UPDRAFT

THE PROGRAMMERS DOCUMENTATION

Team Leader :

Tomas Zamecnik[1]

Team Members :

Cestmir Houska

Jakub Marek

Bohdan Maslowski

Maria Vamosova

Ales Zita

Project Supervisor :

Alexander Wilkie[2]

Faculty of Mathematics and Physics,

Charles University,

Prague,

Czech Republic

07-May-12

[1] pulcik@email.cz

[2] wilkie@cgg.mff.cuni.cz

Table of content

1 INTRODUCTION

1.1 GLIDE PATH VISUALISATION AND PLANNING

There is more to gliding then just to fly the airplane. For instance every competition flight needs to be planned in advance as well as analysed after the flight is done.

The post-flight analysis or so-called de-briefings, is the process during the pilot takes the stored flight path recordings and scrutinises the data or re-plays the flight he made to further analyse the flight or learn from it. As for planning of the competition flight there are sets pre-defined *turn-points* from which usually the three turn-points are chosen to form a competition triangle. Every turn-point is defined by its world coordinates. During the competition flight, the pilot needs to navigate his airplane through the defined set of points in pre-defined order for his flight to be valid. Although this planning can be done by-hand, the use of computer program is very convenient. Therefore a need of software that can be used for these tasks has arisen. To our knowledge there are few free and commercial solutions on the market. However the free software doesn't match the quality of the commercial applications and the commercial applications are expensive. One of the widely spread commercial software for this purposes at the present time is a application called *SeeYou* [1] created by the *Naviter* company, which we use as an inspiration.

Our aim is to create similar (in functionality) open-source product under the GNU license and therefore made this product available to wider glider pilot communities. We try to improve on some of the SeeYou features, which we believe can be done in more user friendly manner. We would not like our product to be compared to SeeYou, rather taken as a free-of-charge alternative for non-heavy users. To achieve this uneasy task we setup the group of skilled programmers and flight enthusiasts, who has spent many nights and day to make this achievable. We call our project **The Updraft**[3].

1.2 THE UPDRAFT PROJECT - OPRAVIT

to create such complex software, which could be used as a previously mentioned glider pilot tool and all the basic procedures can be done. The tools we want to be contained within the final product are :

- flight planning (called task declaration)
- airspace visualisation
- turnpoints and airfields visualisation
- flight path visualisation with colour-coded information about
    - speed
    - vertical speed
    - altitude
    - airspeed

[3] http://updraft.github.com/

We call this project **The Updraft**.

2 THE UPDRAFT PROJECT

2.1 THE TEAM

This subject was originally proposed by our Project Leader *Tomas Zamecnik*. He, as an enthusiastic glider pilot, has pointed out the lack of non-commercial glider-path planning and visualisation software. Group of people fond of flying was established with *Tomas Zamecnik* as a project leader being the glider pilot himself and a group of people being fond of flying. Finally the team consists of 6 people.

Team Members :

- **Tomas Zamecnik**
- **Cestmir Houska**
- **Jakub Marek**
- **Bohdan Maslowski**
- **Maria Vamosova**
- **Ales Zita**

See the "*Appendix A – People*" for further details.

Programming of the Updraft project was divided into several partially separable parts with each of the part dedicated to one team member. Obviously it was impossible to split the workload evenly, so most of the people ended up working on parts of the project together and tight cooperation between team members proved crucial. However there always was a tendency for having one responsible person for each of the project component. The table below shows every member main responsibilities. (The work was well coordinated and the Meetings were held on a week basis as well as cases of team brain-stormings. Edit.) For e-mail communication we used services of *freelists.org* [2] for its accessibility and transparency.

Developer	Updraft component	Main responsibilities
Tomas Zamecnik	Turnpoints plugin, TaskDecl plugin	Leadership and management, Turnpoints plugin, User documentation, TaskDecl plugin, Importing files
Cesmir Houska	Core, GUI	Updraft core, Settings, TaskDecl. GUI, Mouse picking
Jakub Marek	IGC plugin, Core	Core, IGC visualisation, IGC parsing, on-demand rendering (not yet finished), Translations (not yet finished), Bottom panel interface, OpenFile dialog
Bohdan Maslowski	GUI	Mac at all, Installation for all platforms, 2D Map mode
Maria Vamosova	IGC plugin, GUI	OSG Earth implementation, Map layers visualisation, IGC graph drawing, IGC statistics, Camera manipulation, Map Layer groups + left pane interface, Scene manager
Ales Zita	Airspaces plugin, documentation	Airspace plugin, documentation, turnpoint labels, airfields visualisation

Table 1 : Team members main responsibilities.

2.2 TECHNICAL DETAILS OF DEVELOPMENT

The programming language we choose for the Updraft implementation was C++ for both the object oriented characteristic of the language and overall advancement of this language's features and possibilities. To keep the complexity of the language on a manageable level, we decided to use a Google C++ style guide [3]. Google C++ style guide is one of the widely used open-source set of suggestions and recommendation on how to maintain the readability of the code. The style checker was incorporated to the build of the project to ensure the rules would be followed.

On top of the main programming language we decided to use Nokia Qt framework [4] for its multiplatform user interface capabilities. Nokia Qt is a development framework used for creation of applications for many platforms including Windows, Linux, Mac OS, Symbian and many others. One of the goals we set was to have Updraft usable under not only Windows, but under MacOS and Linux operating systems as well, because these are 3 most common operating systems nowadays.

The program was developed on Windows and Linux platforms and tested on Mac. As a development environment we used the Microsoft Visual Studio 2008 [5] under student license on Windows and ...... on Linux.

CMake [6] was used as a cross-platform project build system.

2.3 Repository structure

We used the *Git* [7] version control software and hosted the source code on the *GitHub* [8] web hosting service. We choose GitHub offers free accounts and storing space for open-source project, such as ours, and for very convenient possibility of creating the web page for our product. The web-page address of our product can be found on http://updraft.github.com. The project repository is accessible via internet on https://github.com/updraft. The repository structure of our project is as follows:

- updraft.github.com - Project web page files.
- updraft - Project main repository folder.
  - Docs - Documentation related files
  - Experiments - Project experiments with 3rd part SW installation.
  - SourcesInstallation - Test files for source code installation.
  - Updraft - Source code repository.

2.4 Project specification

2.4.1 Environment

The goal of this endeavour is to create multiplatform desktop application capable of serving as a glider pilot tool for flight planning and post-flight visualisation. The target platforms are Linux, Microsoft Windows or Mac OS. Connection to the internet is not required, but enables some of the functionality.

2.4.2 Supported languages

Application is distributed in Czech and English localisations.

2.4.3 Negative specification

The software is not intended to be used by competition directors or referees or for judging the achievements of any official goals. Application will not be focused on detailed flight planning like calculating optimal speed, water ballast, final glide etc. The application will not run on mobile devices and will not serve as an on-board computer.

2.4.4 List of features

The features include the visualisation of 3D maps including the terrain height, visualisation of flight relevant data, such as airspace divisions, turn-points, airfields, etc., next to the flight planning and visualisations of recorded IGC files.

The tools we contained within the final product are :

- flight planning (called task declaration)
- airspace visualisation
- turnpoints and airfields visualisation
- flight path visualisation with colour-coded information about
  - speed
  - vertical speed
  - altitude
  - airspeed

2.4.5 GUI PRINCIPLES

Most of the action is done by mouse.

3 THIRD PARTY SOFTWARE

As this is an uneasy task, we made our work a little bit easier by utilizing several 3rd party frameworks, which help us to create the application. These are :

1. *Nokia Qt* [4] – the multiplatform User Interface
2. *OSG* [9] (the open scene graph) – as a OpenGL interface for 3D geometry visualization
3. *OSG Earth* [10] – as a tool for the world map visualization

3.1 NOKIA QT

Nokia Qt [4] is a cross-platform development framework used for creation of applications with graphical user interface (GUI) for many of the operating systems including Windows, Linux, Mac OS, Symbian and others. Nokia Qt is a free and open source software distributed under the terms of the GNU Lesser General Public License. As it has been always our goal for Updraft to be multi platform application, we decided to use the Nokia Qt framework as it ideal for this task. Moreover, the Nokia Qt is known to be able to incorporate the OSG efficiently. Nokia Qt uses standard C++ but makes extensive use of special code generator called the *Meta Object Compiler*, or *moc* together with several macros to enrich the language. [11]

3.2 OSG

The *OpenSceneGraph* [9] is an open source cross-platform 3D graphics application programming interface used in fields such as visual simulation, games, virtual reality, scientific visualisations and modeling. The OSG is distributed under OpenSceneGraph Public License based on LGPL licence. It uses the *OpenGL* [12] for 3D graphics acceleration interface. We use the OSG as a base for the OSGEarth framework, which we decided to use for the worldwide map visualisation.

3.3 OSG EARTH

The *OSGEarth* [10] is a terrain rendering toolkit written in C++ and distributed under LGPL license. It is developed by Jason Beverage and Glenn Waldron as a visualisation toolkit for the map, elevation and vector data. OSGEarth utilises the OSG framework as the 3D graphics interface.

Figure 1. The OSGEarth framework example.

4 THE THIRD PARTY DATA

4.1 INTRODUCTION

So far we described the environment we chose for our project development. OSGEarth is a superb framework for visualising the map imagery and elevation data of various sorts. Now for data we decided to use the online/caching strategy. It means that we use the data sets, which are available online for reasons of minimal necessary installation packaging, but cache the data we once downloaded from the internet source to limit the bandwidth requirements to minimum level. In this way the application is capable of visualising the data from all over the world without the necessity of all the data being stored locally and distributed with the application. Note that these data packages are enormous in size for obvious reasons. Therefore the internet connection during the application run is highly advisable but not essential.

Two types of data sources are necessary for visually pleasing display of the world, the map imagery data and the elevation data. There are several free data sources available online for the map imagery, but not so much for the elevation data.

4.2 MAP IMAGERY SOURCES

In order to display the world map we need to use a free database of map imagery data. This data set needs tom bee in one of the format supported by the OSGEarth framework. The complete list of OSGEarth drivers capable of process particular data sources are listed on the OSGEarth web page[4]. The configuration of the particular data sources is written in the `initial.earth` file stored in `/bin/data` folder.

In our case we wanted to have at least two basic views of the virtual world. We pick the sources in following manner: the first one to be very simple for the sake of the visualisation clarity and the second one to be the “realistic” view, i.e. the photomaps with the elevation displayed as a terrain displacement.

Map imagery data sources:

- OpenStreet map [13] - free worldwide map tiles database hosted on under open licence
- ArcGIS online map service [14]- free worldwide satellite tiles imagery database

[4] http://osgearth.org/wiki/TileSourcePlugins

Figure 2. OpenStreet map source.

Figure 3. ArcGIS online map service.

4.3 Elevation data source

As for the elevation (so-called heightfield) data source we use the Ready Map elevation data source [15]. For this data source is designated for development purposes only, we asked the author for permission to use this data set for our project. The permission was granted and the corresponding e-mail is attached to this document. See Appendix C – Elevation dataset usage confirmation.

5 PROGRAM ARCHITECTURE

The application is designed as a core/plug-ins architecture with the core being responsible for user interface, map visualization and plug-in handling. The plug-ins are linked as a dynamic libraries to the main project and forms the main features functionalities. Plug-ins are capable of drawing to the map layer, creating plug-in related menus, mouse click actions, etc.

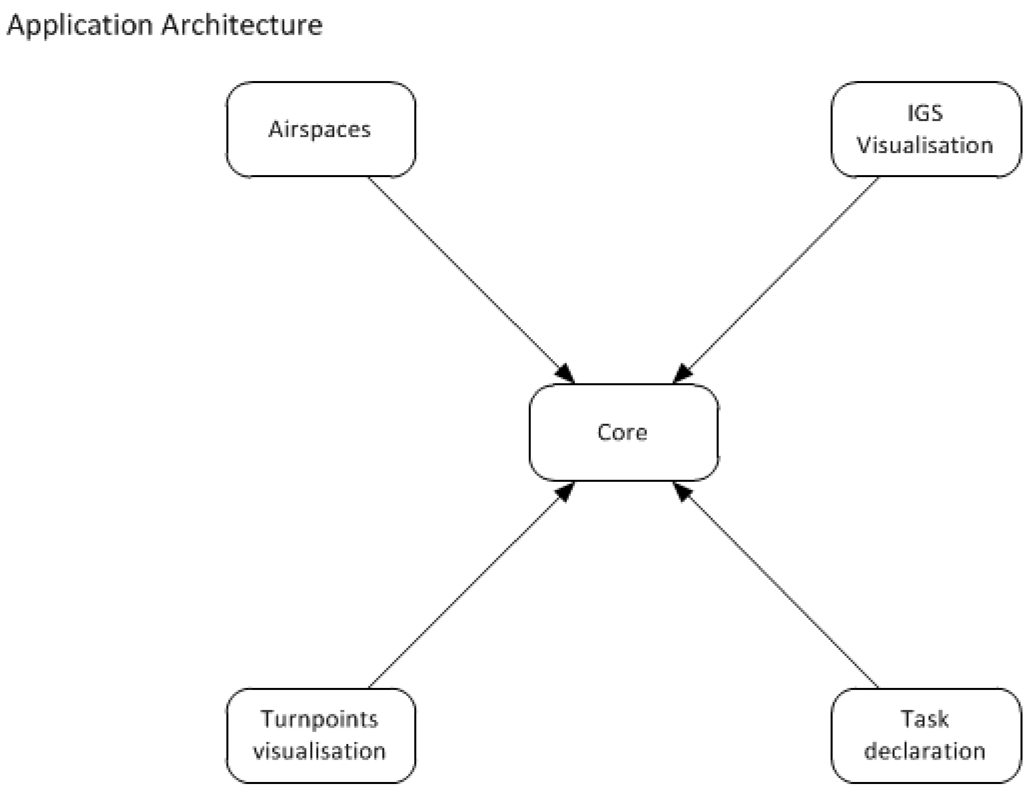

Figure 4. Application architecture diagram.

6 THE CORE

6.1.1 PLUG-IN INTERFACE

Updraft uses Qt's plug-in system to load plug-ins at runtime.

PluginBase

Plug-ins in updraft are derived from *PluginBase* class. This class has methods for plug-in identification (*getName*), initialisation and handling of call-backs from core.

PluginBase class is declared as a Qt plug-in interface using *Q_DECLARE_INTERFACE*.

CoreInterface

Interface *CoreInterface* enables the plug-ins to communicate with *Core*. It is passed to the plug-in during its initialisation phase (every plug-in gets its own instance) and it contains pointer to the plug-in. This makes it possible to attribute all calls to the source plug-in (although this option is currently not used).

CoreInterface is implemented by a core class *CoreImplementation*, however this class contains only thin wrapper around various parts of the high level managers.

PluginManager

On the side of Updraft Core, the class *PluginManager* takes care of loading plug-ins. The public interface of plug-in manager is fairly limited, only listing all loaded plug-ins and retrieving plug-in instance or plug-in directory by name.

7 THE PLUGINS

7.1 INTRODUCTION

7.1.1 PLUG-IN CREATION

A new plug-in must contain the main class inheriting from *PluginBase* and *QObject*. Additionally, the class must be marked as *Q_DECL_EXPORT* and *Q_INTERFACES*(Updraft::PluginBase), and it's implementation must contain *Q_EXPORT_PLUGIN2*(pluginName, PluginClass) outside all functions.

As a convention, we manually add a global variable *g_core* to all plug-ins to simplify access to the core interface in other classes of the plug-in.

7.1.2 MINIMAL PLUG-IN EXAMPLE

This example shows the mandatory steps to be taken for succesfull plug-in creation.

In file `/Updraft/src/plugins/foo/fooplugin.h`

```
#include "../../pluginbase.h"

class Q_DECL_EXPORT FooPlugin
  : public Updraft::PluginBase, public QObject {
  Q_OBJECT
  Q_INTERFACES(Updraft::PluginBase)

 public:
  QString getName();
  void initialize(Updraft::CoreInterface *coreInterface);
  void deinitialize();
};
```

In file `/Updraft/src/plugins/foo/fooplugin.cpp`

```
#include "fooplugin.h"

Updraft::CoreInterface *g_core = NULL;

QString FooPlugin::getName() {
  return "foo";
}

QString FooPlugin::initialize(Updraft::CoreInterface
*coreInterface) {
  g_core = coreInterface;
}
```

```
QString FooPlugin::deinitialize() {}

Q_EXPORT_PLUGIN2(foo, FooPlugin)
```

In file `/Updraft/src/plugins/foo/CMakeLists.txt`

```
cmake_minimum_required(VERSION 2.8)
PLUGIN_BUILD(foo)
```

7.2 AIRSPACES

7.2.1 INTRODUCTION

One the most important information for a pilot planning his flight is to know where he is actually allowed to fly. The air space is divided into several sectors defined by its shape and a top and bottom altitudes. Each such a defined part of the air space belongs to a particular class, which will tell the pilot more about its availability. There are for example military airspaces, no fly zones or restricted areas, the pilot must be aware of, otherwise risking high fine or even worse kind of countermeasures. This type of awareness is vital for every pilot flying. As these information are so crucial, the pilot usually uses the approved maps, which are costly, and cannot rely on open-source data. On the other hand, if the pilot wishes to use the visualisation tool to check the proximity of any airspace to his flight path after the flight is done, the open-source data are usually sufficient approximation of the reality. These may not replace the approved maps but can rather serve as an eye-candy in post flight visualisation. The files containing such a data are commonly accessible via the Internet. Therefore we like to provide the user with a tool for visualisation of these data. As the most common format used in Europe is the OpenAir(tm)[5] text format, we decided to incorporate the OpenAir(tm) airspace definition format parser into the application as well as the visualisation algorithm to draw these information on the map surface. For more detail please see the Appendix B – OpenAir(tm) airspace format definition.

[5] http://www.winpilot.com/UsersGuide/UserAirspace.asp

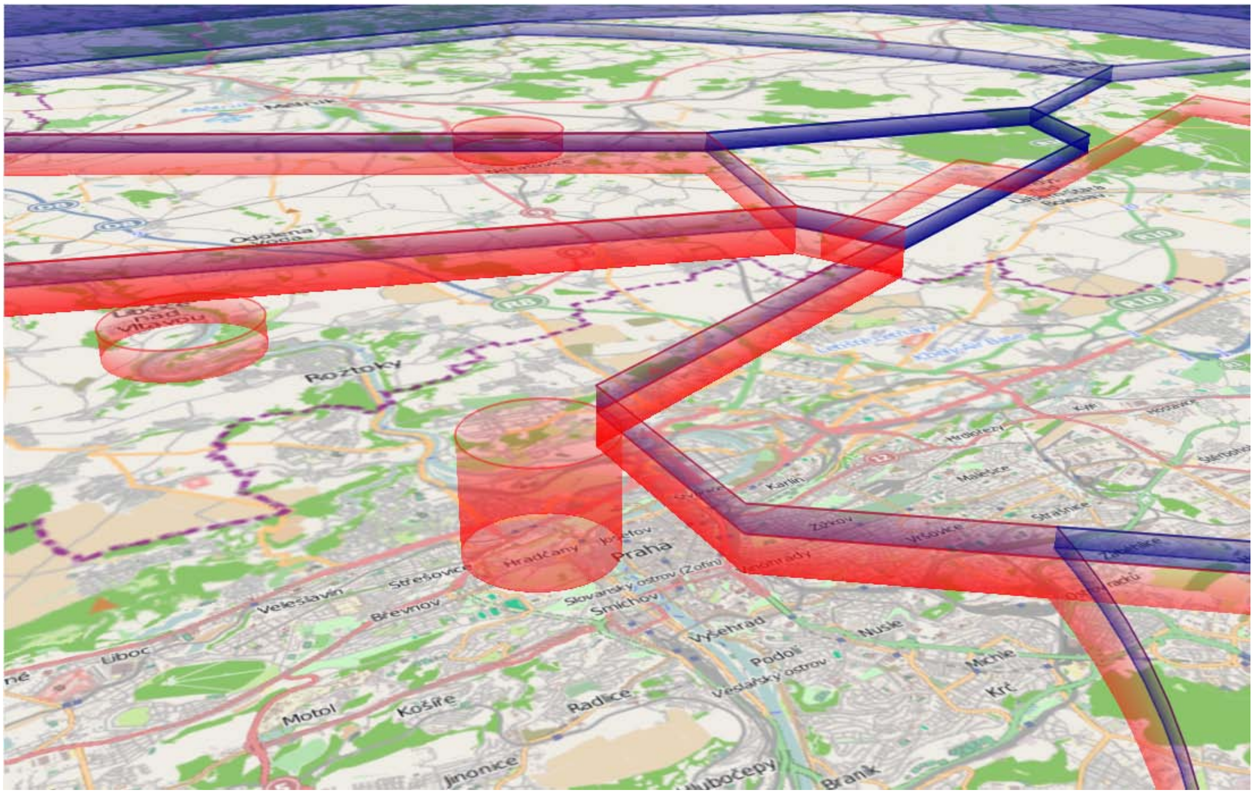

Figure 5. Example of airspace divisions visualisation.

7.2.2 OVERVIEW

The main role of this plug-in is to parse an airspace definition data from the provided file and visualize them.
The basic and in the time of the first program release the only format is the OpenAir(tm) format (see Appendix B – OpenAir(tm) airspace format definition).

7.2.3 DESIGN

The program is divided in two main parts of the airspace processing. The first part consists of the routines which load and parse the text input file containing the provided airspace information. This information is then handed over to the second part of the algorithm, which will draw the defined airspaces into the map layer. We call the first part the *Parser* and the second part the *Airspaces* plug-in. The plug-in is loaded by the core as a part of the application, whereas the Parser is a library called from within the plug-in otherwise forming independent program. A part of the plug-in is the format-specific drawing engine which is called by the plug-in to draw to a particular map layer. The engine is provided with the name of the file the user wishes to process by the *Core* and calls the parser to load the airspaces which the file contains.

The *Parser* as well as the *Airspaces* library is linked to the main project as a dynamic library.

7.2.4 *OpenAir(tm) parser*

This airspace declaration file is in text format and can contain information about several airspaces usually grouped into one logical or geographical group. Within the file the class, name, altitudes and other information is provided for each of the defined airspace. The airspace itself is defined either as a set of points and arcs forming a polygon or as a set of primitives (i.e. circles). During the start phase of the program, the plug-in parses the information from all the files which are saved in */data/airspaces* folder. Each file typically containing several airspaces within one region (e.g. country, city, etc.).

During the parsing phase of the process the file is opened for reading and the parsing process enters the main parsing loop, which ends when all the file content is processed or if the unexpected error occurs, e.g. the file is of incorrect format. In this loop one by one airspace class member is defined and filled with the data in file. This process is done within the particular Airspace class constructor. As stated above, the airspace is defined by the geometric primitives, such as arcs and points forming a closed polygon or a circle. The special data structures for these primitives were defined in the *geometry.cpp* and *geometry.h* files. These are the *Polygon, ArcI, ArcII* and *Circle* classes.

The parsing process itself is quite straight forward. The only tricky factoid of which the programmer must be aware of is that some of the variables the airspace definition file contains are state variables. That means that the validity of such a variable reaches the beyond the currently parsed airspace hence is valid until the end of the file or until re-defined. One example of such a variable are the pen and brush definitions, which are often defined in the beginning of the airspace definition file only and are valid for all the airspaces contained in the file. Another example could be the definition of the centre of co-centric airspaces. These are typically the airspaces surrounding one airport, therefore its centre is defined once per airport.

The Parser library is contained in following files located in `/Updraft/src/libraries/openairspace/` folder.

a. *openairspace.cpp* – the OpenAir(tm) format parser main procedures
b. *openairspace.*h – the OpenAir(tm) parser class declaration and interface
c. *airspace.cpp* – the single airspace class
d. *airspace.*h – the single airspace class declaration and interface
e. *openairspace_global.h* – the parser library main interface
f. *geometry.cpp* – the airspace geometry primitives class definitions
g. *geometry.h* – the airspace geometry primitives class declarations

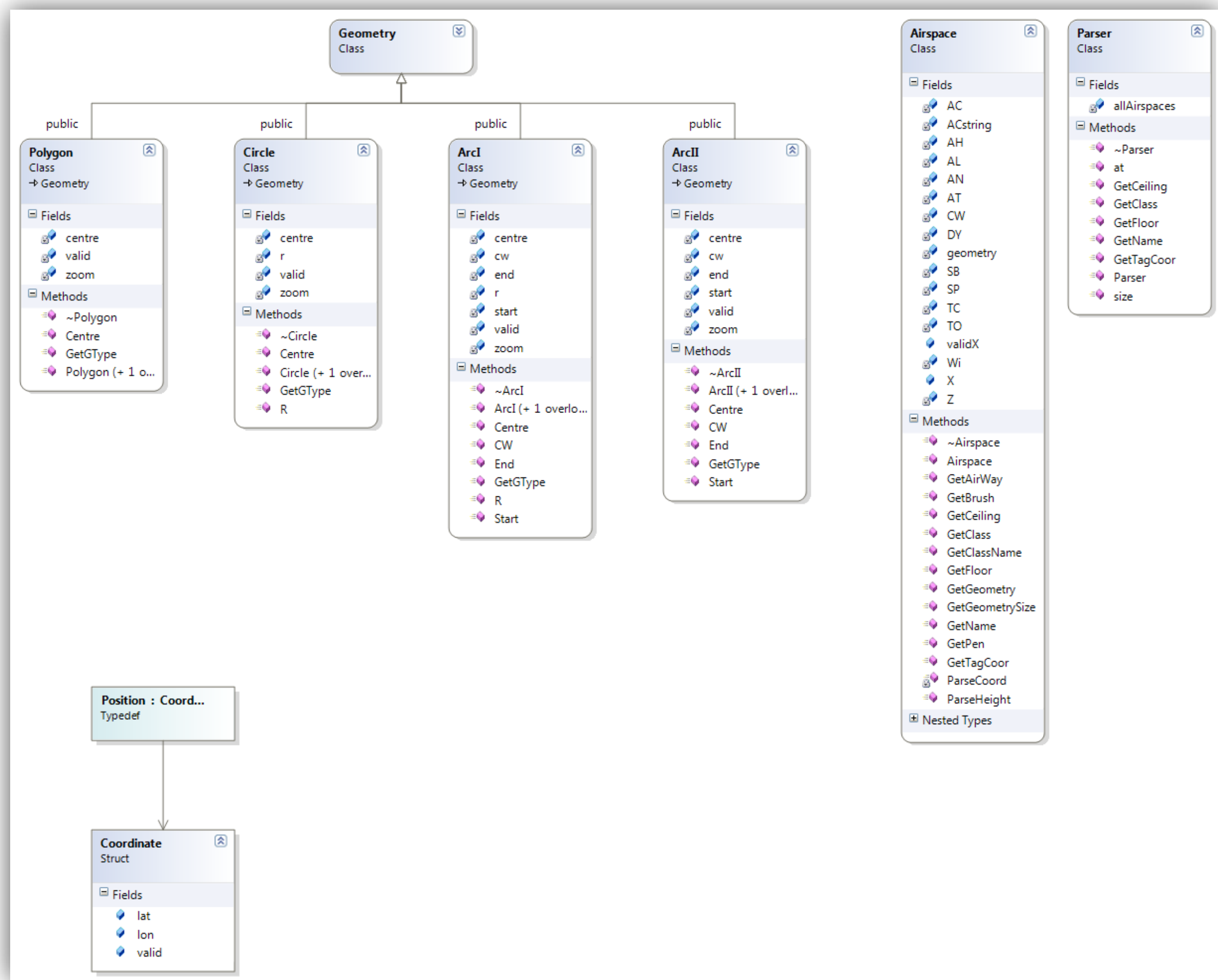

Figure 6 : This graphic shows the classes of the OpenAir(tm) airspace file parser. Each cell shows the variables and method of the particular class.

Output of the parser is the member of *OpenAirspace::Parser* class containing information about all the airspaces described in particular file. The only private variable of this class is the *allAirspaces* array of *OpenAirspace::Airspace* members. Each such a member contains parsed information about one airspace. This concludes the pen and brush colour, name, class, ceiling, floor, geometric primitives forming the airspace boundaries and more.

The OpenAir(tm) *Parser* library is linked to the main project as a dynamic library.

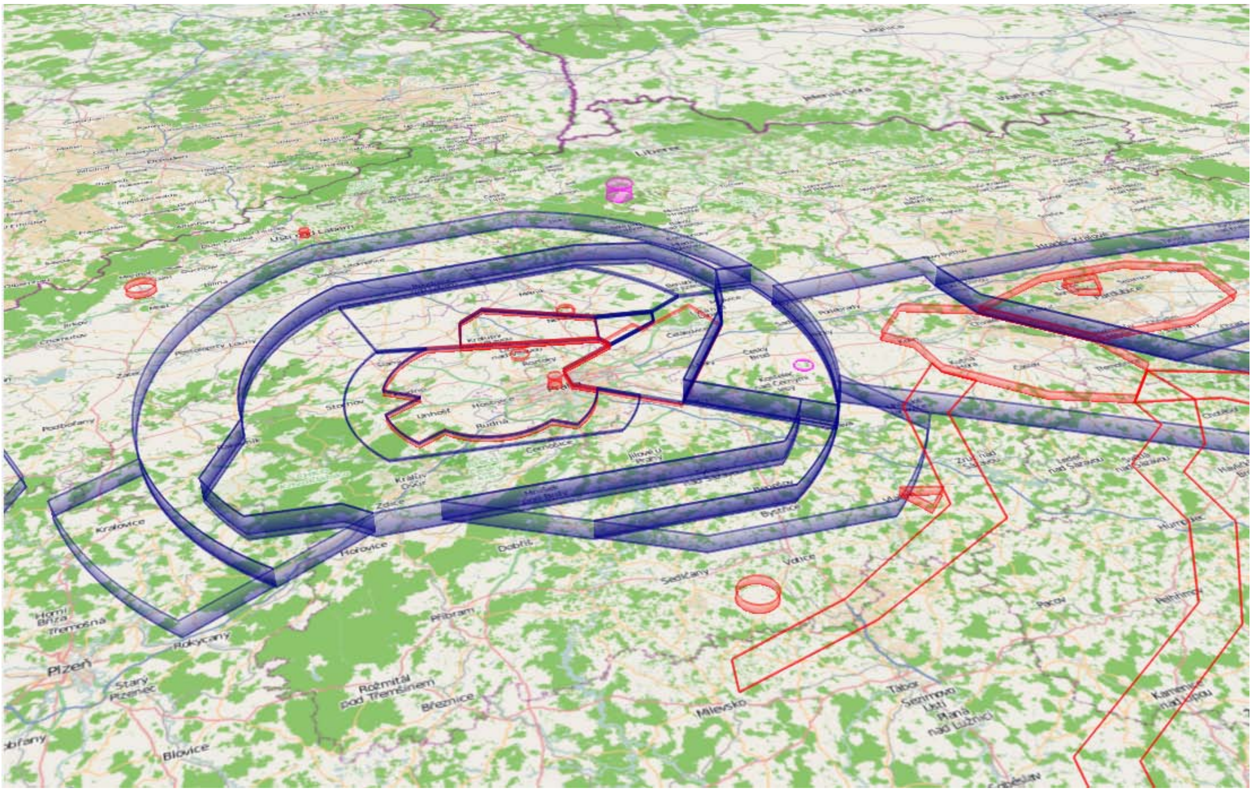

Figure 7. Example of the OpenAir(tm) data visualisations.

7.2.5 OpenAir(tm) drawing engine

The main class connected to the *Core* is the *Airspaces* class. This class is responsible for the communication with the core and for calling the airspace drawing engine and parser. In the initial phase of the plug-in load the *Airspaces* all the files in */data/airspaces* folder are processed. During the program run also the user-imported files are drawn and the file is stored in */data/airspaces* directory. Each of the files, which needs to be processed is given to the drawing engine, specifically the *oaEngine* class constructor. This method is then responsible for calling the parser.

As mentioned above the output of the parser is the member of *OpenAirspace::Parser* class containing information about all the airspaces described in particular file. This member contains in fact an array of parsed airspaces for the given region and is the main outcome of the parsing process. Each of the airspaces in array is then processed in such a way, that first the colour specification is read (this is often common for whole the file or at least for group of airspaces) or the default colour is assigned. Next the geometry is created for each of the airspace's set of points and arcs forming a closed polygon or a circle. The final geometry which is to be drawn consists of bottom polygon and the top polygon. Both the polygons being topologically identical each of them is located at height assigned as a floor and ceiling of the airspace respectively. Data defining the polygons were parsed during the process described above. The array of point pairs (bottom polygon point and corresponding top polygon point) defines the triangle strip OpenGL primitive used to visualize the side of the airspace. As mentioned earlier the OpenAir(tm) format defines the airspace as a circle primitive or as a closed polygon consisting of a set of points and arcs. OpenGL is based on processing the triangle meshes defined as a set of vertices for performance reasons. Thus all the arcs, as well as circles declared must be converted to a set of points allowing the use of OpenGL mesh setup. Granularity of this conversion is defined as a constant within *oaEngine* class. Because all the data defining the airspace are in WGS84 [16] format, many arbitrary algorithms which work with this coordinate system had to be introduced. The main aim of these procedures is to compute the set of points from given airspace geometry. These points can be then used for filling the vertex array of the OpenGL utilising the methods of OSG Earth. These are used to correctly transform the WGS84 coordinates of vertices to the OpenGL world view and for correct evaluation of the terrain height from the height field data provided by OSG Earth.

Such a filled OpenGL vertex array is added to a scene graph node as a set geometry. Each node is now defining one particular airspace. Finally the set of OpenGL switches is used to ensure correct evaluation of transparency effects and visibility of the airspaces. The nodes are accumulated in map nodes array, which is handed back to the *Airspaces* class.

Each of the node in the array is added by *Airspaces* class to the scene tree of OSG as a map layer. In the same time the GUI display tree is created and signals for displaying the defined layers connected.

The Parser library is contained in following files located in `/Updraft/src/plugins/airspaces/` folder.

a. *airspaces.cpp* - the main interface of the plug-in
b. *airspaces.h* - the airspaces plug-in interface
c. *oaengine.cpp* - the main airspace drawing engine
d. *oaengine.h* - the OpenAir(tm) drawing engine class and methods declarations

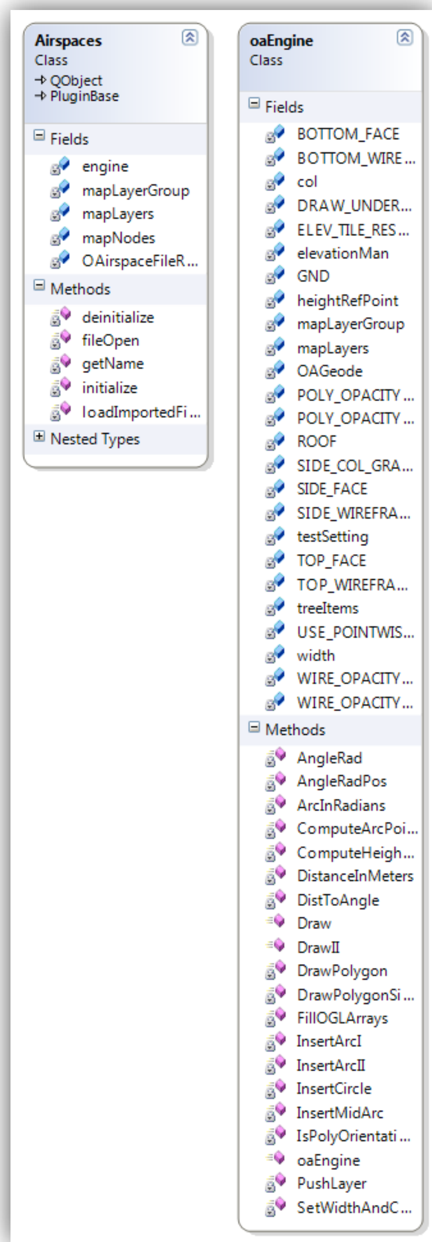

Figure 8 : Airspaces plug-in classes. In each of the cells the variables and methods of the particular class are stated.

The *Airspaces* plug-in library is linked to the main project as a dynamic library.

7.2.6 3RD PARTY SOFTWARE USED

- Nokia Qt [4] for the GUI framework
- OSG [9] – as a OpenGL framework, for the mesh visualisation
- OSG Earth [10] – for the world-view matrix transformations

7.2.7 The Airspaces plug-in diagram

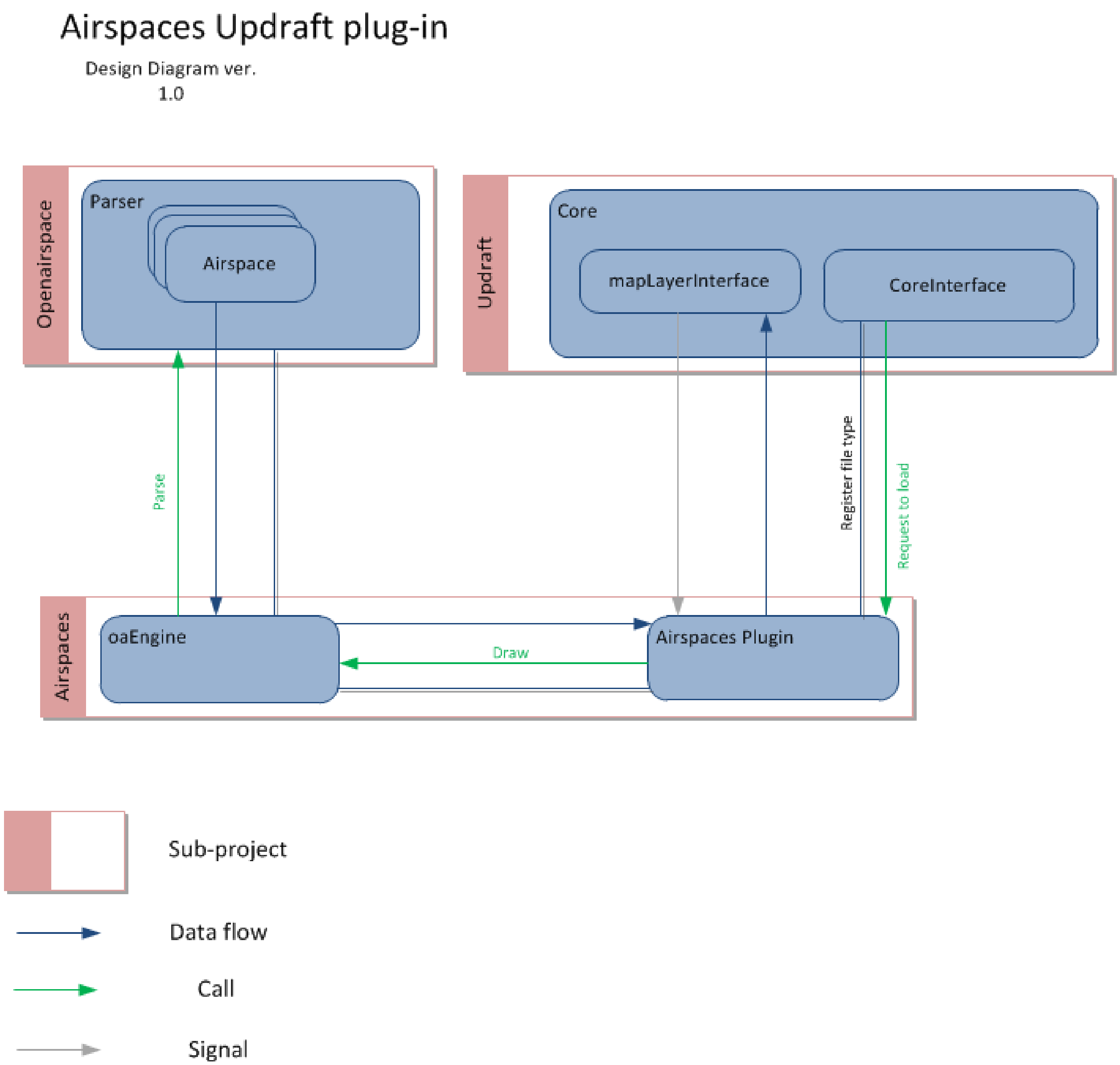

Figure 9 : Airspaces plug-in and OpenAir parser diagram.

7.3 TurnPoints

7.4 Task Declation

7.5 IGC Visualisation

8 PROGRAM INSTALLATION

8.1 WINDOWS

8.2 LINUX

8.3 MAC

9 Conclusion

We have created this application to serve the glider pilots as a glider path planning and flight visualisation tool. It is meant to be a free easy-to use alternative to the commercial product SeeYou.

10 Future work

11 REFERENCES

[1] Naviter, "SeeYou," Naviter, [Online]. Available: http://naviter.si/index.php?option=com_content&task=view&id=9&Itemid=213.

[2] "FreeLists," [Online]. Available: http://www.freelists.org.

[3] Google, "Google C++ Style Guide," [Online]. Available: http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml.

[4] Nokia, "Qt," Nokia, [Online]. Available: http://qt.nokia.com/.

[5] Microsoft, "Visual Studio 2008," Microsoft, [Online]. Available: http://www.microsoft.com/visualstudio/en-us/products/2008-editions.

[6] Open-source, "CMake," [Online]. Available: http://www.cmake.org/.

[7] L. Torvalds. [Online]. Available: http://git-scm.com/.

[8] "GitHub," [Online]. Available: http://www.github.com.

[9] Open-source, "The OpenSceneGraph," [Online]. Available: http://www.openscenegraph.org.

[10] G. W. Jason Beverage, "OSG Earth," Pelican Mapping, [Online]. Available: http://osgearth.org.

[11] Wikipedia, "Qt framework," [Online]. Available: http://en.wikipedia.org/wiki/Qt_(framework).

[12] K. Gtoup, "OpenGL," [Online]. Available: http://www.opengl.org/.

[13] Open-source, "OpenStreetMap," [Online]. Available: http://tile.openstreetmap.org/.

[14] ESRI, "ESRI Imagery World 2D," [Online]. Available: http://server.arcgisonline.com/ArcGIS/rest/services/ESRI_Imagery_World_2D/MapServer.

[15] G. W. Jason Beverage, "Ready map elevation tiles," Pelican Mapping, [Online]. Available: http://readymap.org/readymap/tiles/1.0.0/9/.

[16] Wikipedia, "World Geodetic System," [Online]. Available: http://en.wikipedia.org/wiki/World_Geodetic_System.

12 TABLES

13 FIGURES

14 APPENDIX A – PEOPLE

Alexander Wilkie – Computer Graphics Senior Lecturer on Faculty of Mathematics and Physics by Charles University, Prague, Czech Republic; email:

Tomas Zamecnik – Student of Computer Science master degree on Faculty of Mathematics and Physics by Charles University, Prague, Czech Republic; email:

Cestmir Houska – Student of Computer Science master degree on Faculty of Mathematics and Physics by Charles University, Prague, Czech Republic; email:

Jakub Marek – Student of Computer Science master degree on Faculty of Mathematics and Physics by Charles University, Prague, Czech Republic; email:

Bohdan Maslowski – Student of Computer Science master degree on Faculty of Mathematics and Physics by Charles University, Prague, Czech Republic; email:

***Maria Vamosova* -** Student of Computer Science master degree on Faculty of Mathematics and Physics by Charles University, Prague, Czech Republic; email:

Ales Zita – Student of Computer Science master degree on Faculty of Mathematics and Physics by Charles University, Prague, Czech Republic; email:

15 APPENDIX B – OPENAIR(TM) AIRSPACE FORMAT DEFINITION

Following is the definition of the OpenAir(tm) format as provided on the winpilot[6] web page.

OPEN AIR (tm) TERRAIN and AIRSPACE DESCRIPTION LANGUAGE
Version 1.0
December 10, 1998
Updated October 15, 1999
Send comments to jerryp@winpilot.com

AIRSPACE related record types:
==============================

AC class ; class = Airspace Class, see below:
R restricted
Q danger
P prohibited
A Class A
B Class B
C Class C
D Class D
GP glider prohibited
CTR CTR
W Wave Window

AN string ; string = Airspace Name
AH string ; string = Airspace Ceiling
AL string ; string = Airspace Floor
AT coordinate ; coordinate = Coordinate of where to place a name label on the map (optional)
; NOTE: there can be multiple AT records for a single airspace segment

TERRAIN related record types (WinPilot version 1.130 and newer):

[6] http://www.winpilot.com/UsersGuide/UserAirspace.asp

==============================

TO {string} ; Declares Terrain Open Polygon; string = name (optional)
TC {string} ; Declares Terrain Closed Polygon; string = name (optional)
SP style, width, red, green, blue ; Selects Pen to be used in drawing
SB red, green, blue ; Selects Brush to be used in drawing

Record types common to both TERRAIN and AIRSPACE
===

V x=n ; Variable assignment.
; Currently the following variables are supported:
; D={+|-} sets direction for: DA and DB records
; '-' means counterclockwise direction; '+' is the default
; automatically reset to '+' at the begining of new airspace segment
; X=coordinate : sets the center for the following records: DA, DB, and DC
; W=number : sets the width of an airway in nm (NYI)
; Z=number : sets zoom level at which the element becomes visible (WP version 1.130 and newer)

DP coordinate ; add polygon pointC
DA radius, angleStart, angleEnd ; add an arc, angles in degrees, radius in nm (set center using V X=...)
DB coordinate1, coordinate2 ; add an arc, from coordinate1 to coordinate2 (set center using V X=...)
DC radius ; draw a circle (center taken from the previous V X=... record, radius in nm
DY coordinate ; add a segment of an airway (NYI)

16 Appendix C – Elevation dataset usage confirmation

Od: "Glenn Waldron" <gwaldron@pelicanmapping.com>
Komu: "Ales Zita" <ales.zita@volny.cz>
Předmět: Re: ReadyMap elevation data
Datum: 10.12.2011 - 15:53:03

Ales,

You are welcome to use the server for your project and for a
freeware app.
Thanks for asking, and if you have any issues or questions, don't
hesitate
to email me.

Glenn Waldron / Pelican Mapping / @glennwaldron

On Sat, Dec 10, 2011 at 5:57 AM, Ales Zita <ales.zita@volny.cz>
wrote:

> From: Ales Zita <ales.zita@volny.cz>
> Subject: ReadyMap elevation data
>
> Message Body:
> Hello,
>
> I'd like to ask for permission to use the world-wide elevation
> data for
> > our school project.
>
> We are a group of 6 people developing the the glider flightpath
> visualisation tool in osgEarth environment as a school project at
> Faculty
> > of Mathematics and Physics by the Charles University in Prague.
> As this is non commertial project, we cannot afford any paid
> elevation
> > data sources. Therefore we would very much appreciate the use of
> your
> > ReadyMap global elevation dataset accessible via the osgEarth api.
>
> Should this project become available as a freeware at some point,
> can we
> > continue to use your server as a souce of the elevation data?
>
> Thank you
>
> Kind regards
>
> Ales Zita

>
> --
> This mail is sent via contact form on Pelican Mapping
> http://pelicanmapping.com

17 APPENDIX D – DVD CONTENT