwagon joystick.
- `--native-ctrls=`*params*: Open connection using the FG native Controls protocol.
- `--native-fdm=`*params*: Open connection using the FG Native FDM protocol.
- `--native=`*params*: Open connection using the FG Native protocol.
- `--nmea=`*params*: Open connection using the NMEA protocol.
- `--opengc=`*params*: Open connection using the OpenGC protocol.
- `--props=`*params*: Open connection using the interactive property manager.
- `--pve=`*params*: Open connection using the PVE protocol.
- `--ray=`*params*: Open connection using the RayWoodworth motion chair protocol.
- `--rul=`*params*: Open connection using the RUL protocol.
- `--atc610x`: Enable atc610x interface.

3.4.13 Debugging options

- `--trace-read=`*params*: Trace the reads for a property; multiple instances are allowed.
- `--trace-write=`*params*: Trace the writes for a property; multiple instances are allowed.

3.5 Joystick support

Could you imagine a pilot in his or her Cessna controlling the machine with a keyboard alone? For getting the proper feeling of flight you will need a joystick/yoke plus rudder pedals, right? However, the combination of numerous types of joysticks, flightsticks, yokes, pedals etc. on the market with the several target operating systems, makes joystick support a nontrivial task in ***FlightGear***.

Beginning with version 0.8.0, ***FlightGear*** has an integrated joystick support, which automatically detects any joystick, yoke, or pedals attached. Just try it! If this does work for you, lean back and be happy!

Unfortunately, given the several combinations of operating systems supported by ***FlightGear*** (possibly in foreign languages) and joysticks available, chances are your joystick does not work out of the box. Basically, there are two alternative approaches to get it going, with the first one being preferred.

3.5.1 Built-in joystick support

General remarks

In order for joystick autodetection to work, a joystick bindings xml file must exist for each joystick. This file describes what axes and buttons are to be used to control which functions in ***FlightGear***. The associations between functions and axes or buttons are called ”bindings”. This bindings file can have any name as long as a corresponding entry exits in the joysticks description file

```
/FlightGear/joysticks.xml
```

which tells ***FlightGear*** where to look for all the bindings files. We will look at examples later.

FlightGear includes several such bindings files for several joystick manufacturers in folders named for each manufacturer. For example, if you have a CH Products joystick, look in the folder

```
/FlightGear/Input/Joysticks/CH
```

for a file that might work for your joystick. If such a file exists and your joystick is working with other applications, then it should work with ***FlightGear*** the first time you run it. If such a file does not exist, then we will discuss in a later section how to create such a file by cutting and pasting bindings from the examples that are included with ***FlightGear***.

Verifying your joystick is working

Does your computer see your joystick? One way to answer this question under Linux is to reboot your system and immediately enter on the command line

```
dmesg | grep Joystick
```

which pipes the boot message to grep which then prints every line in the boot message that contains the string ”Joystick”. When you do this with a Saitek joystick attached, you will see a line similar to this one:

```
input0: USB HID v1.00 Joystick [SAITEK CYBORG 3D USB] on
usb2:3.0
```

This line tells us that a joystick has identified itself as SAITEK CYBORG 3D USB to the operating system. It does not tell us that the joystick driver sees your joystick. If you are working under Windows, the method above does not work, but you can still go on with the next paragraph.

Confirming that the driver recognizes your joystick

FlightGear ships with a utility called js_demo. It will report the number of joysticks attached to a system, their respective ”names”, and their capabilities. Under Linux, you can run js_demo from the folder `/FlightGear/bin` as follows:

```
$ cd /usr/local/FlightGear/bin
$ ./js_demo
```

Under Windows, open a command shell (Start|All Programs|Accessoires), go to the ***FlightGear*** binary folder and start the program as follows (given ***FlightGear*** is installed under `c:\Flightgear`)

```
cd \FlightGear\bin
js_demo.exe
```

On our system, the first few lines of output are (stop the program with ˆC if it is quickly scrolling past your window!) as follows:

```
Joystick test program.
Joystick 0: ''CH PRODUCTS CH FLIGHT SIM YOKE USB ''
Joystick 1: ''CH PRODUCTS CH PRO PEDALS USB''
Joystick 2 not detected
Joystick 3 not detected
Joystick 4 not detected
Joystick 5 not detected
Joystick 6 not detected
Joystick 7 not detected
+--------------------JS.0----------------------+--------------------JS.1----------------------+
| Btns Ax:0 Ax:1 Ax:2 Ax:3 Ax:4 Ax:5 Ax:6 | Btns Ax:0 Ax:1 Ax:2 |
+----------------------------------------------+----------------------------------------------+
| 0000 +0.0 +0.0 +1.0 -1.0 -1.0 +0.0 +0.0 .  | 0000 -1.0 -1.0 -1.0 .  .  .  .  .  |
```

First note that js_demo reports which number is assigned to each joystick recognized by the driver. Also, note that the ”name” each joystick reports is also included between quotes. We will need the names for each bindings file when we begin writing the binding xml files for each joystick.

Identifying the numbering of axes and buttons

Axis and button numbers can be identified using js_demo as follows. By observing the output of js_demo while working your joystick axes and buttons you can determine what axis and button numbers are assigned to each joystick axis and button. It should be noted that numbering generally starts with zero.

The buttons are handled internally as a binary number in which bit 0 (the least significant bit) represents button 0, bit 1 represents button 1, etc., but this number is displayed on the screen in hexadecimal notation, so:

0001 ⇒ button 0 pressed
0002 ⇒ button 1 pressed
0004 ⇒ button 2 pressed
0008 ⇒ button 3 pressed
0010 ⇒ button 4 pressed
0020 ⇒ button 5 pressed
0040 ⇒ button 6 pressed
... etc. up to ...
8000 ⇒ button 15 pressed
... and ...
0014 ⇒ buttons 2 and 4 pressed simultaneously
... etc.

For Linux users, there is another option for identifying the ”name” and the numbers assigned to each axis and button. Most Linux distributions include a very handy program, ”jstest”. With a CH Product Yoke plugged into the system, the following output lines are displayed by jstest:

```
jstest /dev/js3

Joystick (CH PRODUCT FLIGHT SIM YOKE USB) has 7 axes and 12 buttons.  Driver version is 2.1.0

Axes:  0:  0 1:  0 2:  32767 3:32767 4:32767 5:  0 5:  0

Buttons:  0:off 1:off 2:off 3:off 4:off 5:off 6:off 7:off ...  11:off
```

Note the ”name” between parentheses. This is the name the system associates with your joystick.

When you move any control, the numbers change after the axis number corresponding to that moving control and when you depress any button, the ”off” after the button number corresponding to the button pressed changes to ”on”. In this way, you can quickly write down the axes numbers and button numbers for each function without messing with binary.

Writing or editing joystick binding xml files

At this point, you have confirmed that the operating system and the joystick driver both recognize your joystick(s). You also know of several ways to identify the joystick ”name” your joystick reports to the driver and operating system. You will need a written list of what control functions you wish to have assigned to which axis and button and the corresponding numbers.

Make the following table from what you learned from js_demo or jstest above (pencil and tablet is fine). Here we assume there are 5 axes including 2 axes associated with the hat.

Axis	Button
elevator = 0	view cycle = 0
rudder = 1	all brakes = 1
aileron = 2	up trim = 2
throttle = 3	down trim = 3
leftright hat = 4	extend flaps = 4
foreaft hat = 5	retract flaps = 5
	decrease RPM = 6
	increase RPM = 7

We will assume that our hypothetical joystick supplies the ”name” QUICK STICK 3D USB to the system and driver. With all the examples included with ***FlightGear***, the easiest way to get a so far unsupported joystick to be auto detected, is to edit an existing binding xml file. Look at the xml files in the subfolders of `/FlightGear/Input/Joysticks/`. After evaluating at several of the xml binding files supplied with ***FlightGear***, we decide to edit the file `/FlightGear/Input/Joysticks/Saitek/Cyborg-Gold-3d-USB.xml`. This file has all the axes functions above assigned to axes and all the button functions above assigned to buttons. This makes our editing almost trivial.

Before we begin to edit, we need to choose a name for our bindings xml file, create the folder for the QS joysticks, and copy the original xml file into this directory with this name.

```
$ cd /usr/local/FlightGear/Input/Joysticks
$ mkdir QS
$ cd QS
$ cp /usr/local/FlightGear/Input/Joysticks/Saitek/
Cyborg-Gold-3d-USB.xml QuickStick.xml
```

Here, we obviously have supposed a Linux/UNIX system with ***FlightGear*** being installed under `/usr/local/FlightGear`. For a similar procedure under Windows with ***FlightGear*** being installed under `c:`***FlightGear***, open a command shell and type

```
c:
cd /FlightGear/Input/Joysticks
mkdir QS
cd QS
copy /FlightGear/Input/Joysticks/Saitek/
Cyborg-Gold-3d-USB.xml QuickStick.xml
```

Next, open `QuickStick.xml` with your favorite editor. Before we forget to change the joystick name, search for the line containing <name>. You should find the line

```
<name>SAITEK CYBORG 3D USB</name>
```

and change it to

```
<name>QUICK STICK 3D USB</name>.
```

This line illustrates a key feature of xml statements. They begin with a <tag> and end with a </tag>.

You can now compare your table to the comment table at the top of your file copy. Note that the comments tell us that the Saitek elevator was assigned to axis 1. Search for the string

```
<axis n="2">
```

and change this to

```
<axis n="1">.
```

Next, note that the Saitek rudder was assigned to axis 2. Search for the string

```
<axis n="2">
<axis n="1">.
```

Continue comparing your table with the comment table for the Saitek and changing the axis numbers and button numbers accordingly. Since QUICKSTICK USB and the Saitek have the same number of axes but different number of buttons, you must delete the buttons left over. Just remember to double check that you have a closing tag for each opening tag or you will get an error using the file.

Finally, be good to yourself (and others when you submit your new binding file to a ***FlightGear*** developers or users archive!), take the time to change the comment table in the edited file to match your changed axis and button assignments. The new comments should match the table you made from the js_demo output. Save your edits.

Telling *FlightGear* about your new bindings xml file

Before ***FlightGear*** can use your new xml file, you need to edit the file
`/FlightGear/joysticks.xml`,
adding a line that will include your new file if the "name" you entered between the name tags matches the name supplied to the driver by your joystick. Add the following line to `joysticks.xml`.

```
<js-named include="Input/Joysticks/QS/QuickStick.xml"/>
```

Some hints for Windows users

Basically, the procedures described above should work for Windows as well. If your joystick/yoke/pedals work out of the box or if you get it to work using the methods above, fine. Unfortunately there may be a few problems.

The first one concerns uwers of non-US Windows versions. As stated above, you can get the name of the joystick from the program js_demo. If you have a non-US version of Windows and the joystick .xml files named above do not contain that special name, just add it on top of the appropriate file in the style of

```
<name>Microsoft-PC-Joysticktreiber </name>
```

No new entry in the base `joysticks.xml` file is required.

Unfortunately, there is one more loophole with Windows joystick support. In case you have two USB devices attached (for instance a yoke plus pedals), there may be cases, where the same driver name is reported twice. In this case, you can get at least the yoke to work by assigning it number 0 (out of 0 and 1). For this purpose, rotate the yoke (aileron control) and observe the output of js_demo. If figures in the first group of colons (for device 0) change, assignment is correct. If figures in the second group of colons (for device 1) change, you have to make the yoke the preferred device first. For doing so, enter the Windows "Control panel", open "Game controllers" and select the "Advanced" button. Here you can select the yoke as the "Preferred" device. Afterward you can check that assignment by running js_demo again. The yoke should now control the first group of figures.

Unfortunately, we did not find a way to get the pedals to work, too, that way. Thus, in cases like this one (and others) you may want to try an alternative method of assigning joystick controls.

3.5.2 Joystick support via .fgfsrc entries

Fortunately, there is a tool available now, which takes most of the burden form the average user who, maybe, is not that experienced with XML, the language which these files are written in.

For configuring your joystick using this approach, open a command shell (command prompt(DOS shell under windows, to be found unter Start—All programs—Accessories). Change to the directory `/FlightGear/bin` via e.g. (modify to your path)

```
cd c:\FlightGear\bin
```

and invoke the tool fgjs via

```
fgjs
```

on a UNIX/Linux machine, or via

```
fgjs.exe
```

on a Windows machine. The program will tell you which joysticks, if any, where detected. Now follow the commands given on screen, i.e. move the axis and press the buttons as required. Be careful, a minor touch already "counts" as a movement. Check the reports on screen. If you feel something went wrong, just re-start the program

After you are done with all the axis/switches, the directory above will hold a file called `fgfsrc.js`. If the ***FlightGear*** base directory `FlighGear` does not already contain an options file `.fgfsrc` (under UNIX)/`system.fgfsrc` (under Windows) mentioned above, just copy

`fgfsrc.js` into `.fgfsrc` (UNIX)/`system.fgfsrc` (Windows)

and place it into the directory ***FlightGear*** base directory `FlighGear`. In case you already wrote an options file, just open it as well as `fgfsrc.js` with an editor and copy the entries from `fgfsrc.js` into `.fgfsrc`/`system.fgfsrc`. One hint: The output of `fgjs` is UNIX formatted. As a result, Windows Editor may not display it the proper way. I suggest getting an editor being able to handle UNIX files as well. My favorite freeware file editor for that purpose, although somewhat dated, is PFE still, to be obtained from

http://www.lancs.ac.uk/people/cpaap/pfe/.

The the axis/button assignment of `fgjs` should, at least, get the axis assignments right, its output may need some tweaking. There may be axis moving the opposite way the should, the dead zones may be too small etc. For instance, I had to change

```
--prop:/input/joysticks/js[1]/axis[1]/binding/factor=-1.0
```

into

```
--prop:/input/joysticks/js[1]/axis[1]/binding/factor=1.0
```

(USB CH Flightsim Yoke under Windows XP). Thus, here is a short introduction into the assignments of joystick properties.

Basically, all axes settings are specified via lines having the following structure:

```
--prop:/input/joysticks/js[n]/axis[m]
/binding/command=property-scale
--prop:/input/joysticks/js[n]/axis[m]
/binding/property=/controls/steering option
--prop:/input/joysticks/js[n]/axis[m]
/binding/dead-band=db --prop:/input/joysticks/js[n]/axis[m]
/binding/offset=os --prop:/input/joysticks/js[n]/axis[m]
/binding/factor=fa
```

where

n	=	number of device (usually starting with 0)
m	=	number of axis (usually starting with 0)
steering option	=	elevator, aileron, rudder, throttle, mixture, pitch
dead-band	=	range, within which signals are discarded; useful to avoid jittering for minor yoke movements
offset	=	specifies, if device not centered in its neutral position
factor	=	controls sensitivity of that axis; defaults to +1, with a value of -1 reversing the behavior

You should be able to at least get your joystick working along these lines. Concerning all the finer points, for instance, getting the joystick buttons working, John Check has written a very useful README being included in the base package to be found under `FlightGear/Docs/Readme/Joystick.html`. In case of any trouble with your input device, it is highly recommended to have a look into this document.

3.6 A glance over our hangar

The following is a Table 1 of all the aircraft presently available for use with ***FlightGear***. In the first column, you will find the name of the aircraft, the second one tells the start option, the third one names the FDM (flight dynamics management model, see Sec. 1.4), and the last column includes some remarks. Here, "no exterior model" means, that there is no aircraft specific external view available until now. As a result, you will see the default blue-yellow glider, when you change to the external view, which might look a bit strange in certain cases.

Moreover, this list is complete insofar as it covers all aircraft available via the `--aircraft=` option. There are more aircraft available via the start options mentioned in Sec. 3.4.4. To find aout which ones, have a look into the folder `Aircraft-uiuc`.

Tab. 1: *Presently available aircraft.*

Aircraft type	Start option	FDM	Remarks
Boeing 747	`--aircraft=747-yasim`	YASim	
BA A4 Hawk	`--aircraft=a4-yasim`	YASim	
Boeing 747	`--aircraft=747-uiuc`	UIUC	no exterior model
North American X-15	`--aircraft=X15`	JSBSim	experimental supersonic plane
Airwave Xtreme 150	`--aircraft=airwaveXtreme150-v1-nl-uiuc`	UIUC	experimental GA
Beech 99	`--aircraft=beech99-uiuc`	UIUC	no exterior model
Beech 99	`--aircraft=beech99-v1-uiuc`	UIUC	no exterior model
Cessna 172	`--aircraft=c172-3d`	JSBSim	sports a 3D cockpit
Cessna 172	`--aircraft=c172-3d-yasim`	YASim	sports a 3D cockpit
Cessna 172	`--aircraft=c172-ifr`	JSBSim	with IFR panel
Cessna 172	`--aircraft=c172-larcsim`	LaRCsim	
Cessna 172	`--aircraft=c172`	JSBSim	default
Cessna 172	`--aircraft=c172-uiuc`	UIUC	
Cessna 172	`--aircraft=c172-yasim`	YASim	
Cessna 172p	`--aircraft=c172p-3d`	JSBSim	sports a 3D cockpit
Cessna 172p	`--aircraft=c172p`	JSBSim	
Cessna 172	`--aircraft=c172x`	JSBSim	flight dynamics testbed
Cessna 182	`--aircraft=c182`	JSBSim	
Cessna 310	`--aircraft=c310`	JSBSim	
Cessna 310	`--aircraft=c310-uiuc`	UIUC	
Cessna 310	`--aircraft=c310-yasim`	YASim	twin-prop machine
Cessna 310U3A	`--aircraft=c310u3a-3d`	JSBSim	twin-prop machine, 3D cockpit
Cessna 310U3A	`--aircraft=c310u3a`	JSBSim	twin-prop machine
Douglas DC-3	`--aircraft=dc3-yasim`	YASim	
BA Harrier	`--aircraft=harrier-yasim`	YASim	no exterior model
Piper Cub J3 Trainer	`--aircraft=j3cub-yasim`	YASim	
Siai Marchetti S.211	`--aircraft=marchetti-uiuc`	UIUC	no exterior model
Siai Marchetti S.211	`--aircraft=marchetti-v1-uiuc`	UIUC	no exterior model
Space Shuttle	`--aircraft=shuttle`	JSBSim	no exterior model
UFO	`--aircraft=ufo`	JSBSim	'White Project' (UNESCO)
1903 Wright Flyer	`--aircraft=wrightFlyer1903-v1-nl-uiuc`	UIUC	historical model
X-24B	`--aircraft=x24b`	JSBSim	USAF/NACA reentry testbed
Cessna 172	`--aircraft=c172-610x`	JSBSim	full screen, hi-res panel (IFR)

Chapter 4

In-flight: All about instruments, keystrokes and menus

The following 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. Later on, they were supplemented by several menu entries, making the interface more accessible, particularly for beginners, and providing additional functionality.

For getting a real feeling of flight, you should definitely consider getting a joystick or – preferred – a yoke plus rudder pedals. In any case, you can specify your device of choice for control via the `--control-mode` option, i.e. select joystick, keyboard, mouse. The default setting is joystick. Concerning instruments, there are again two alternatives: You can use the panel or the HUD.

A short leaflet based on this chapter can be found at

http://www.flightgear.org/Docs/InstallGuide/FGShortRef.html.

A version of this leaflet can also be opened via ***FlightGear***'s help menu.

4.1 Starting the engine

Depending on your situation, when you start the simulator the engines may be on or off. When they are on you just can go on with the start. When they are off, you have to start them first. The ignition switch for starting the engine is situated in the lower left corner of the panel. It is shown in Fig. 4.

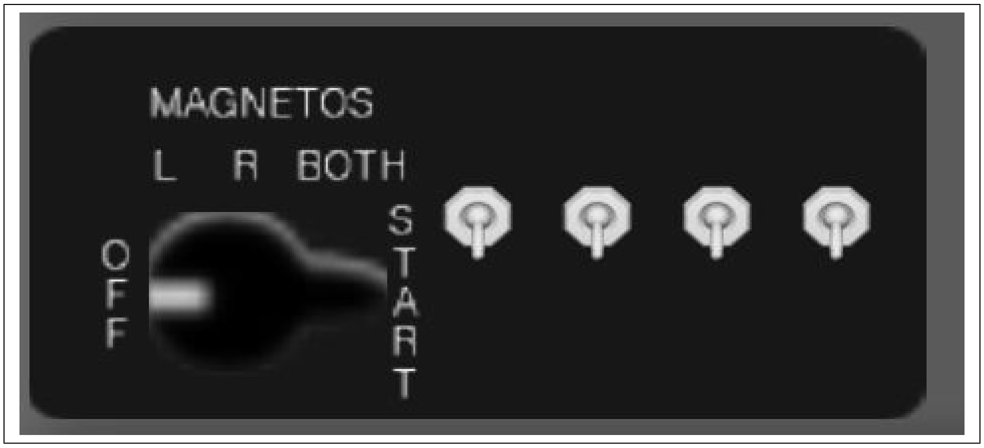

Fig. 4: *The ignition switch.*

It has five positions: ”OFF”, ”L”, ”R”, ”BOTH”, and ”START”. The extreme right position is for starting the engine. For starting the engine, put it onto the position ”BOTH” using the mouse first.

Keep in mind that the mixture lever has to be at 100 % (all the way in) for starting the engine – otherwise you will fail. In addition, advance the throttle to about 25 %.

Operate the starter using the SPACE key now. When pressing the SPACE key you will observe the ignition switch to change to the position ”START” and the engine to start after a few seconds. Afterwards you can bring the throttle back to idle (all the way out).

In addition, have a look if the parking brakes are on (red field lit). If so, press the ”B” button to release them.

4.2 Keyboard controls

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

After activating `NumLock` the following main keyboard controls for driving the plane should work:

Tab. 2: *Main keyboard controls for **FlightGear** on the numeric keypad with activated* `NumLock` *key:*.

Key	Action
9/3	Throttle
4/6	Aileron
8/2	Elevator
0/,	Rudder
5	Center aileron/elevator/rudder
7/1	Elevator trim

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

Tab. 3: *View directions accessible after de-activating* `NumLock` *on the numeric keypad.*

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

Besides, there are several more options for adapting display on screen:

Tab. 4: *Display options*

Key	Action
P	Toggle instrument panel on/off
c	Toggle3D/2D cockpit
s	Cycle panel style full/mini
Shift-F5/F6	Shift the panel in y direction
Shift-F7/F8	Shift the panel in x direction
Shift-F3	Read a panel from a property list
i/I	Minimize/maximize HUD
h/H	Change color of HUD/toggle HUD off forward/backward
x/X	Zoom in/out
v	Cycle view modes
W	Toggle full screen mode on/off (3dfx only)
z/Z	Change visibility (fog) forward/backward
F8	Toggle fog on/off
F2	Refresh Scenery tile cache
F4	Force Lighting update
F9	Toggle texturing on/off
F10	Toggle menu on/off

The autopilot is controlled via the following keys:

Tab. 5: *Autopilot and related controls.*

Key	Action
Ctrl + A	Altitude hold toggle on/off
Ctrl + G	Follow glide slope 1 toggle on/off
Ctrl + H	Heading hold toggle on/off
Ctrl + N	Follow NAV 1 radial toggle on/off
Ctrl + S	Autothrottle toggle on/off
Ctrl + T	Terrain follow toggle on/off
Ctrl + U	Add 1000 ft. to your altitude (emergency)
Enter	Increase autopilot heading
F6	Toggle autopilot target: current heading/waypoint
F11	Autopilot altitude dialog
F12	Autopilot heading dialog

Ctrl + T is especially interesting as it makes your little Cessna behave like a cruise missile. Ctrl + U might be handy in case you feel you're just about to crash.

(Shouldn't real planes sport such a key, too?)

In case the autopilot is enabled, some of the numeric keypad keys get a special meaning:

Tab. 6: *Special action of keys, if autopilot is enabled.*

Key	Action
8 / 2	Altitude adjust
0 / ,	Heading adjust
9 / 3	Autothrottle adjust

There are several keys for starting and controlling the engine :

Tab. 7: *Engine control keys*

Key	Action
SPACE	Fire starter on selected engine(s)
!	Select 1st engine
@	Select 2nd engine
#	Select 3rd engine
$	Select 4th engine
{	Decrease Magneto on Selected Engine
}	Increase Magneto on Selected Engine
~	Select all Engines

Beside these basic keys there are miscellaneous keys for special actions; some of these you'll probably not want to try during your first flight:

Tab. 8: *Miscellaneous keyboard controls.*

Key	Action
B	Toggle parking brake on/off
b	Apply/release all brakes
g	Toggle landing gear down
,	Left gear brake (useful for differential braking)
.	Right gear brake (useful for differential braking)
l	Toggle tail-wheel lock)
]/[	Extend/Retract flaps
p	Toggle pause on/off
a/A	Speed up/slow down (time acceleration)
t/T	Time speed up/slow down
m/M	Change time offset (warp) used by t/T forward/backward
Shift-F2	Save current flight to `fgfs.sav`
Shift-F1	Restore flight from `fgfs.sav`
F3	Save screen shot under `fgfs-screen.ppm`
Shift-F4	Re-read global preferences from `preferences.xml`
Shift-F10	Toggle data logging of FDM on/off
ESC	Exit program

Note: If you have difficulty processing the screenshot `fgfs-screen.ppm` on a windows machine, just recall that simply pressing the ”Print” key copies the screen to the clipboard, from which you can paste it into any graphics program.

Finally: Starting from ***FlightGear*** 0.7.7 these key bindings are no longer hard coded, but user-adjustable. You can check and change these setting via the file `keyboard.xml` to be found in the main ***FlightGear*** directory. This is a human readable plain ASCII file. Although it's perhaps not the best idea for beginners to start just with modifying this file, more advanced users will find it useful to change key bindings according to what they like (or, perhaps, know from other simulators).

4.3 Menu entries

By default, the menu is disabled after starting the simulator (you don't see a menu in a real plane, do you?). You can turn it on either using the toggle F10 or just by moving the mouse pointer to the top left corner of the display. In casse you want the menu to disappear just hit F10 again or move the mouse to the bottom of the screen.

At present, the menu provides the following functions.

- **File**
  - **Exit** Exits the program.

  - **Print** Exits the program.
  - **Snap Shot** Saves a normal resolution Screen Shot under `fgfs-screen-XXX.ppm`.
  - **Hires Snap Shot** Saves a high resolution Screen Shot under `fgfs-screen-XXX.ppm`.
  - **Reset** Resets you to the selected starting position. Comes handy in case you got lost or something went wrong.
  - **Save flight** Saves the current flight, by default to `fgfs-screen-XXX.ppm`.
  - **Load flight** Loads the current flight, by default from `fgfs-screen-XXX.ppm`.

- **View**

  - **Properties** Provies access to numerous properies managed via ***FlightGear***'s property manager. This is actually a quite powerful tool allowing to set all the values in the property tree. Obviously, this is a good place to crash the program by entering a ”bad” value.
  - **HUD Alpha** Toggles antialiasing of HUD lines on/off.
  - **Pilot Offset** Allows setting a different viewpoint (useful for R/C flying).
  - **Toggle Panel** Toggles instrument panel on/off.

- **Environment**

  - **Goto Airport** Enter the airport ID. For details on how to get the IDs see Section 3.4.6.

- **Autopilot**

  - **Toggle HUD format** Toggles figures of latitude/longitude in HUD.
  - **Adjust AP Settings** Allows input of several autopilot parameters.
  - **Clear Route** Clears current route.
  - **Skip Current Waypoint** Self explaining.
  - **Add Waypoint** Adds waypoint to waypoint list.
  - **Set Altitude** Sets altitude manually.
  - **Set Heading** Sets heading manually.

- **Network** (supposes compile option `--with-network-olk`)

  - **Toggle Display** Toggle call sign etc. on/off.

  - **Enter Callsign** Enter your call sign.
  - **Scan for Daemons** Scan for daemons on the net.
  - **Register for FGD** Register for ***FlightGear*** Daemon.
  - **Unregister for FGD** Unregister from ***FlightGear*** Daemon.

- **Help**

  - **Help** Opens your browser and displays and overview on several help options (including links to this Guide as well as to the FAQ).

4.4 The Instrument Panel

The Cessna instrument panel is activated by default when you start ***FlightGear***, but can be de-activated by pressing the ”P” key. While a complete description of all the functions of the instrument panel of a Cessna is beyond the scope of this guide, we will at least try to outline the main flight instruments or gauges.

All panel levers and knobs can be operated with the mouse To change a control, just click with the left/middle mouse button on the corresponding knob/lever.

Fig. 5: *The panel.*

Let us start with the most important instruments any simulator pilot must know. In the center of the instrument panel (Fig. 5), in the upper row, you will find the artificial horizon (attitude indicator) displaying pitch and bank of your plane. It has pitch marks 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 provide a speed indication in knots 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 fully retracted. The white arc indicates the range of speed with flaps in action. The 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 you must never exceeded - at least as long as you wan't brake your plane.

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 would indicate a standard turn, i.e. a turn of 360 degrees in exactly two minutes.

Below the plane, still in the turn indicator, is the 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. A simple rule says: "Step onto the ball", i.e. step onto the left rudder pedal in case the ball is on the l.h.s.

If you don't have pedals or lack the experience to handle the proper ratio between aileron/rudder automatically, you can start ***FlightGear*** with the option `--enable-auto-coordination`.

To the r.h.s of the artificial horizon you will find the altimeter showing the height above sea level (not ground!) in hundreds of feet. Below the altimeter is the vertical speed indicator indicating the rate of climbing or sinking of your plane in hundreds of feet per minute. While you may find it more convenient to use then the altimeter in cases, keep in mind that its diplay usually has a certain lag in time.

Further below the vertical speed indicator is the RPM (rotations per minute) indicator, which displays the rotations per minute in 100 RPMs. The green arc marks the optimum region for long-time flight.

The group of the main instruments further includes the gyro compass being situated below the artificial horizon. Besides this one, there is a magnetic compass sitting on top of the panel.

Four of these gauges being arranged in the from of a "T" are of special importance: The air speed indicator, the artificial horizon, the altimeter, and the compass should be scanned regularly during flight.

Besides these, there are several supplementary instruments. To the very left you will find the clock, obviously being an important tool for instance for determining turn rates.Below the clock there are several smaller gauges displaying the technical state of your engine. Certainly the most important of them is the fuel indicator - as any pilot should know.

The ignition switch is situated in the lower left corner of the panel (cf. Fig. 4). It has five positions: ”OFF”, ”L”, ”R”, ”BOTH”, and ”START”. The first one is obvious. ”L” and ”R” do not refer to two engines (actually the Cessna does only have one) but to two magnetos being present for safety purposes. The two switch positions can be used for test puposes during preflight. During normal flight the switch should point on ”BOTH”. The extreme right position is for using a battery-powered starter (to be operated with the SPACE key in flight gear).

Like in most flight simulators, you actually get a bit more than in a real plane. The red field directly below the gyro compass displays the state of the brakes, i.e., it is lit in case of the brakes being engaged. The instruments below indicate the position of youryoke. This serves as kind of a compensation for the missing forces you feel while pushing a real yoke. Three of the arrows correspond to the three axes of your yoke/pedal controlling nose up/down, bank left/right, rudder left/right, and throttle. (Keep in mind: They do **not** reflect the actual position of the plane!) The left vertical arrow indicates elevator trim.

The right hand side of the panel is occupied by the radio stack. Here you find two VOR receivers (NAV), an NDB receiver (ADF) and two communication radios (COMM1/2) as well as the autopilot.

The communication radio is used for communication with air traffic facilities; it is just a usual radio transceiver working in a special frequency range. The frequency is displayed in the ”COMM” field. Usually there are two COM transceivers; this way you can dial in the frequency of the next controller to contact while still being in contact with the previous one.

The COM radio can be used to display ATIS messages as well. For this purpose, just to dial in the ATIS frequency of the relevant airport.

The VOR (Very High Frequency Omni-Directional Range) receiver is used for course guidance during flight. The frequency of the sender is displayed in the ”NAV” field. In a sense, a VOR acts similarly to a light house permitting to display the position of the aircraft on a radial around the sender. It transmits one omni-directional ray of radio waves plus a second ray, the phase of which differs from the first one depending on its direction (which may be envisaged as kind of a ”rotating” signal). The phase difference between the two signals allows evaluating the angle of the aircraft on a 360 degrees circle around the VOR sender, the so-called radial. This radial is then displayed on the gauges NAV1 and NAV2, resp., left to frequency field. This way it should be clear that the VOR dispaly, while indicating the position of the aircraft relative to the VOR sender, does not say anything about the orientation of the plane.

Below the two COM/NAV devices is an NDB receiver called ADF (automatic direction finder). Again there is a field displaying the frequency of the facility. The ADF can be used for navigation, too, but contrary to the VOR does not show the position of the plane in a radial relative to the sender but the direct heading from the aircraft to the sender. This is displayed on the gauge below the two NAV gauges.

Above the COMM1 display you will see three LEDs in the colors blue, amber, and white indicating the outer, middle, and, inner, resp. marker beakon. These show the distance to the runway threshold during landing. They to not require the input of a frequency.

Below the radios you will find the autopilot. It has five keys for WL = ”Wing-Leveler”, ”HDG” = ”Heading”, NAV, APR = ”Glide-Slope”, and ALT = ”Altitude”. These keys when engaged hold the corresponding property.

A detailed description of the workings of these instruments and their use for navigation lies beyond this Guide; if you are interested in this exciting topic, we suggest consulting a book on instrument flight (simulation). Besides, this would be material for a yet to be written ***FlightGear*** Flight School.

It should be noted, that you can neglect these radio instruments as long as you are strictly flying according to VFR (visual flight rules). For those wanting to do IFR (instrument flight rules) flights, it should be mentioned that ***FlightGear*** includes a huge database of navaids worldwide.

Finally, you find the throttle, mixture, and flap control in the lower right of the panel (recall, flaps can be set via [and] or just using the mouse).

As with the keyboard, the panel can be re-configured using configuration files. As these have to be plane specific, they can be found under the directory of the corresponding plane. As an example, the configuration file for the default Cessna C172 can be found at `FlightGear/Aircraft/c172/Panels` as c172-panel.xml. The accompanying documentation for customizing it (i.e. shifting, replacing etc. gauges and more) is contained in the file `README.xmlpanel` written by John Check, to be found in the source code in the directory `docs-mini`.

Since version 0.8.0, ***FlightGear*** has a 3D cockpit including a 3D cockpit as an alternative to the 2D panel mentioned above (see Fig. 6). This one can be activated using the option `--aircraft=c172-3d`. Its functionality is the same as that of the 2D panel mentioned above, but it gives a much more realistic view, while instruments may be better readable in the 2D cockpit.

Fig. 6: *The 3D cockpit of the Cessna 172.*

4.5 The Head Up Display

At current, there are two options for reading off the main flight parameters of the plane: One is the instrument panel already mentioned, while the other one is the HUD (**H**ead **U**p **D**isplay) . Neither are HUDs used in usual general aviation planes nor in civilian ones. Rather they belong to the equipment of modern military jets. However, some might find it easier to fly using the HUD even with general aviation aircraft. Several Cessna pilots might actually love to have one, but technology is simply too expensive for implementing HUDs in general aviation aircraft. Besides, the HUD displays several useful figures characterizing simulator performance, not to be read off from the panel.

The HUD shown in Fig. 7 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. The Cessna 172 takes off at around 55 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 delivers some additions information. On the upper left you will find date and time. Besides, latitude and longitude, resp., of your current position are shown on top.

You can change color of the **HUD** using the ”H” or ”h” key. Pressing ethe toggle ”i/I” minimizes/maximizes the HUD.

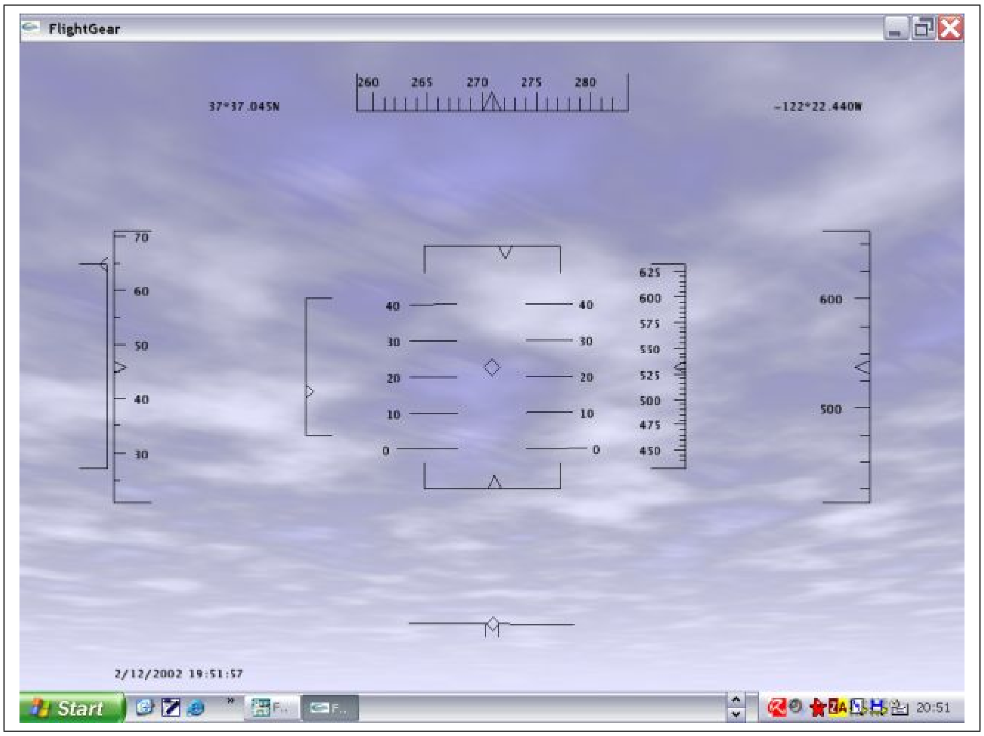

Fig. 7: *The HUD, or Head Up Display.*

4.6 Mouse controlled actions

Besides just clicking the menues, your mouse has got certain valuable functions in ***FlightGear***.

There are three mouse modi. In the normal mode (pointer curser) panel’s controls can be operated with the mouse. To change a control, click with the left/middle mouse button on the corresponding knob/lever. While the left mouse button leads to small increments/decrements, the middle one makes greater ones.

Clicking on the left hand side of the knob/lever decreases the value, while clicking on the right hand side increases it.

Right clicking the mouse activates the simulator control mode (cross hair cursor). This allows control of aileron/elevator via the mouse in absence of a joystick/yoke (enable `--enable-auto-coordination` in this case). If you have a joystick you certainly will not make use of this mode

Right clicking the mouse another time activates the view control mode (arrow cursor). This allows changing direction of view, i.e. pan and tilt the view, via the mouse.

Right clicking the mouse once more resets it into the initial state.

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/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. It is not suggested to simply "kill" the simulator by clicking the text window.

4.7 Some further reading for pilot students

In view of that fact, that there is not yet a ***FlightGear*** specific flight course, here are some useful hints to texts for those who want to learn piloting a plane.

First, a quite comprehensive manual is the Aeronautical Information Manual, published by the FAA, and being online available at

http://www.faa.gov/ATPubs/AIM/.

This is the Official Guide to Basic Flight Information and ATC Procedures by the FAA. It contains a lot of information on flight rules, flight safety, navigation, and more. If you find this a bit too hard reading, you may prefer the FAA Training Book,

http://avstop.com/AC/FlightTraingHandbook/,

which covers all aspects of flight, beginning with the theory of flight and the working of airplanes, via procedures like takeoff and landing up to emergency situations. This is an ideal reading for those who want to learn some basics on flight but don't (yet) want to spend bucks on getting a costly paper pilot's handbook.

While the handbook mentioned above is an excellent introduction on VFR (visual fligtht rules), it does not include flying according to IFR (instrument flight rules). However, an excellent introduction into navigation and flight according to Instrument Flight Rules written by Charles Wood can be found at

http://www.navfltsm.addr.com/.

Another comprehensive but yet readable text is John Denker's ”See how it flies”, available at

http://www.monmouth.com/ jsd/how/htm/title.html.

This is a real online text book, beginning with Bernoulli's principle, drag and power, and the like, with the later chapters covering even advanced aspects of VFR as well as IFR flying

Part III

Appendices

Appendix A

Missed approach: If anything refuses to work

In the following section, we tried to sort some problems according to operating system, but if you encounter a problem, it may be a wise idea to look beyond ”your” operating system – just in case. If you are experiencing problems, we would strongly advise you to first check the FAQ maintained by Cameron Moore at

http://www.flightgear.org/Docs/FlightGear-FAQ.html.

Moreover, the source code contains a directory `docs-mini` containing numerous ideas on and solutions to special problems. This is also a good place to go for further reading.

A.1 FlightGear Problem Reports

The best place to look for help is generally the mailing lists, specifically the **[Flightgear-User]** mailing list. If you happen to be running a CVS version of ***FlightGear***, you may want to subscribe to the **[Flightgear-Devel]** list. Instructions for subscription can be found at

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

It's often the case that someone has already dealt with the issue you're dealing with, so it may be worth your time to search the mailing list archives at

http://www.mail-archive.com/flightgear-users%40flightgear.org/
http://www.mail-archive.com/flightgear-devel%40flightgear.org/.

There are numerous developers and users reading the lists, so questions are generally answered. However, messages of the type

FlightGear does not compile on my system. What shall I do?
are hard to answer without any further detail given, aren't they? Here are some things to consider including in your message when you report a problem:

- **Operating system:** (Linux Redhat 7.0.../Windows 98SE...)
- **Computer:** (Pentium III, 1GHz...)
- **Graphics board/chip:** (Diamond Viper 770/NVIDIA RIVA TNT2...)
- **Compiler/version:** (Cygnus version 1.0...)
- **Versions of relevant libraries:** (PLIB 1.2.0, Mesa 3.0...)
- **Type of problem:** (Linker dies with message...)
- **Steps to recreate the problem:** Start at KSFO, turn off brakes ...

One final remark: Please avoid posting binaries to these lists! List subscribers are widely distributed, and some users have low bandwidth and/or metered connections. Large messsages may be rejected by the mailing list administrator. Thanks.

A.2 General problems

- ***FlightGear*** runs SOOO slow.
  If ***FlightGear*** says it's running with 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 Appendix B.

- Either `configure` or `make` dies with not found ***PLIB*** headers or libraries. Make sure you have the latest version of ***PLIB*** ($>$ version 1.2) compiled and installed. Its headers like `pu.h` have to be under `/usr/include/plib` and its libraries, like `libplibpu.a` should be under `/lib`. Double check there are no stray ***PLIB*** headers/libraries sitting elsewhere!

  Besides check careful the error messages of `configure`. In several cases it says what is missing.

A.3 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.)

- 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.

  You should also be sure to *always keep the* latest version of ***PLIB*** on your system. Lots of people have failed miserably to compile ***FlightGear*** just because of an outdated plib.

- Missing permissions
  In case you are using XFree86 before release 4.0 the ***FlightGear*** binary may need to be setuid root in order to be capable of accessing some accelerator boards (or a special kernel module as described earlier in this document) based on 3DFX chips. So you can either issue a

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

  to give the ***FlightGear*** binary the proper rights or install the 3DFX module. The latter is the "clean" solution and strongly recommended!

- Non-default install options
  FlightGear will display a lot of diagnostics while starting up. If it complains about bad looking or missing files, check that you installed them in the way they are supposed to be installed (i.e. with the latest version and in the 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 in general
  Make sure you have the latest (official) version of gcc. Old versions of gcc are a frequent source of trouble! On the other hand, some versions of the RedHat 7.0 reportedly have certain problems compiling ***FlightGear*** as they include a preliminary version of GCC.

- Problems with linking
  There may be several reasons; however in case you get a message like

  `libmk4.so.0`: cannot open shared object file

  the reason is a missing library package called Metakit. This is provided with Simgear in packed form. Unpack and install it first.

A.4 Potential problems under Windows

- 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 within a MS-DOS shell. Double-clicking via Explorer does never work (unless you set the environment variable `FG_ROOT` in `autoexec.bat` or otherwise). Rather double-click `runfgfs.bat`. For more details, check Chapter 3.

  Another cause of grief might be that you did not download the most recent versions of the base package files required by ***FlightGear***, or you did not download any of them at all. Have a close look at this, as the scenery/texture format is still under development and may change frequently. For more details, check Chapter 2.

  Next, if you run into trouble at runtime, do not use windows utilities for unpacking the `.tar.gz`. If you did, try it in the Cygnus shell with `tar xvfz` instead.

- ***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`.

- I am unable to build ***FlightGear*** under MSVC/MS DevStudio.
  By default, ***FlightGear*** is build with GNU GCC. The Win32 port of GNU GCC is known as Cygwin. For hints on Makefiles required for MSVC for MSC DevStudio have a look into

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

  In principle, it should be possible to compile ***FlightGear*** with the project files provided with the source code.

- Compilation of ***FlightGear*** dies.
  There may be several reasons for this, including true bugs. However, before trying to do anything else or report a problem, make sure you have the latest

version of the ***Cygwin*** compiler, as described in Section **??**. In case of doubt, start `setup.exe` anew and download and install the most recent versions of bundles as they possibly may have changed.

Appendix B

Some words on OpenGL graphics drivers

FlightGear's graphics engine is based on a graphics library called OpenGL. Its primary advantage is its 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 have some sort of OpenGL support installed for your video card.

A good review on OpenGL drivers can be found at

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

Specific information is collected for windows at

http://www.x-plane.com/SYSREQ/v5ibm.html

and for Macintosh at

http://www.x-plane.com/SYSREQ/v5mac.html.

An excellent place to look for documentation about Linux and 3-D accelerators is the *Linux Quake HOWTO* at

http://www.linuxquake.com.

This should be your first aid in case something goes wrong with your Linux 3-D setup.

Unfortunately, there are so many graphics boards, chips and drivers out there that we are unable to provide a complete description for all systems. Given the present market dominance of NVIDIA combined with the fact that their chips have indeed been proven powerful for running ***FlightGear***, we will concentrate on NVIDIA drivers in what follows.

B.1 NVIDIA chip based cards under Linux

Recent Linux distributions include and install anything needed to run OpenGL programs under Linux. Usually there is no need to install anything else.

If for whatever reason this does not work, you may try to download the most recent drivers from the NVIDIA site at

http://www.nvidia.com/Products/Drivers.nsf/Linux.html

At present, this page has drivers for all NVIDIA chips for the following Linux distributions: RedHat 7.1, Redhat 7.0, Redhat 6.2, Redhat 6.1, Mandrake 7.1, Mandrake 7.2, SuSE 7.1, SuSE 7.0 in several formats (.rpm, .tar.gz). These drivers support OpenGL natively and do not need any additional stuff.

The page named above contains a detailed `README and Installation Guide` giving a step-by-step description, making it unnecessary to copy the material here.

B.2 NVIDIA chip based cards under Windows

Again, you may first try the drivers coming with your graphics card. Usually they should include OpenGL support. If for whatever reason the maker of your board did not include this feature into the driver, you should install the Detonator reference drivers made by NVIDIA (which might be a good idea anyway). These are available in three different versions (Windows 95/98/ME, Windows 2000, Windows NT) from

http://www.nvidia.com/products.nsf/htmlmedia/detonator3.html

Just read carefully the Release notes to be found on that page. Notably do not forget to uninstall your present driver and install a standard VGA graphics adapter before switching to the new NVIDIA drivers first.

B.3 3DFX chip based cards under Windows

With the Glide drivers no longer provided by 3DFX there seems to be little chance to get it running (except to find older OpenGL drivers somewhere on the net or privately). All pages which formerly provided official support or instructions for 3DFX are gone now. For an alternative, you may want to check the next section, though.

B.4 An alternative approach for Windows users

There is now an attempt to build a program which detects the graphics chip on your board and automatically installs the appropriate OpenGL drivers. This is called OpenGL Setup and is presently in beta stage. It's home page can be found at

http://www.glsetup.com/.

We did not try this ourselfes, but would suggest it for those completely lost.

B.5 3DFX chip based cards under Linux

Notably, with 3DFX now having been taken over by NVIDIA, manufacturer's support already has disappeared. However with XFree86-4.x (with x at least being greater than 1) Voodoo3 cards are known to be pretty usable in 16 bit colour mode. Newer cards should work fine as well. If you are still running a version of Xfree86 3.X and run into problems, consider an upgrade. The recent distributions by Debian or SuSE have been reported to work well.

B.6 ATI chip based cards under Linux

There is excellent support for ATI chips in XFree86-4.1 and greater. Lots of AGP boards based on the Rage128 chip - from simple Rage128 board to ATI Xpert2000 - are pretty usuable for FlightGear. Since XFree86-4.1 you can use early Radeon chips - up to Radeon7500 with XFree86-4.2.

B.7 Building your own OpenGL support under Linux

Setting up proper OpenGL support with a recent Linux distribution should be pretty simple. As an example SuSE ships everything you need plus some small shell scripts to adjust the missing bits automagically. If you just want to execute prebuilt binaries of FlightGear, then you're done by using the supplied FlightGear package plus the mandantory runtime libraries (and kernel modules). The package manager will tell you which ones to choose.

In case you want to run a selfmade kernel, you want to compile FlightGear yourself, you're tweaking your X server configuration file yourself or you even run a homebrewn Linux ”distribution” (this means, you want to compile everything yourself), this chapter might be useful for you.

Now let's have a look at the parts that build OpenGL support on Linux. First there's a Linux kernel with support for your graphics adapter.

Examples on which graphics hardware is supported natively by Open Source drivers are provided on

http://dri.sourceforge.net/status.phtml.

There are a few graphics chip families that are not directly or no more than partly supported by XFree86, the X window implementation on Linux, because vendors don't like to provide programming information on their chips. In these cases - notably IBM/DIAMOND/now: ATI FireGL graphics boards and NVIDIA GeForce based cards - you depend on the manufacturers will to follow the ongoing development of the XFree86 graphics display infrastructure. These boards might prove to deliver impressing performance but in many cases - considering the CPU's speed you find in today's PC's - you have many choices which all lead to respectable performance of FlightGear.

As long as you use a distribution provided kernel, you can expect to find all necessary kernel modules at the approriate location. If you compile the kernel yourself, then you have to take care of two submenues in the kernel configuration menue. You'll find them in the "Character devices" menue. Please notice that AGP support is not compulsory for hardware accelerated OpenGL support on Linux. This also works quite fine with some PCI cards (3dfx Voodoo3 PCI for example, in case you still own one). Although every modern PC graphics card utilizes the AGP 'bus' for fast data transfer.

Besides "AGP Support" for your chipset - you might want to ask your mainboard manual which one is on - you defnitely want to activate "Direct Rendering Manager" for your garphics board. Please note that recent releases of XFree86 - namely 4.1.0 and higher might not be supported by the DRI included in older Linux kernels. Also newer 2.4.x kernels from 2.4.8 up to 2.4.17 do not support DRI in XFree86-4.0.x.

After building and installing your kernel modules and the kernel itself this task might be completed by loading the 'agpgart' module manually or, in case you linked it into the kernel, by a reboot in purpose to get the new kernel up and running. While booting your kernel on an AGP capable mainboard you may expect boot messages like this one:

```
> Linux agpgart interface v0.99 (c) Jeff Hartmann
> gpgart:  Maximum main memory to use for agp memory:  439M
> agpgart:  Detected Via Apollo Pro chipset
> agpgart:  AGP aperture is 64M @ 0xe4000000
```

If you don't encounter such messages on Linux kernel boot, then you might have missed the right chipset. Part one of activation hardware accelerated OpenGL support on your Linux system is now completed.

The second part consists of configuring your X server for OpenGL. This is not a big deal as it simply consists of to instructions to load the appropriate modules on startup of the X server. This is done by editing the configuration file `/etc/X11/XF86Config`. Today's Linux distributions are supposed to provide a tool that does this job for you on your demand. Please make shure there are these two instructions:

```
  Load ''glx''
  Load ''dri''
```

in the "Module" section your X server configuration file. If everything is right the X server will take care of loading the appropriate Linux kernel module for DRI support of your graphics card. The right Linux kernel module name is determined by the 'Driver' statement in the "Device" section of the XF86Config. Please see three samples on how such a "Device" section should look like:

```
Section ''Device''
  BoardName ''3dfx Voodoo3 PCI''
  BusID ''0:8:0''
  Driver ''tdfx''
  Identifier ''Device[0]''
  Screen 0
  VendorName ''3Dfx''
EndSection

Section ''Device''
  BoardName ''ATI Xpert2000 AGP''
  BusID ''1:0:0''
  Driver ''ati''
  Option ''AGPMode'' ''1''
  Identifier ''Device[0]''
  Screen 0
  VendorName ''ATI''
EndSection

Section ''Device''
  BoardName ''ATI Radeon 32 MB DDR AGP''
  BusID ''1:0:0''
  Driver ''radeon''
  Option ''AGPMode'' ''4''
  Identifier ''Device[0]''
  Screen 0
  VendorName ''ATI''
```

```
EndSection
```

By using the Option "AGPMode" you can tune AGP performance as long as the mainboard and the graphics card permit. The BusID on AGP systems should always be set to "1:0:0" - because you only have one AGP slot on your board - whereas the PCI BusID differs with the slot your graphics card has been applied to. 'lspci' might be your friend in desperate situations. Also a look at the end of /var/log/XFree86.0.log, which should be written on X server startup, should point to the PCI slot where your card resides.

This has been the second part of installing hardware accelerated OpenGL support on your Linux box.

The third part carries two subparts: First there are the OpenGL runtime libraries, sufficient to run existing appliactions. For compiling FlightGear you also need the suiting develoment headers. As compiling the whole X window system is not subject to this abstract we expect that your distribution ships the necessary libraries and headers. In case you told your package manager to install some sort of OpenGL support you are supposed to find some OpenGL test utilities, at least there should be 'glxinfo' or 'gl-info'.

These commandline utilities are useful to say if the previous steps where successfull. If they refuse to start, then your package manager missed something because he should have known that these utilities usually depend on the existence of OpenGL runtime libraries. If they start, then you're one step ahead. Now watch the output of this tool and and have a look at the line that starts with

OpenGL renderer string:

If you find something like

```
OpenGL renderer string:  FireGL2 / FireGL3 (Pentium3)
```

or

```
OpenGL renderer string:  Mesa DRI Voodoo3 20000224
```

or

```
OpenGL renderer string:  Mesa DRI Radeon 20010402
AGP 4x x86
```

```
OpenGL renderer string:  Mesa GLX Indirect
```

mind the word 'Indirect', then it's you who missed something, because OpenGL gets dealt with in a software library running solely on your CPU. In this case you might want to have a closer look at the preceding paragraphs of this chapter. Now please make shure all necessary libraries are at their proper location. You will need three OpenGL libraries for running FlightGear. In most cases you will find them in /usr/lib/:

```
/usr/lib/libGL.so.1
/usr/lib/libGLU.so.1
/usr/lib/libglut.so.3
```

These may be the libraries itself or symlinks to appropriate libraries located in some other directories. Depending on the distribution you use these libraries might be shipped in different packages. SuSE for example ships libGL in package 'xf86_glx', libGLU in 'xf86glu' and libglut in 'mesaglut'. Additionally for FlightGear you need libplib which is part of the 'plib' package.

For compiling FlightGear yourself - as already mentioned - you need the appropriate header files which often reside in /usr/include/GL/. Two are necessary for libGL and they come in - no, not 'xf86glx-devel' (o.k., they do but they do not work correctly) but in 'mesa-devel':

```
/usr/include/GL/gl.h
/usr/include/GL/glx.h
```

One comes with libGLU in 'xf86glu-devel':

```
/usr/include/GL/glu.h
```

and one with libglut in 'mesaglut-devel'

```
/usr/include/GL/glut.h
```

The 'plib' package comes with some more libraries and headers that are too many to be mentioned here. If all this is present and you have a comfortable compiler environment, then you are ready to compile FlightGear and enjoy the result.

Further information on OpenGL issues of specific XFree86 releases is avaliable here:

http://www.xfree86.org/<RELEASE NUMBER>/DRI.html

Additional reading on DRI:

http://www.precisioninsight.com/piinsights.html

In case you are missing some 'spare parts':

http://dri.sourceforge.net/res.phtml

B.8 OpenGL on MacIntosh

OpenGL is pre-installed on Mac OS 9.x and later. You may find a newer version than the one installed for Mac OS 9.x at http://www.apple.com/opengl

You should receive the updates automatically for Mac OX 10.x.

One final word: We would recommend that you test your OpenGL support with one of the programs that accompany the drivers, to be absolutely confident that it is functioning well. There are also many little programs, often available as screen savers, that can be used for testing. It is important that you are confident in your graphics acceleration because ***FlightGear*** will try to run the card as fast as possible. If your drivers aren't working well, or are unstable, you will have difficulty tracking down the source of any problems and have a frustrating time.

Appendix C

Landing: Some further thoughts before leaving the plane

C.1 A not so Short History of *FlightGear*

The ***FlightGear*** project goes back to a discussion among a group of net citizens in 1996 resulting in a proposal written by David Murr who, unfortunately, dropped out of 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 have changed over time, the spirit of that proposal has clearly been retained up to the present time.

Actual coding started in the summer of 1996 and by the end of that year essential graphics routines were completed. At that time, programming was mainly performed and coordinated by Eric Korpela from Berkeley University. Early code ran under Linux as well as under DOS, OS/2, Windows 95/NT, and Sun-OS. This was found to be quite an ambitious project as it involved, among other things, writing all the graphics routines in a system-independent way entirely from scratch.

Development slowed and finally stopped in the beginning of 1997 when Eric was completing 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 who re-launched the project in the middle of 1997. His idea was as simple as it was powerful: 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 (developed by Bruce Jackson from NASA), seemed to be well suited to the approach.

Curt took this one apart and re-wrote several of the routines such as to make them build as well as run on the intended target platforms. The key idea in doing so was to exploit a system-independent graphics platform: OpenGL.

Fig. 8: *LaRCsim's Navion is still available in **FlightGear**.*

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

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

for the U.S., and

http://edcwww.cr.usgs.gov/landdaac/gtopo30/gtopo30.html,

resp., for other countries. Those freely accessible scenery data, in conjunction with scenery building tools included with ***FlightGear***, are an important feature enabling anyone to create his or her own scenery.

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

- The display of sun, moon and stars have been a weak point for PC flight simulators for a long time. It is one of the great achievements of ***FlightGear*** to include accurate modeling and display of sun, moon, and planets very early. The corresponding astronomy code was implemented in fall 1997 by Durk Talsma.

- Texture support was added by Curt Olson in spring 1998. This marked a significant improvement in terms of reality. You may recall that Microsoft Flight Simulator had non-textured scenery up until version 4.0. Some high-quality textures were submitted by Eric Mitchell for the ***FlightGear*** project.

- A HUD (head up display) was added based on code provided by Michele America and Charlie Hotchkiss in the fall of 1997 and was improved later by Norman Vine. While not generally available for real Cessna 172, the HUD conveniently reports the actual flight performance of the simulation and may be of further use in military jets later.

- After improving the scenery and texture support frame rate dropped down to a point where ***FlightGear*** became unflyable in spring 1998. This issue was resolved by exploiting hardware OpenGL support, which became available at that time, and implementing view frustrum culling (a rendering technique that ignores the part of the scenery not visible in a scene), done by Curt Olson. Taking these measures made ***FlightGear*** flyable again as long as they included a 3-D graphics board that featured hardware OpenGL support. With respect to frame rate one should keep in mind that the code, at present, is in no way optimized, which leaves room for further improvements.

- A rudimentary autopilot implementing heading hold was contributed by Jeff Goeke-Smith in April 1998. It was improved by the addition of an altitude hold and a terrain following switch in October 1998 and further developed by Norman Vine later.

- The basis for a menu system was laid based on another library, the Portable Library ***PLIB***, in June 1998. After having been idle for a time, the first working menu entries came to life in spring 1999.

  PLIB underwent rapid development later. It has been distributed as a separate package by Steve Baker with a much broader range of applications in mind, since spring 1999. It has provided the basic graphics rendering engine for ***FlightGear*** since fall 1999.

- Friedemann Reinhard developed early instrument panel code, which was added in June 1998. Unfortunately, development of that panel slowed down later, partly because of OpenGL compatibility problems. Finally, David Megginson decided to rebuild the panel code from scratch in January 2000. This led to a rapid addition of new instruments and features to the panel, resulting in nearly all main instruments being included until spring 2001. A handy minipanel was added in summer 2001.

- In 1998 there was basic audio support, i. e. an audio library and some basic background engine sound. This was later integrated into the above-mentioned portable library, ***PLIB***. This same library was extended to support joystick/yoke/rudder in October 1999, again marking a huge step in terms of realism. To adapt on different joystick, configuration options were introduced in fall 2000.

- In September 1998 Curt Olson succeeded in creating a complete terrain model for the U.S. The scenery is available worldwide via a clickable map at:

  http://www.flightgear.org/Downloads/world-scenery.html.

- Networking/multiplayer code has been integrated by Oliver Delise and Curt Olson starting fall 1999. This effort is aimed at enabling ***FlightGear*** to run concurrently on several machines over a network, either an Intranet or the Internet, coupling it to a flight planner running on a second machine, and more. There emerged several approaches for remotely controlling FlightGear over a Network during 2001. Notably there was added support working together wirth the ”Atlas” moving map program. Besides, an embedded HTTP server developed late in 2001 by Curt Olson can now act a property manager for external programs.

- Christian Mayer, together with Durk Talsma, contributed weather code in the winter of 1999. This included clouds, winds, and even thunderstorms.

- Manually changing views in a flight simulator is in a sense always ”unreal” but nonetheless required in certain situations. A possible solution was supplied by Norman Vine in the winter of 1999 by implementing code for changing views using the mouse. Alternatively, you can use a hat switch for this purpose, today.

- Finally, LaRCsims Navion was replaced as the default aircraft when the Cessna 172 was stable enough in February 2000 - a move most users will welcome. There are now several flight model options to choose from at runtime: a modified and improved LaRCsim Cessna 172 developed by Tony Peden, Jon Berndt’s X15, and Christian Mayer’s hot air balloon. Jon Berndt has invested a lot of time in a more realistic and versatile flight model with a more powerful aircraft configuration method. ***JSBSim***, as it has come to be called, may eventually replace LaRCsim as the default flight dynamics model (FDM), and it is planned to include such features as fuel slosh effects, turbulence, complete flight control systems, and other features not often found all together in a flight simulator. As an alternative, Andy Ross added another flight dynamics model called ***YASim*** (Yet Another Flight Dynamics Simulator) which aims at simpliciy of use, by the end of 2001. This one bought us flight modles for a 747, an A4, and a DC-3.

- The scenery was further improved by adding geographic features including lakes, rivers, and coastlines later, an effort still going on. Since the end of 2000, there was again stronger focus on scenery. Textured runways were added by Dave Cornish in spring 2001. Light textures add to the visual impression at night. To cope with the constant growth of scenery data, a binary scenery format was introduced in spring 2001.

- A fully operational radio stack and working radios were added to the panel by Curt Olson in spring 2000. A huge database of Navaids contributed by Robin Peel allows IFR navigation since then.

- A property manager was implemented by David Megginson in fall 2000. It allows parsing a file called `.fgfsrc` under UNIX/Linux and `system.fgfsrc` under Windows for input options. This plain ASCII file has proven useful in submitting the growing number of input options, and notably the joystick settings. This has proven a useful concept, and joystick, keyboard, and panel settings are no longer hard coded but set using *.xml files since spring 2001 thanks to work mainly by David Megginson and John Check.

- There was support added for static objects to the scenery in 2001, which permits placing buildung, static planes, trees and so on in the scenery. However, despite a few profs systematic includion of these landmarks is still missing.

- There was basic ATC support added in fall 2001 by David Luff. This is not yet fully implemented, but displaying ATIS messages is already possible.

- A magneto switch with proper functions was added at the end of 2001 by John Check and David Megginson.. Actually, several panels were vastly improved during 2001 by John and others.

During development there were several code reorganization efforts. Various code subsystems were moved into packages. As a result, presetnly code is organized as follows:

The base of the graphics engine is **OpenGL**, a platform independent graphics library. Based on OpenGL, the Portable Library ***PLIB*** provides basic rendering, audio, joystick etc. routines. Based on ***PLIB*** is ***SimGear***, which includes all of the basic routines required for the flight simulator as well as for building scenery. On top of ***SimGear*** there are (i) ***FlightGear*** (the simulator itself), and (ii) ***TerraGear***, which comprises the scenery building tools.

This is by no means an exhaustive history and most likely some people who have made important contributions have been left out. Besides the above-named contributions there was a lot of work done concerning the internal structure by: Jon S. Berndt, Oliver Delise, Christian Mayer, Curt Olson, Tony Peden, Gary R. Van Sickle, Norman Vine, and others. A more comprehensive list of contributors can be found in Chapter C as well as in the `Thanks` file provided with the code. Also, the ***FlightGear*** Website contains a detailed history worth reading of all of the notable development milestones at

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

C.2 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 (!) mail 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 at

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. Besides, there is a very low-traffic list for announcements.

The following names the people who did the job (this information was essentially taken from the file `Thanks` accompanying the code).

A1 Free Sounds (techie@mail.ev1.net)
Granted permission for the flightgear project to use some of the sound effects from their site. Homepage under

http://www.a1freesoundeffects.com/

Raul Alonzo (amil@las.es)
Mr. Alonzo is the author of Ssystem and provided his kind permission for using the moon texture. Parts of his code were used as a template when adding the texture. Ssystem Homepage can be found at:

http://www1.las.es/˜amil/ssystem/.

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

Michael Basler (pmb@epost.de)
Author of Installation and Getting Started. Flight Simulation Page at

http://www.geocities.com/pmb.geo/flusi.htm

Jon 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 simulate arbitrary aircraft. Jon maintains a page dealing with Flight Dynamics at:

http://jsbsim.sourceforge.net

Special attention to X15 is paid in separate pages on this site. Besides, Jon contributed via a lot of suggestions/corrections to this Guide.

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@kingmont.com)
Provided a big chunk of online space to store USA scenery for ***FlightGear***.

Bernie Bright (bbright@bigpond.net.au)
Many C++ style, usage, and implementation improvements, STL portability and much, much more. Added threading support and a threaded tile pager.

Bernhard H. Buckel (buckel@mail.uni-wuerzburg.de)
Contributed the README.Linux. Contributed several sections to earlier versions of Installation and Getting Started.

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

Ralph Carmichael (ralph@pdas.com)
Support of the project. The Public Domain Aeronautical Software web site at

http://www.pdas.com

has the PDAS CD-ROM for sale containing great programs for astronautical engineers.

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.

John Check (j4strngs@rockfish.net)
John maintains the base package CVS repository. He contributed cloud textures, wrote an excellent Joystick howto as well as a panel howto. Moreover, he contributed new instrument panel configurations. ***FlightGear*** page at

http://rockfish.net/fg/.

Dave Cornish (dmc@halcyon.com)
Dave created new cool runway textures.

Oliver Delise (delise@mail.isis.de)
Started a FAQ, Documentation, Public relations. Working on adding some networking/multi-user code. Founder of the FlightGear MultiPilot Project at

http://www.isis.de/members/˜odelise/progs/flightgear.

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.

Dave Eberly (eberly@magic-software.com)
Contributed some sphere interpolation code used by Christian Mayer's weather data base system. On Dave's web site there are tons of really useful looking code at

http://www.magic-software.com.

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.

Bruce Finney (bfinney@gte.net)
Contributed patches for MSVC5 compatibility.

Jean-loup Gailly and **Mark Adler** (zlib@gzip.org)
Authors of the zlib library. Used for on-the-fly compression and decompression routines,

http://www.cdrom.com/pub/infozip/zlib/.

Mohit Garg (theprotean_1@hotmail.com)
Contributed to the manual.

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

Neetha Girish (neethagirish@usa.net)
Contributed the changes for the xml configurable HUD.

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.

Habibe (habibie@MailandNews.com)
Made RedHat package building changes for SimGear.

Mike Hill (mikehill@flightsim.com)
For allowing us to concert and use his wonderful planes, available form

http://www.flightsimnetwork.com/mikehill/home.htm,

for ***FlightGear***.

Erik Hofman (erik.hofman@a1.nl)
Contributed SGI IRIX support and binaries.

Charlie Hotchkiss (clhotch@pacbell.net)
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://dcb.larc.nasa.gov/www/DCBStaff/ebj/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.

Ove Kaaven (ovek@arcticnet.no)
Contributed the Debian binary.

Richard Kaszeta (bofh@me.umn.edu)
Contributed screen buffer to ppm screen shot routine. Also helped in the early development of the ”altitude hold autopilot module” by teaching Curt Olson the basics of Control Theory and helping him code and debug early versions. Curt’s

”Boss” Bob Hain (bob@me.umn.edu) also contributed to that. Further details available at:

http://www.menet.umn.edu/ curt/fgfs/Docs/Autopilot/AltitudeHold/AltitudeHold.html.

Rich’s Homepage is at

http://www.menet.umn.edu/ kaszeta.

Tom Knienieder (tom@knienieder.com)
Ported the audio library first to OpenBSD and IRIX and after that 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@who.net)
Redid the Makefile system so it is simpler and more robust.

Kyler B Laird (laird@ecn.purdue.edu)
Contributed corrections to the manual.

David Luff (david.luff@nottingham.ac.uk)
Contributed heavily to the IO360 piston engine model.

Christian Mayer (flightgear@christianmayer.de)
Working on multi-lingual conversion tools for fgfs as a demonstration of technology. Contributed code to read Microsoft Flight Simulator scenery textures. Christian is working on a completely new weather subsystem. Donated a hot air balloon to the project.

David Megginson (david@megginson.com)
Contributed patches to allow mouse input to control view direction yoke. Contributed financially towards hard drive space for use by the flight gear project. Updates to README.running. Working on getting fgfs and ssg to work without textures. Also added the new 2-D panel and the save/load support. Further, he developed new panel code, playing better with OpenGL, with new features. Developed the property manager and contributed to joystick support.

Cameron Moore (cameron@unbeatenpath.net)
FAQ maintainer. Reigning list administrator. Provided man pages.

Eric Mitchell (mitchell@mars.ark.com)
Contributed some topnotch scenery textures being all original creations by him.

Alan Murta (amurta@cs.man.ac.uk)

http://www.cs.man.ac.uk/aig/staff/alan/software/

Created the Generic Polygon Clipping library.

Phil Nelson (phil@cs.wwu.edu)
Author of GNU dbm, a set of database routines that use extendible hashing and work similar to the standard UNIX dbm routines.

Alexei Novikov (anovikov@heron.itep.ru)
Created European Scenery. Contributed a script to turn fgfs scenery into beautifully rendered 2-D maps. Wrote a first draft of a Scenery Creation Howto.

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 subsystem as well as the graphics stuff. Homepage at
http://www.menet.umn.edu/ curt/

noindent **Brian Paul**
We made use of his TR library and of course of Mesa:
http://www.mesa3d.org/brianp/TR.html, http://www.mesa3d.org

Tony Peden (apeden@earthlink.net)
Contributions on flight model development, including a LaRCsim based Cessna 172. Contributed to ***JSBSim*** the initial conditions code, a more complete standard atmosphere model, and other bugfixes/additions. His Flight Dynamics page can be found at:

http://www.nwlink.com/˜apeden.

Robin Peel (robin@cpwd.com)
Maintains worldwide airport and runway database for ***FlightGear*** as well as X-Plane.

Alex Perry (alex.perry@ieee.org)
Contributed code to more accurately model VSI, DG, Alticude. Suggestions for improvements of the layout of the simulator on the mailing list and help on documentation.

Friedemann Reinhard (mpt218@faupt212.physik.uni-erlangen.de)
Development of an early 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". Also wrote a patch to support a first joystick with more than 2 axis.

Andy Ross (andy@plausible.org)
Contributed a new configurable FDM called YASim (Yet Another Fligth Dynamics Simulator, based on geometry information rather than aerodynamic coefficients.

Paul Schlyter (pausch@saaf.se)
Provided Durk Talsma with all the information he needed to write the astro code. Mr. Schlyter is also willing to answer astro-related questions whenever one needs to.

http://welcome.to/pausch

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

Phil Schubert (philip@zedley.com)
Contributed various textures and engine modelling.

http://www.zedley.com/Philip/index.htm.

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 Cessna.

Michael Smith (msmith99@flash.net)
Contributed cockpit graphics, 3-D models, logos, and other images. Project Bonanza

http://members.xoom.com/ConceptSim/index.html.

Durk Talsma (d.talsma@chello.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. Contributed 2-D cloud layer. Website at

http://people.a2000.nl/dtals.

UIUC - Department of Aeronautical and Astronautical Engineering
Contributed modifications to LaRCsim to allow loading of aircraft parameters from a file. These modifications were made as part of an icing research project.

Those did the coding and made it all work:
Jeff Scott jscott@students.uiuc.edu
Bipin Sehgal bsehgal@uiuc.edu
Michael Selig m-selig@uiuc.edu

Moreover, those helped to support the effort:
Jay Thomas jthomas2@uiuc.edu
Eunice Lee ey-lee@students.uiuc.edu
Elizabeth Rendon mdfhoyos@md.impsat.net.co
Sudhi Uppuluri suppulur@students.uiuc.edu

U. S. Geological Survey

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

Provided geographic data used by this project.

Mark Vallevand (Mark.Vallevand@UNISYS.com)
Contributed some METAR parsing code and some win32 screen printing routines.

Gary R. Van Sickle (tiberius@braemarinc.com)
Contributed some initial GameGLUT support and other fixes. Has done some interesting preliminary work on a binary file format. Check

http://www.woodsoup.org/projs/ORKiD/fgfs.htm.

Martin Spott (Martin.Spott@uni-duisburg.de)
Co-Author of the ”Getting Started”.

Norman Vine (nhv@yahoo.com)
Provided more numerous URL’s to the ”FlightGear Community”. Many performance optimizations throughout the code. Many contributions and much advice for the scenery generation section. Lots of Windows related contributions. Contributed wgs84 distance and course routines. Contributed a great circle route autopilot mode based on wgs84 routines. Many other GUI, HUD and autopilot contributions. Patch to allow mouse input to control view direction. Ultra hires tiled screen dumps. Contributed the initial ’goto airport’ and ’reset’ functions and the initial http image server code

Roland Voegtli (webmaster@sanw.unibe.ch)
Contributed great photorealistic textures. Founder of European Scenery Project for X-Plane:

http://www.g-point.com/xpcity/esp/

Carmelo Volpe (carmelo.volpe@mednut.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. Contributed to the Mac part of Getting Started, too.

Ed Williams (Ed_Williams@compuserve.com).
Contributed magnetic variation code (impliments Nima WMM 2000). We've also borrowed from Ed's wonderful aviation formulary at various times as well. Website at http://www.best.com/~williams/index.html,

Jean-Claude Wippler (jcw@equi4.com)
Author of MetaKit - a portable, embeddible database with a portable data file format used in ***FlightGear***. Please see the following URL for more info:

http://www.equi4.com/metakit

Woodsoup Project

http://www.woodsoup.org

While ***FlightGear*** no longer uses Woodsoup servies we appreciate the support provied to our project during the time they hosted us. Once they provided computing resources and services so that the ***FlightGear*** project could have a real home.

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.

C.3 What remains to be done

If you read (and, maybe, followed) this guide up to this point you may probably agree: ***FlightGear***, even in its present state, is not at all for the birds. It is already a flight simulator which sports even several selectable flight models, several planes with panels and even a HUD, terrain scenery, texturing, all the basic controls and weather.

Despite, ***FlightGear*** needs – and gets – further development. Except internal tweaks, there are several fields where ***FlightGear*** needs basics improvement and development. A first direction is adding airports, streets, and more of those things bringing scenery to real life and belonging to realistic airports. Another task is further implementation of the menu system, which should not be too hard with the basics being working now. A lot of options at present set via command line or even during compile time should finally make it into menu entries. Finally, ***FlightGear*** lacks any ATC until now.

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.

First, I was very glad to see Martin Spott entering the documentation effort. Martin provided not only several updates and contributions (notably in the OpenGL section) on the Linux side of the project but also several general ideas on the documentation in general

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

Next, Bernhard Buckel wrote several sections of early versions of that Guide and contributed at lot of ideas to it.

Jon S. Berndt supported me by critical proofreading of several versions of the document, pointing out inconsistences and suggesting improvements.

Moreover, I gained a lot of help and support from Norman Vine. Maybe, without Norman's answers I would have never been able to tame different versions of the ***Cygwin*** – ***FlightGear*** couple.

We were glad, our Mac expert Darrell Walisser contributed the section on compiling under Mac OS X. In addition he submitted several Mac related hints and fixes.

Further contributions and donations on special points came from John Check, (general layout), Oliver Delise (several suggestions including notes on that chapter), Mohit Garg (OpenGL), Kyler B. Laird (corrections), Alex Perry (OpenGL), and Kai Troester (compile problems).

Besides those whose names got lost withing the last-minute-trouble we'd like to express our gratitude to the following people for contributing valuable 'bug fixes' to this version of Getting Started (in random order): Cameron Moore, Melchior

Franz, David Megginson, Jon Berndt, Alex Perry,, Dave Perry,, Andy Ross, Erik Hofman.

Index