CZECH TECHNICAL UNIVERSITY IN PRAGUE
Faculty of Nuclear Sciences and Physical Engineering

Development and refactoring of application for MRI data processing

Vývoj a refaktorizace aplikace pro zpracování dat z magnetické rezonance

Bachelor's Degree Project

Author:	**Roman Yaremchuk**
Supervisor:	**Ing. Tomáš Oberhuber, Ph.D.**
Language advisor:	**Mgr. Hana Čápová**
Academic year:	2020/2021

Katedra: matematiky | Akademický rok: 2020/2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student:	Roman Yaremchuk
Studijní program:	Aplikace přírodních věd
Studijní obor:	Aplikovaná informatika
Název práce (česky):	Vývoj a refaktorizace aplikace pro zpracování dat z magnetické rezonance
Název práce (anglicky):	Development and refactoring of application for MRI data processing

Pokyny pro vypracování:

1) Seznamte se s knihovnou Qt pro vývoj grafického uživatelského rozhraní.
2) Seznamte se s existující implementací aplikace Cameo pro zpracování tagovaných snímků z magnetické rezonance.
3) Popište její návrh, proveďte vhodnou refaktorizaci a odstraňte případné chyby v kódu.
4) Naimplementujte uživatelské rozhraní pro výpočet tenzorů deformací.
5) Napište stručný návod pro uživatele.

Doporučená literatura:

1) L. Z. Eng, Qt5 C++ GUI Programming Cookbook: Practical recipes for building cross-platform GUI applications, widgets, and animations with Qt 5 (2nd Edition). Packt Publishing, 2019.

2) S. C. Bushong, G. Clarke, Magnetic Resonance Imaging: Physical and Biological Principles (4th Edition). Mosby, 2014.

3) K. Naik, P. Tripathy, Software Testing and Quality Assurance: Theory and Practice (1st Edition). Wiley-Spektrum, 2008.

Jméno a pracoviště vedoucího bakalářské práce:

Ing. Tomáš Oberhuber, Ph.D.
KM FJFI ČVUT v Praze, Trojanova 339/13, 120 00 Praha 2

Jméno a pracoviště konzultanta:

Datum zadání bakalářské práce:	31.10.2020
Datum odevzdání bakalářské práce:	7.7.2021

Doba platnosti zadání je dva roky od data zadání.

V Praze dne 30.10.2020

[illegible]
....
garant oboru

[illegible]
....
vedoucí katedry

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta jaderná a fyzikálně inženýrská
-1-

[illegible]
....
děkan

Acknowledgment:
I would like to thank my supervisor Ing. Tomáš Oberhuber, Ph.D. for his expert guidance and express my gratitude to Mgr. Hana Čápová for her language assistance.

Author's declaration:
I declare that this Bachelor's Degree Project is entirely my own work and I have listed all the used sources in the bibliography.

Prague, July 7, 2021 Roman Yaremchuk

Název práce:

Vývoj a refaktorizace aplikace pro zpracování dat z magnetické rezonance

Autor: Roman Yaremchuk

Obor: Aplikovaná informatika

Druh práce: Bakalářská práce

Vedoucí práce: Ing. Tomáš Oberhuber, Ph.D., Katedra matematiky, FJFI, České vysoké učení technické v Praze, Trojanova 13, 120 00 Praha 2

Abstrakt: Hlavním cílem tohoto projektu je studium a vylepšení stávající implementace grafického uživatelského rozhraní programu Cameo. Po teoretickém seznámení se základy zobrazování magnetickou rezonancí je práce zaměřena na funkce knihovny Qt, kterou lze použít pro vytvoření grafického uživatelského rozhraní. Dále se práce zabývá strukturou zdrojového kódu programu Cameo a testováním GUI. Poté následuje uživatelská příručka, která popisuje, jak s programem pracovat.

Klíčová slova: magnetická rezonance, Qt, testování, zpracování medicínských dat.

Title:

Development and refactoring of application for MRI data processing

Author: Roman Yaremchuk

Abstract: The main goal of this project is to study and improve the existing graphical user interface implementation of the Cameo program. After theoretical introduction to the basics of the magnetic resonance imaging, the work is focused on the features of the Qt library, which can be used for creating the graphical user interface. Then the work deals with the structure of the source code of the Cameo program and GUI testing. Thereafter, the user guide follows that describes how to work with the program.

Key words: magnetic resonance, medical data processing, Qt, testing.

Contents

List of Abbreviations

- MRI - magnetic resonance imaging.
- DICOM Digital Imaging and Communications in Medicine).
- GUI - Graphical User Interface.
- ISO - International Organization for Standardization.
- CT - Computed tomography.
- CSS - Cascading Style Sheets.
- JSON - JavaScript Object Notation.
- PHP - Hypertext Preprocessor.
- QML - Qt Modeling Language.
- API - Application Programming Interface.
- Jpg - Joint Photographic Expert Group.
- Bmp - Bitmap.
- Png - Portable Network Graphics.

List of Figures

Introduction

There are various techniques of heart examinations. Some of them require surgical intervention and some do not. However, if there is any possibility to avoid the surgical intervention and get reliable information, this is always the best option. Almost all non-surgical techniques require using a special device. Different types of the devices can be used for different types of examination. Moreover, some devices can slightly harm human body. Nonetheless, possible problems with heart may be more dangerous.

Artificial intelligence, machine learning and computer vision are getting popular these days. These trends speed up a lot of different kinds of processes that can be extremely useful in medicine. Today there is a need to implement new algorithms and improve existing ones in order to process the data from the devices used for the heart examination.

Programs that are able to evaluate the condition of the human heart are very useful and reliable. In addition, the software is supposed to display and offer different kinds of the manipulation with the medical data, process the medical data, etc. The source of the data can be a special device that is used for scanning of the internal organs. Moreover, the device should be widely used so that the software can be more accessible because different devices may produce different data.

The magnetic resonance imaging is a powerful technique that allows scanning the human's internal organs safety and the result is a short sequence of images. The length of the output sequence is sufficient to evaluate the state of the heart. Additionally, this technology is able to produce special images with tags. Each tag moves and deforms according to the deformation of the heart. As a result, it is possible to study the deformation of the tags and evaluate the state of the heart.

The Cameo program was developed by a student from the faculty of Nuclear Sciences and Physical Engineering some time ago. The program can be used for processing, displaying and evaluating the state of the human heart using tags generated by magnetic resonance imaging scanner. However, the program required improvements in graphical user interface, fixing critical bugs, creating tests, etc. We put a lot of effort to fix the most significant issues and make the program executable and useful.

The study of the Cameo program is divided into several parts: theoretical basics that tell about the principles of the magnetic resonance technology, the structure of the program, tests for graphical user interface and the special user manual of the Cameo program.

Chapter 1

Magnetic resonance imaging

Magnetic resonance imaging (MRI) is a special approach used in radiology to provide images of the inside of the body using the MRI scanner. It is based on the principles of nuclear magnetic resonance that allows creating magnetic fields and radio waves. In contrast to computed tomography scan and positron emission tomography, this method does not include usage of X-rays or ionizing radiation which is a huge advantage. That is why MRI is widely used in medical facilities for diagnosis and monitoring the patient's condition with no radiation exposure. However, the process of getting detailed images using MRI produces an unpleasant loud sound and takes more time compared to CT scans. Moreover, people with medical implants made of metal inside the body in some cases are not able to get MRI scan safely. Usually, the patient is supposed to enter a narrow tube in order to start medical examination. The MRI scanner is able to detect the signals of excitation of hydrogen atoms that make up the human body that occur as a result of strong magnetic field and are excited by a resonant magnetic excitation pulse. The highest concentration of hydrogen can be found in water and fat, therefore, MRI scanners show the locations of water and fat. The output contrast between tissues can be changed by changing the magnetic pulse.

Figure 1.1: MRI scanner.

1.1 Tagged MRI

MRI tagging or myocardial tagging is a method of measuring and detecting motion and form-changing in the human heart without any surgical intervention. Tags can be understood as regions of tissue with longitudinal magnetization. The magnetization is modified before imaging. Thus, they are showing up dark in colour in the following MR images. The MRI tagging technique can be used for monitoring and evaluating the state of many parameters of the human heart and even the most indistinguishable parameters may be noticed.

This method is widely accepted and used in many clinical trials. However, it requires some time to collect the necessary information and using complicated algorithms in order to process the input images. Due to these reasons, the MRI tagging is not widely used in contrast to the other methods of examination of the heart.

Advantages

- Direct heart examination
- Potentially more accurate results

Disadvantages:

- Tags may become less visible at the end of the recorded sequence of images
- Low number of frames per one heart beat

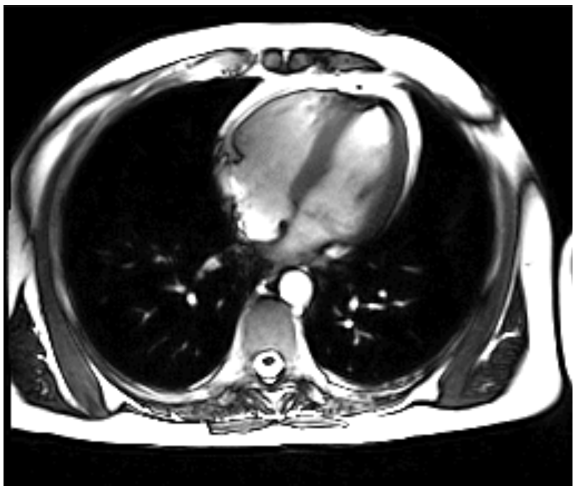


Figure 1.2: MRI image of the human heart with no tags.

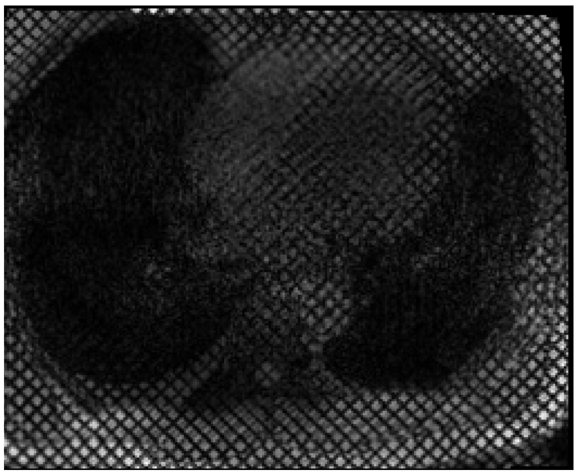


Figure 1.3: MRI image of the human heart with tags.

The mortality from heart diseases is very high around the world. Especially in developed countries. For instance, the cardiac arrhythmia is very common disease today. As a result, this is the cause of death of 1-2% of people in developed countries. Even though there are new methods of treatment, the majority of people cannot recover. According to statistics 30% people who have arrhythmia and undergone therapy did not have any benefit from the therapy and around 40% of patients began to experience a worsening of the condition due to disease progression.

The MRI tagging can improve the arrhythmia specification and help people to choose a treatment that may be more accurate according to the condition of the heart. The MRI tagging provide information

10 about the myocardium deformation of a particular parts. This information is very useful because it is able to show the degree of asynchrony of the heart.

Chapter 2

Qt

Qt is a powerful framework that allows creating cross-platform applications and graphical user interfaces. The applications created using Qt are very flexible because they can be run on different operation systems (Linux, Windows, Android, etc.) and on different hardware with minor possible changes in the source code. Moreover, Qt can be understood as a widget toolkit. This framework is used in more than 70 industries. It supports both commercial and non commercial use. Examples of the user applications created with Qt for Desktop are as follows:

- Autodesk 3ds Max
- Source 2 game engine tools
- VirtualBox
- Mathematica

The story of Qt starts in 1990 by two Norwegian programmers Eirik Chambe-Eng and Haavard Nord. In 1992 they founded the Trolltech company. As of today, the Qt Company is engaged in development of Qt.

2.1 Language Bindings

One of the most powerful features of Qt is language bindings. The API of Qt is written in the C++ programming language that makes the software made using this framework easily executed on different platforms. Moreover, Qt supports QML (Qt Modeling Language) which is a markup language very similar to CSS and JSON. It allows defining user interface and its behaviour. In addition, there are other bindings: Python, Go, PHP, Java, etc. The only Python bindings are officially supported. The rest of bindings are third party.

2.2 Main features

Qt comes as a package of very useful tools that make the development easier and faster. Some of the tools need to be mentioned.

2.2.1 Widgets

Widgets are the most fundamental elements that are used for creating a graphical user interface using Qt. Typical examples of widgets can be buttons, labels, text fields, spin boxes, etc. Each widget is extended by `QWidget` class which in turn is extended by `QObject` class. The `QObject` is the base class of any Qt object.

2.2.2 Qmake

Qmake is a tool that is used for generating makefiles. The main purpose is to make a makefile that will build an application in order to be executable. Qmake is able to configure a makefile for a particular platform. In short, it helps to make the process of building applications for a particular platform much faster and easier with minimal effort. It supports a wide range of operation systems such as Windows, Linux, macOS, IOS, etc. Qmake uses project file as an input (.pro) that tells what should be included in the final makefile. The project file usually includes the path to the source code files (.cpp) and headers (.h) and a number of additional preferences.

Initially this tool was created by the Trolltech company and was a part of the Qt framework.

2.2.3 Signals and Slots

Signals and slots is the most significant feature of Qt. This function allows the objects to communicate with each other by signals.

The most typical situation is when there are two widgets. Let's say one of them is altered and now the second one has to be informed about this alteration. This principle is similar to the callback which, however, is less intuitive in contrast to signals and slots.

The signal is sent once a predefined event occurs. Qt offers a lot of built-in events but it can also be its own method. A slot is a function that is called once the signal is emitted. Qt offers its own predefined slots but also a custom slot can be created. In other words, the signal and the slot are linked together. Moreover, one signal can be linked with several slots or one slot can be linked with several signals.

The signals and slots require the object to be inherited from the `QObject` or to be a subclass that is inherited from the `QObject`.

2.2.4 Qt creator

The Qt creator is a cross-platform IDE that is used for developing applications. It is a very powerful set of tools that includes many different functions. It provides creating applications for different platforms such as Windows, Linux, Android, etc. Moreover, it offers making applications for a browser. The development of the Qt Creator started in 2007 and initially the name was Workbench.

The Qt Creator has its own debugger and code editor. There is a number of possible project managers, for example: qmake, CMake.

The code editor has basic functionality that belongs to all famous IDEs. It is able to highlight the code, auto complete and offer refactor options (extracting method, changing names of variables, etc).

The Qt Designer is a part of Qt creator and it allows the developers visually designing and building a graphical user interface using widgets. It has a wide range of tools for customizing windows, layouts and a visual side of each element. The GUI created using this tool can be easily accessed via the code and each widget can be connected with a particular function using signals and slots.

2.2.5 Qt Test

Qt Test is a framework that is used for unit testing of applications that are made using Qt. It has various functions that can be found in all widely used testing frameworks. One of the functions is testing graphical user interface.

The main features of Qt Test are as follows:

1. It is Relatively small in contrast to the other frameworks.
2. Tests can be added very easily just in a few lines of code.
3. Tests can be executed more than once with different input data.
4. The Qt Test is able to simulate keyboard and mouse.
5. Qt Test does not require to be used in Qt Creator only.

To add a new test, it is necessary to add a method as a private slot. Qt Test will run tests one-by-one according to the order in the class definition. There are predefined default methods that are automatically added in the class-tester. They are not testing methods but they are useful for preparing for testing. The most useful ones are as follows:

- The `initTestCase()` is called before all test methods to prepare system for testing.
- The `cleanupTestCase()` is called last to clean temporary files, restore default preferences.
- The `init()` is called before each testing method to initialize the program before testing.

Chapter 3

Program structure

Cameo is a program that allows browsing and working with images made using the MRI scanner, display additional information about the patient and provide the analysis of the heart muscle. The input images are supposed to be tagged so that the study of myocardium can be run. The grid tracker is a part of Cameo and it allows analyzing the movement of the heart muscle by tracking the grid in the sequence of MRI images. The principle is that Cameo generates its grid and puts it above each image. The generated grid is supposed to follow the grid in the MRI images and as a result, the program is able to detect possible problems with the heart muscle.

The main structure of the Cameo program will be described in this section. The main structure includes the relationships between the main classes, methods and basics information of the GUI architecture.

3.1 DICOM files

DICOM (Digital Imaging and Communications in Medicine) is a special medical standard that is used for creating, storing and visualizing medical images or documents of examined patients.

It was designed by the DICOM Standard Committee and is a part of ISO standard. Is supported by medical equipment manufacturers and software developers. With this format it is possible to create, store, and transmit the whole sequences of images or just particular ones. There are two information layers of DICOM:

- DICOM file is an object made of tags for representing one or more frames
- DICOM Network Protocols is a layer responsible for transmitting over networks that support the TCP/IP protocols

3.2 GUI

The `Ui_MainWindow` contains declarations of all the main static and interactive graphical objects. It defines the main widgets, layouts, buttons, labels, etc.

In this picture, the important parts are highlighted in red, yellow and blue.

- Red rectangle: `GV_image`. The area of the selected DICOM image.
- Yellow rectangle: `GV_series`. The area of the series of available DICOM images.

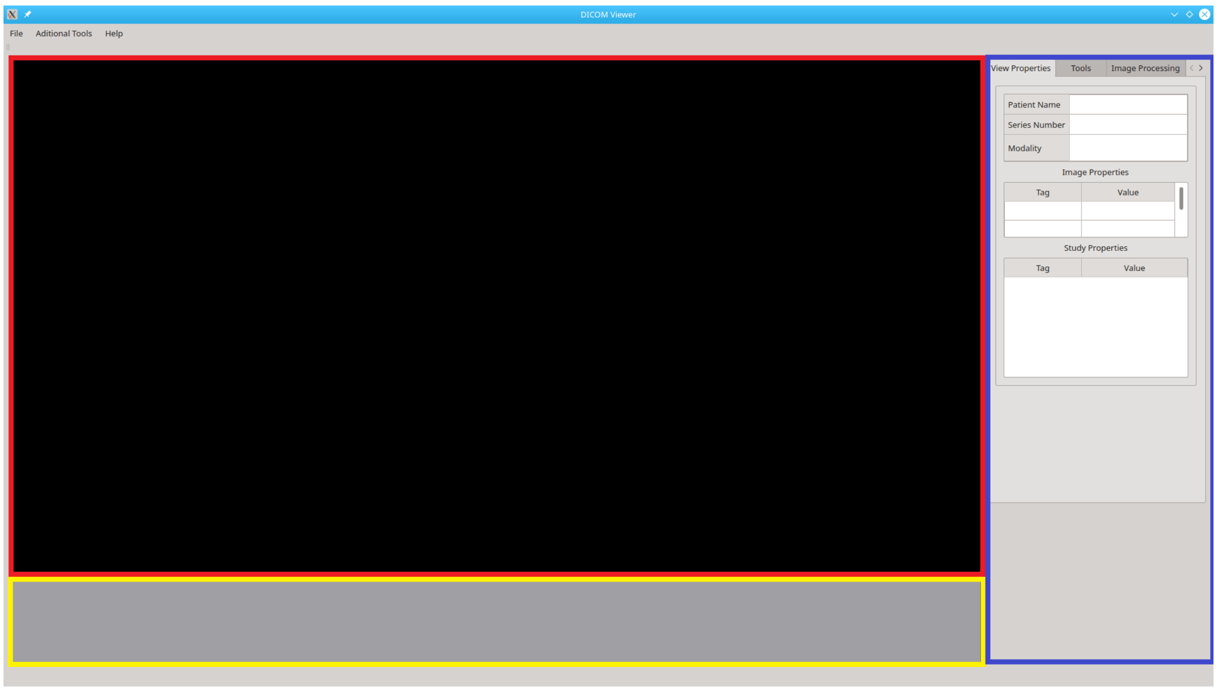


Figure 3.1: Cameo application.

- Blue rectangle: `tabWidget`. The area with the main tools that can be applied to the opened series of DICOM images.

All declared objects of the interface should necessarily be connected with an appropriate method. It happens in the `MainWindow` class inside the constructor with `QObject::connect()`.

The `MainWindow` class inherits `QMainWindow` and it is responsible for the program's GUI logic (e.g, making buttons inactive when needed), reading user's input signals and sending to the other classes. One of the most important features of the Cameo program is the ability to view DICOM images. These input images can be loaded by pressing the open-button in "File" drop-down menu.

3.3 Workflow with DICOM files

The Cameo program also offers to save the selected area of DICOM files. It can be enabled in the "Tools" tab. When this feature is enabled, the green rectangle with the dashed line appears on top of the image and by changing the height and the width the selected area changes.

The `CutGraphicsItem` class inherits `QGraphicsRectItem` and is responsible for the functionality. It contains the `paint()` method that draws the rectangle. Before the rectangle is drawn, the `xMappedToScene` and `yMappedToScene` methods are called. They adjust the top left point of the image on the scene for correct display. In addition, the class has mouse events methods. These methods are responsible for changing the area of interest with the mouse by clicking on the edge of the green rectangle and dragging it inside or out. To determine the mouse click and dragging, there are two bool variables for each side.

```
bool rightSideSelected, leftSideSelected, bottomSideSelected, upperSideSelected;
bool rightSideToSelect, leftSideToSelect, bottomSideToSelect, upperSideToSelect;
```

When the cursor clicks the edge of the green rectangle, the appropriate variable `*SideToSelect` becomes true. When the left mouse button is released, the `ToSelect` variables are refreshed. However, if the left mouse button is hold and the cursor is moving, it indicates that the side is selected and appropriate variable `*SideSelected` becomes true and it allows changing the area of interest.

3.4 Workflow with files

The `OpenDicomSeries()` method can be executed by the “Open” button, it pops up a window in which the input file can be chosen and saves the path of the file. In the case of previously opened file, the method closes it, clears `DicomScene imageScene` and `QGraphicsScene seriesScene`, frees up the memory and makes inactive some parts of the interface that are supposed to be active only when the input is loaded.

This method instantiates the `DicomQImageItems` object that does the logic of processing and opening DICOM files by passing a path of the image to the constructer.

```
dicomQImageItems = new DicomQImageItems(fileName);
```

It scans the directory where the chosen file is located for other DICOM images to make a row of images. If the input images have been opened successfully, the method `isDicomQImageLoaded()` returns true. In the end, `MainWindow::openDicomSeries()` will load and show the DICOM information stored in the input files and the method returns true. Otherwise, if `dicomQImageItems->isDicomQImageLoaded()` return false, the `MainWindow::openDicomSeries()` will return false and shows up a message box.

```
 QMessageBox *messageBox = new QMessageBox(this);
 messageBox->setText("DICOM series can not be loaded");
```

Moreover, this method can be called also using keyboard shortcut ctrl+o

```
QShortcut *openFileShortcut;
openFileShortcut = new QShortcut(QKeySequence("Ctrl+O"), this);
```

Images will be stored in dicomQImageItems class and may be accessed by `getElement(imageIndex)`. The `MainWindow` class also allows users to save one single or more files in several ways:

- Save scene
- Save view
- Save only selected areas of an image

Furthermore, there are two sets of options how to save DICOM images: save only a single image that is currently on scene or save multiple images that are selected. All the mentioned features use the same method at the beginning.

This method asks the user to choose the directory where the files should be saved, gives a prefix to the files and offers to choose the format of the image: jpg, bmp or png. Every operation is checked using a boolean variable and if the directory was not chosen or the program failed to save a file, the method will return false.

The `MainWindow` class also provides zoom in and zoom out operations by holding the control button and scrolling the mouse wheel. It is done using `eventFilter` method that allows monitoring incoming events of a certain object and process them. In addition, it requires to install the `eventFilter` which can be done in the `openDicomSeries()` method only when the DICOM files were successfully opened. The `eventFilter` is attached directly to the `GV_image` object which is responsible for displaying the selected image.

```
bool MainWindow::openDicomSeries()
{
    ui->GV_image->scene()->installEventFilter(this);
    return true;
}
```

Furthermore, it is supposed to define a new behaviour based on events. The main idea of this method is to accept or deny the event, that is why it returns a bool variable.

The `ScaleFator` variable defines the speed of increasing or decreasing of an opened DICOM image based on changing `delta` of scroll event.

Moreover, this method is responsible for changing the angle of the grid. The angle changing will be described in the next section.

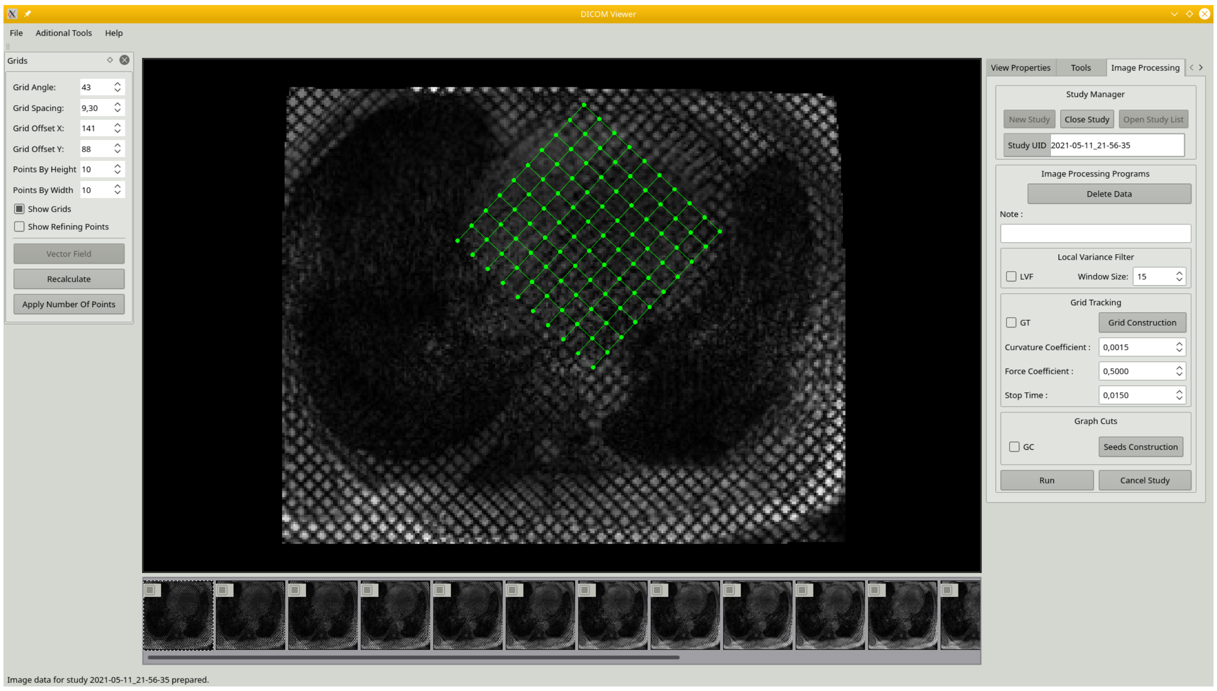
```
bool MainWindow::eventFilter(QObject *obj, QEvent *event)
{
    if (event->type() == QEvent::GraphicsSceneWheel)
    {
        ui->GV_image->setTransformationAnchor(QGraphicsView::AnchorUnderMouse);
        double scaleFactor = 1.15;
        bool ok = QApplication::keyboardModifiers() & Qt::ControlModifier;
        if (ok)
        {
            QGraphicsSceneWheelEvent *scrollevent = static_cast<↩
                QGraphicsSceneWheelEvent *>(event);
            if (scrollevent->delta() > 0)
            {
                ui->GV_image->scale(scaleFactor, scaleFactor);
            }
            else
            {
                ui->GV_image->scale(1/scaleFactor, 1/scaleFactor);
            }
        }
        //else... change grid's angle

        return true;
    }

    return false;
}
```

3.5 Grid Construction

When the DICOM images are successfully loaded, it is possible to run grid construction that allows examining the condition of the heart muscle. The button is located in the Image Processing tab. It calls the `gridTrackingGUI(bool showGrids)` method that sets the window of grid preferences to be active and instantiates the `GridCatalog` object with the input from the grid window.



Grid Angle: 43
Grid Spacing: 9,30
Grid Offset X: 141
Grid Offset Y: 88
Points By Height 10
Points By Width 10
[✓ Show Grids]
Show Refining Points

Study UID 2021-05-11_21-56-35
Note :
LVF Window Size: 15
GT
Curvature Coefficient : 0,0015
Force Coefficient : 0,5000
Stop Time : 0,0150
GC

Image data for study 2021-05-11_21-56-35 prepared.

Figure 3.2: Cameo with constructed grid.

The `GridCatalog` class can be understood as the entry point to the logic of the grid construction. It creates MultiVector files in the directory of DICOM images and continues by creating grid for every single frame with given parameters. One single grid is created using the `GridGraphicsItem` class that gets the input parameters and runs the grid construction. The last step is storing the grid in the list of the pointers of the `GridGraphicsItem`.

```
for(int f = 0; f < framesInDicomSeries; f++)
{
    GridGraphicsItem* grid = new GridGraphicsItem;
    grid->setX(this->x);
    grid->setY(this->y);
    grid->setWidth(this->width);
    grid->setHeight(this->height);
    grid->runGridConstruction( angle, spacing, offsetX, offsetY );
    grids.append( grid );
}
```

- `SetX` and `SetY` - set left top x and y coordinates of grid in image scene.
- `setWidth` and `setHeight` - set width and height of the covered area by the grid. The area can be affected with changing the area of the interest with the tool located in the tab of tools. Area of the interest is defined with image's width and height by default.
- `angle` - angle of the whole grid.

- `spacing` - defines distance between curves of the grid.
- `offsetX` and `offsetY` - values in x and y axis that shift the whole grid from the point where the grid has been initialized.

The `runGridConstruction` method checks if the grid has been created. If so, it deletes the grid and calls the method that initializes grid. The initializing method belongs to `grid-tracker`. In the end, it creates grid widgets so that the grid is displayed and ready to work.

Furthermore, the grid window appears on the left side of the main window. The `GridWindow` class is responsible for this operation, it extends `QDockWidget`. In the constructor it provides memory allocation for spin boxes, labels and buttons with adjusting maximum and minimum values of a particular fields, connecting signals and slots and so on. This class will get access to the user input fields for the `MainWindow` class.

Once the grid is ready, it can be moved, rotated or the distance between single nodes can be changed. The rotation can be set manually by typing numbers in the special field in the grid window or by holding shift and scrolling mouse wheel. It is done using the `eventFilter` method mentioned above. If any of the grid parameters were changed, the function - `applyGridsParameters` is called. It sets the angle, offsets and spacing from the UI spin boxes and runs `resetGrids(numberOfPointsByHeight, numberOfPointsByWidth)`.

3.6 Arguments

The arguments are implemented in the `ProcessArguments` class. The constructor takes the list of the input arguments, the number of the input arguments and the `MainWindow` class. The `MainWindow` class in necessary because it contains some methods that are implemented specially for executing using input arguments.

The `ProcessArguments` classs is implemented using `getopt.h` library that allows parsing the input arguments. The class contains an array of structures of type `option` where the all arguments are defined.

```
    const struct option longopts[] = {
        { "input", required_argument, 0, 'i' },
        { "help", no_argument, 0, 'h' },
        { "grid", optional_argument, 0, 'g' },
        { "angle", required_argument, 0, 'a' },
        { "spacing", required_argument, 0, 's' },
        { "offsetX", required_argument, 0, 'x' },
        { "offsetY", required_argument, 0, 'y' },
        { 0, 0, 0, 0 },
    };
```

As can be seen in the code listing above, each structure has a full argument key word, obligation, flag and the short name of the argument. Furthermore, flags are used for particular arguments as an indicator that argument appeared once.

We used `getopt_long` method that allows making a regular expression of short names of the arguments and parse the input arguments in the cycle.

Chapter 4

Tests

In this chapter, the details of testing of the Cameo program user interface is described. The main goal in GUI testing is not to affect the structure of the program as much as possible. It can be done by avoiding strong dependencies with minimal changes in the source code of the program. However, some minor actions in the source code should be provided in order to achieve the desired result.

4.1 Unit Testing

Unit testing is a technique that allows ensuring that each unit of the program works stably and according to the expectations. The "unit" can be represented by a module, method, function or even an object. The idea is to take a relatively small piece of the source code to test so that tests become short and easy to maintain. The unit testing is frequently used at the stage of the development.

The unit tests are able to save time and money during the development process. However, wrong tests may lead to critical problems that would possibly require a lot of time to solve.

Three are three main reasons to conduct unit testing:

1. Helps the code to be reusable. Since tests are already prepared, it is possible to ensure that the code works as it should work in another project or module.
2. Allows adding changes safety and quickly.
3. Helps to reveal bugs at early stages.

4.1.1 Basic principles

The unit tests are usually automated and can be executed at any time. Additionally, the unit tests can be manual. If there's a possibility to automate the tests, the manual tests do not make sense. The reason for having the manual tests may consist in specific requirements of the project. Most of the time, developers use a special frameworks that are designed for the test automation. Moreover, these frameworks provide logs and an additional information, for example, time of the execution.

Another interesting principle is based on isolation. The developer can create a special environment into which the method to be tested can be copied. In this way, it can help to reveal unsafe dependencies and fix them.

Advantages:

- The unit tests allow improving code structure and ensure that the code works in the same way.

- Tests can be written even if not all modules are finished.
- Bugs can be easily found.

Disadvantages:

- This testing approach is unable to reveal all bugs because it is focused on small pieces of code.
- It is challenging to write readable and quality tests because they can be very specific.

4.2 GUI Testing

This way of testing is focused on the graphical user interface functionality. It is an extremely useful technique because even if the structure of the program is well tested, it does not guarantee that the behaviour of the program from the user's side will be working as it is expected. Additionally, this way of testing is supposed to check UI elements: positions, colours, etc.

There are two methods how GUI testing can be performed:

Manual Testing. This method requires a human presence. The tester is supposed to check if each UI element or a particular part of GUI is working as expected. Undoubtedly, the tester should know the product requirements and understand the behaviour of the particular elements.

This approach is used when GUI is ready to use but it is so fresh and unstable that it requires a human attendance to be tested. In addition, manual testing is suitable when GUI of the program constantly has new significant changes.

Automated GUI testing. Special tools are used to automate the testing process. These tools analyze the user's or tester's interactions with the given application and then collect and store this data. The collected data can be used by developers to evaluate new features and the effectiveness UI.

The GUI testing can become challenging when it requires to know the state of the particular structure of the program. Sometimes it is impossible to validate if the test is completed successfully without knowing the value of the variable in the source code. Having access to the source code, the GUI testing may become more effective and reliable.

Advantages:

- The tests are provided from the side of the user.
- The GUI testing helps to check if the UI elements are on the places where they are supposed to be.
- As a result, it improves the product and makes it more stable.

Disadvantages:

- It can be less effective when the testers cannot have the access to the source code of the program.
- Requires additional software and human resources.
- Significant changes in the GUI may lead to refactor or even re-implementing existing tests.

4.3 Dependencies

To ensure better test independence from the main project, it was decided to create a completely separate new project and include all the existing source codes and the header files of Cameo. The project configuration file differs from the configuration file of the main project in that it has `QT += testlib` adjustment, which allows using the QTest library. The rest of the project configuration file is completely identical to the main one with the only one extra class included. The class is responsible for the tests and also it has different file `main.cpp` because the `main.cpp` needs to be configured for the QTest. The extra class is called `CameoGUITesting` and is set to be a friend class of the `MainWindow` class to get access to the private methods and fields. It is necessary because,e.g., buttons are private objects.

```
friend class CameoGUITesting;
```

Since the `MainWindow` class is the main one for the whole GUI functionality, it is reasonable to give an access to `CameoGUITesting` to each method and field.

4.4 Preparations for GUI testing

`CameoGUITesting` is defined as the main class for the QTest library and it means that the methods that are declared as the private slots will be executed sequentially one-by-one and each one is understood as one single test case. Each method has `QVERIFY(bool)` macro at the end that defines the test success in the case of true value or failure in the case of false one. For example, `QVERIFY` checks if the files are successfully opened or saved.

The main function should be changed as well. Usually at the end of the main function there is `QApplication::exec()` method that allows the application to be controlled by Qt and the program enters the event loop. It means that Qt will be able to handle operation system and user events. These incoming events may be passed to the program's widgets. Once the user exits the application, Qt interrupts the event loop, returns integer value and the main method returns. To get the same result, it is supposed to enter the event loop using QTest and therefore the `QTest::qExec(&t, argc, argv)` is used where `t` is the object of the `CameoGUITesting` class. As a result, the program execution will be stopped as soon as the testing class finishes its tests.

In addition, it is important to implement a method that is able to focus on the pop-up window with warning information to skip. It is implemented in the `PressEnterWhenMessageBox()` method that stores in the `QWidgetList` top level widgets and by iterating over all of the widgets, it is possible to find one that inherits from the `QMessegeBox` object and by simulating the key click, the enter key can be pressed and the message is skipped. Otherwise, it would be impossible to continue GUI testing since the message box is in focus until enter is not clicked.

4.5 GUI testing of Cameo application

The first called method of `CameoGUITesting` class is `Init()`. It prepares Cameo for automatic tests by setting the `testMode` boolean variable of the `MainWindow` class true and the path to the folder where the images will be saved. The `testMode` variable will allow bypassing the window in which the user is supposed to choose the folder to store images because QTest library does not support the button simulations for all kinds of windows.

First of all, to simulate a button click it is needed to find the object to be clicked on. It is done using `findChild` method and since the majority of the GUI logic is implemented in the `MainWindow` class, it is supposed to look for the particular object in the `MainWindow` object. Once the object has been found, it is possible to use the `mouseClick` method to simulate user's mouse click which is a part of the QTest library.

```
QPushButton *pb = w->findChild<QPushButton*>("PB_newStudy");
QTest::mouseClick(pb, Qt::LeftButton);
```

However, this method works, e.g., with the `QWidget` objects but it does not work with the GUI elements that do not inherit from `QWidget`. For this reason, the `QAction` objects that are represented as buttons of drop-down menus do not support the mouse click simulation. This problem can be solved using keyboard shortcuts:

- The Alt + the first letter of the headers of the drop-down menu allows selecting a particular drop-down menu and open it
- The Enter button replaces mouse click
- The keyboard arrows allow to navigate in the drop-down menus

Since there are some repetitive actions, it is logical to put them in the separate methods. Each helping method calls pause and a particular selection. The pause allows waiting the end of the previous operation and execute the next one. The pause time is calculated in milliseconds. These methods should not be placed in the private slots. Otherwise, the Qt test would execute them separately as the testing methods.

```
void CameoGUITesting::KeyDownOnDropDownMenu(QMenu *mainMenu)
{
    QTest::qWait(300);
    QTest::keyClick(mainMenu, Qt::Key_Down);
}
```

The list of the helping methods:

- `PressEnterWhenMessageBox()` is used for pressing "Ok" button in message window in order to skip it. This kind of window can appear when user is trying to upload a file of an unexpected format.
- `FileDropDownMenu()` helps to open File drop-down menu in the Cameo main window. It allows to start navigating to particular buttons.
- `KeyDownOnDropDownMenu(QMenu *mainMenu)` this method allows to select the next bottom button. It is possible to pass an input parameter which is represented by the QMenu class in order to choose what the menu is necessary to interact with.
- `KeyEnterOnDropDownMenu(QMenu *mainMenu)` when the needed button was chosen, with this method it is possible to simulate "Enter" and trigger the selected button in a particular menu.
- `ImageProcessingTab()` is used for selecting the Image Processing tab in Cameo window. It is located in the right side and contains many tabs. The Image Processing tab contains the button that executes the grid construction.
- `CleanSpinBoxField()` while testing the functions of the grid, it is required to be sure that all data will be typed with no additional unexpected data. For this reason, the initial content of the input fields is removed before setting testing parameters of the grid (e.g. angle, spacing etc.).

The `SaveSingleArea()` test provides opening the drop-down menu, navigating to the particular button in it and checking if the image has been saved. The whole test is based on GUI. At the beginning, it is necessary to store the objects of interest,i.e., the main menu and the sub menu. The sub menu is a part of the drop-down menu and has three types of saving one image. If the image is saved, at the left bottom corner of the window the message appears indicating whether the operation completed successfully.

```
void CameoGUITesting::SaveSingleArea()
{
    QMenu *mainMenu;
    mainMenu = w->findChild<QMenu*>("menuFile");
    QMenu *subSaveMenu;
    subSaveMenu = w->findChild<QMenu*>("menuSave_Image");

    FileDropDownMenu();
    KeyDownOnDropDownMenu(mainMenu);
    KeyEnterOnDropDownMenu(mainMenu);
    KeyDownOnDropDownMenu(subSaveMenu);
    KeyDownOnDropDownMenu(subSaveMenu);
    KeyEnterOnDropDownMenu(subSaveMenu);

    QTest::qWait(300);

    QString message = "Displayed area of interest saved.";
    QVERIFY(message == w->statusBar()->currentMessage());
}
```

Available tests provide:

- opening a DICOM files
- opening files with a wrong format
- saving one image in 3 different ways
- saving the whole series of images in 3 different ways
- starting study of the opened DICOM file
- running the grid construction
- changing the grid parameters

The list of the available tests with a short description about each one:

1. `OpenWrongFile()` opens a file of an unexpected format.
2. `OpenNormalFile()` opens a file of the expected format. Further tests expect that this test was completed successfully.
3. `SaveSingleView()` provides saving the main view of the opened DICOM file (including zoom value).
4. `SaveSingleScene()` provides saving the main view. The Zoom value does not affect the result.

5. `SaveSingleArea()` provides saving the selected area of the image. The selected area is represented by the green rectangle with a dotted line.

6. `SaveAllSelectedAreasToFile()` saves all the DICOM images with the selected area. It works in the same way as `SaveSingleArea()` but for all the available images. It compares a short message of the result of the operation that appears once it is completed.

7. `SaveAllSelectedViewsToFile()` saves all the DICOM images with the configured zoom. It works in the same way as `SaveSingleView()` but for all the available images. It compares a short message of the result of the operation that appears once it is completed.

8. `SaveAllSelectedScenesToFile()` saves all the DICOM images and ignores the value of the zoom. It works in the same way as `SaveSingleScene()` but for all the available images. It compares a short message of the result of the operation that appears once it is completed.

9. `ConstructGrid()` runs grid construction by opening a special tab in the tools tab where the needed button is located.

10. `ChangeGridAngle()` removes the initial value of the grid and types the testing input value. At the end, compares the value of the angle in the text field with the value in the grid class.

11. `ChangeGridSpacing()` removes the initial value of the grid and types the testing input value. At the end, compares the value of the spacing in the text field with the value in the grid class.

12. `ChangeGridOffsetX()` removes the initial value of the grid and types the testing input value. At the end, compares the value of the offset along the the x-axis in the text field with the value in the grid class.

13. `ChangeGridOffsetY()` removes the initial value of the grid and types the testing input value. At the end, compares the value of the offset along y-axis in the text field with the value in the grid class.

14. `ChangeGridPointsHeight()` changes the initial value of the amount of the points in height.

15. `ChangeGridPointsWidth()` changes the initial value of the amount of the points in width.

Since it is a separate project which is connected to the Cameo source code. At the end of testing AddressSanitizer prints in the console the list of memory leaks. More detailed information regarding this tool can be found in the next chapter.

```
********* Start testing of CameoGUITesting *********
Config: Using QtTest library 5.9.5, Qt 5.9.5 (x86_64-little_endian-lp64 shared (↩
    dynamic) release build; by GCC 7.4.0)
PASS   : CameoGUITesting::initTestCase()
PASS   : CameoGUITesting::Init()
PASS   : CameoGUITesting::OpenWrongFile()
PASS   : CameoGUITesting::OpenNormalFile()
PASS   : CameoGUITesting::SaveSingleView()
PASS   : CameoGUITesting::SaveSingleScene()
PASS   : CameoGUITesting::SaveSingleArea()
PASS   : CameoGUITesting::SaveAllSelectedAreasToFile()
PASS   : CameoGUITesting::SaveAllSelectedViewsToFile()
PASS   : CameoGUITesting::SaveAllSelectedScenesToFile()
PASS   : CameoGUITesting::ConstructGrid()
PASS   : CameoGUITesting::ChangeGridAngle()
PASS   : CameoGUITesting::ChangeGridSpacing()
PASS   : CameoGUITesting::ChangeGridOffsetX()
PASS   : CameoGUITesting::ChangeGridOffsetY()
PASS   : CameoGUITesting::ChangeGridPointsHeight()
PASS   : CameoGUITesting::ChangeGridPointsWidth()
PASS   : CameoGUITesting::cleanupTestCase()
Totals: 18 passed, 0 failed, 0 skipped, 0 blacklisted, 35900ms
```

Chapter 5

Memory leaks

The C++ programming language supports the three main types of memory allocations.

The dynamic memory allocation allows allocating memory when it is needed while the app is running. The memory is allocated on the heap which is much larger than the stack. To allocate memory, the `new` operator is used. The `new` operator returns the pointer to the newly allocated memory.

Static memory allocation is less flexible because the size is fixed and cannot be changed, which leads to impossibility of the memory re-usage in contrast to the dynamic memory allocations. Moreover, memory is allocated on the stack. The amount of memory on the stack is small and very limited. For example, function parameters and local variables use the static memory allocation and are stored on the stack.

5.1 AddressSanitizer

However, using dynamic memory leads to memory leaks. The memory leak is a dynamically allocated memory that was not freed. This is the key feature between other types of memory allocations because once something was allocated dynamically, the developer should clean up the used memory once it is not needed. If the developer does not monitor the allocated memory, it may accumulate and cause the program to crash.

The AddressSanitizer is a tool that allows locating memory-related errors, mainly memory leaks, dangling pointers, etc. This tool is a part of Clang , GCC, Xcode and MSVC compilers.

In order to use this tool, it is necessary to add two lines to a Qt project file.

```
CONFIG += sanitizer sanitize_address\
QMAKE_CXXFLAGS += "-fsanitize=address"
```

5.2 Workflow with AddressSanitizer

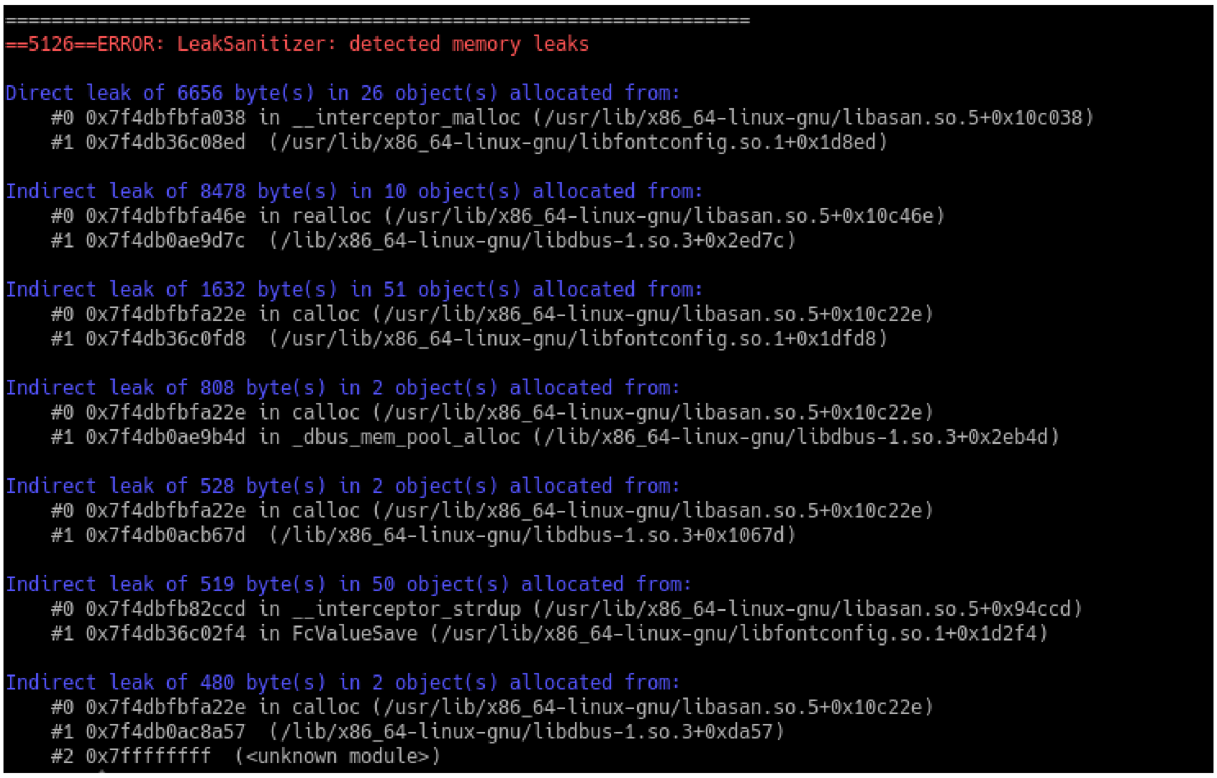
When the program finishes working, the AddressSanitizer prints logs into the console output with the list of direct and indirect memory leaks.

- **Direct leak** - is an allocated memory block with no pointer to it. As a consequence, memory becomes unreachable.

- **Indirect leak** - the result of a direct leak. A situation when the data stored in the leaked block of memory (direct leak) points to another memory block. In other words, leaked memory (direct leak) points to another block of memory and since the direct leak is unreachable, the data it points to becomes unreachable too and this is called an indirect leak.

There is a very high probability that even a small and simple program can finish working with direct and indirect leaks. It can happen because external libraries are used and there is no opportunity to fix the leak of memory because the source code of these libraries is unreachable.

The AddressSanitizer is able to print informative logs. Once the program exits, the AddressSanitizer prints out the list of detected direct and indirect leaks. Each leak is represented in bytes and in the number of allocations. By fixing direct leaks the indirect ones are fixed as well because in most cases they are provoked by direct ones.



```
=================================================================
==5126==ERROR: LeakSanitizer: detected memory leaks

Direct leak of 6656 byte(s) in 26 object(s) allocated from:
    #0 0x7f4dbfbfa038 in __interceptor_malloc (/usr/lib/x86_64-linux-gnu/libasan.so.5+0x10c038)
    #1 0x7f4db36c08ed  (/usr/lib/x86_64-linux-gnu/libfontconfig.so.1+0x1d8ed)

Indirect leak of 8478 byte(s) in 10 object(s) allocated from:
    #0 0x7f4dbfbfa46e in realloc (/usr/lib/x86_64-linux-gnu/libasan.so.5+0x10c46e)
    #1 0x7f4db0ae9d7c  (/lib/x86_64-linux-gnu/libdbus-1.so.3+0x2ed7c)

Indirect leak of 1632 byte(s) in 51 object(s) allocated from:
    #0 0x7f4dbfbfa22e in calloc (/usr/lib/x86_64-linux-gnu/libasan.so.5+0x10c22e)
    #1 0x7f4db36c0fd8  (/usr/lib/x86_64-linux-gnu/libfontconfig.so.1+0x1dfd8)

Indirect leak of 808 byte(s) in 2 object(s) allocated from:
    #0 0x7f4dbfbfa22e in calloc (/usr/lib/x86_64-linux-gnu/libasan.so.5+0x10c22e)
    #1 0x7f4db0ae9b4d in _dbus_mem_pool_alloc (/lib/x86_64-linux-gnu/libdbus-1.so.3+0x2eb4d)

Indirect leak of 528 byte(s) in 2 object(s) allocated from:
    #0 0x7f4dbfbfa22e in calloc (/usr/lib/x86_64-linux-gnu/libasan.so.5+0x10c22e)
    #1 0x7f4db0acb67d  (/lib/x86_64-linux-gnu/libdbus-1.so.3+0x1067d)

Indirect leak of 519 byte(s) in 50 object(s) allocated from:
    #0 0x7f4dbfb82ccd in __interceptor_strdup (/usr/lib/x86_64-linux-gnu/libasan.so.5+0x94ccd)
    #1 0x7f4db36c02f4 in FcValueSave (/usr/lib/x86_64-linux-gnu/libfontconfig.so.1+0x1d2f4)

Indirect leak of 480 byte(s) in 2 object(s) allocated from:
    #0 0x7f4dbfbfa22e in calloc (/usr/lib/x86_64-linux-gnu/libasan.so.5+0x10c22e)
    #1 0x7f4db0ac8a57  (/lib/x86_64-linux-gnu/libdbus-1.so.3+0xda57)
    #2 0x7fffffff  (<unknown module>)
```

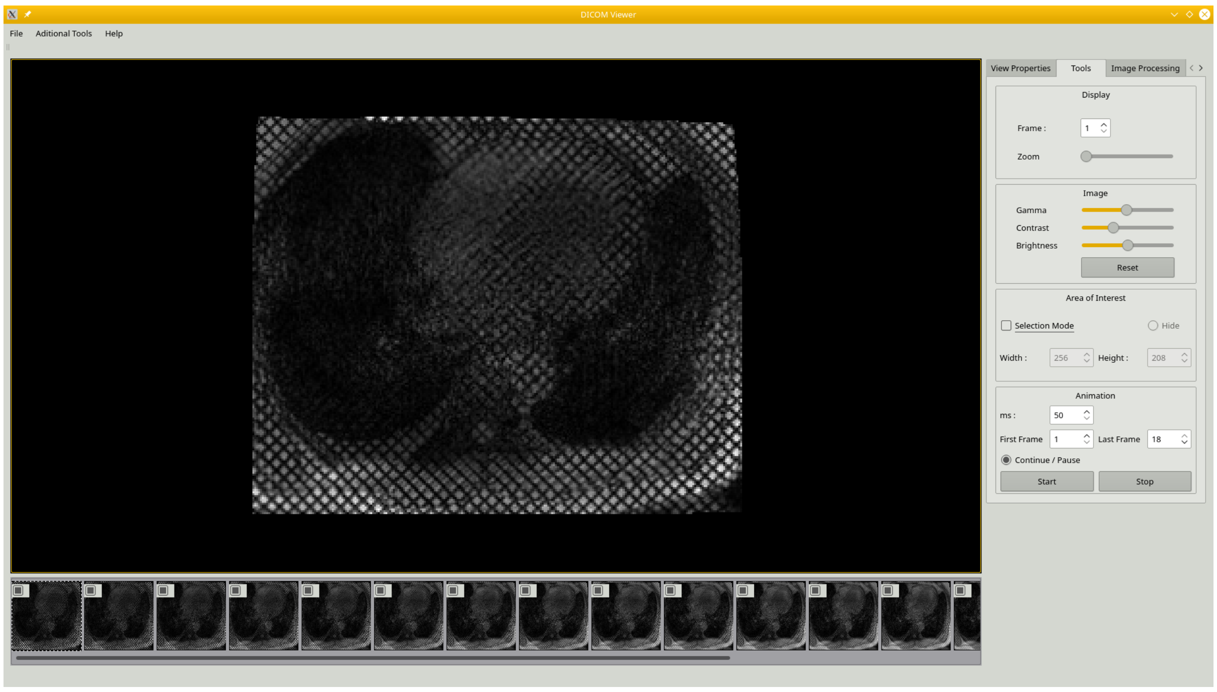
Figure 5.1: AddressSanitizer - Report Example.

Chapter 6

User Guide

User manuals are very effective because they tell users, who are not familiar with the application, how to use it effectively. This is especially important for the medical applications. Even well structured, useful, and stable the application can be unusable without proper documentation. Therefore, working with the Cameo application will be described in this chapter.

6.1 Main window overview



Frame : 1

Width : 256 Height : 208

ms : 50

First Frame 1 Last Frame 18

Figure 6.1: Cameo GUI - example.

Once the application is executed, the main window appears. It consists of the following three main sections:

- **Selected DICOM image.** In this area the selected DICOM file can be viewed. In addition, some manipulations are possible with an opened image.

- **Row of DICOM images.** In this section all the images that were in the same folder with the opened DICOM file appear. Any of the available images can be selected and browsed.
- **Tabs of additional tools.** There are 4 tabs with different functionalities. Parameters of the images can be changed in this section, patients' information can be viewed, etc. All available functions will be described in the next sections.

Moreover, there is a panel with drop-down menus that offer additional operations with the files.

6.2 Menu bar

The menu bar consists of the following three menus: File, Additional Tools and Help.

6.2.1 File

The File drop-down menu allows opening files, saving files and exiting the program.

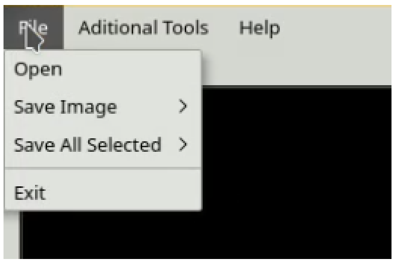


Figure 6.2: File drop-down menu.

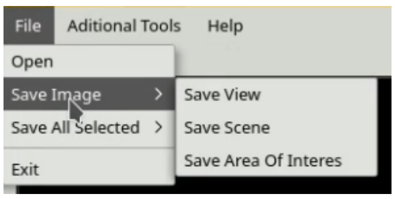


Figure 6.3: Save image drop-down menu.

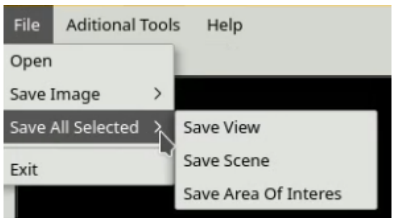


Figure 6.4: Save all selected drop-down menu.

- **Open.** The Open button allows opening a new file.
- **Save Image.** Save Image is another drop down menu. It allows saving the selected image in three different ways: save view, scene, area of interest.
- **Save All Selected.** Save All Selected is another drop-down menu that has the same functionality as the previous menu but for all the images in the given series.

6.2.2 Additional Tools

The Additional tools drop-down menu contains the three tools that can be applied to the DICOM images. Each tool appears as a separate section of the main window and can be detached and become a separate window.

- Grid tools
- LVF (Local Variance Filter) tools
- Graph Cuts tools

6.2.3 Help

In the Help drop-down menu there is the only one button About. It opens a small message box that contains the name and email of the developer.

6.3 Tool tabs

Tool tabs are located on the right section of the main program. Tool tabs consist of the four tabs:

- View Properties
- Tools
- Image processing
- History

6.3.1 View properties

View properties are used to display information about the patient that contains the DICOM images. Moreover, the width and the height of the input images are displayed.

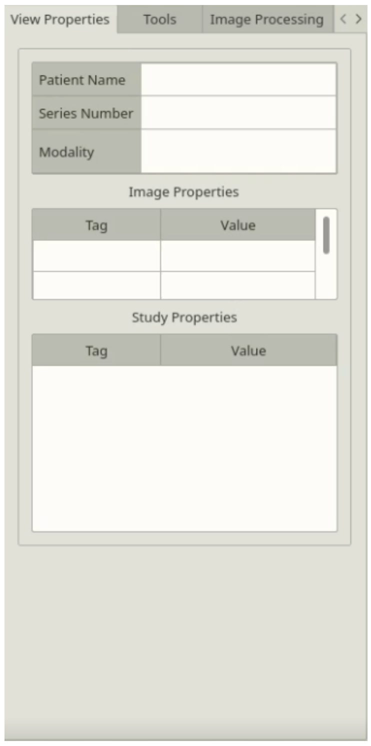


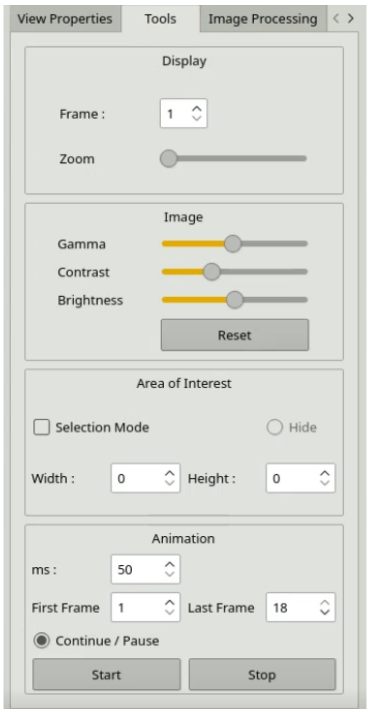
Figure 6.5: Cameo GUI - View Properties tab.

6.3.2 Tools

The tools tab is divided into four parts:

- **Display.** In this part there is a slider that sets zoom and a spin-box that shows the number of the current frame. Moreover, the current frame can be changed manually.
- **Image**. There are three sliders that adjust gamma, contrast and brightness. This feature can be useful if the image is of poor quality. In addition, there is Reset button that restores default values of the sliders.
- **Area of interest.** This part is used for creating a green rectangle above the image in order to select a part of Interest of an image.

  There are two radio buttons: Selection mode and Hide. The Selection mode shows the green rectangle and the Hide radio-button hides the green rectangle when needed. If Selection mode is deselected, the green rectangle disappears. Additionally, there are two text fields that provide setting the width and the height of the green rectangle manually.
- **Animation.** Since a series of the images are expected, there are basic animation preferences. There are three text fields that set the time for the next frame (measured in milliseconds), the number of the starting frame and the number of the ending frame. There are two buttons Start and Stop at the bottom. Stop button stops the animations and sets the current frame to the starting frame. There is also a Continue / Pause radio button that stops and resumes the animations. (add pic of area of interest)



Frame : 1

Width : 0 Height : 0

ms : 50

First Frame 1 Last Frame 18

Figure 6.6: Cameo GUI - Tools tab.

6.3.3 Image Processing

The Image processing tab is responsible for providing a study of the given sequence of the DICOM images. It is divided into two main sections:

- **Study manager.** In order to use functions of the Image processing tab, the study should be started. The Start study button is located in the Study manager section. Moreover, there are buttons Close study, Open Study list and the text field where generated ID of the study appears (Study UID).

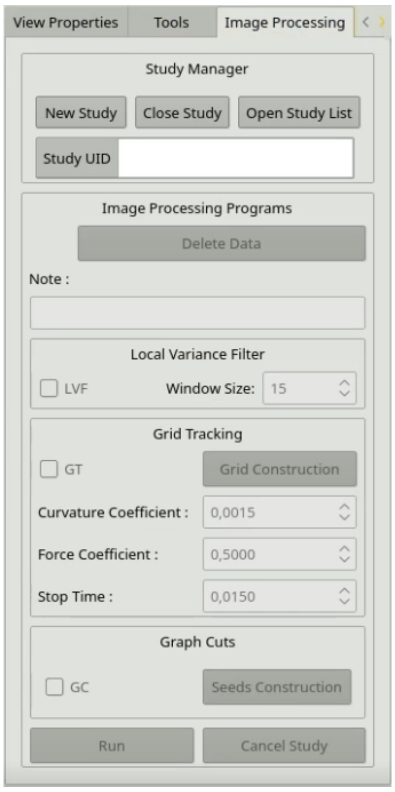
- **Image Processing Programs.** It is a set of programs that are able to process and analyse the given DICOM images.

  Local Variance Filter - is an algorithm used for detecting ventricle of the heart. Since the blood comes inside the ventricle, the grid can become inaccurate because of the blood flow. The algorithm helps to detect where the ventricles are.

  Grid Tracking - this program is described in the Grid Construction section, Program Structure chapter. In the Grid Tracking section there is Grid Construction button that creates grid. Moreover, there are additional spin-boxes with parameters of Curvature Coefficient, Force Coefficient, Stop Time. Once the process of the grid construction finishes, the Grid window appears where the grid can be adjusted.

  Graph Cuts - is an algorithm used for an image segmentation. Helps to segment ventricles from the images after the Local Variance Filter algorithm.

At the bottom of this section there are Run and Cancel Study buttons. The Run button executes study by running Graph Cuts, Local Variance Filter or Grid Tracking. Additionally, any particular program can be selected separately by selecting the radio-button associated with the particular program (LVF, GT, GC).



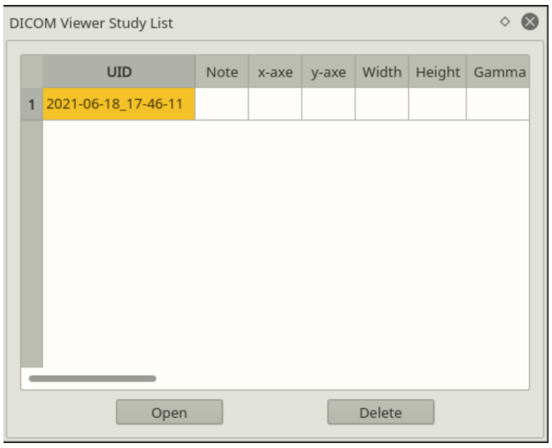
Window Size: 15

Curvature Coefficient : 0,0015

Force Coefficient : 0,5000

Stop Time : 0,0150

Figure 6.7: Cameo GUI - Image Processing tab.


DICOM Viewer Study List

	UID	Note	x-axe	y-axe	Width	Height	Gamma
1	2021-06-18_17-46-11						

Figure 6.8: The Study list window.

The Close Study button stops the current study and the Open Study List button opens small window with the list of studies that was conducted on the given DICOM series.

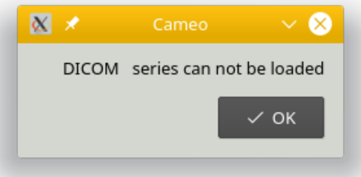
6.3.4 History

In the History section the information about individual actions can be found. It includes the actions of the grid executing, changing the grid parameters, etc.

Figure 6.9: Cameo GUI - History tab.

6.4 Opening file

There are two ways how to open a file: using the file drop-down menu (the first button "Open file") or using a keyboard shortcut Ctrl+O. Both methods lead to the same window where a particular file can be selected. It is supposed to open the DICOM files of the **ima** format. All the files that are in the same folder as the selected file will be opened in the Cameo program. However, if the selected file has a format which is not **ima**, a small message box will appear with a short error message.



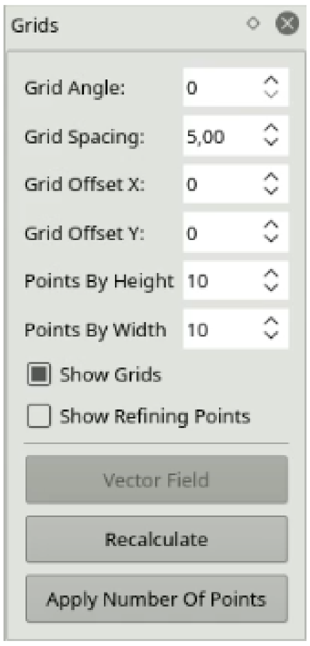
DICOM series can not be loaded

Figure 6.10: Wrong file format message box.

Some buttons are not active because they provide functionality that requires input files. Once the files are loaded, some buttons are unblocked.

6.5 Grid Window

Once the DICOM files are opened, it is possible to start grid construction. When the grid construction finishes, the Grid Window appears on the left side of the main window.


Grids

Grid Angle: 0

Grid Spacing: 5,00

Grid Offset X: 0

Grid Offset Y: 0

Points By Height 10

Points By Width 10

[✓] Show Grids

[ ] Show Refining Points

Figure 6.11: Cameo GUI - Grid Window.

In this window there are some parameters that can be changed and the changes apply immediately.

- Grid Angle - the angle of the given grid. Can be changed from 0 up to 90 degrees.
- Grid Spacing - the distance between nodes. It can be changed from 0 up to 20?
- Grid offset X and Grid offset Y - the distance from the initial coordinates.
- Points By Height and Width - the number of nodes of the grid.

6.6 Available Manipulations With Grid

We made possible a grid manipulation using the manual preferences. Moreover, the grid parameters can be changed using shortcuts.

- Shift + Holding the left mouse button allows moving the grid according the cursor.
- Shift + the Mouse wheel allows changing the angle of the grid.

These new features make the working process with the grid much faster, convenient and intuitive.

6.7 Installation

The required libraries and additional software will be described in this section.

6.7.1 Operating system

The Cameo program was tested on KUbuntu 18.04.

6.7.2 Libraries

The Cameo program requires some libraries in order to work properly. The list of libraries:

- **DCMTK.** This is a set of libraries and applications that allow working with the DICOM file format. This is an open source software includes the ability to browse the DICOM images, creating and changing the file format of DICOM images and more. It is possible to use a command line to install the library:

```
    sudo apt-get install dcmtk libdcmtk2-dev
```

- **Qt library.** It was tested on Qt 5.15.2 with GCC 7.3.1 and GCC 10. It is possible to use command line to install Qt:

```
    sudo apt-get install dcmtk libdcmtk2-dev
```

- **TNL.** This library (Template Numerical Library) was created at CTU at the faculty of Nuclear Sciences and Physical Engineering. The project leader is Tomáš Oberhuber. The main goal of the program is to offer efficient and fast numerical solvers. The implementation is done in the C++ programming language and uses the most recent programming paradigms. As a result, it is more flexible and fast. There are a lot of similar libraries, but most of them are done in the C programming language, It means they are not using objects and it leads to the code complexity. The library is based on the CUDA technology from the Nvidia company. Moreover, TNL supports multi-core processors and video cards.

  The library may require installing the development packages for Debian Linux distributive:

```
    sudo apt-get install build-essential
```

  Then, additional libraries

```
    sudo apt-get install cmake libbz2-dev libcr-dev mpich2 mpich2-doc ↩
        libcppunit-dev
```

  Finally, the system is ready to download the TNL library:

```
    git clone https://mmg-gitlab.fjfi.cvut.cz/gitlab/tnl/tnl-dev.git
```

  The last step is to execute installation from the TNL directory.

```
sudo ./install
```

- **Image-flow.** This library require Java and one more additional library.

  Java:

  ```
  sudo apt-get install openjdk-7-jre icedtea-7-plugin openjdk-7-jdk openjdk↩
      -7-source openjdk-7-doc
  ```

  Additional library:

  ```
  sudo apt-get install libsdl-gfx1.2-dev
  ```

  To apply changes to the TNL library, it is supposed to run installation of the TNL one more time using command **sudo ./install** from the main folder of the TNL.

Conclusion

In this bachelor project, we made the existing application, which is used for displaying and processing the MRI images, executable and stable. We achieved it by deep studying the source code of the whole program. We also became familiar with features of the Qt framework and its tools that can be used for creating the graphical user interface.

Once the most significant bugs were solved, we started working on the graphical user interface testing. It was very important because we made some changes in the existing GUI. It was decided to use the Qt Test because it offers mouse and keyboard simulation. Moreover, the program initially was written using the Qt framework and, therefore, the Qt Test is very suitable for the application. Working on the tests helped to better understand the logic and relationship between the existing GUI elements.

The next step was to improve the initial graphical user interface. The goal was to simplify it and make it more intuitive. We have deleted some buttons and reduced additional actions. Moreover, we added new functionalities for the grid such as rotating by holding the shift button and scrolling the mouse wheel, moving the grid by holding the shift button and clicking to the place where the grid is supposed to be moved. Moreover, we added zoom functionality that allows zooming by holding the control button with the mouse wheel scrolling. Another feature we added is selecting the area of interest using the mouse cursor so the process of selecting the area of interest became very similar to changing size of the window in Windows or Linux. The GUI tests were also changed to work with new user interface.

When the tests were ready, we started working on memory issues. The tool we used is called AddressSanitizer. It is a part of Clang and GCC compilers and it is able to show the memory leaks. We fixed all the memory leaks that the AddressSanitizer found and the memory consumption dropped.

To simplify the work with the application, it was decided to make functionality for the input arguments. As a result, it is possible to run the program from the terminal with input file to open. Moreover, it is possible to run grid construction and pass parameters of the grid as arguments.

To simplify the work with the application, it was decided to make the functionality for the input arguments. As a result, it is possible to run the program from the terminal with the input file to open. Moreover, it is possible to run the grid construction and pass parameters of the grid as arguments.

Bibliography

[1] L. Z. Eng: *Qt5 C++ GUI Programming Cookbook: Practical recipes for building cross-platform GUI applications, widgets, and animations with Qt 5 (2nd Edition)*. Packt Publishing, 2019.

[2] S. C. Bushong, G. Clarke: *Magnetic Resonance Imaging: Physical and Biological Principles (4th Edition)*. Mosby, 2014.

[3] K. Naik, P. Tripathy: *Software Testing and Quality Assurance: Theory and Practice (1st Edition)*. Wiley-Spektrum, 2008.

[4] Moore CC, McVeigh ER, Zerhouni EA. Quantitative tagged magnetic resonance imaging of the normal human left ventricle. Top Magn Reson Imaging. 2000;11(6):359-371. doi:10.1097/00002142-200012000-00005

[5] Dr Joachim Feger: *MR tagging*, Radiopaedia [online], `radiopaedia.org`

[6] Nicholas Sherriff: *Learn Qt 5: Build modern, responsive cross-platform desktop applications with Qt, C++, and QML*. Packt Publishing, 2018.

[7] Qt 5.15, Qt Group, Documentation [online]. `doc.qt.io/qt-5.15/`

[8] Qt Test, Qt Group, Documentation [online], `doc.qt.io/qt-5/qtest-overview.html`

[9] AddressSanitizer, Clang Project, AddressSanitizer Documentation [online], `clang.llvm.org/docs/AddressSanitizer.html`