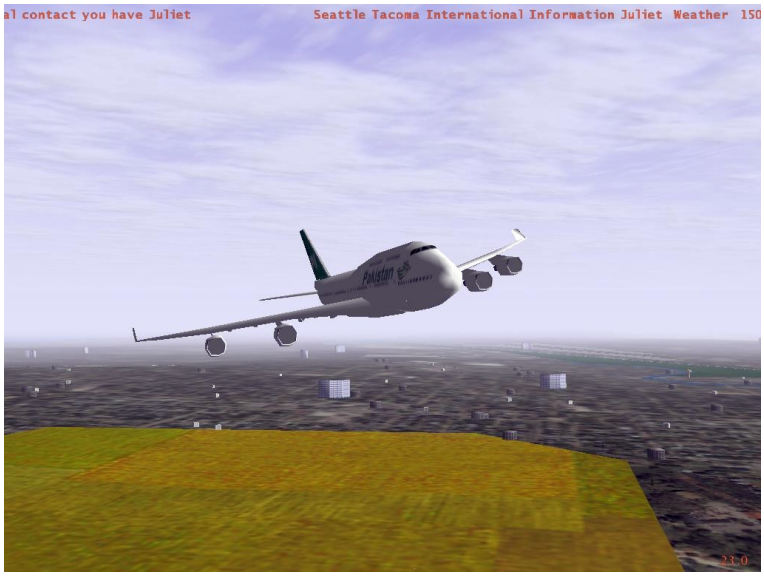
FlightGear Flight Simulator – Installation and Getting Started

Michael Basler and Martin Spott

including contributions by
Jon Berndt, Stuart Buchanan,
Bernhard Buckel, Cameron Moore,
Curt Olson, Dave Perry,
Michael Selig, Darrell Walisser,
and others



Getting Started Version 0.7
November 17, 2005
Manual was written for ***FlightGear*** version 0.9.3 with updates for version 0.9.9.

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 Guide" is meant to give beginners a guide in getting ***FlightGear*** up and running, and themselves into the air. It is not intended to provide complete documentation of all the features and add-ons of ***FlightGear*** but, instead, aims to give a new user the best start to exploring what ***FlightGear*** has to offer.

This guide is split into three parts and is structured as follows.

Part I: Installation

Chapter 1, *Want to have a free flight? Take **FlightGear***, introduces ***FlightGear***, provides background on the philosophy behind it and describes the system requirements.

In Chapter 2, *Preflight: Installing **FlightGear***, you will find instructions for installing the binaries and additional scenery and aircraft.

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.

Part III: Tutorials

Chapter 5, *Tutorials*, provides information on the many tutorials available for new pilots.

Chapter 6, *A Cross Country Flight Tutorial*, describes a simple cross-country flight in the San Fransisco area that can be run with the default installation.

Appendices

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

Appendix C, *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.

Appendix B, *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.

In the final Appendix D, *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:	B, 2
Installation under Macintosh:	2
Operation	
Program start (all users):	3
Keycodes, Panel, Mouse... (all users):	4
Troubleshooting	
General issues:	A
Graphics problems:	C
Optionally	1, D

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 FlightGear icon (on Windows), or textttfgfs having set the environmental variables described in Chapter 3 (on Linux).

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 pre-built aircraft, but there's just something special about building one yourself.

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.

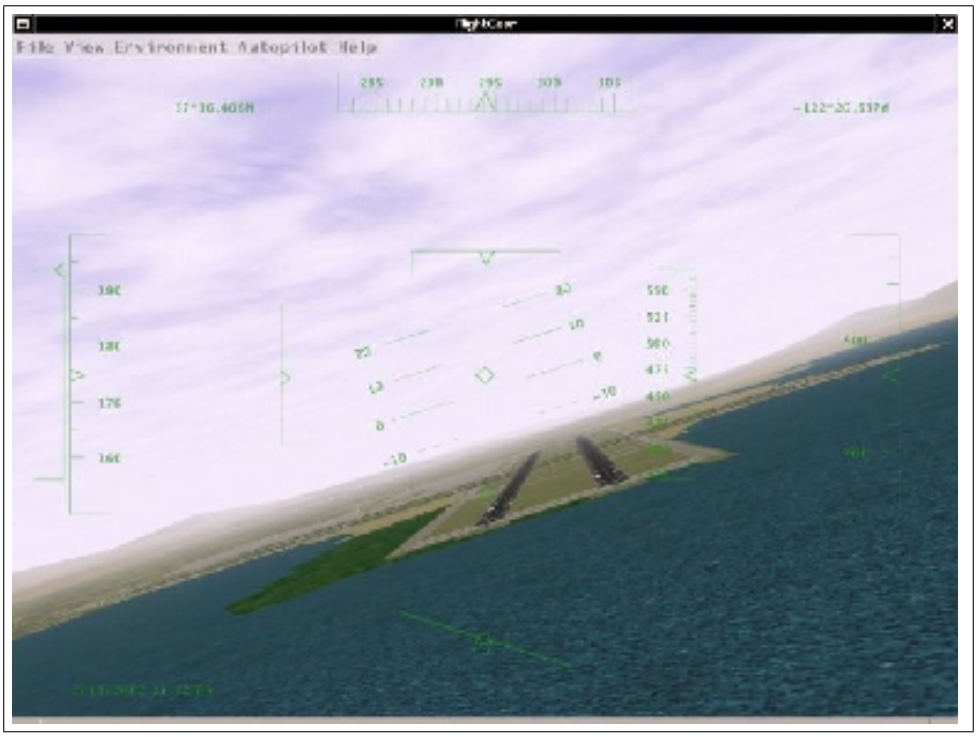


Fig. 1: ***FlightGear*** *under UNIX: Bad approach to San Francisco International - by one of the authors of this manual...*

1.2 System Requirements

In comparison to other recent flight simulators, the system requirements for ***FlightGear*** are not extravagant. A decent PIII/800, 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 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

Previously the ***FlightGear*** source code existed in 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. This policy has been obsoleted by practical reasons - mostly because stable releases got out-ouf-date and behind in features very fast and the so called development releases had been proven to be of comparable stability.

You are invited to fetch the "latest official release" which the pre-compiled binaries are based on. It is available from

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

If you really want to get the very latest and greatest (and, at times, buggiest) code, you can use a tool called anonymous cvs 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/.

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/.

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 flightmodels. 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 "UFO" slew mode, 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, developed by a team at the University of Illinois at Urbana-Champaign. This work was initially geared

toward modeling aircraft in icing conditions together with a smart icing system to better enable pilots to fly safely in an icing encounter. While this research continues, the project has expanded to include modeling "nonlinear" aerodynamics, which result in more realism in extreme attitudes, such as stall and high angle of attack flight. Two good examples that illustrate this capability are the Airwave Xtreme 150 hang glider and the 1903 Wright Flyer. For the hang glider, throttle can be use to fly to gliding altitude or Ctrl-U can be used to jump up in 1000-ft increments. Try your hand at the unstable Wright Flyer and don't stall the canard! Considerable up elevator trim will be required for level flight. In general, the aerodynamics are probably very close to the original Wright Flyer as they are partly based on experimental data taken on a replica tested recently at the NASA Ames Research Center. Also included are two more models, a Beech 99 and Marchetti S-211 jet trainer, which are older generation UIUC/FGFS models and based on simpler "linear" aerodynamics. More details of the UIUC flight model and a list of aircraft soon to be upgraded can be found on their website at

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

Note that the 3D models of the UIUC airplanes can be downloaded from a site maintained by Wolfram Kuss

http://home.t-online.de/home/Wolfram.Kuss/

It is even possible to drive FlightGear's scene display using an external FDM running on a different computer or via named pipe on the local machine – 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 are 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*

To run ***FlightGear*** you need to install the binaries. Once you've done this you may install additional scenery and aircraft if you wish.

Pre-compiled binaries for the latest release are available for

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

To download them go to

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

and follow the instructions provided on the page.

If you are running on another OS, or wish to compile for yourself, see Appendix B, *Building the plane: Compiling the program*.

2.1 Installing scenery

The ***FlightGear*** base package contains scenery for a small area around San Fransisco, but the entire world is available at a high level of detail, so you will almost certainly wish to install extra scenery at some point.

The scenery is based on SRTM elevation data (accurate to 30m in the USA, and 90m elsewhere and) VMap0 land use data. Additionally, various people have created buildings, bridges and other features to enrich the environment.

You can download scenery in 10 degree by 10 degree chunks from a clickable map at

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

Curt Olson also provides the USA or the entire world along with the latest FlightGear release on DVD from here http://cdrom.flightgear.org/

If you are interested in generating your own scenerey, have a look at http://www.terragear.org/, the tools that generate the scenery for ***FlightGear***.

Finally, an alternative data set was produced by William Riley and is available from http://www.randdtechnologies.com/fgfs/newScenery/world-scenery.html.

Whatever scenery you shoose to download, it should be kept in a separate directory from the scenery delivered with the binaries.

To do this, create a `WorldScenery` directory in the ***FlightGear*** data directory, usually

`c:\Program Files\FlightGear\data`

on Windows or

`/usr/local/share/FlightGear/data`

on *nix.

Underneath this directory create `Terrain` and `Objects` subdirectories. These are used for terrain information and buildings/bridges/structures respectively.

Unpack the downloaded scenery into the `WorldScenery/Terrain`. 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 into an installation located at

`/usr/local/share/FlightGear`.

Once your installation is complete, you'll have the following directories

```
/usr/local/FlightGear/data/WorldScenery/Objects/
/usr/local/FlightGear/data/WorldScenery/Terrain/w120n30/w112n30
/usr/local/FlightGear/data/WorldScenery/Terrain/w120n30/w112n31
...
/usr/local/FlightGear/data/WorldScenery/Terrain/w120n30/w120n39
```

As well as the scenery itself, objects such as bridges, skyscrapers, radio masts can be downloaded from http://fgfsdb.stockill.org. See the website for more information. You can exploit `FG_SCENERY` environmental variable or the `--fg-scenery=`*path* command line option if you want to install different scenery sets in parallel or want to have scenery sitting in another place. These are more fully described in Chapter 3.

2.2 Installing aircraft

The base ***FlightGear*** package contains only a small subset of the aircraft that are available for ***FlightGear***. Developers have created a wide range of aircraft, from WWII fighters like the Spitfire, to passenger planes like the Boeing 747.

You can download aircraft from

http://www.flightgear.org/Downloads/aircraft/index.shtml

Simply download the file and unpack it into the `FlightGear/data/Aircraft` subdirectory of your installation. Next time you run ***FlightGear***, the new aircraft will be available.

2.3 Installing documentation

Most of the packages named above include the complete ***FlightGear*** documentation including a ṗdf 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 ḣtml 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

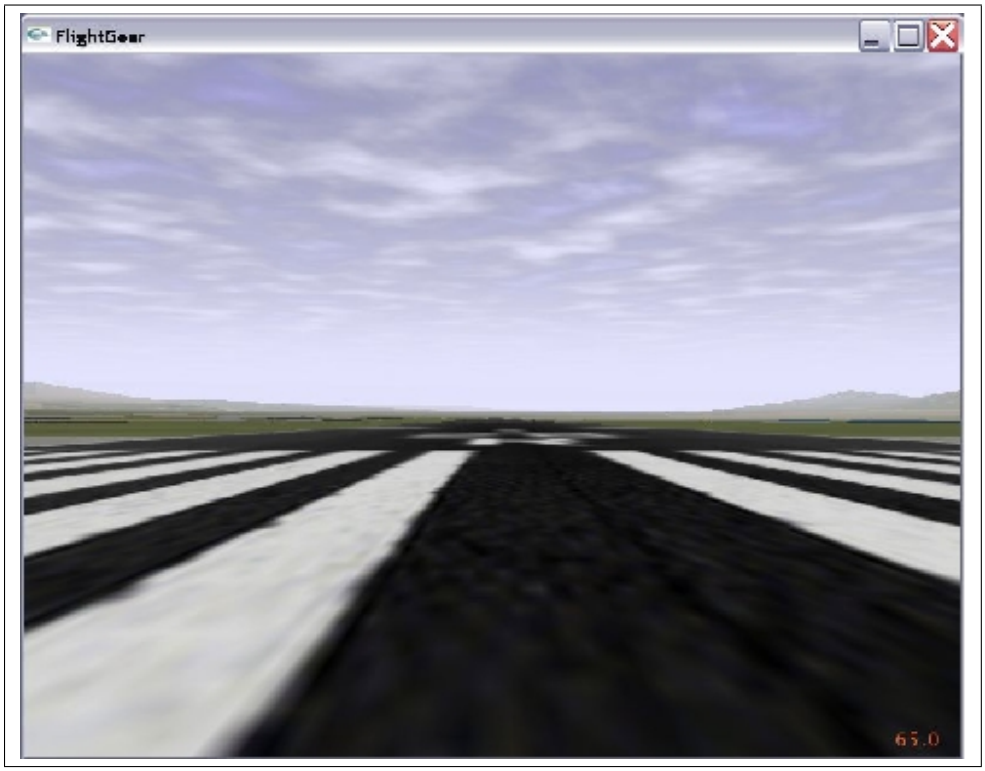


Fig. 3: *Ready for takeoff. Waiting at the default startup position at San Francisco Itl., KSFO.*

Before you can run ***FlightGear***, you need to have a couple of environmental variables set.

- You must add `/usr/local/share/FlightGear/lib` to your `LD_LIBRARY_PATH`
- `FG_HOME` must be set to the root of your ***FlightGear*** installation. e.g. `/usr/local/share/FlightGear`.
- `FG_ROOT` must be set to the date directory of your ***FlightGear*** installation. e.g. `/usr/local/share/FlightGear/data`.
- `FG_SCENERY` should be a list of scenery directories, separated by ":". This works like `PATH` when searching for scenery.

  e.g. `$FG_ROOT/Scenery:$FG_ROOT/WorldScenery`.

To add these in the Bourne shell (and compatibles):

```
LD_LIBRARY_PATH=/usr/local/share/FlightGear/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH
FG_HOME=/usr/local/share/FlightGear
export FG_HOME
FG_ROOT=/usr/local/share/FlightGear/data
export FG_ROOT
FG_SCENERY=$FG_ROOT/Scenery:$FG_ROOT/WorldScenery
export FG_SCENERY
```

or in C shell (and compatibles):

```
setenv LD_LIBRARY_PATH=\
  /usr/local/share/FlightGear/lib:$LD_LIBRARY_PATH
setenv FG_HOME=/usr/local/share/FlightGear
setenv FG_ROOT=/usr/local/share/FlightGear/data
setenv FG_SCENERY=\
  $FG_HOME/Scenery:$FG_ROOT/Scenery:$FG_ROOT/WorldScenery
```

Once you have these environmental variables set up, simply start ***FlightGear*** by running `fgfs --option1 --option2...`. Command-line options are described in Chapter 3.4.

3.2 Launching the simulator under Windows

The pre-built windows binaries come complete with a graphical wizard to start ***FlightGear***. Simply double-click on the `FlightGear Launcher` Start Menu item, or the icon on the Desktop. The launcher allows you to select

- your aircraft
- the start airport and runway
- time of day

- current weather
- ... and whole lot of other environmental settings

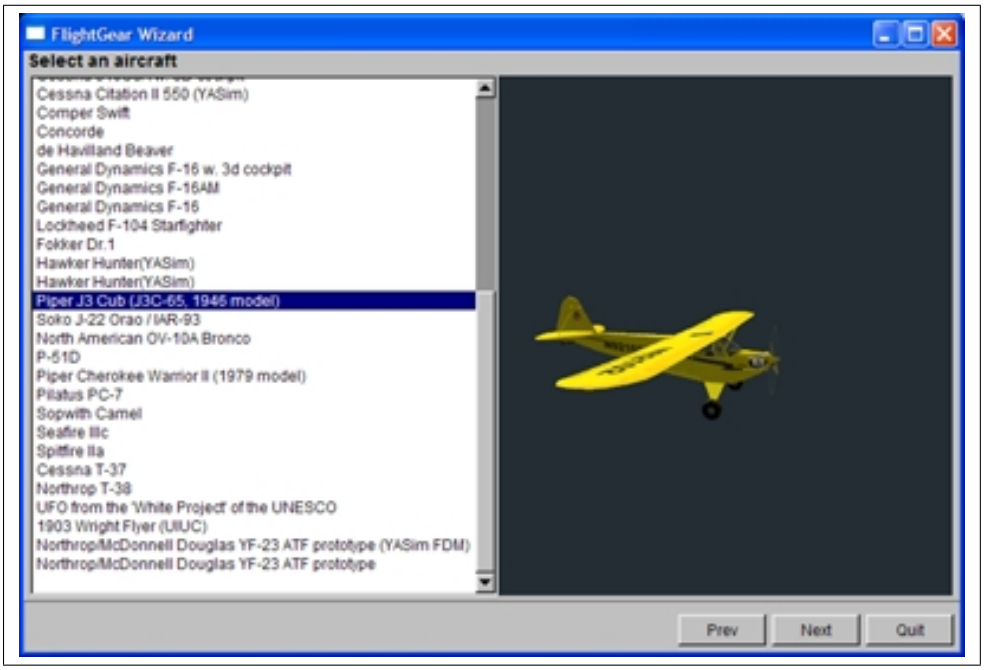


Fig. 4: *The FlightGear Launcher*

The first time your run it, you will be asked to set your `FG_ROOT` variable (normally `c:\Program Files\FlightGear\data`) and `FG_SCENERY`. This should be a list of the directories where you have installed scenery, typically `c:\Program Files\FlightGear\data\scenery` and `c:\Program Files\FlightGear\data\WorldScenery`.

Alternatively, you can run FlightGear from the command line. Open a command shell, change to the directory where your binary resides (typically something like `c:\Program Files\FlightGear\Win32\bin`), set the environment variables by typing

```
SET FG_HOME="c:\Program Files\FlightGear"
SET FG_ROOT="c:\Program Files\FlightGear\data"
SET FG_SCENERY="c:\Program Files\FlightGear\data\Scenery";\
 "c:\Program Files\FlightGear\data\WorldScenery"
```

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

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

Of course, you can create a batch file with a Windows text editor (like notepad) using the 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
```

or

```
./fgfs-X.X.X-version-date --fg-root=\~/fgfs-base-X.X.X --option1
```

3.4 Command line parameters

Following is a complete list and short description of the numerous command line options available for ***FlightGear***Most of these options are exposed through the FlightGear launcher delivered with the Windows binaries.

If you have options you re-use continually, you can include them in a preferences file. 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).

- On Unix systems, create a `.fgfsrc` file in your home directory.
- On Windows, create a `system.fgfsrc`, in the top `FG_ROOT` directory (e.g. `c:\Program Files\FlightGear`).

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.
- `--enable-random-objects`: Include random scenery objects (buildings/trees). This is the default.
- `--disable-random-objects`: Exclude random scenery objects (buildings/trees).
- `--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:\Program Files\Internet Explorer\iexplore.exe"` (Note the " " because of the spaces!).

- `--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-feet`: 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 etcare all loaded in a consistent way. For a full list, see Sec. 3.6 below.

- `--show-aircraft`: Print a sorted 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, pipe, ada, null`. Default value is `jsb` (***JSBSim***). larcsim is the flight model which ***FlightGear*** inherited from the LaRCSim simulatorẏasim is Any Ross' Yet Another Flight Dynamics Simulator. Magic is a slew mode (which drives the UFO aircraft). Balloon is a hot air balloon. External refers to remote control of the simulator via TCP socket, pipe is for local control via named pipe. 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). Values may be specified as a range by using a clon separator; e.g. 180:220@10:15

- `--random-wind`: Adds random wind to make flying more challenging

3.4.5 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 `/Flight Gear/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).

3.4.6 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=`*degrees*: 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.

3.4.7 HUD Options

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

3.4.8 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.

3.4.9 Network Options

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

3.4.10 Route/Waypoint Options

- `--wp=`*ID[@alt]*: Allows specifying a waypoint for the GC autopilot; it is possible to specify multiple waypoints (i.eả 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.

3.4.11 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.

3.4.12 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 etcòn the market with the several target operating systems, makes joystick support a nontrivial task in ***FlightGear***.

Beginning with version 0.8.0, ***FlightGear*** has a reworked 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 auto-detection 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 exists 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| Accessories), 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 ^ C 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
... etcùp 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 PRODUCTS CH FLIGHT SIM YOKE USB ) has 7 axes and 12 buttons.  Driver version is 2.1.0
Testing...(interrupt to exit)
Axes:  0:  0 1:  0 2:  0 3:  0 4:  0 5:  0 6:  0 Buttons:  0:off 1:off 2:off 3:on 4:off 5:off 6:off 7:off
8:off 9:off 10: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 paper 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 sub-folders of `/FlightGear/Input/Joysticks/`. After evaluating 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="1">
```

and change this to

```
<axis n="0">.
```

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

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

and change this to

```
<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.

Several users have reported that the numbers of axes and buttons assigned to functions may be different with the same joystick under Windows and Linux. The above procedure should allow one to easily change a binding xml file created for a different operating system for use by their operating system.

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 users 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 from 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 under windows, to be found under 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
```

on a Windows machine. The program will tell you which joysticks, if any, were 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 and switches, the directory above will hold a file called `fgfsrc.js`. If the ***FlightGear*** base directory `FlightGear` 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 `FlightGear`. 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 (and oldie but goldie in this respect is PFE, just make a web search for it). My favorite freeware file editor for that purpose, although somewhat dated, is still PFE, 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 axes moving the opposite way they 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` (one line)
`--prop:/input/joysticks/js[`*n*`]/axis[`*m*`]/binding`
`/property=/controls/`*steering option* (one line)
`--prop:/input/joysticks/js[`*n*`]/axis[`*m*`]/binding`
`/dead-band=`*db* (one line)
`--prop:/input/joysticks/js[`*n*`]/axis[`*m*`]/binding`
`/offset=`*os* (one line)
`--prop:/input/joysticks/js[`*n*`]/axis[`*m*`]/binding`
`/factor=`*fa* (one line)

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 model provided with the base package. As a result, you will see the default blue-yellow glider, when you change to the external view. However, you can download external views for these models from Wolfram Kuss' site at

http://home.t-online.de/home/Wolfram.Kuss/.

Moreover, this list is complete insofar as it covers all aircraft available via the `--aircraft=` option.

Tab. 1: *Presently available aircraft in **FlightGear**.*

Aircraft type	Start option	FDM	Remarks
Boeing 747	`--aircraft=747-yasim`	YASim	
BA A4 Hawk	`--aircraft=a4-yasim`	YASim	
North American X-15	`--aircraft=X15`	JSBSim	experimental supersonic plane
Airwave Xtreme 150	`--aircraft=airwaveXtreme150-v1-nl-uiuc`	UIUC	hang glider!
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-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-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-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.eśelect 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.

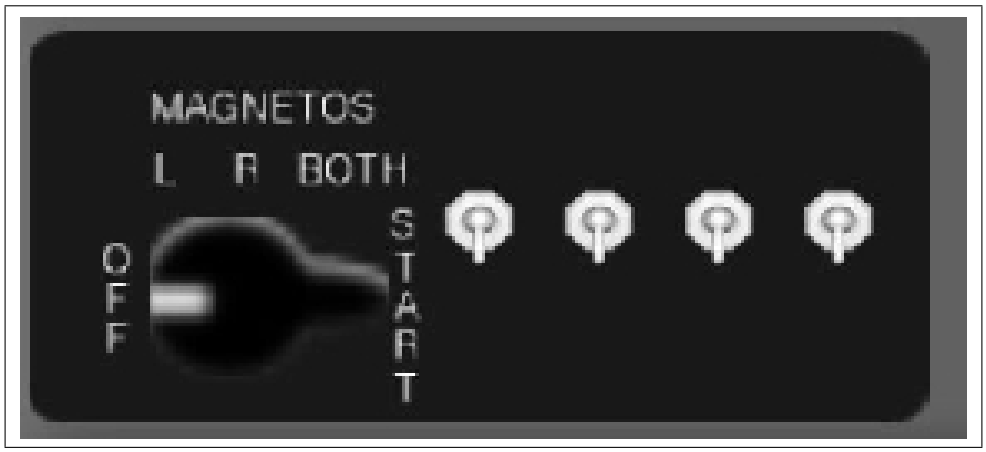


Fig. 4: *The ignition switch.*

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`*. [U.S. keyboard uses "." instead of ","]*

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 (if both are available)
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/V	Cycle view modes forth an back
Ctrl-c	Set view modes to pilot's view
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 ftto 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. [U.S. keyboard uses "." instead of ","]*

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/G	Toggle landing gear up/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-F9	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 case 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**
  - **Load flight** Loads the current flight, by default from `fgfs.sav`.
  - **Save flight** Saves the current flight, by default to `fgfs.sav`.

  - **Reset** Resets you to the selected starting position. Comes handy in case you got lost or something went wrong.
  - **Hires Snap Shot** Saves a high resolution Screen Shot under `fgfs-screen-XXX.ppm`.
  - **Snap Shot** Saves a normal resolution Screen Shot under `fgfs-screen-XXX.ppm`.
  - **Print** Prints screen shot (Linux only).
  - **Exit** Exits the program.

- **View**
  - **Properties** Provides access to numerous properties 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 3.4.5.

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

- **Network** (supposes compile option `--with-network-olk`)
  - **Unregister for FGD** Unregister from ***FlightGear*** Daemon.
  - **Register for FGD** Register for ***FlightGear*** Daemon.
  - **Scan for Daemons** Scan for daemons on the net.
  - **Enter Callsign** Enter your call sign.
  - **Toggle Display** Toggle call sign etcòn/off.

- **Help**

  - **Help** Opens your browser and displays an 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.

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 won'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.eȧ 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.eṡtep onto the left rudder pedal in case the ball is on the l.h.s.

Fig. 5: *The panel.*

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 display 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 display, 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, respṁarker beacon. 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.

You can change the numbers for the radios using the mouse. For this purpose, click left/right to the circular knob below the corresponding number. The corresponding switch left to this knob can be used for toggling between the active/standby frequency.

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.eṡhifting, replacing etcġauges 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 the toggle “i/I” minimizes/maximizes the HUD.

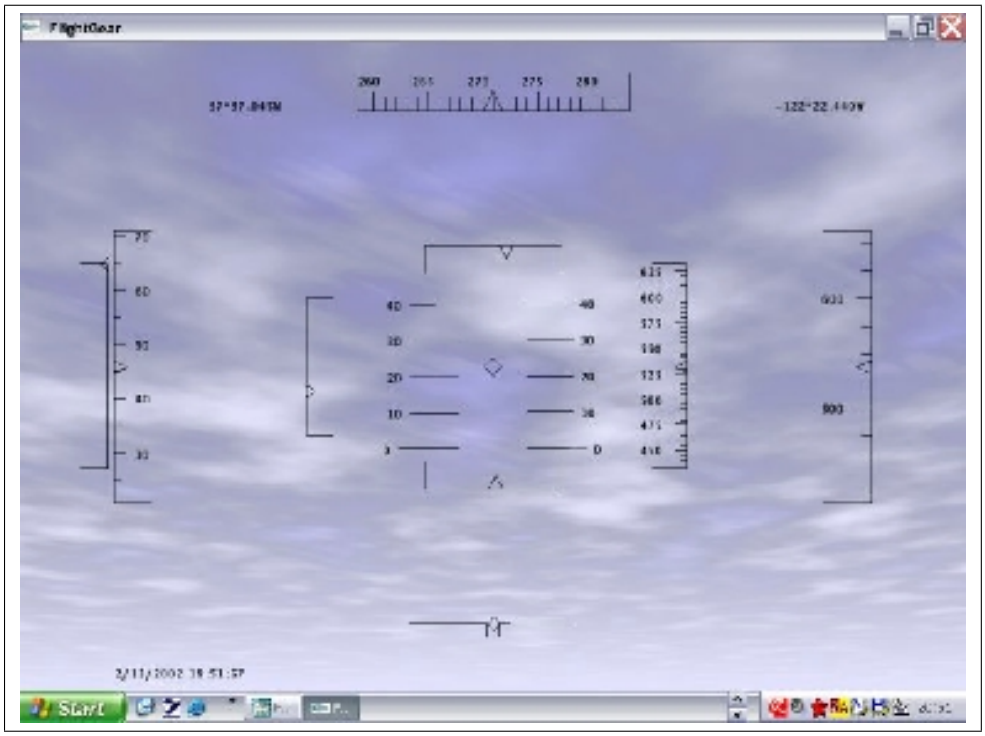


Fig. 7: *The HUD, or Head Up Display.*

4.6 Mouse controlled actions

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

There are three mouse modes. In the normal mode (pointer cursor) 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.

Part III

Tutorials

Chapter 5

Tutorials

5.1 FlightGear Tutorials

A range of ***FlightGear*** tutorials are available from various sources targetted at different users.

Eric Brasseur has written a very good tutorial for people completely new to ***FlightGear***, and flying in general. It also describes many basic principles of flight. Accessible here:

http://www.4p8.com/eric.brasseur/flight_simulator_tutorial.html.

Secondly, there is an excellent tutorial written by David Megginson – being one of the main developers of ***FlightGear*** – on flying a basic airport circuit specifically using ***FlightGear***. This document includes a lot of screen shots, numerical material etc., and is available from

http://www.flightgear.org/Docs/Tutorials/circuit.

A tutorial describing cross-country flight in FlightGear can be found in the following section.

5.2 Other Tutorials

There are many non-***FlightGear*** specific tutorials, many of which are applicable. First, a quite comprehensive manual of this type 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 flight 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

Chapter 6

A Cross Country Flight Tutorial

6.1 Introduction

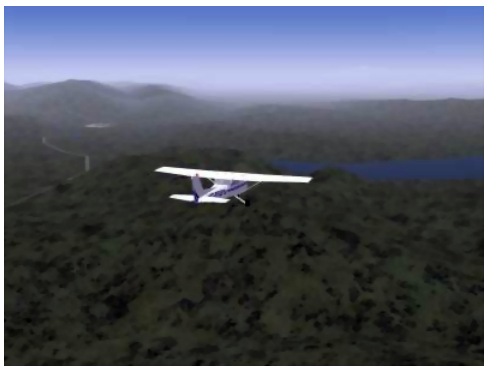


Figure 6.1: Flying over the San Antonio Dam to Livermore

This tutorial simulates a cross-country flight from Reid-Hillview (KRHV) to Livermore (KLVK) under Visual Flight Rules (VFR). Both airports are included in the standard FlightGear package, so no additional scenery is required.

I'll assume that you are happy taking off, climbing, turning, descending and landing in FlightGear. If not, have alook at the tutorials listed above. This tutorial is designed to follow on from them and provide information on some of the slightly more complicated flight systems and procedures.

6.1.1 Disclaimer and Thanks

A quick dislaimer. I'm not a pilot. Most of this information has been gleaned from various non-authoritive sources. If you find an error or misunderstanding, please let me know. Mail me at stuart_d_buchanan -at- yahoo.co.uk.

I'd like to thank the following people for helping make this tutorial accurate and readable. Benno Schulenberg, Sid Boyce. Vassilii Khachaturov, James Briggs.

6.2 Flight Planning

Before we begin, we need to plan our flight. Otherwise we'll be taking off not knowing whether to turn left or right.

First, have a look at the Sectional for the area. This is a map for flying showing airports, navigational aids, and obstructions. There are two scales of sectionals for VFR flight - the 1:500,000 sectionals themselves, and a number of 1:250,000 VFR Terminal Area Charts which cover particularly busy areas.

They are available from pilot shops, or on the web from various sources. http://aviationtoolbox.org has TIFF files of all the current sectionals and VFR Terminal Area Charts. We want the VFR Terminal Area Chart for San Francisco, which is available under the raw_data/FAA/sectionals/current/Terminal-Area-Charts/ directory as a 29MB download. An extract from that chart is shown in Figure 6.2.

If that is too big, http://mapserver.maptech.com has an Aeronautical Chart tab which allows them to be viewed quite easily in smaller views. Search for Reid-Hillview and Livermore.

If you want a map of the entire area showing exactly where the plane is, you can use http://atlas.sourceforge.net/Atlas. This is a moving-map program that connects to FlightGear.

So, how are we going to fly from Reid-Hillview to Livermore?

We'll be taking off from runway 31R at KRHV. KRHV is the ICAO code for Reid-Hillview airport, and is shown in the FlightGear wizard. (It is marked on the sectional as RHV for historic reasons. To get the ICAO code, simply prefix a 'K'.)

The 31 indicates that the magnetic heading of the runway is around 310 degrees, and the R indicates that it's the runway on the right. As can be seen from the sectional, there are two parallel runways at KRHV. This is to handle the large amount of traffic that uses the airport. Each of the runways can be used in either direction. Runway 31 can be used from the other end as runway 13. So, the runways available are 13R, 13L, 31R, 31L. Taking off and landing is easier done into the wind, so when the wind is coming from the North West, runways 31L and 31L will be in use. The name of the runway is written in large letters at the beginning and is easily seen from the air.

Once we take off we'll head at 350 degrees magnetic towards Livermore (KLVK). We'll fly at about 3,500ft about sea-level. This puts us at least 500ft above any terrain or obstructions like radio masts on the way.

We'll fly over the Calaveras Reservoir then the San Antonio Reservoir. These are both large bodies of water and we can use them as navigation aids to ensure we stay on the right track.

Once we get about 10 miles out of Livermore (above the San Antonia Reservoir), we'll contact the Livermore Air Traffic Control (ATC) to find out where we should land. We'll then join the circuit and land.

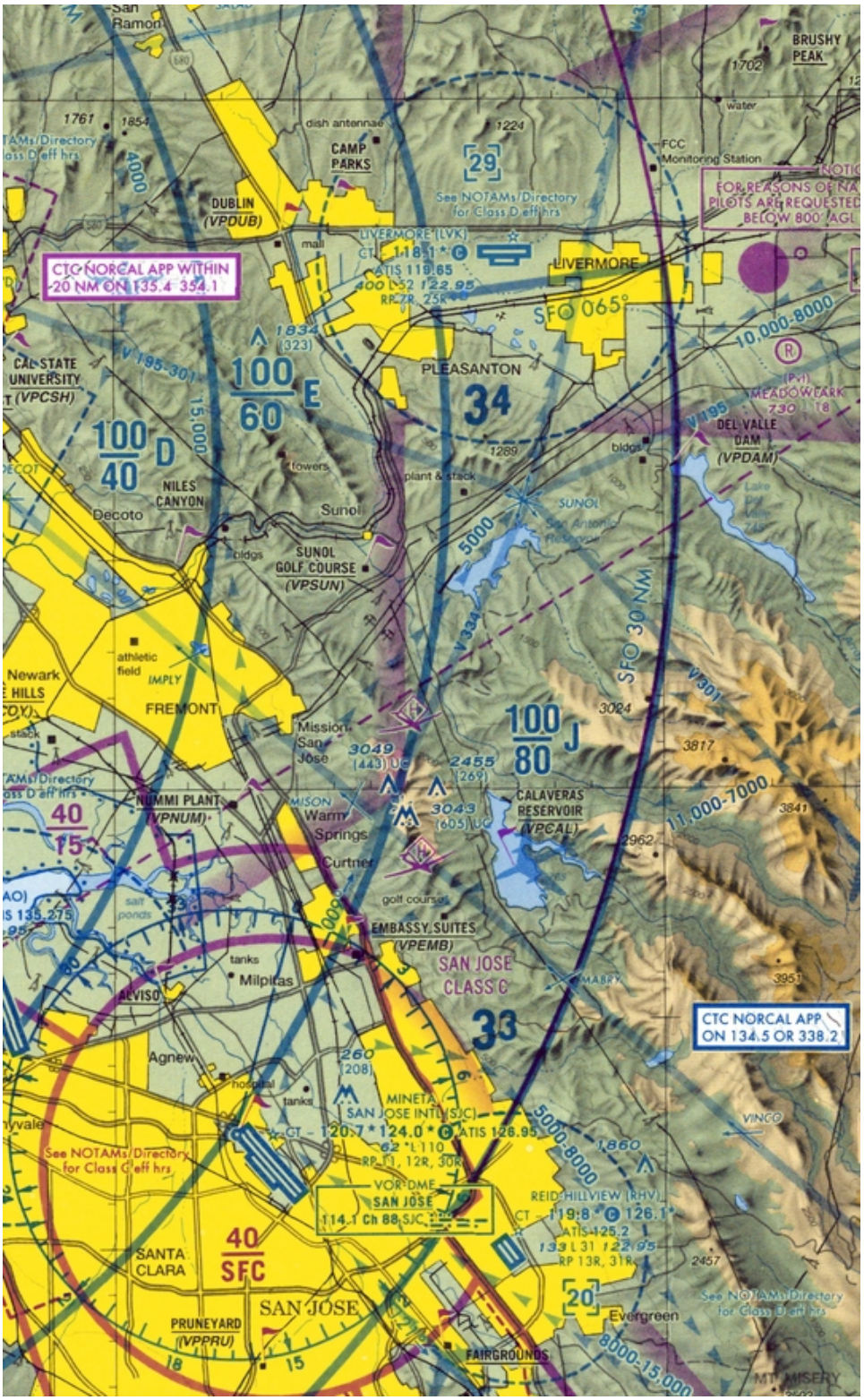


Figure 6.2: Sectional extract showing Reid-Hillview and Livermore airports

6.3 Getting Up

OK, we know where we're going and how we'll get there. Time to get started.

Start FlightGear using the Wizard (or command-line if you prefer). We want to use a C172P and take off from runway 31R at Reid-Hillview of Santa Clara County (KRHV). Dawn is a nice time to fly in California.

If you want, you can fly in the current weather at KRHV by clicking the Advanced button on the final screen of the Wizard, selecting Weather from the left-hand pane, selecting 'Fetch real weather' and clicking OK.

Figure 6.3: On the runway at KRHV

6.3.1 Pre-Flight

Before we take off, we need to pre-flight the aircraft. In the real world, this consists of walking around the aircraft to check nothing has fallen off, and checking we have enough fuel.

In our case, we'll take the opportunity to check the weather, set our altimeter and pre-set things that are easier to do when you're not flying.

The weather is obviously important when flying. We need to know if there is any sort of cross-wind that might affect take-off, at what altitude any clouds are (this is a VFR flight - so we need to stay well away from clouds at all times), and any wind that might blow us off course.

We also need to calibrate our altimeter. Altimeters calculate the current alititude indirectly by measuring air pressure, which decreases as you ascend. However, weather systems can affect the air pressure and lead to incorrect altimeter readings, which can be deadly if flying in mountains.

6.3.2 ATIS

Conveniently, airports broadcast the current sea-level pressure along with useful weather and airport information over the ATIS. This is a recorded message that is

broadcast over the radio. However, to listen to it, we need to tune the radio to the correct frequency.

The ATIS frequency is displayed on the sectional (look for 'ATIS' near the airport), but is also available from within FlightGear. To find out the frequencies for an airport (including the tower, ground and approach if appropriate), use the ATC/AI menu and select Frequencies. Then enter the ICAO code (KRHV) into the dialog box. The various frequencies associated with the airport are then displayed. Duplicates indicate that the airport uses multiple frequencies for that task, and you may use either.

Either way, the ATIS frequency for Reid-Hillview is 125.2MHz.

6.3.3 Radios

We now need to tune the radio. The radio is located in the Radio Stack to the right of the main instruments. There are actually two independent radio systems, 1 and 2. Each radio is split in two, with a communications (COMM) radio on the left, and a navigation (NAV) radio on the right. We want to tune COMM1 to the ATIS frequency.

Figure 6.4: The C172 communications stack with COMM1 highlighted

The radio has two frequencies, the active frequency, which is currently in use, and the standby frequency, which we tune to the frequency we wish to use next. The active frequency is shown on the left 5 digits, while the standby frequency is shown on the right. We change the standby frequency, then swap the two over, so the standby becomes active and the active standby. This way, we don't lose radio contact while tuning the radio.

To change the frequency, click on the grey knob below the standby frequency (highlighted in Figure 6.5), just to the right of the 'STBY'. Using the left mouse button changes the number after the decimal place, using the middle button changes the numbers before the decimal place. Click on the right side of the button to change the frequency up, and the left of the button to change the frequency down. Most of the FlightGear cockpit controls work this way. If you are having difficulty clicking on the correct place, press Ctrl-C to highlight the hot-spots for clicking.

Figure 6.5: COMM1 adjustment knob

Figure 6.6: COMM1 switch

Once you have changed the frequency to 125.2, press the white button between the words 'COMM' and 'STBY' to swap the active and standby frequencies (highlighted in Figure 6.6). After a second or so, you'll hear the ATIS information.

6.3.4 Altimeter and Compass

Figure 6.7: Altimeter calibration knob

Listen for the 'Altimeter' setting. If you are not using 'real weather', the value will be 2992, which is standard and already set on the plane. If you are using 'real weather', then the altimeter value is likely to be different. We therefore need to set the altimeter to the correct value. To do this, use the knob at the bottom left of the altimeter (circled in red in Figure 6.7), in the same way as you changed the radio frequency. This changes the value in the little window on the right of the altimeter, which is what you are trying to set, as well as the altitude displayed by the altimeter.

The other way to set the altimeter is to match it to the elevation above sea-level of the airport. The elevation is listed on the sectional. For KRHV it is 133ft. This means you can double-check the pressure value reported over ATIS.

Figure 6.8: Heading adjust knob

We will also take the opportunity to set the heading bug on the compass to 350 - our bearing from KRHV to KLVK. To do this, use the the red button on the compass housing (highlighted in Figure 6.8), just as you've done before. Use the left mouse button for small adjustments, and middle mouse button to make big adjustments. The value of 350 is just anti-clockwise of the labeled value of N (North - 0 degrees).

Figure 6.9: Take-off from KRHV

6.3.5 Take-Off

OK, now we've done that we can actually take off!. In my case this usually involves weaving all over the runway, and swerving to the left once I've actually left the ground, but you'll probably have better control than me. Once above 1000ft, make a gentle turn to the right to a heading of 350 degrees. As we've set the heading bug, it will be easy to follow. We're aiming for a fairly prominent valley.

Continue climbing to 3,500 ft at around 500-700 fpm. Once you reach that altitude, reduce power, level off to level flight and trim appropriately. Check the power again and adjust so it's in the green arc of the RPM guage. We shouldn't run the engine at maximum RPM except during take-off.

6.4 Cruising

OK, we've taken off and are on our way to Livermore. Now we can make our life a bit easier by using the autopilot and our plane more fuel efficient by tuning the engine. We'll also want to check we're on-course

6.4.1 The Autopilot

We can make our life a bit easier by handing some control of the aircraft over to 'George' - the autopilot.

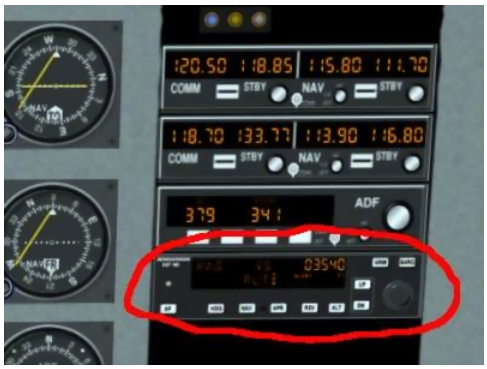


Figure 6.10: The C172 Autopilot

The autopilot panel is located towards the bottom of the radio stack (highlighted in Figure 6.10). It is easily distinguishable as it has many more buttons than the other components on the stack. It can work in a number of different modes, but we are only interested in one of them for this flight - HDG. As the names suggest, HDG will cause the autopilot to follow the heading bug on the compass, which we set earlier.

To set the autopilot, press the AP button to switch the autopilot on, then press the HDG button to activate heading mode. While the autopilot is switched on, it will use the trim controls to keep the plane on the heading. You can change the heading bug, and the autopilot will maneuver appropriately. However, the autopilot doesn't make any allowances for wind speed or direction, it only sets the heading of the airplane. If flying in a cross-wind, the plane may be pointed in one direction, but be travelling in quite another.

You should use the trim controls to keep a level flight. You can use the autopilot for this, but it is a bit more complicated.

Once the aircraft has settled down under the autopilot's control, we can pay more attention to the outside world and higher level tasks.

6.4.2 Navigation

As we noted above, we're going to be travelling over a couple of reservoirs. When you leveled off, the first (Calaveras) was probably right in front of you. You can use them to check your position on the map. If it looks like you're heading off course, twist the heading bug to compensate.

6.4.3 Mixture

As altitude increases, the air gets thinner and contains less oxygen. This means that less fuel can be burnt each engine cycle. The engine in the C172 is simple and doesn't automatically adjust the amount of fuel to compensate for this lack of oxygen. This results in an inefficient fuel burn and a reduction in power because the

Figure 6.11: The Calaveras Reservoir

Figure 6.12: The Calaveras Reservoir

fuel-air mixture is too 'rich'. We can control the amount of fuel entering the engine every cycle using the mixture control. This is the red lever next to the throttle. By pulling it out, we 'lean' the mixture. We don't want the mixture too rich, nor too lean. Both these conditions don't produce as much power as we'd like. Nor do we want it perfect, because this causes the fuel-air to explode, rather than burn in a controlled manner, which is a quick way to trash an engine.

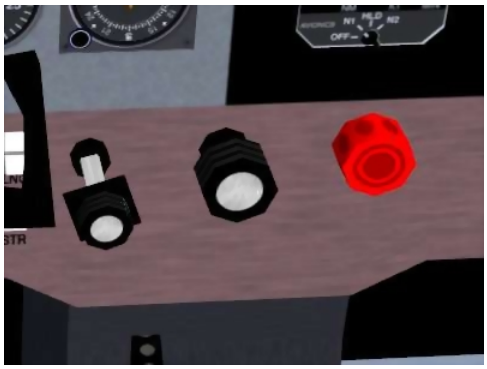


Figure 6.13: Mixture Control

The mixture is controlled by the red lever to the right of the yoke. You may need to pan your cockpit view to see it.

To pan the cockpit view, right-click with the mouse-button within the FlightGear window until the cursor becomes a double-headed arrow. Moving the mouse now pans the view. Once you can see the mixture lever clearly, right-click again to change the mouse back to the normal arrow.

Figure 6.14: Fuel Flow and EGT guages

Pull the mixture lever out slowly (use Ctrl-C to see the hot spots), leaning the mixture. As you do so, you'll see various engine instruments (on the left of the panel) change. Fuel flow will go down (we're burning less fuel), EGT (Exhaust Gas Temperature) will go up (we're getting closer to a 'perfect mixture') and RPM will increase (we're producing more power). Pull the mixture lever out until you see the EGT go off the scale, then push it in a bit. We're now running slightly rich

of peak. While at 3,500ft we don't need to lean much, at higher altitudes leaning the engine is critical for performance.

6.5 Getting Down

Once you reach the second reservoir (the San Antonio Reservoir), we need to start planning our descent and landing at Livermore. Landing is a lot more complicated than taking off, assuming you want to get down in one piece, so you may want to pause the simulator (press 'p') while reading this.

6.5.1 Air Traffic Control

In the Real World, we'd have been in contact with Air Traffic Control (ATC) continually, as the bay area is quite congested in the air as well as on the ground. ATC would probably provide us with a 'flight following' service, and would continually warn us about planes around us, helping to avoid any possible collisions. The FlightGear skies are generally clear of traffic, so we don't need a flight following service. If you want to change the amount of traffic in the sky, you can do so from the AI menu.

Livermore Airport is Towered (towered airports are drawn in blue on the sectional), so we will need to communicate with the tower to receive instructions on how and where to land.

Before that, we should listen to the ATIS, and re-adjust our altimeter, just in case anything has changed. This is quite unlikely on such a short flight, but if flying hundreds of milesm it might make a difference. To save time when tuning radios, you can access the Radio Settings dialog from the Equipment menu. The Livermore ATIS frequency is 119.65MHz.

An ATIS message also has a phonetic letter (Alpha, Bravo, ... Zulu) to identify the message. This phonetic is changed each time the recorded message is updated. When first contacting a tower, the pilot mentions the identifier, so the tower can double-check the pilot has up to date information.

Besides the altitude and weather information, the ATIS will also say which runway is in use. This is useful for planning our landing. Normally, due to the prevalent Westerly wind, Livermore has runways 25R and 25L in use.

Once you've got the ATIS, tune the radio to Livermore Tower. The frequency is 118.1MHz. Depending on the level of AI traffic you have configured on your system, you may hear Livermore Tower talking to other aircraft that are landing or departing. This information is not played over the speakers, it is only displayed on the screen.

Once the frequency goes quiet, press the ' key. This will bring up the ATC menu. Click on the radio button on the left to select what you wish to say (you only have one option), then OK.

Your transmission will be displayed at the top of the screen. It will indicate who you are (type and tail number), where you are (e.g. 6 miles south), that you are landing, and the ATIS you have.

After a couple of seconds, Livermore Tower will respond, addressing you by name and telling you what runway to use, which pattern is in use and when to contact them, for example

> "Golf Foxtrot Sierra, Livermore Tower, Report left downwind runway two five left."

To understand what this means, we'll have to describe the Traffic Pattern.

6.5.2 The Traffic Pattern

With the number of aircraft flying around, there have to be standard procedures for take-off and landing, otherwise someone might try to land on-top of an aircraft taking off.

The Traffic Pattern is a standard route all aircraft must follow when near an airport, either taking off or landing. The traffic pattern has four stages (or 'legs'), shown in Figure 6.15. The 'downwind' mentioned above refers to one of these, the one with the number 3.

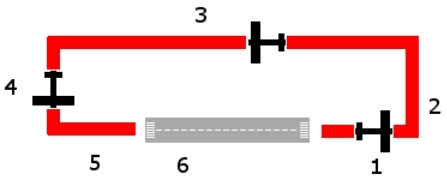


Figure 6.15: The Traffic Pattern

1. Aircraft take off from the runway and climb. If they are leaving the airport, they just continue climbing straight ahead until clear of the pattern and then do whatever they like. If they are returning to the runway (for example to practise landing), they continue climbing until they reach a couple of hundred feet below 'pattern altitude'. This varies from country to country, but is usually between 500ft and 1000ft Above Ground Level (AGL). This is called the *upwind* leg.

2. The pilot makes a 90 degree left-hand turn onto the *crosswind* leg. They continue their climb to 'pattern altitude' and level out.

3. After about 45 seconds to a minute on the crosswind leg, the pilot again makes a 90 degree left turn onto the *downwind* leg. Aircraft arriving from other airports join the pattern at this point, approaching from a 45 degree angle away from the runway.

4. When a mile or so past the end of the runway (a good guide is when the runway is 45 degrees behind you), the pilot turns 90 degrees again onto the *base* leg and begins the descent to the runway, dropping flaps as appropriate. A descent rate of about 500fpm is good.

5. After about 45 seconds the pilot turns again onto the *final* leg. It can be hard to estimate exactly when to perform this turn. Final adjustments for landing are made. I usually have to make small turns to align with the runway properly.

6. The aircraft lands. If the pilot is practising take-offs and landings, full power can be applied and flaps retracted for takeoff, and the aircraft can take off once more. This is known as 'touch-and-go'.

Most patterns at left-handed, i.e. all turns are to the left, as described above. Right-hand patterns also exist, and are marked as 'RP' on the sectional. ATC will also advise you what pattern is in use.

6.5.3 Approach

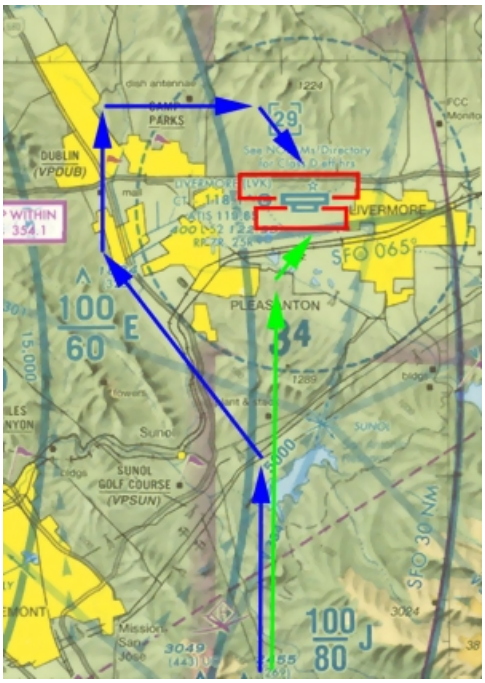


Figure 6.16: Sectional extract showing approaches to Livermore

We're approaching Livermore airport from the South, while the runways run East/West. Due to the prevailing Westerly wind, we'll usually be directed to either runway 25R or 25L. 25R uses a right-hand pattern, while 25L uses a left-hand

pattern. Both the patterns are illustrated in Figure 6.16. Depending on the runway we've been assigned, we'll approach the airport in one of two ways. If we've been asked to land on runway 25R, we'll follow the blue line in the diagram. If we've been asked to land on runway 25L, we'll follow the green line.

We also need to reduce our altitude. We want to end up joining the pattern at pattern altitude, about 1,000ft above ground level (AGL). Livermore airport is at 400 ft above sea-level (ASL), so we need to descend to an altitude of 1400 ASL.

We want to begin our maneuvers well before we reach the airport. Otherwise we're likely to arrive too high, too fast, and probably coming from the wrong direction. Not the best start for a perfect landing :).

So, let's start descending immediately.

1. First switch off the autopilot by pressing the AP switch.
2. Return mixture to fully rich (pushed right in). If we were landing at a high airport, we'd just enrich the mixture slightly and re-adjust when we reached the pattern.
3. Apply carb-heat. This stops ice forming when the fuel and air mix before entering the cylinder, something that can often happen during descent in humid air. The carb-heat lever is located between the throttle and mixture. Pull it out to apply heat.
4. Reduce power quite a bit. Otherwise we might stress the airframe due to over-speeding.
5. Drop the nose slightly to start the descent.
6. Trim the aircraft.

Use your location relative to the airport and the two towns of Pleasanton and Livermore to navigate yourself to the pattern following the general guide above.

Once you're established on the downwind leg, you'll need to report to ATC again. Do this in the same way as before. They will then tell you where you are in the queue to land. 'Number 1' means there are no planes ahead of you, while 'Number 9' means you might want to go to a less busy airport! They'll also tell you who is ahead of you and where. For example 'Number 2 for landing, follow the Cessna on short final' means that there is a single aircraft in front of you that is currently on the final leg of the pattern. When they land and are clear of the runway, they'll tell ATC, who can then tell you 'Number 1 for landing'.

6.5.4 VASI

Once on final, you'll notice two sets of lights on the left of the runway (enhanced in Figure 6.17). This is the VASI and provides a nice visual clue as to whether you're too low or too high on approach. Each set of lights can either be white or

Figure 6.17: On Final at Livermore with VASI on the left

red. White means too high, red means too low. White and red together means just perfect. On a Cessna approaching at 60kts, a descent rate of about 500fpm should be fine. If you are too high, just decrease power to increase your descent rate to 700fpm. If you are too low, increase power to decrease your descent rate to 200fpm.

6.5.5 Go Around

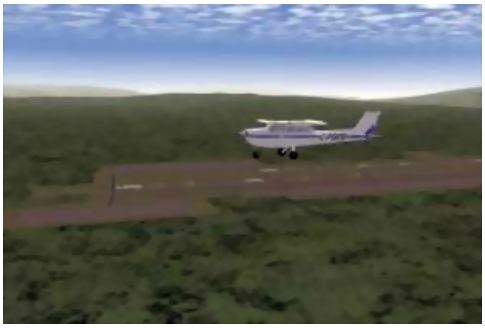


Figure 6.18: Missed approach at Livermore

If for some reason it looks like you're going to mess up the landing you can abort the landing and try again. This is called a 'Go Around'. To do this

1. Apply full power
2. Wait until you have a positive rate of climb - i.e. your altitude is increasing according to the altimeter.
3. Raise your flaps to 10 degrees (first-stage).
4. Tell ATC you are 'going around'
5. Climb to pattern height

6. If you aborted on final approach, continue over the runway to re-join the pattern on the crosswind leg. If on base, fly past the turn for final, then turn and fly parallel to the runway on the opposite side from downwind to rejoin on the crosswind leg.

7. Fly the complete pattern, telling ATC when you are on downwind, and try again.

6.5.6 Clearing the Runway

Figure 6.19: Landing at Livermore

Once you're on the ground, you should taxi off the runway, then tell ATC you are clear. At high-altitude airports, you would lean the engine to avoid fouling the spark-plugs with an over-rich mixture. Find somewhere nice to park, shut down the engine by pulling mixture to full lean, then throttle off and magnetos to off (knob on the bottom left of the panel). Switch off the avionics master switch, tie down the aircraft, then go get that hamburger!

I hope this tutorial is of some use. If you have any comments, please let me know at stuart_d_buchanan {at} yahoo.co.uk.

Part IV

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 . . .

For getting a trace of the output which ***FlightGear*** produces, then following command may come in handy (may need to be modified on some OSs or may not work on others at all, though):

```
%FG_ROOT/BIN/fgfs >log.txt 2>&1
```

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 messages 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 C.

- 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.ewith 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.

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 GCCṪhe Win32 port of GNU GCC is known as CygwinḞor 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 B. 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

Building the plane: Compiling the program

This appendix describes how to build ***FlightGear*** on several systems. In case you are on a Win32 (i. e. Windows95/98/ME/NT/2000/XP) platform or any of the other platforms for which binary executables are available, you may not want to go through that potentially troublesome process but skip this section for now and go back to the chapter on installing ***FlightGear*** one. (Not everyone wants to build his or her plane himself or herself, right?) However, there may be good reason for at least trying to build the simulator:

- In case you are on a UNIX/Linux platform there may be no pre-compiled binaries available for your system. In practice it is common to install programs like this one on UNIX systems by recompiling them.
- There are several options you can set during compile time only.
- You may be proud you did.

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

As you will notice, this Chapter is far from being complete. Basically, we describe compiling for two operating systems only, Windows and Linux, and for only one compiler, the GNU C compiler. ***FlightGear*** has been shown to be built under different compilers (including Microsoft Visual C) as well as different systems (Macintosh) as well. The reason for these limitations are:

- Personally, we have access to a Windows machine running the Cygnus compiler only.
- According to the mailing lists, these seem to be the systems with the largest user base.

- These are the simplest systems to compile ***FlightGear*** on. Other compilers may need special add-ons (workplace etc.) or even modification of the code.
- The GNU compiler is free in the same sense of the GPL as ***FlightGear*** is.

You might want to check Section A, *Missed approach*, if anything fails during compilation. In case this does not help we recommend sending a note to one of the mailing lists (for hints on subscription see Chapter D).

There are several Linux distributions on the market, and most of them should work. Some come even bundled with (often outdated) versions of ***FlightGear***. However, if you are going to download or buy a distribution, Debian (Sarge) is highly recommended by most people. SuSE and Ubuntu works well, too.

Contrary to Linux/Unix systems, Windows usually comes without any development tools. This way, you first have to install a development environment. On Windows, in a sense, before building the plane you will have to build the plant for building planes. This will be the topic of the following section, which can be omitted by Linux users.

B.1 Preparing the development environment under Windows

There is a powerful development environment available for Windows and this even for free: The Cygnus development tools, resp. ***Cygwin***. Their home is at

http://sources.redhat.com/cygwin/,

and it is always a good idea to check back what is going on there now and then.

Nowadays, installing ***Cygwin*** is nearly automatic. First, make sure the drive you want ***Cygwin***, ***PLIB***, ***SimGear*** and ***FlightGear*** to live on, has nearly 1 GB of free disk space. Create a temporary directory and download the installer from the site named above to that directory. (While the installer does an automatic installation of the Cygnus environment, it is a good idea to download a new installer from time to time.)

Invoke the installer now. It gives you three options. To avoid having to download stuff twice in case of a re-installation or installation on a second machine, we highly recommended to take a two-step procedure. First, select the option `Download from Internet`. Insert the path of your temporary directory, your Internet connection settings and then choose a mirror form the list. Near servers might be preferred, but may be sometimes a bit behind with mirroring. We found

ftp://mirrors.rcn.net

a very recent and fast choice. In the next windows the default settings are usually a good start. Now choose `Next`, sit back and wait.

If you are done, invoke the installer another time, now with the option `Install from local directory`. After confirming the temporary directory you can

select a root directory (acting as the root directory of your pseudo UNIX file system). Cygnus does not recommend taking the actual root directory of a drive, thus choose `c:/Cygwin` (while other drives than `c:` work as well). Now, all ***Cygwin*** stuff and all ***FlightGear*** stuff lives under this directory. In addition, select

```
Default text file type:  Unix
```

In addition, you have the choice to install the compiler for all users or just for you.

The final window before installation gives you a selection of packages to install. It is hard, to provide a minimum selection of packages required for ***FlightGear*** and the accompanying libraries to install. We have observed the following (non minimum) combination to work:

- `Admin` skip
- `Archive` install
- `Base` install
- `Database` skip
- `Devel` install
- `Doc` install
- `Editors` skip
- `Graphics` install
- `Interpreters` install
- `Libs` install
- `Mail` skip
- `Net` skip
- `Shells` install
- `Text` install
- `Utils` install
- `Web` skip
- `XFree86` do not install!

Note XFree86 must be not installed for building ***FlightGear*** and the accompanying libraries. If it is installed you have to deinstall it first. Otherwise ***FlightGear***'s configuration scripts will detect the XFree86 OpenGL libraries and link to them, while the code is not prepared to do so.

As a final step you should include the binary directory (for instance: `c:/Cygwin/bin`) into your path by adding `path=c:\Cygwin\bin` in your `autoexec.bat` under Windows 95/98/MEUnder WindowsNT/2000/XP, use the `Extended` tab under the `System properties` page in Windows `control panel`. There you'll find a button `Environment variables`, where you can add the named directory.

Now you are done. Fortunately, all this is required only once. At this point you have a nearly UNIX-like (command line) development environment. Because of this, the following steps are nearly identical under Windows and Linux/Unix.

B.2 Preparing the development environment under Linux

Linux, like any UNIX, usually comes with a compiler pre-installed. On the other hand, you still have to make sure several required libraries are present.

First, make sure you have all necessary OpenGL libraries. Fortunately, most of the recent Linux distributions (i.e. SuSE-7.3) put these already into the right place. (There have been reports, though, that on Slackware you may have to copy the libraries to `/usr/local/lib` and the headers to `/usr/local/include` by hand after building `glut-3.7`). Be sure to install the proper packages: Besides the basic X11 stuff you want to have - SuSE as an example - the following packages: mesa, mesa-devel, mesasoft, xf86_glx, xf86glu, xf86glu-devel, mesaglut, mesaglut-devel and plib.

Also you are expected to have a bunch of tools installed that are usually required to compile the Linux kernel. So you may use the Linux kernel source package to determine the required dependencies. The following packages might prove to be useful when fiddling with the ***FlightGear*** sources: automake, autoconf, libtool, bison, flex and some more, that are not required to build a Linux kernel.

Please compare the release of the Plib library with the one that ships with your Linux distribution. It might be the case that ***FlightGear*** requires a newer one that is not yet provided by your vendor.

B.3 One-time preparations for Linux and Windows users

There are a couple of 3rd party libraries which your Linux or Windows system may or may not have installed, i.ethe ***ZLIB*** library. You can either check your list of installed packages or just try building ***SimGear***: It should exit and spit an error message (observe this!) if one of these libraries is missing.

If you make this observation, install the missing libraries, which is only required once (unless you re-install your development environment).

Both libraries come bundled with ***SimGear***, which links to them, but does not automatically install them. For installing either of them, get the most recent file `SimGear-X.X.X.tar.gz` from

http://www.simgear.org/downloads.html

Download it to `/usr/local/source`. Change to that directory and unpack ***SimGear*** using

`tar xvfz SimGear-X.X.X.tar.gz`.

You will observe a directory `src-libs` which contains the two names libraries.

B.3.1 Installation of *ZLIB*

`cd` into `SimGear-X.X.X/scr-libs` and unpack ***ZLIB*** using

`tar xvfz zlib-X.X.X.tar.gz`.

Next, change to the newly created directory `zlib-X.X.X` and type

```
./configure
make
make install
```

Under Linux, you have to become root for being able to `make install`, for instance via the `su` command.

You may want to consult the Readme files under `SimGear-X.X.X/scr-libs` in case you run into trouble.

B.4 Compiling *FlightGear* under Linux/Windows

The following steps are identical under Linux/Unix and under Windows with minor modifications. Under Windows, just open the ***Cygwin*** icon from the Start menu or from the desktop to get a command line.

To begin with, the ***FlightGear*** build process is based on four packages which you need to built and installed in this order:

- PLIB
- SimGear
- FlightGear, program
- FlightGear, base (data - no compilation required)

1. First, choose an install directory for FlightGear. This will not be the one your binaries will live in but the one for your source code and compilation files. We suggest

   `cd:/usr/local/`

   `mkdir source`

2. Now, you have to install a support library ***PLIB*** which is absolutely essential for the building process. ***PLIB*** contains most of the basic graphics rendering, audio, and joystick routines. Download the latest stable version of ***PLIB*** from

   http://plib.sourceforge.net/

   to `/usr/local/source`. Change to that directory and unpack ***PLIB*** using

   `tar xvfz plib-X.X.X.tar.gz`.

   `cd` into `plib-X.X.X` and run

   ```
   ./configure
   make
   make install.
   ```

   Under Linux, you have to become root for being able to `make install`, for instance via the `su` command.

   Confirm you now have ***PLIB***'s header files (as `ssg.h` etc.) under `/usr/include/plib` (and nowhere else).

3. Next, you have to install another library ***SimGear*** containing the basic simulation routines. Get the most recent file `SimGear-X.X.X.tar.gz` from

   http://www.simgear.org/downloads.html

   Download it to `/usr/local/source`. Change to that directory and unpack ***SimGear*** using

   `tar xvfz SimGear-X.X.X.tar.gz`.

   `cd` into `SimGear-X.X.X` and run

   ```
   ./configure
   make
   make install
   ```

   Again, under Linux, you have to become root for being able to `make install`, for instance via the `su` command.

4. Now, you're prepared to build ***FlightGear*** itself, finally. Get `FlightGear-X.X.X.tar.gz` from

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

   and download it to `/usr/local/source`. Unpack ***FlightGear*** using

   `tar xvfz FlightGear-X.X.X.tar.gz`.

`cd` into `FlightGear-X.X.X` and run

```
./configure
```

configure knows about numerous options, with the more relevant ones to be specified via switches as

- `--with-network-olk`: Include Oliver Delise's multi-pilot networking support,
- `--with-new-environment`: Include new experimental environment subsystem,
- `--with-weathercm`: Use WeatherCM instead of FGEnvironment,
- `--with-plib`=PREFIX: Specify the prefix path to ***PLIB***,
- `--with-simgear`=PREFIX: Specify the prefix path to ***SimGear***,
- `--prefix=/XXX`: Install ***FlightGear*** in the directory `XXX`.
- `--disable-jsbsim`: Disable ***JSBSim***m FDM (in case of trouble compiling it).
- `--disable-yasim`: Disable ***YASim*** FDM (in case of trouble compiling it).
- `--disable-larcsim`: Disable ***LaRCsim*** FDM (in case of trouble compiling it).
- `--disable-uiuc`: Disable UIUC FDM (in case of trouble compiling it).

A good choice would be `--prefix=/usr/local/FlightGear`. In this case ***FlightGear***'s binaries will live under `/usr/local/FlightGear/bin`. (If you don't specify a `--prefix` the binaries will go into `/usr/local/bin` while the base package files are expected under `/usr/local/share/FlightGear`.)

Assuming `configure` finished successfully, run

```
make
make install.
```

Again, under Linux, you have to become root for being able to `make install`, for instance via the `su` command.

Note: You can save a significant amount of space by stripping all the debugging symbols off the executable. To do this, make a

```
cd /usr/local/FlightGear/bin
```

to the directory in the `install tree` where your binaries live and run

```
strip *.
```

This completes building the executable and should result in a file `fgfs` (Unix) or `fgfs.exe` (Windows) under `/usr/local/FlightGear/bin`

Note: If for whatever reason you want to re-build the simulator, use the command `make distclean` either in the `SimGear-X.X.X` or in the `FlightGear-X.X.X` directory to remove all the build. If you want to re-run `configure` (for instance because of having installed another version of ***PLIB*** etc.), remove the files `config.cache` from these same directories before.

B.5 Compiling *FlightGear* under Mac OS X

For compiling under Mac OS X you will need

- Mac X OS 10.1+ with developer tools installed.
- 500MB disk (minimum) free disk space.
- Fearlessness of command line compiling.

This will need a bit more bravery than building under Windows or Linux. First, there are less people who tested it under sometimes strange configurations. Second, the process as described here itself needs a touch more experience by using CVS repositories.

First, download the development files. They contain files that help simplify the build process, and software for automake, autoconf, and plib:

http://expert.cc.purdue.edu/~walisser/fg/fgdev.tar.gz

or

http://homepage.mac.com/walisser

Once you have this extracted, make sure you are using TCSH as your shell, since the setup script requires it.

Important for Jaguar users:

If you run Mac OS X 10.2 or later, gcc 3.1 is the default compiler. However, only version 2.95 works with ***FlightGear*** as of this writing. To change the default compiler, run this command (as root). You'll only have to do this once and it will have a global effect on the system.

```
sudo gcc_select 2
```

1. Setup the build environment:
   ```
   cd fgdev
   source bin/prepare.csh
   ```
2. Install the latest versions of the automake and autoconf build tools:
   ```
   cd $BUILDDIR/src/automake-X.X.X
   ./configure --prefix=$BUILDDIR
   ```

```
make install
rehash
cd $BUILDDIR/src/autoconf-X.XX
./configure --prefix=$BUILDDIR
make install
rehash
```

3. Download PLIB
   ```
   cd $BUILDDIR/src
   setenv CVSROOT :pserver:anonymous@cvs.plib.sourceforge.net:/cvsroot/plib
   cvs login
   ```
   Press <enter> for password
   ```
   cvs -z3 checkout plib
   ```

4. Build PLIB
   ```
   cd $BUILDDIR/src/plib
   ./autogen.sh
   ./configure --prefix=$BUILDDIR
   make install
   ```

5. Get the ***SimGear*** sources
   ```
   cd $BUILDDIR/src
   setenv CVSROOT :pserver:cvsguest@cvs.simgear.org:/var/cvs/SimGear-0.3
   cvs login
   ```
   Enter <guest> for password
   ```
   cvs -z3 checkout SimGear
   ```

6. Build ***SimGear***
   ```
   cd $BUILDDIR/src/SimGear
   ./autogen.sh
   ./configure -prefix=$BUILDDIR
   make install
   ```

7. Get the ***FlightGear*** sources
   ```
   cd $BUILDDIR/src
   setenv CVSROOT :pserver:cvsguest@cvs.flightgear.org:/var/cvs/FlightGear-0
   cvs login
   ```
   Enter <guest> for password
   ```
   cvs -z3 checkout FlightGear
   ```

8. Build ***FlightGear***
   ```
   cd $BUILDDIR/src/FlightGear
   patch -p0 < ../jsb.diff
   ./autogen.sh
   ```

```
./configure -prefix=$BUILDDIR
-with-threads -without-x (one line)
make install
```

9. Get the base data files (if you don't have them already)
   ```
   cd $BUILDDIR
   setenv CVSROOT :pserver:cvsguest@cvs.flightgear.org:/var/cvs/Flig
   cvs login
   ```
   Password is "guest"
   ```
   cvs -z3 checkout data
   ```

10. Move data files (if you have them already)
    just make a symlink or copy data files to "fgfsbase" in $BUILDDIR
    alternatively adjust `--fg-root=xxx` parameter appropriately

11. Run FlightGear
    ```
    cd $BUILDDIR
    src/FlightGear/src/Main/fgfs
    ```

B.6 Compiling on other systems

Compiling on other UNIX systems - at least on IRIX and on

Solaris, is pretty similar to the procedure on Linux - given the presence of a working GNU C compiler. Especially IRIX and also recent releases of Solaris come with the basic OpenGL libraries. Unfortunately the "glut" libraries are mostly missing and have to be installed separately (see the introductory remark to this chapter). As compilation of the "glut" sources is not a trivial task to everyone, you might want to use a pre-built binary. Everything you need is a static library "libglut.a" and an include file "glut.h". An easy way to make them usable is to place them into `/usr/lib/` and `/usr/include/GL/`. In case you insist on building the library yourself, you might want to have a look at FreeGLUT

http://freeglut.sourceforge.net/

which should compile with minor tweaks. Necessary patches might be found in

ftp://ftp.uni-duisburg.de/X11/OpenGL/freeglut_portable.patch

Please note that you do **not** want to create 64 bit binaries in IRIX with GCC (even if your CPU is a R10/12/14k) because GCC produces a broken "fgfs" binary (in case the compiler doesn't stop with "internal compiler error"). Things look better since Eric Hofman managed to tweak the ***FlightGear*** sources for proper compiling with MIPSPro compiler.

There should be a workplace for Microsoft Visual C++ (MSVC6) included in the official ***FlightGear*** distribution. Macintosh users find the required CodeWarrior files as a `.bin` archive at

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

Numerous (although outdated, at times) hints on compiling on different systems are included in the source code under `docs-mini`.

B.7 Installing the base package

If you succeeded in performing the steps named above, you will have a directory holding the executables for ***FlightGear***. This is not yet sufficient for performing ***FlightGear***, though. Besides those, you will need a collection of support data files (scenery, aircraft, sound) collected in the so-called base package. In case you compiled the latest official release, the accompanying base package is available from

ftp://www.flightgear.org/pub/flightgear/Shared/fgfs-base-X.X.X.tar.gz.

This package is usually quite large (around 25 MB), but must be installed for ***FlightGear*** to run. There is no compilation required for it. Just download it to `/usr/local` and install it with

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

Now you should find all the ***FlightGear*** files under `/usr/local/Flightgear` in the following directory structure::

```
/usr/local/Flightgear
/usr/local/Flightgear/Aircraft
/usr/local/Flightgear/Aircraft-uiuc
...
/usr/local/Flightgear/bin
...
/usr/local/Flightgear/Weather.
```

B.8 For test pilots only: Building the CVS snapshots

It you are into adventures or feel you're an advanced user, you can try one of the recent bleeding edge snapshots at

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

In this case you have to get the most recent Snapshot from ***SimGear*** at

http://www.simgear.org/downloads.html

as well. But be prepared: These are for development and may (and often do) contain bugs.

If you are using these CVS snapshots, the base package named above will usually not be in sync with the recent code and you have to download the most recent developer's version from

http://rockfish.net/fg/.

We suggest downloading this package `fgfs_base-snap.X.X.X.tar.gz` to a temporary directory. Now, decompress it using

`tar xvfz fgfs_base-snap.X.X.X.tar.gz`.

Finally, double-check you got the directory structure named above.

Appendix C

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.

C.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. Please enshure to replace any OpenGL related libraries with those that are shipped with the NVIDIA driver - not only user space libraries but also those in the X server extension modules directory.

C.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.

C.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.

C.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 ourselves, but would suggest it for those completely lost.

C.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 color 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.

C.6 ATI chip based cards under Linux

There is 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 mostly usable for FlightGear. Since XFree86-4.1 you can use early Radeon chips - up to Radeon7500 with XFree86-4.2, up to Radeon9100 with XFree86-4.3. Be careful with stock XFree86-4.3.0, it was released with (known) bugs in the Radeon driver. Ongoing development provides functional drivers for R100 (Radeon7000 up to 7500) and R200 (Radeon8500 and 9100) chips.

ATI provides an alternative with their binary drivers. You need to build a kernel module using a script that is supplied with the package and add several X server modules into your XFree86 tree. In most cases, the RPM installation will do that for you.

C.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 self-made kernel, you want to compile ***FlightGear*** yourself, you're tweaking your X server configuration file yourself or you even run

a homebrewed 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/dri_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 appropriate location. If you compile the kernel yourself, then you have to take care of two sub-menus in the kernel configuration menu. You'll find them in the "Character devices" menu. 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 definitely want to activate "Direct Rendering Manager" for your graphics 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 chip set. 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 OpenGLṪhis 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 sure 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 developmental 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 command-line 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 CPUİn this case you might want to have a closer look at the preceding paragraphs of this chapter. Now please make sure 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 available 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/documentation.phtml

C.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 OSX.

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 D

Landing: Some further thoughts before leaving the plane

D.1 A Sketch on the History of *FlightGear*

History may be a boring subject. However, from time to time there are people asking for the history of ***FlightGear***. As a result, we'll give a short outline.

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.

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://edc.usgs.gov/geodata/

for the U.S., and

http://edcdaac.usgs.gov/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.

D.1.1 Scenery

- Texture support was added by Curt Olson in spring 1998. This marked a significant improvement in terms of reality. Some high-quality textures were submitted by Eric Mitchell for the ***FlightGear*** project. Another set of high-quality textures was added by Erik Hofman ever since.

- 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. 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.

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

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

- Scenery was further improved by adding geographic features including lakes, rivers,and coastlines later, an effort still going on. 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. Runway lighting was introduced by Curt Olson in spring 2001. Finally, a completely new set of scenery files for the whole world was created by William Riley based on preparatory

documentation by David Megginson in summer 2002. This is based on a data set called VMap0 as an alternative to the GSHHS data used so far. This scenery is a big improvement as it has world wide coverage of main streets, rivers, etc., while it's downside are much less accurate coast lines. ***FlightGear***'s base scenery is based on these new scenery files since summer 2002. The complete set is available via a clickable map, too, from

http://www.randdtechnologies.com/fgfs/newScenery/world-scenery.html.

- There was support added for static objects to the scenery in 2001, which permits placing buildings, static planes, trees and so on in the scenery. However, despite a few proofs of concept systematic inclusion of these landmarks is still missing.
- The world is populated with random ground objects with appropriate type and density for the local ground cover type since summer 2002. This marks a mayor improvement of reality and is mainly thanks to work by D. Megginson.

D.1.2 Aircraft

- 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.
- 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.
- Friedemann Reinhard developed early instrument panel code, which was added in June 1998. Unfortunately, development of that panel slowed down later. 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.
- Finally, LaRCsims Navion was replaced as the default aircraft when the Cessna 172 was stable enough in February 2000 - as move most users will welcome. There are now several flight model and airplane options to choose from at runtime. 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, did 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 simplicity of use and is based on fluid dynamics, by the end of 2001. This one bought us flight models for a 747, an A4, and a DC-3. Alternatively, a group around Michael Selig from the UIUC group provided another flight model along with several planes since around 2000.

- 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. 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.. Moreover, several panels were continually improved during 2001 and 2002 by John and others. ***FlightGear*** now allows flying ILS approaches and features a Bendix transponder.

- In 2002 functional multi-engine support found it's way into ***FlightGear***. ***JSBSim*** is now the default FDM in ***FlightGear***.

- Support of ''true" 3D panels became stable via contributions from John Check and others in spring 2002. In addition, we got movable control surfaces like propellers etc., thanks to David Megginson.

D.1.3 Environment

- 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.

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

D.1.4 User Interface

- The foundation 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.

- In 1998 there was basic audio support, i. eàn 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. Joystick support was further improved by adding a self detection feature based on xml joystick files, by David Megginson in summer 2002.

- 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 for the "Atlas" moving map program. Besides, an embedded HTTP server developed by Curt Olson late in 2001 can now act a property manager for external programs.

- 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.

- 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 shown to be 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.

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

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 etcroutines. 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 D 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/

D.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
Granted permission for the ***FlightGear*** project to use some of the sound effects from their site. Homepage under

http://www.a1freesoundeffects.com/

Raul Alonzo
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
Contributed to the HUD code.

Michael Basler
Author of Installation and Getting Started. Flight Simulation Page at

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

Jon S. Berndt
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
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
Provided a big chunk of online space to store USA scenery for ***FlightGear***!.

Bernie Bright
Many C++ style, usage, and implementation improvements, STL portability and much, much more. Added threading support and a threaded tile pager.

Bernhard H. Buckel
Contributed the README.Linux. Contributed several sections to earlier versions of Installation and Getting Started.

Gene Buckle
A lot of work getting ***FlightGear*** to compile with the MSVC++ compiler. Numerous hints on detailed improvements.

Ralph Carmichael
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
Provided some initial code to parse the 30 arcsec DEM files found at:

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

John Check
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://www.rockfish.net/fg/.

Dave Cornish
Dave created new cool runway textures plus some of our cloud textures.

Oliver Delise
Started a FAQ, Documentation, Public relations. Working on adding some networking/multi-user code. Founder of the FlightGear MultiPilot

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
Contributed some sphere interpolation code used by Christian Mayer's weather data base system.

Francine Evans
Wrote the GPL'd tri-striper we use.

http://www.cs.sunysb.edu/~stripe/

Oscar Everitt
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
Contributed patches for MSVC5 compatibility.

Melchior Franz
Contributed joystick hat support, a LED font, improvements of the telnet and the http interface. Notable effort in hunting memory leaks in ***FlightGear***, ***SimGear***, and ***JSBSim***.

Jean-loup Gailly and **Mark Adler**
Authors of the zlib library. Used for on-the-fly compression and decompression routines,
http://www.gzip.org/zlib/.

Mohit Garg
Contributed to the manual.

Thomas Gellekum
Changes and updates for compiling on FreeBSD.

Neetha Girish
Contributed the changes for the xml configurable HUD.

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

Michael I. Gold
Patiently answered questions on OpenGL.

Habibe
Made RedHat package building changes for SimGear.

Mike Hill
For allowing us to concert and use his wonderful planes, available form

http://www.flightsimnetwork.com/mikehill/home.htm,
for ***FlightGear***.

Erik Hofman
Major overhaul and parameterization of the sound module to allow aircraft-specific sound configuration at runtime. Contributed SGI IRIX support (including binaries) and some really great textures.

Charlie Hotchkiss
Worked on improving and enhancing the HUD code. Lots of code style tips and code tweaks.

Bruce Jackson (NASA)
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
Contributed the Debian binary.

Richard Kaszeta
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 ŕBossŕBob Hain 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.kaszeta.org/rich/.

Tom Knienieder
Ported the audio library first to OpenBSD and IRIX and after that to Win32.

Reto Koradi
Helped with setting up fog effects.

Bob Kuehne
Redid the Makefile system so it is simpler and more robust.

Kyler B Laird
Contributed corrections to the manual.

David Luff
Contributed heavily to the IO360 piston engine model.

Christian Mayer
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
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. Random ground cover objects

Cameron Moore
FAQ maintainer. Reigning list administrator. Provided man pages.

Eric Mitchell
Contributed some topnotch scenery textures being all original creations by him.

Anders Morken
Former maintainer of European web pages.

Alan Murta
Created the Generic Polygon Clipping library.

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

Phil Nelson
Author of GNU dbm, a set of database routines that use extendible hashing and work similar to the standard UNIX dbm routines.

Alexei Novikov
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
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/

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
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.

Robin Peel
Maintains worldwide airport and runway database for ***FlightGear*** as well as X-Plane.

Alex Perry
Contributed code to more accurately model VSI, DG, Altitude. Suggestions for improvements of the layout of the simulator on the mailing list and help on documentation.

Friedemann Reinhard
Development of an early textured instrument panel.

Petter Reinholdtsen
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
Contributed code to add "brakes". Also wrote a patch to support a first joystick with more than 2 axis. Did the job to create scenery based on VMap0 data.

Andy Ross
Contributed a new configurable FDM called ***YASim*** (Yet Another Flight Dynamics Simulator, based on geometry information rather than aerodynamic coefficients.

Paul Schlyter
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://www.welcome.to/pausch/

Chris Schoeneman
Contributed ideas on audio support.

Phil Schubert
Contributed various textures and engine modeling.

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

Jonathan R. Shewchuk
Author of the Triangle program. Triangle is used to calculate the Delauney triangulation of our irregular terrain.

Gordan Sikic
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
Contributed cockpit graphics, 3-D models, logos, and other images. Project Bonanza

Martin Spott
Co-Author of the "Getting Started".

Durk Talsma
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
Bipin Sehgal
Michael Selig

Moreover, those helped to support the effort:
Jay Thomas
Eunice Lee
Elizabeth Rendon
Sudhi Uppuluri

U. S. Geological Survey

Provided geographic data used by this project.

http://edc.usgs.gov/geodata/

Mark Vallevand
Contributed some METAR parsing code and some win32 screen printing routines.

Gary R. Van Sickle
Contributed some initial GameGLUT support and other fixes. Has done preliminary work on a binary file format. Check

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

His Ćygwin Tipsṕage might be helpful for you at

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

Norman Vine
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 ǵoto airportánd ŕesetf́unctions and the initial http image server code

Roland Voegtli
Contributed great photorealistic textures. Founder of European Scenery Project for X-Plane:

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

Carmelo Volpe
Porting ***FlightGear*** to the Metro Works development environment (PC/Mac).

Darrell Walisser
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
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://williams.best.vwh.net/.

Jim Wilson
Wrote a major overhaul of the viewer code to make it more flexible and modular. Contributed many small fixes and bug reports. Contributed to the PUI property browser and to the autopilot.

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

http://www.equi4.com/metakit/

Woodsoup Project

While ***FlightGear*** no longer uses Woodsoup servies we appreciate the support provided 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.
http://www.woodsoup.org/

Robert Allan Zeh
Helped tremendously in figuring out the Cygnus Win32 compiler and how to link with ḋll's. Without him the first run-able Win32 version of ***FlightGear*** would have been impossible.

D.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, buildings, and more of those things bringing scenery to real life and belonging to realistic airports and cities. 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), Kai Troester (compile problems), Dave Perry (joystick support), and Michael Selig (UIUC models).

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, and Julian Foad.

Index