



Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelor Thesis

Alexei Figueroa

Development of a graphical user interface for X-ray
simulation of computed tomography images

*Fakultät Technik und Informatik
Department Informations- und
Elektrotechnik*

*Faculty of Engineering and Computer Science
Department of Information and
Electrical Engineering*

Alexei Figueroa

Development of a graphical user interface for X-ray simulation of computed tomography images

Bachelor Thesis based on the examination and study regulations for the Bachelor of Engineering degree programme

Information Engineering

at the Department of Information and Electrical Engineering

of the Faculty of Engineering and Computer Science

of the University of Applied Sciences Hamburg

Supervising examiner: Prof.Dr. Robert Heß

Second examiner: Prof. Dr. Marc Hensel

Day of delivery September 8th 2017

Alexei Figueroa

Title of the Bachelor Thesis

Development of a graphical user interface for X-ray simulation of computed tomography images

Keywords

Computed tomography, X-ray simulation, graphical user interface, wxWidgets, model view presenter, C++, Contrast to Noise Ratio

Abstract

The work detailed in the present document builds on top of the software provided by Prof.Dr. Robert Heß and exposes the development of a Graphical User Interface application that supports the workflow of the whole simulation process of a CT system, to serve as a tool for further research into the optimization of the contrast in medical images with the aim of reducing patient radiation doses.

CONTENTS

Abbreviations	vi
1 Introduction	1
2 Theoretical background	3
2.1 Photon Interaction with matter	3
2.2 Computed Tomography	5
2.3 Image Reconstruction and Filtered Back Projection	8
2.4 Contrast to noise ratio	10
2.5 Simulation Workflow	10
2.6 Architecture design patterns	11
2.6.1 Model view controller	11
2.6.2 Model view presenter	12
2.7 wxWidgets.....	13
3 Requirement Specification.....	14
3.1 Previous work and existing code.....	14
3.2 Technical requirements	15
3.3 Functional Requirements	15
3.3.1 Parameters	15
3.3.2 Client area and presentation of results.....	19
3.3.3 Storage and data persistence.....	20
3.3.4 Behavioural aspects	20
3.4 Other Requirements	20

4	Design of the solution.....	22
4.1	General layout of the user interface.....	22
4.2	Simulation State Behaviour.....	23
4.3	Architecture and separation of concerns	27
5	Implementation Details	32
5.1	Graphical Layout	32
5.2	Extended functionality on previously existing classes	42
5.2.1	Phantom related modifications	42
5.2.2	cInputData modifications.....	43
5.2.3	Other modifications	46
5.3	Additional classes.....	46
5.3.1	Model Classes.....	46
5.3.2	View Classes	47
5.3.3	Presenter Classes	48
6	Test and validation	50
6.1	Data persistence tests.....	50
6.2	Parameters tests	51
6.3	GUI state behaviour tests.....	52
6.4	Comparison test with command line application	54
7	Summary	58
7.1	Further work	58
	List of tables	60
	List of figures	61
	References	63
	Appendix A.....	65

Abbreviations

CNR Contrast to noise ratio

CT Computed tomography

CTN Computed tomography number

FBP Filtered back-projection

GUI Graphical User Interface

MVC Model view controller

MVP Model view presenter

ROI Region of interest

SNR Signal to noise ratio

1 Introduction

The discovery of the X-rays by W.C Roentgen in the late XIX century led to a tremendous development in the field of medical imaging. The fact that non-invasive procedures can yield the present level of accurate description of the internal constitution of the human body is as impressive and major as it is the work of all the scientists and engineers that have developed the related technologies. Computed Tomography (CT) scanners are used worldwide to support medical assessment, their capability of delivering a high contrast three-dimensional (in some cases with motion) reconstruction model of the examined human body sections make them not only intuitive for medical practitioners but also far superior to the 2D radiographic techniques [1].

Unfortunately, the leverage of the benefits of all the X-ray based medical imaging technologies, comes at the cost of exposing the subjects of examination to an unnatural level of radiation that brings with itself an inherent risk of developing cancer. This level of exposure, generally qualified as *dose*, can certainly be controlled. However, it is proportional to the accuracy of the results of CT implying that a lower dose yields an undesired lower level of certainty in the medical images. The latter comprises a critical point of concern for all the people developing and using the CT technologies.

Due to the underlying requirement for medical practitioners to be able to distinguish the different structures and tissue composition of the human body, the accuracy of the resulting models (or images) that are produced by CT-scanners can be characterized with the help of *Contrast to Noise* analysis. This measure can be thought of a target function for optimization of the imaging system, in the spirit of subjecting the patients to the lowest radiation dose possible achieving the needed accuracy of results for sensible medical assessment. This is imaginable since there are many variables related not only to the patient's body structure, the examined body parts, but also the CT-scanner itself that can be fine-tuned and alter the contrast behaviour in the resulting images.

Experimentally the process of optimizing the contrast of an image produced by a CT system involves several measurements for different configurations. This is not a tractable endeavour considering the logistics required for such experiments as well as the multidimensional space of all the variables that need to be explored. Nevertheless, the mathematical description of

the physical phenomenon taking place in a CT-scanner is possible and the optimization of the contrast in the resulting medical images can be tackled by simulating the system with software.

Prof. Dr. Robert Heß ,lecturer from the Hochschule für Angewandte Wissenschaften(HAW) Hamburg implemented such simulation software, namely a mathematical model describing the physical functioning of a CT-scanner for a single slice two-dimensional reconstructed image. This is materialized in a command line application developed using the C++ programming language. The accuracy of this model is validated to the extent of the work presented in [2].

The work detailed in the present document builds on top of the software provided by Prof.Dr. Robert Heß and exposes the development of a Graphical User Interface application that supports the workflow of the whole simulation process of a CT system, to serve as a tool for further research into the optimization of the contrast in medical images with the aim of reducing patient radiation doses.

2 Theoretical background

2.1 Photon Interaction with matter

X-rays are known to have a very high material-dependant capability of matter penetration. However, the number of photons i.e., the radiation intensity, decreases exponentially while running through an object along the incident direction. This attenuation is due to absorption and scattering. The main phenomena driving this attenuation for the energy levels of medical imaging CT scanners are the photoelectric absorption, the Compton and the Rayleigh scattering [3].

The transmission of a monoenergetic photon *pencil-beam* through a material is described by an exponential equation

$$I(x) = I_0 e^{-\mu_l x} \quad (2.1)$$

Where $I(x)$ is the beam intensity transmitted through a thickness x of absorber, $I(0)$ is the intensity recorded with no absorber present, and μ_l is the linear attenuation coefficient of the absorber at the photon energy of interest [4]. This coefficient is an additive combination of both scattering and absorption coefficients which are material dependent [3].

If the attenuation coefficients are extended to vary spatially and depend on Energy (2.1) can be extended as follows:

$$I(s) = \int_0^{E_{max}} I_0(E) e^{-\int_0^s \mu(E,x) dx} dE \quad (2.2)$$

Where the argument of the exponential function is the line integral summing up the attenuation coefficients for an energy level over a distance s [3].

The physical processes that are modelled with (2.1), namely the photoelectric effect, the Compton and Rayleigh scattering are briefly described next.

The photoelectric effect

This effect is an atomic absorption process in which an atom absorbs totally the energy of an incident photon. The photon disappears and the energy absorbed is used to eject an orbital electron from the atom. The ejected electron is called photoelectron [4].

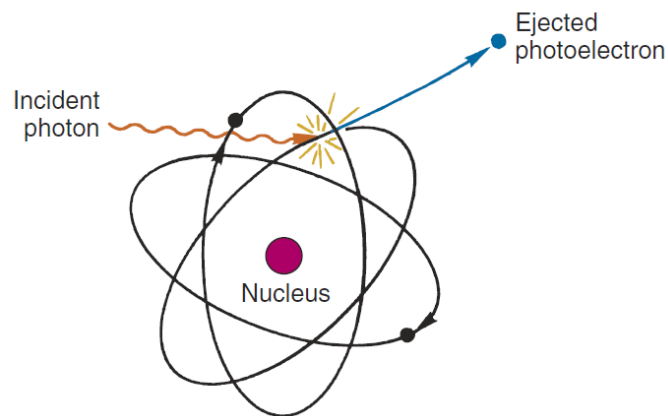


Figure 2.1 Schematic representation of the photoelectric effect [4]

Compton scattering

This effect is a collision between a photon and a loosely bound outer-shell orbital electron of an atom. Since the incident photon energy greatly exceeds the binding energy of the electron to the atom, the interaction looks like a collision between the photon and a free electron. The photon does not disappear, instead it is deflected through a scattering angle θ . The energy of the scattered photon is related to the scattering angle θ as follows:

$$E_{sc} = \frac{E_0}{1 + \frac{E_0}{0.511}(1 - \cos \theta)} \quad (2.3)$$

Where E_0 and E_{sc} are the incident and scattered photon energies in MeV , respectively [4].

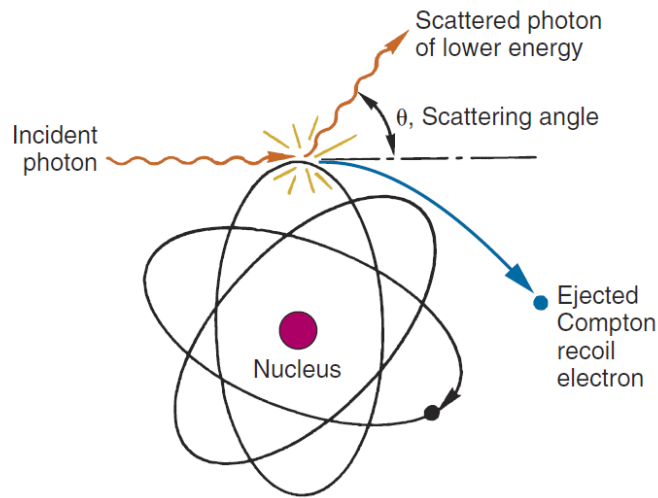


Figure 2.2 Schematic representation of Compton scattering [4]

Rayleigh scattering

This type of scattering interaction occurs between a photon and an atom as whole. Because of the considerable mass of the atom, very little recoil energy is absorbed by the atom and the incident photon is therefore deflected with essentially no loss of energy [4].

2.2 Computed Tomography

The main purpose of Computed Tomography is to measure and compute the spatial distribution of the linear attenuation coefficients $\mu(x, y)$. This value also often referred to as CT value is normalized to the attenuation coefficient of water. For an arbitrary tissue T with attenuation coefficient μ_T the CT value is defined as:

$$CT\ value = \frac{(\mu_T - \mu_{water})}{\mu_{water}} * 1000\ HU \quad (2.4)$$

Where HU stands for Hounsfield units in honour of the inventor of CT. The CT value scale is defined by the two fixed points “air = -1000HU” and “water = 0 HU”. For every CT scanning unit, these fixed points are set using phantom measurements for each tube voltage value and each x-ray filtration choice available [1].

Figure 2.3 illustrates a frontal view of a third-generation CT scanner with its main geometrical features. The fan angle is described by φ and $\Delta\varphi$ stands for the thickness of a single detector.

In these systems, the X-ray source moves along a circle trajectory determined by the coordinates $(-FCD \sin \gamma, FCD \cos \gamma)$ if the rotation of the system is counter clockwise. FCD stands for the focus centre distance, and γ stands for the projection angle defined by the central beam [3].

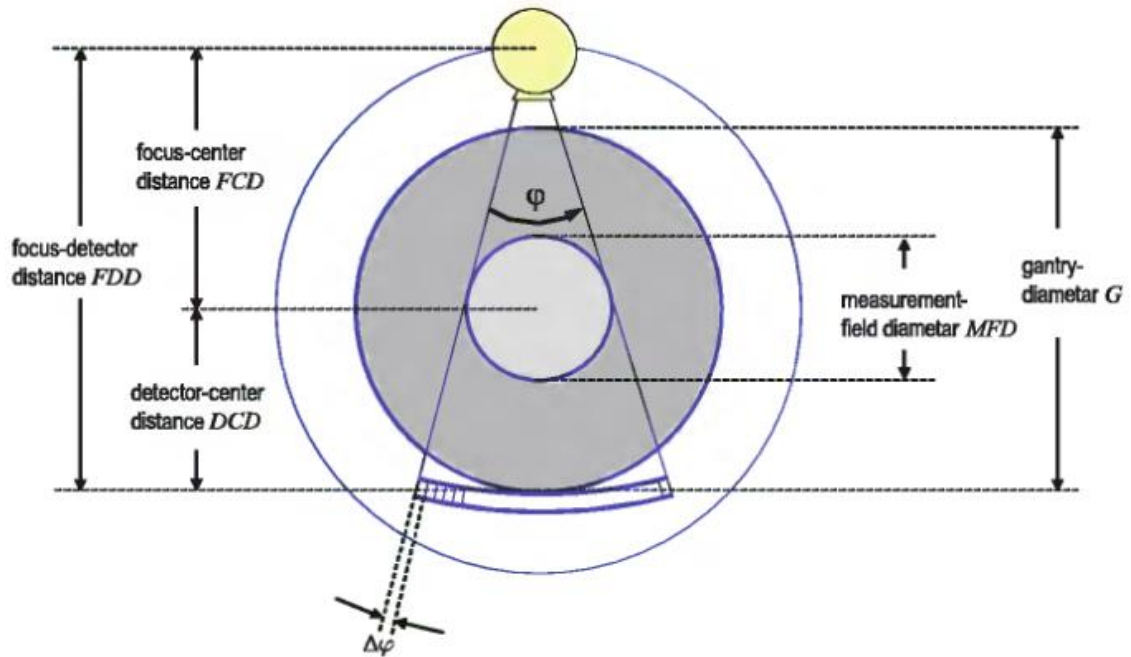


Figure 2.3 Geometry of Fan-beam CT scanners of the third generation [3]

Even though systems of the third generation are mainly used in practice, the geometry of parallel projections i.e. systems of the first generation of CT (*pencil-beam*), is more intuitive to follow the mathematics of image reconstruction. In these systems, the X-ray source moves in parallel to the array detectors. Figure 2.4 presents an illustration of such geometry here the rotation angle of the X-ray source is depicted as γ .

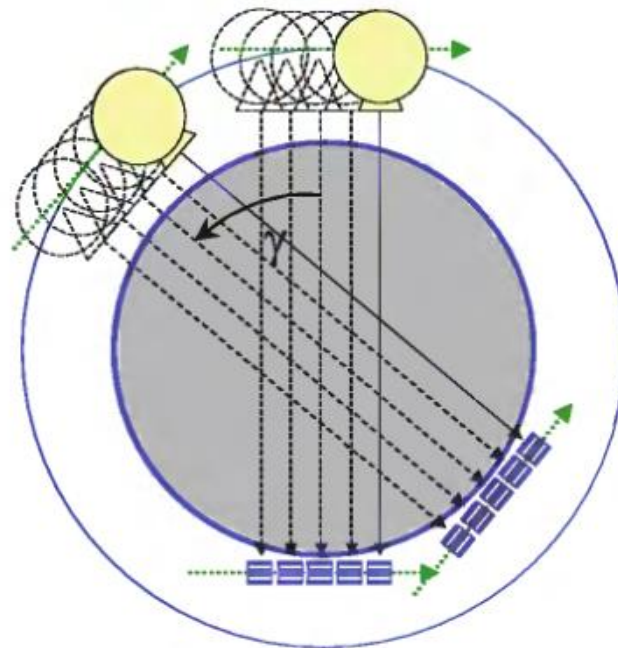


Figure 2.4 Illustration of a first-generation pencil beam CT system [3]

The detector array of the CT scanner measures an intensity projection (*shadow*) for each angle in the rotation of the X-ray source. When these projections are arranged as a function of the rotation angle γ and the detector ξ a **Sinogram** is achieved. This process follows the mathematical description of the Radon transform which for a two-dimensional case would stand as follows:

$$R_2\{f(x, y)\} = p(\gamma, \xi) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) \delta(x \cos \gamma + y \sin \gamma - \xi) dx dy \quad (2.5)$$

Where δ is the Dirac delta function, $x \cos \gamma + y \sin \gamma$ is the Hesse normal form of a line L with a normal vector pointing to the origin of magnitude ξ spanning an angle γ (in the context of CT the path of an X-ray beam) and $f(x, y)$ is the function transformed [3] [1]. A simple interpretation of this equation is to focus on the inner integral and use the sifting (masking) property of the δ function over the parametrized line L , then the outer integral only denotes the summation of the values of $f(x, y)$ along this line. Intuitively what is measured by the detectors of CT scanner is the X-ray intensity and not directly the attenuation coefficients of the test object. However, from (2.2) the line integral of the attenuation coefficients equals the natural logarithm of the ratio of the incident intensity and the intensity measured by the detectors [3] [1].

Figure 2.5 **b** shows the Radon transform of a simple object composed of two rectangles depicted in Figure 2.5 **a**. Here the Radon transform is plotted in a Cartesian scheme showing the angle of the rotation of the system γ and the detector position ξ . From this graphical perspective, it is clear how the Sinogram name became a synonym of the Radon transform in the context of CT.

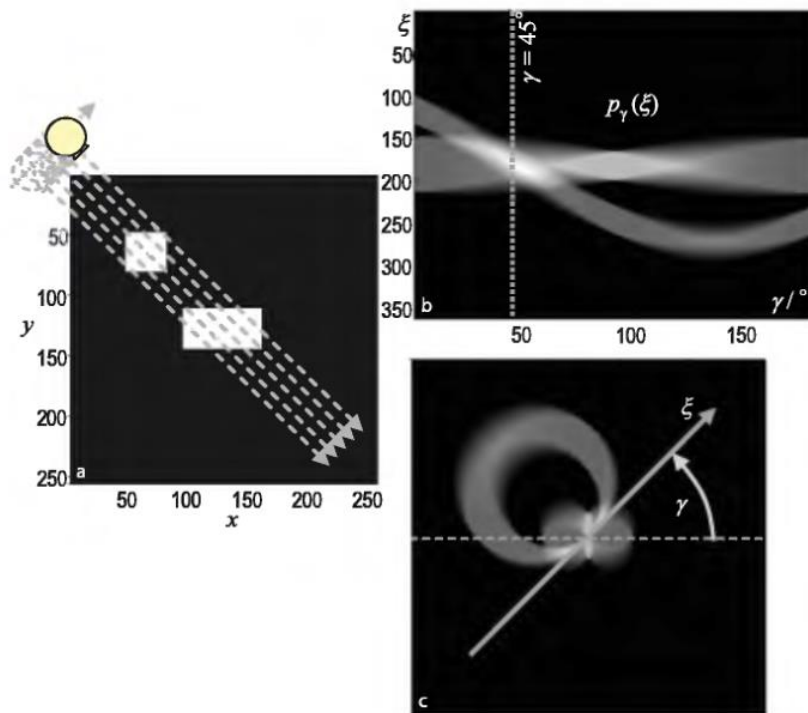


Figure 2.5 Cartesian Radon space, Sinogram of a synthetic image [3]

- a)** Synthetic 256x256 pixel image. The homogeneous attenuation values of two objects are simulated by gray values set to 1. **b)** Cartesian Radon space of synthetic image provided in a (*Sinogram*). **c)** Radon space of the synthetic image in polar coordinates

2.3 Image Reconstruction and Filtered Back Projection

Image reconstruction implies finding the inverse Radon transform. For this purpose, the Fourier-slice theorem is introduced. Computing the Fourier transform of $p(\gamma, \xi)$ (2.5) with

respect to the variable ξ and using the masking property of the δ function on the argument of the exponential function:

$$P(\gamma, u) = \int_{-\infty}^{\infty} p(\gamma, \xi) e^{-2\pi i u \xi} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) e^{-2\pi i u (x \cos \gamma + y \sin \gamma)} dx dy \quad (2.6)$$

Comparing this result to the two-dimensional Fourier transform of $f(x, y)$:

$$F(u_x, u_y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) e^{-2\pi i (u_x x + u_y y)} dx dy \quad (2.7)$$

The identity that is known as the Fourier slice theorem is shown:

$$F(u \cos \gamma, u \sin \gamma) = P(\gamma, u) \quad (2.8)$$

Which can be interpreted as the Fourier transform of the projections with respect to the distance parameter ξ being equal to the Fourier transform of the object ($f(x, y)$) expressed in polar coordinates $(u_x, u_y) = (u \cos \gamma, u \sin \gamma)$.

Performing the inverse Fourier transform of equation (2.7) with respect to the parametrization of $u_x = u \cos \gamma$ and $u_y = u \sin \gamma$ with the substitution of the variables of integration to u and γ with $du_x du_y = u du d\gamma$ the following result is achieved:

$$f(x, y) = \int_0^{\pi} \int_{-\infty}^{\infty} |u| P(\gamma, u) e^{2\pi i u (x \cos \gamma + y \sin \gamma)} du d\gamma \quad (2.9)$$

The inner integral of the equation corresponds to the multiplication of the Fourier transform of a projection on an angle γ with a ramp function $|u|$ which corresponds to a convolution in the spatial domain or a filtering process, in this case a high-pass impulse response. Finally, the outer integral corresponds to the actual back-projection i.e. the integration of the filtered projection over a sinusoidal curve $\xi = x \cos \gamma + y \sin \gamma$. This analytical method of deriving $f(x, y)$ from the Radon transform $p(\gamma, \xi)$ is called Filtered Back-Projection (FBP) [3] [1].

2.4 Contrast to noise ratio

One of the most important features of CT in comparison to radiography is the capability of producing images of higher contrast. Image contrast is defined by the difference in intensity of two neighbouring picture elements or regions [1]. Noise is an inherent component in a CT image and an important measure to characterize image quality is the signal to noise ratio SNR:

$$SNR = \frac{\text{signal level}}{\text{noise level}} = \frac{\mu}{\sigma} \quad (2.10)$$

Where μ and σ denote the mean and standard deviation of a signal. Unfortunately, the SNR in the context of CT is proportional to the square root of the dose, hence it can not be arbitrarily increased [3]. When the ideas of image contrast and SNR are considered for assessing image quality the contrast-to-noise ratio CNR is introduced:

$$CNR = \frac{\overline{CTN_1} - \overline{CTN_2}}{\sqrt{\frac{\sigma_1^2 + \sigma_2^2}{2}}} \quad (2.11)$$

Where $\overline{CTN_1}$, $\overline{CTN_2}$ and σ_1 , σ_2 are the mean and standard deviation of the CT values of two regions of interest (ROI) defined on the CT image. This expression is of great interest when comparing results coming from CT images of different systems, the work in [2] uses this ratio to compare the resulting images of a CT scanner and a simulation.

2.5 Simulation Workflow

The simulation process can be subdivided into 4 main activities:

- Setting the simulation parameters.
- Simulating the X-Ray transmission through an object.
- Reconstructing the raw transmission data into an image.
- Analysing the reconstructed image.

An activity diagram of the overall workflow of an uninterrupted simulation is shown in Figure 2.6. Each of the stages is dependent upon results of one or more previous stages.

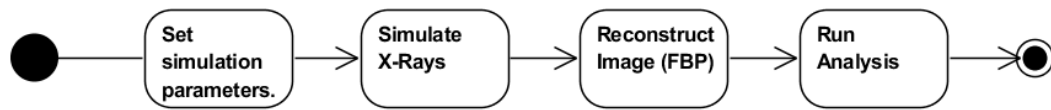


Figure 2.6 Activity diagram of a simulation workflow

Generally, the simulation parameters can be subdivided into the following categories:

1. Materials.
2. Geometry of the X-Ray system.
3. Incident beam properties.
4. Test object (Phantom) properties.
5. Image reconstruction settings.
6. Regions of interest for the analysis.

The *X-ray simulation* stage uses as inputs the settings of: the materials, the geometry of the system, the incident beam and test object data, and yields as an output the Sinogram (or Radon transform) of both the X-ray transmission as well as the noise in the form of standard deviations. This implies that the image reconstruction and analysis settings are independent from this stage and modifying them will not cause any effect in its result.

The *image reconstruction* stage only takes into consideration its settings and the Sinogram and noise results coming from the X-ray simulation. Thus, this stage produces an image resulting from the filtered back projection algorithm of both sinograms.

Finally, the *analysis* stage uses the definition of the regions of interest as well as an already reconstructed image, and yields as outputs measured values, namely mean, standard deviation and pixel count per region of interest.

2.6 Architecture design patterns

2.6.1 Model view controller

The model view controller MVC architecture considers three roles. The *Model* is an object that represents some information about the domain. It encapsulates the appropriate data, and exports application-specific processing procedures. *Controllers* call these procedures on behalf of the user. The *Model* also provides methods to access its data that are used by *View* components to acquire the information to be displayed. The *View* represents the display of

the *Model* in the user interface. The *View* only displays information, the actual changes to the information are handled by the *Controller*. The *Controller* takes the user input and manipulates the *Model* which causes the view to update appropriately. The change-propagation mechanism maintains a registry of the dependent components within the *Model*. All the *Views* and selected *Controllers* register their need to be informed about changes. Changes to the state of the model trigger the change-propagation mechanism [5] [6].

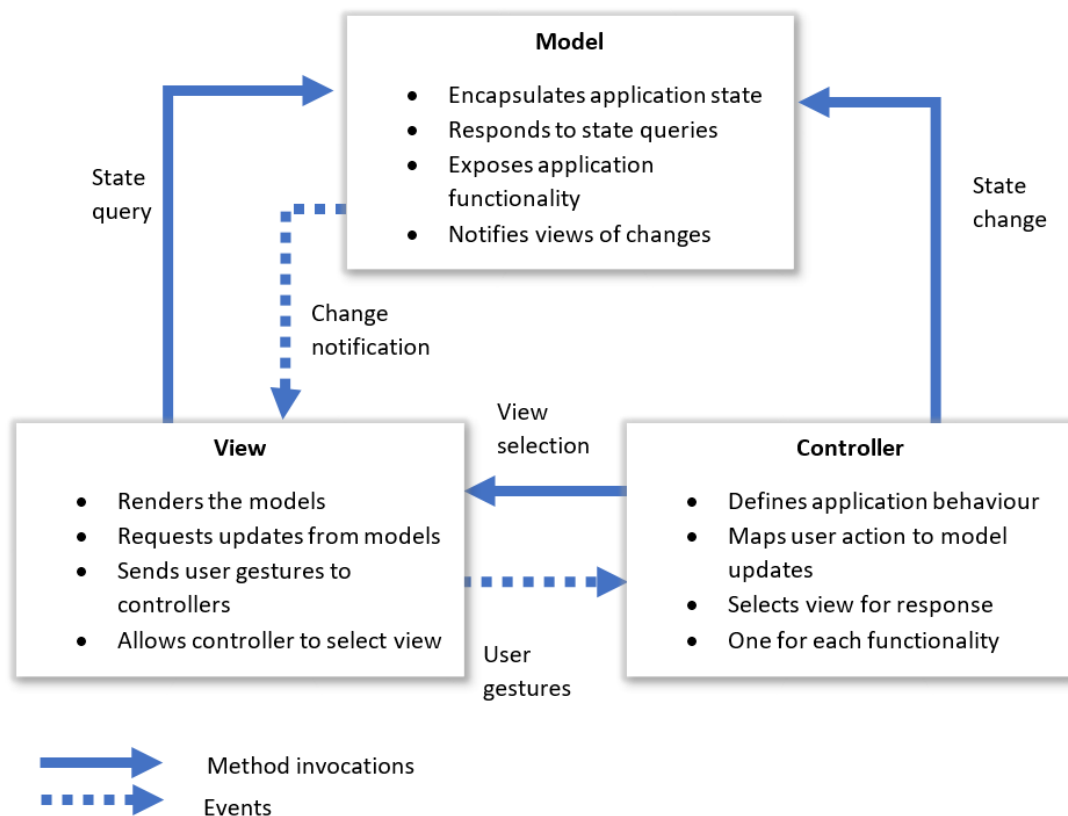


Figure 2.7 MVC role description and interactions [7]

2.6.2 Model view presenter

Model view presenter leverages on the ideas of MVC and aims as well for the separation of the functionality among the different abstractions within an application with a GUI. The View in MVP holds the structure of the widgets that compose a GUI, it doesn't contain a definition

of how these widgets should behave to user interaction. The handling of the actions started by the user take place in the Presenter. The fundamental handlers of user input exist in the widgets however, they are forwarded to the Presenter. The update of the View by the Model could happen in the same manner as in MVC shown in Figure 2.7, nevertheless one of the variations of this approach: *Passive View* involves a complete update of the View guided by the Presenter. This approach is highly beneficial for testing purposes [7] [8]. Figure 2.8 depicts the roles and interactions of MVP using a *Passive View*.

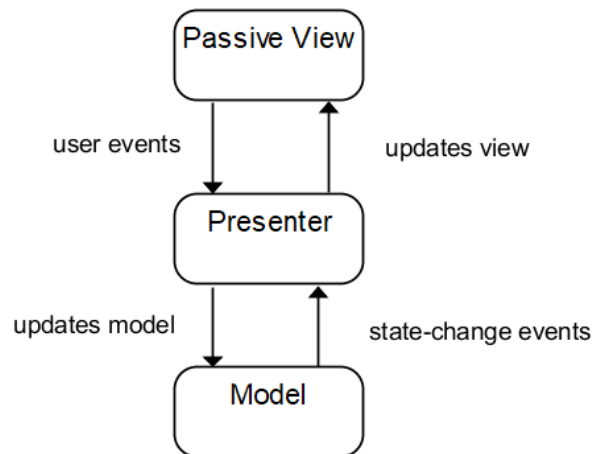


Figure 2.8 Interactions between the roles of MVP [10]

2.7 wxWidgets

wxWidgets is an open source toolkit for writing desktop or mobile applications with graphical user interfaces (GUIs) in the C++ programming language. It is a framework in the sense that provides default application functionality and eases the software implementation. The wxWidgets library contains a large number of classes and methods for the programmer to use and customize, it is a cross-platform framework that besides providing GUI facilities also provides classes for files and streams, multiple threads, application settings, inter-process communication, online help, database access among others.

A very distinguishing feature of wxWidgets is the fact that it provides native look and feel. wxWidgets uses the native widgets of the Operating System (OS) where it is deployed whenever it is possible. wxWidgets is a very mature project with a large and very active community [4].

3 Requirement Specification

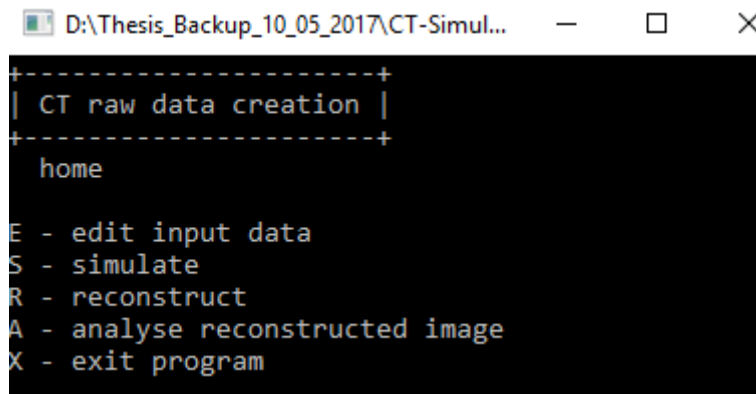
In a general statement the main requirement of this work is to provide a graphical user interface application that fully encapsulates the workflow of the simulation of X-rays in the context of computed tomography images shown in 2.5.

Target users of the solution are preferably engineering students or professionals with an affinity for software development and the X-ray field of studies. This means that the software delivered with this work is not intended for commercial purposes. All the concepts, parameters, views and layout assume from the final user a minimum degree of familiarization not only with X-rays but also technicalities such as the programming languages used, the frameworks, libraries and architecture of the software deliverable of this thesis. The latter implies that the spirit of the software presented in this document is to be a tool that can also be extended by the user.

3.1 Previous work and existing code

As a starting point for the development of the main deliverable of this work, a fully functional command line application was provided. It supports the whole simulation workflow described in 2.5. All the algorithms following the mathematical models that simulate the X-ray transmission, the filtered back projection as well as the analysis on the regions of interest is readily available.

A copy of this work is included within the digital material attached to this document. This application was developed fully using the C++ programming language and the Microsoft Visual Studio IDE.



```
D:\Thesis_Backup_10_05_2017\CT-Simul...
+-----+
| CT raw data creation |
+-----+
home

E - edit input data
S - simulate
R - reconstruct
A - analyse reconstructed image
X - exit program
```

Figure 3.1 Home screen of the command line application for simulation of CT

3.2 Technical requirements

The graphical user interface application delivered with this work must comply with the following technical specification:

- It must be programmed using the C++ language and built with the MSVC 14 compiler or more recent.
- It is deployable under a Windows 10 environment.
- It is as responsive timewise at least as the existent command line application.
- It capitalizes on solid object orientation to provide further extensibility of the abstractions already in place, namely the test-object related classes.
- It reuses the functionality already available in the implemented classes that are related to the mathematical models used for simulation.

3.3 Functional Requirements

3.3.1 Parameters

In the spirit of enhancing the final user experience in the sense of conducting different scenarios of the whole process of simulation, the functionality of the graphical user interface implemented must support modifying all the parameters already in use by the algorithms provided for X-ray simulation, image reconstruction and analysis in the existing command line application. Furthermore, the outputs of each of these simulation stages must be presented at runtime within the working areas of the interface. The parameters that must be configurable through the delivered application are listed next. All the required value ranges for the parameters already in place in the command line application provided are taken as a basis, nevertheless a couple of additional fields are introduced.

Materials

The delivered software should support the management of an arbitrary number of materials, namely it must provide the functionality for adding, deleting and editing them. Every material consists of an arbitrary number of elements. Each element considers an atomic number and a fraction. Additionally, every material must have a name, a density and can be assigned a colour to be painted within the model of the test objects.

The ranges of the mentioned parameters can be summarized as follows:

Field	Existing parameter representation	Value range
Material name	Alphanumeric string	N/A
Density	Number with 4 decimal places	0.001,...,999
Colour	N/A	0,...,2 ²⁴
Atomic number of Elements	Integer number	1,...,100
Fraction	Number with 4 decimal places	1,...,100

Table 3.1 Material parameters

Geometry

The geometry of the CT system modelled by the software can be summarized by the following parameters and their respective value ranges:

Field	Existing parameter representation	Value range
Radius of focal spot	Integer number	1,...,1000
Fan angle in degrees	Integer number	1,...,180
Number of detectors	Integer number	1,...,9999
Detector thickness	Integer number	0,...,99
Number of projections	Integer number	1,...,9999

Table 3.2 Geometry parameters

Incident beam

The incident beam parameters are summarized as follows:

Field	Existing parameter representation	Value range
Tube voltage	Integer number	80,100,120,140

Beam hardening material	Integer number	Representing an index of the existing materials
Current-time product	Number with 4 decimal places	0.0001,...,1000000

Table 3.3 Incident beam parameters

Additionally, physical filters are simulated in the trajectory between the X-ray tube and the test object. Functionality for adding, editing and deleting these filters must be provided. Each of these filters has a name, material and thickness, and can be also defined by an arbitrary number of *bowtie samples* which are defined by an angle with respect to the focal point of the beam and a thickness in millimetres.

Field	Existing parameter representation	Value range
Filter name	N/A	Alphanumeric string
Material	Integer number	Representing an index of the existing materials
Thickness	Number with 2 decimal places	0,...,20
Angle of bowtie sample	Integer number	0,...,15
Thickness at bowtie sample	Integer number	0,...,200

Table 3.4 Filter parameters

Test object, (Phantom)

Two existing two-dimensional test objects were provided with the command line application and the software delivered should provide basic functionality to manage them. These test objects and their respective configurable settings are presented next.

Head phantom: composed of 6 cylinders – circles in the context of a single slice. These represent a *cover*, a *filling* and 4 *details*. Each of the *details* has a radius, a position defined by x and y coordinates and an index corresponding to a material in a list of all the available materials. The parameters for a single *detail* and their corresponding value ranges are as follows:

Field	Existing parameter representation	Value range
Radius	Number with 1 decimal place	0.1,...,1000
X position	Integer number	Representing an index of the existing materials

Y position		Integer number	0,...,20
Material Index		Integer number	0,...,15

Table 3.5 parameters of the cylindrical geometry of head and cylinder phantoms

The parameters for the *cover* and the *filling* are only the radius and material index, the combination of the two radii defines the thickness of the cover and the extent of the filling.

Cylinder phantom: described geometrically as a single cylinder. Just like the *cover* or *filling* it is only defined by the cylinder radius and a material index.

An illustration of a possible 2-dimensional cross section of these test objects is shown in Figure 3.2.

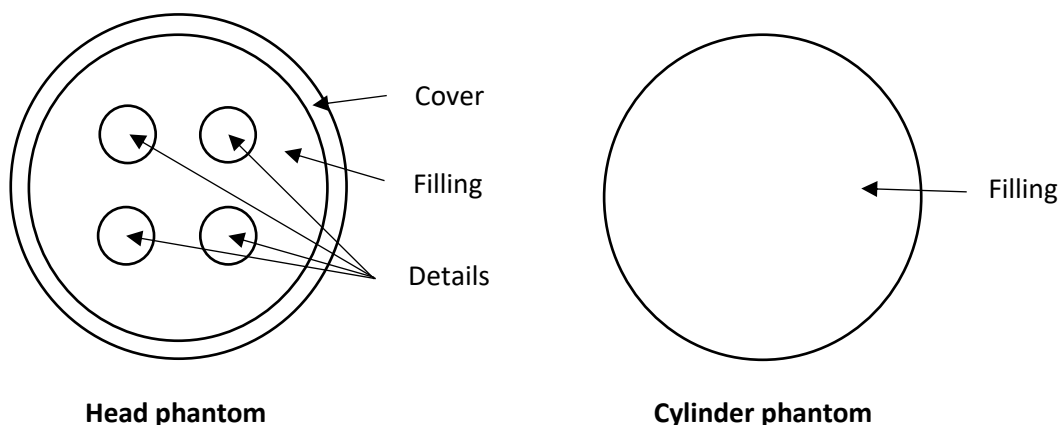


Figure 3.2 Illustration of the geometry of the existing test objects

Image Settings

The parameters of the image reconstruction can be summarized as follows:

Field	Existing parameter representation	Value range
Number of pixels in x direction	Integer number	1,...,4096
Number of pixels in y direction	Integer number	1,...,4096
Centre position in x direction	Integer number	-500,...,500

Centre position in y direction	Integer number	-500,...,500
Pixel size	Number with 4 decimal places	0.001,...,100

Table 3.6 parameters of the reconstructed image

Regions of interest (ROIs)

The software is meant to support an arbitrary number of regions of interest for the analysis of the reconstructed image. Each region of interest is defined by a shape, its size and coordinates, and can be added, edited or deleted. The parameters per ROI can be summarized as follows:

Field	Existing parameter representation	Value range
ROI type	A discrete value	Disc, rectangle
Centre position in x direction in mm	Integer number	-999,...,999
Centre position in y direction	Integer number	-999,...,999
Size in x direction	Number with 4 decimal places	0.001,...,999
Size in y direction	Number with 4 decimal places	0.001,...,999

Table 3.7 parameters of regions of interest ROIs

3.3.2 Client area and presentation of results

As shown in section 2.5, the X-ray simulation, image reconstruction and image analysis yield results that are of interest for the user. The presentation of these must be undertaken within the application to be delivered, namely:

- Two Sinogram images resulting from the x-ray transmission simulation, one for the noiseless transmission data and one for the noise.
- The image resulting from the reconstruction.
- The count of pixels, mean and standard deviation for each ROI defined for the analysis.

Additionally, each one of these images and data could be exported to a file of their own. i.e. The images to a compatible image format, and the results of the analysis to a CSV (comma separated values) file. Furthermore, the results coming from different analysis runs within the same session are not discarded but also displayed and can be exported.

Finally, each of the modelled test objects currently existing (head and cylinder phantom) should support the means of providing an illustration of their geometry consistent with their respective settings as well as the material configuration. This illustration should be drawn as well in the client area.

All the data that is presented as images in the client area needs to be drawn following the directions of the axes: horizontal (x-axis) pointing to the right, and vertical (y-axis) pointing down.

3.3.3 Storage and data persistence

It is of paramount importance to be able to reproduce simulation results, for this the settings of each individual simulation stage need to be persistent over time. To achieve this the implementation of the deliverable software must support loading and storing of the parameters so that the algorithms running the actual simulation can yield the same results over different sessions.

This functionality must be implemented using the XML object description language in files of the same extension. The provided command line tool already supports this behaviour for some of the classes using the C++ *libxml* library, however a formal requirement is to re-implement the existing functionality and extend it by the usage of the XML facilities provided with the *wxWidgets* framework.

3.3.4 Behavioural aspects

The command line application provided as basis for this work splits the simulation process into stages as per chapter 2.5. This incremental division of the simulation is also required for the GUI application involving as well the presentation of the intermediate results. Emphasis is placed in the fact that the application state, i.e. the results displayed and controls used must coherently follow the state of the simulation.

3.4 Other Requirements

Open source dependencies: As stated earlier the software deliverable is intended as a tool that ideally will be further developed and will serve mainly academic and research purposes. Building on top of open source libraries and finally delivering as an open source is an expected feature.

wxWidgets is the framework to be used to build the graphical user interface, this is coherent with the open source perspective and encouraged by the first examiner of this work. A

modern graphic layout is expected, namely a ribbon top menu with a client area for presenting purposes.

Intermediate result data persistence: Strictly speaking the overall input parameters of a complete simulation should be enough to reproduce the intermediate results of the composing simulation stages, i.e. the Sinograms, the reconstructed images and the corresponding metrics for each ROI. Nevertheless, providing storage of these intermediate results could be of use for future analysis, considering the time and computational cost of the simulation.

4 Design of the solution

4.1 General layout of the user interface

Following the requirements specified in the previous chapter, the navigation through the application must be implemented by using a top ribbon bar which supports the following functionality:

- **Management of the session data of the simulation:** Storing and loading simulation files with their relevant settings.
- **Parameters modification:** Access to all the parameters as per subsection 3.3.1 for the simulation process.
- **Control of the simulation workflow:** The application provides access to the algorithms for X-ray simulation, reconstruction and analysis of regions of interest. Furthermore, the application follows a state behaviour coherent with the simulation workflow.
- **Visualisation of results:** For each simulation that provides an output the application must be able to present it in a sensible manner for the user.

Following this guideline, the main window's application layout is subdivided as follows:

- A ribbon bar with the following panels:
 - File: with the controls needed for loading and saving a file including the intermediate results available at current state of the simulation.
 - X-ray settings and materials: With the controls needed to add, delete and edit materials, as well as access to the parameters regarding the CT system geometry, incident beam and test object.
 - Reconstruction and analysis: consisting of the controls providing access to the image settings and regions of interest (ROIs).
 - Simulation control and view: comprising the required controls to run each stage of the simulation as well as to set the displayed results in the client area.
- A client area that presents the intermediate results of the simulation, i.e. Sinograms of x-ray transmission and noise, the reconstructed image and the metrics for each region of interest specified.

The grouping of the user interface controls in such ribbon pages is due to mainly the functionality previously detailed secondly: a coherence for the intermediate results that the parameters relate to and a less importantly: the space of the window (e.g. Reconstruction and analysis page)

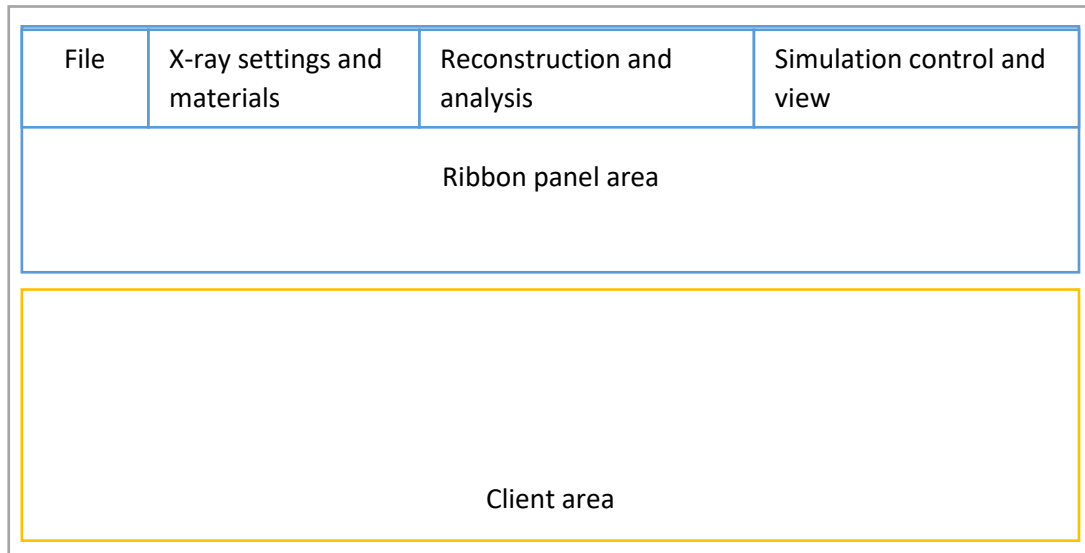


Figure 4.1 Layout of the main window of the application

Some of the parameters specified in chapter 3 support an arbitrary number of instances. These are mainly the materials, the filters and the regions of interest. Instances of these abstractions can be added, deleted and modified. To accomplish this functionality separate dialogs are needed for adding and editing each of the instance's parameters. Additionally, each of the test objects implements its own dialog to provide access to its parameters. To summarize the additional dialogs needed are:

- Material dialog
- Filter dialog
- ROI dialog
- Head-phantom dialog
- Cylinder dialog

Each of them gives access to the corresponding parameters as per section 3.3.1.

4.2 Simulation State Behaviour

In the context of the simulation workflow detailed in section 2.5, the direct independence of some settings from different simulation stages leads to a more general categorization of the parameters.

- X-ray simulation parameters
- Reconstruction parameters
- Analysis parameters

The next table summarizes the preconditions and results yielded by each activity given this grouping of the parameters.

Activity	Preconditions	Results
X-ray simulation (Radon transform)	X-ray Simulation parameters	Sinogram of X-ray transmission and Noise
Reconstruct Image (Filtered Back Projection)	Reconstruction parameters, Available Sinogram data of X-ray transmission and Noise	Reconstructed image
Analysis	Analysis parameters, available Reconstructed Image	Pixel count, mean and standard deviation of regions of interest specified in the reconstructed image.

Table 4.1 Preconditions and results of the simulation stages

Figure 4.2 presents an alternative activity diagram considering again an uninterrupted simulation, however this time the configuration of the parameters is split following the logic above, and hence happening before each corresponding simulation stage that produces an output. This new outline also introduces validation points after all the simulation stages that generate outputs, redirecting the flow to the corresponding parameter configuration. This different perspective of the workflow of the whole process is relevant since it illustrates the iterative nature of experimentally testing the outcomes of the mathematical model behind each simulation stage i.e. how do the results vary with respect to changes in the corresponding settings. Even though this perspective of the simulation process is slightly more complex, it depicts how the overall workflow can be further subdivided, and provided the results of all the intermediate stages can be stored, this alternative workflow shows as well how the simulation could be possibly interrupted and resumed.

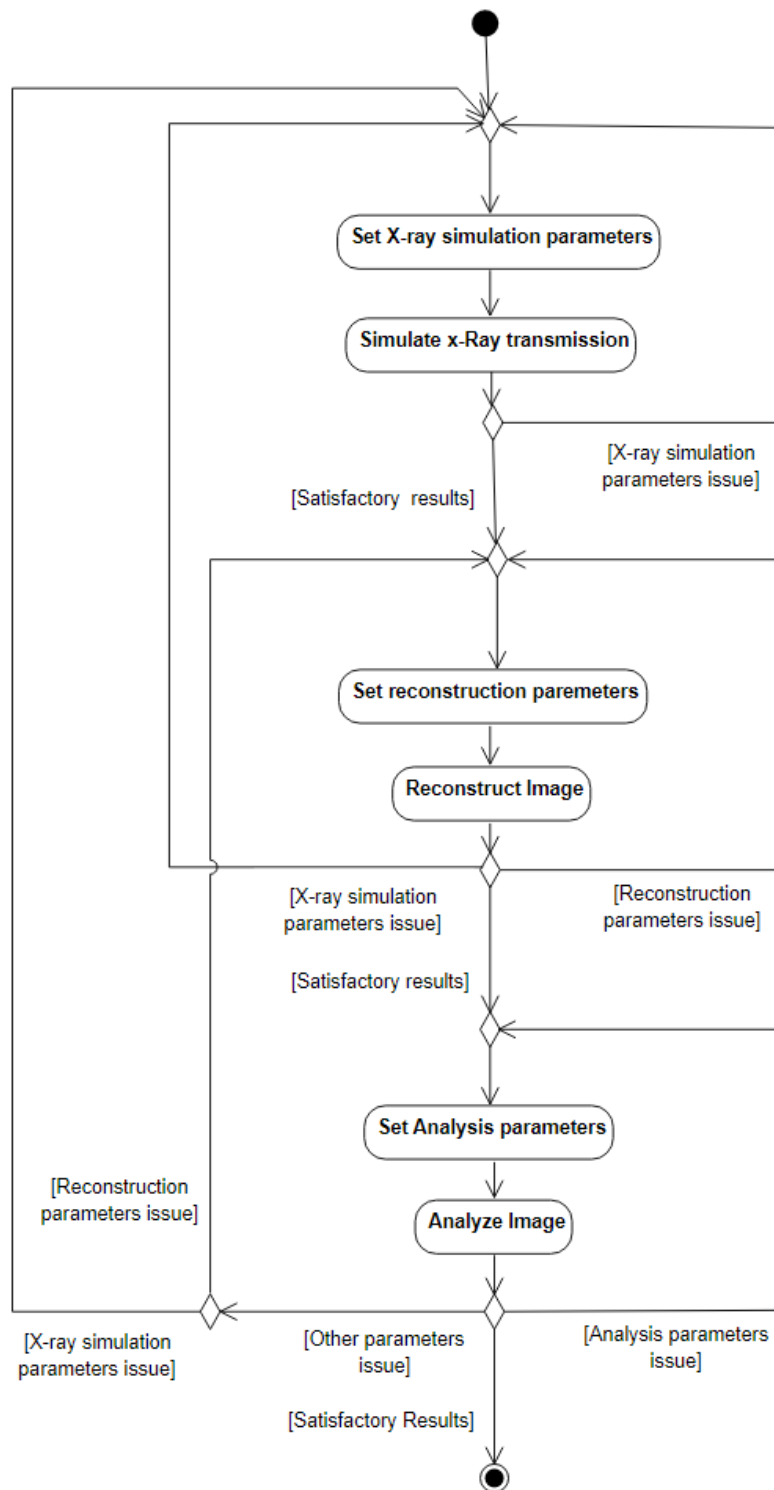


Figure 4.2 Activity diagram of the simulation with intermediate validation points

Following the conclusion that groups of parameters are exclusive for individual stages, the states of the simulation process with their transitions can be seen in figure 4.4. Every transition forward in the states i.e. along the direction of the simulation workflow towards the analysis results, is achieved by running a specific algorithm. These specific transitions consume a subset of the overall simulation settings and yield their corresponding outputs.

The following table details the results available at each state within the simulation.

Simulation State	Results Available
Nothing run	None
X-ray simulation run	<ul style="list-style-type: none"> • Transmission Sinogram • Sinogram of noise in standard deviations
Reconstruction run	<ul style="list-style-type: none"> • Transmission Sinogram • Sinogram of noise in standard deviations. • Reconstructed image
Analysis run	<ul style="list-style-type: none"> • Transmission Sinogram • Sinogram of noise in standard deviations. • Reconstructed image • Statistics of the regions of interest within the reconstructed image

Table 4.2 Results available at each simulation state

The cumulative nature of the results only highlights how every stage leverages upon the previous one, this conclusion is also evident in the state diagram, however the fact that the settings are split into groups that relate only to an individual simulation stage could be exploited to revert only to a state that might already include intermediate results. The latter is of relevance mainly for the X-ray simulation and reconstruction algorithms that are computationally expensive.

The transitions; backwards in the context of the simulation workflow, are depicted in Figure 4.3 with the modification of the parameters that would trigger them.

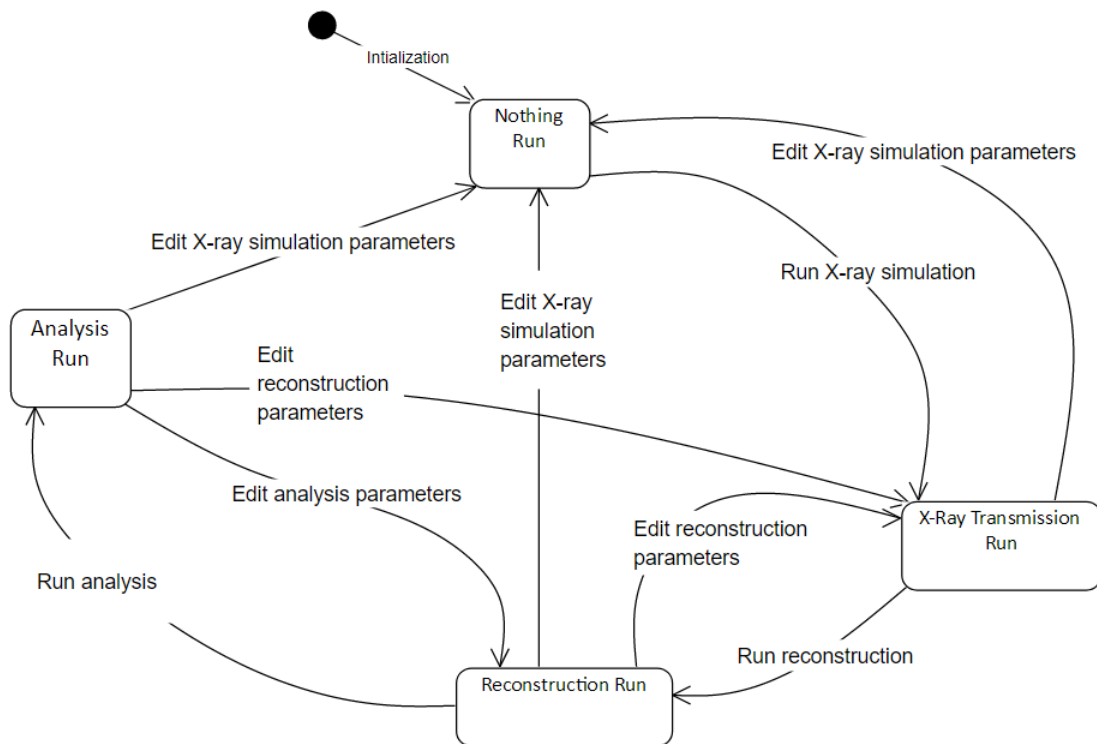


Figure 4.3 State diagram of the simulation considering parameter modifications

If data persistence can be guaranteed by the delivered software for all the results and parameters at every state, the simulation can also be interrupted and resumed in different sessions, this is also a motivation for the state division of the simulation workflow.

4.3 Architecture and separation of concerns

One of the main aspects to consider to develop the application presented in this work is how the newly developed graphical user interface classes and code integrate with the underlying command line application facilities that are already implemented, namely the algorithms involving the radon transform, the filtered back projection and the analysis of the regions of interest. These algorithms constitute a big portion of the domain logic of the application. Consequently, the design of the software must mainly focus on the presentational logic and the complementary adaptations the existing code must be subject of.

wxWidgets as a GUI framework not only provides access to the native controls of the operating system but also implements basic functionality such as method binding to user triggered events as well as extended functionality for common use cases (message dialog

display, file explorer integration among others) [7]. For a simple application, the domain and presentational logic can be well specified within the event handlers of each GUI control.

The simulation state behaviour in the previous section exposes how the presentational aspects of the GUI must be dynamic and coherent. In a trivial case, when the application starts, since no simulation has been run, no result must be available and the GUI functionality related to results of stages that depend on intermediate outputs (image reconstruction or ROI analysis) must be disabled. Extending this idea further, when the simulation has reached an intermediate state, a change in a parameter that reverts this state should disable or alter the GUI controls accordingly. This dynamic aspect of the GUI hints the need for a more general design than just embedding all the functionality in event handlers.

Two design approaches were evaluated for this purpose, these are the Model-view-controller MVC and Model-view-presenter MVP design patterns. As stated per 2.6. the MVP pattern is a specialisation of the MVC. Both approaches were considered since MVC (and consequently MVP) is a design pattern that mainly tackles the separation of the presentation layer and the domain models of the application [5] [8] [6].

In the context of this work the *Model* is the abstraction of the simulation: the staged transformation of input parameters into a set of results that holds a state. The *View* would mainly comprise the GUI controls directly related to the outputs of the simulation (Sinograms, image and ROI statistics) and the input parameters coherent with these results. The main question is what abstraction do the controls that drive the changes in the state of the simulation (parameter and button event handlers) should follow, and the answer resides with the decision between of what suits best among a *controller* or *presenter* to the application. This decision is mainly a matter of complexity of implementation and maintenance. The latter is a crucial point since this work is meant to be a development tool, i.e. it should readily provide further extensibility and more importantly it should not hinder the main purpose of the application delivered, which is to provide a framework to evaluate the dependence of the contrast to noise analysis on the parameters for CT simulation.

The sequence diagram of Figure 4.4 depicts the call mechanism of a normal parameter change of a hypothetical implementation of MVC. The *SimulationView* would comprise any of the UI objects that present to the user the outputs of the simulation (bitmaps, tables) or a text control holding an input parameter coherent with the simulation state. The *SimulationController* would comprise the UI functionality that trigger the changes in the *SimulationModel* given a user command (mainly button and parameter event handlers). Finally, the *SimulationModel* encapsulates the simulation logic (the existing algorithms and their outputs) and updates the *SimulationView* whenever its state changes.

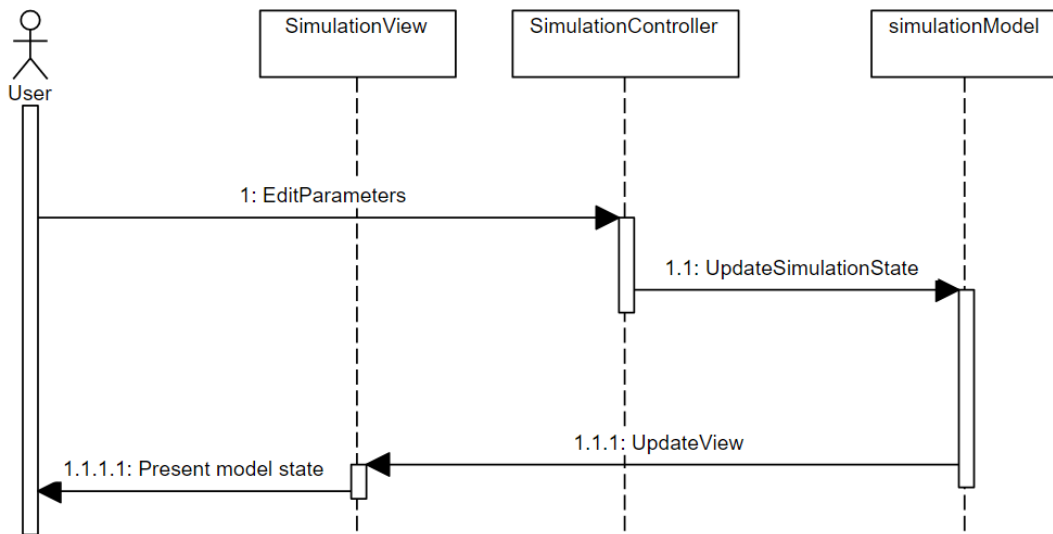


Figure 4.4 MVC user parameter change interaction

In this architecture, the *SimulationView* is only updated by the *SimulationModel* and ideally the *SimulationController* and the *SimulationView* are completely isolated. A technical way to achieve the Model-View relationship could be by configuring the updating methods of the *SimulationView* to listen to events within the *SimulationModel*. For example, a panel with a bitmap in the client area of the application will listen to a custom event dispatched by the simulation model every time the Filtered Back Projection algorithm is run so that the bitmap is loaded with the image resulting from the algorithm.

Although this choice of architecture does achieve a separation of presentation and domain logic there are a few disadvantages:

- The implementation of the *Model-View* interaction is not trivial, and in the spirit of further development of the application, it might only yield in hindering the simplicity of the source code.
- The *wxWidgets* framework implicitly encourages combining both *View* and *Controller* objects when defining windows elements like dialogues or frames that serve as containers for both declaration of UI and event handling. Enforcing a division of these two concepts might as well just overcomplicate the final solution.

An alternative design strategy is the Model-View-Presenter architecture, Figure 4.5 shows the sequence of calls of a user interacting with the application.

The *SimulationView* represents any of the GUI controls related explicitly to the simulation (parameters, results) and their respective event handlers. The *SimulationView* outsources the handling of events to the *SimulationPresenter* which controls the changes in the *SimulationModel* and the corresponding synchronisation of *SimulationView*.

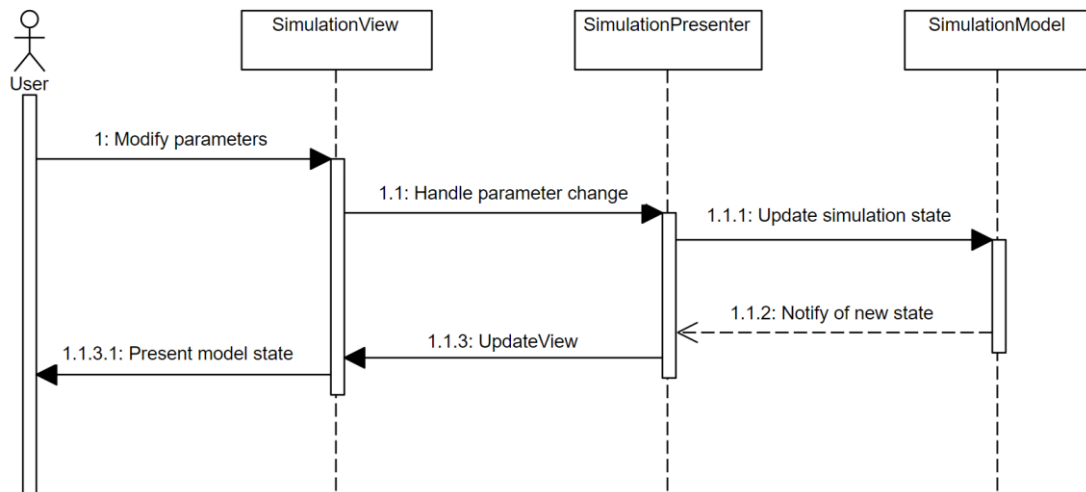


Figure 4.5 MVP user parameter change interaction

The choice of this design pattern is preferred to the previously introduced MVC implementation since it presents the following advantages:

- Reduces the *View* abstraction ideally to declarative programming, i.e. mainly the construction of the GUI and forwarding event handlers. This is desired in the sense that the entire GUI can be replaced with a different user interface and it enables a better setup for unit testing.
- Delegates the synchronisation of the *View* and the *Model* to the *Presenter* in a way that doesn't overly compromise complexity.

The MVP design is therefore chosen for structuring the software classes, there are exceptions to the extent of the strictness of the implementation presented in this work. These relaxations have been made for the sake of simplicity. MVP was mainly used to control the state nature of the simulation model and the synchronization of the GUI. In cases where this was not necessary, the structure of forms and event handlers implicitly supported with *wxWidgets* is preferred. This is the case of exporting images and editing local data within the *View*.

Figure 4.5 illustrates the architecture of the GUI application inspired in the MVP design pattern. The classes *cInputData*, *cFBP* and *cCTRawData* are provided in the command line application that served as a basis for this work. *cInputData* comprehends all parameters of all the simulation stages, *cFBP* implements the image reconstruction algorithm and *cCTRawData* implements the X-Ray simulation algorithm. *cAnalysis* implements the last stage of the simulation where statistics per each ROI are computed, this functionality was originally included in the *cFBP* class in the command line application but in the spirit of having abstractions for each one of the stages, the analysis is specified separately in its own class.

The four classes representing the four stages of the simulation are aggregated into a *cSimulation* class which is meant to hold the state of the simulation abstraction. It also provides the endpoint for storage in the filesystem. This class integrates the domain logic and stands in the context of MVP as the *Model*.

ctMainFrame is the class that encompasses the GUI declaration and construction, it extends the corresponding *wxWidgets* classes to implement the main window, the ribbon bar and client area. This class would stand for the View in the context of MVP, it holds a local instance of *cInputData* to be later synchronized with the *cInputData* object of the *Model* by the *Presenter*.

Finally, *cSimulationPresenter* is the intermediate class that holds a reference to a single *ctMainFrame* and *cSimulation* instances. This class mainly synchronizes the states of the *ctMainFrame* and *cSimulation* objects and implements the logic of the event handlers that drive the state of change in the *cSimulation* object.

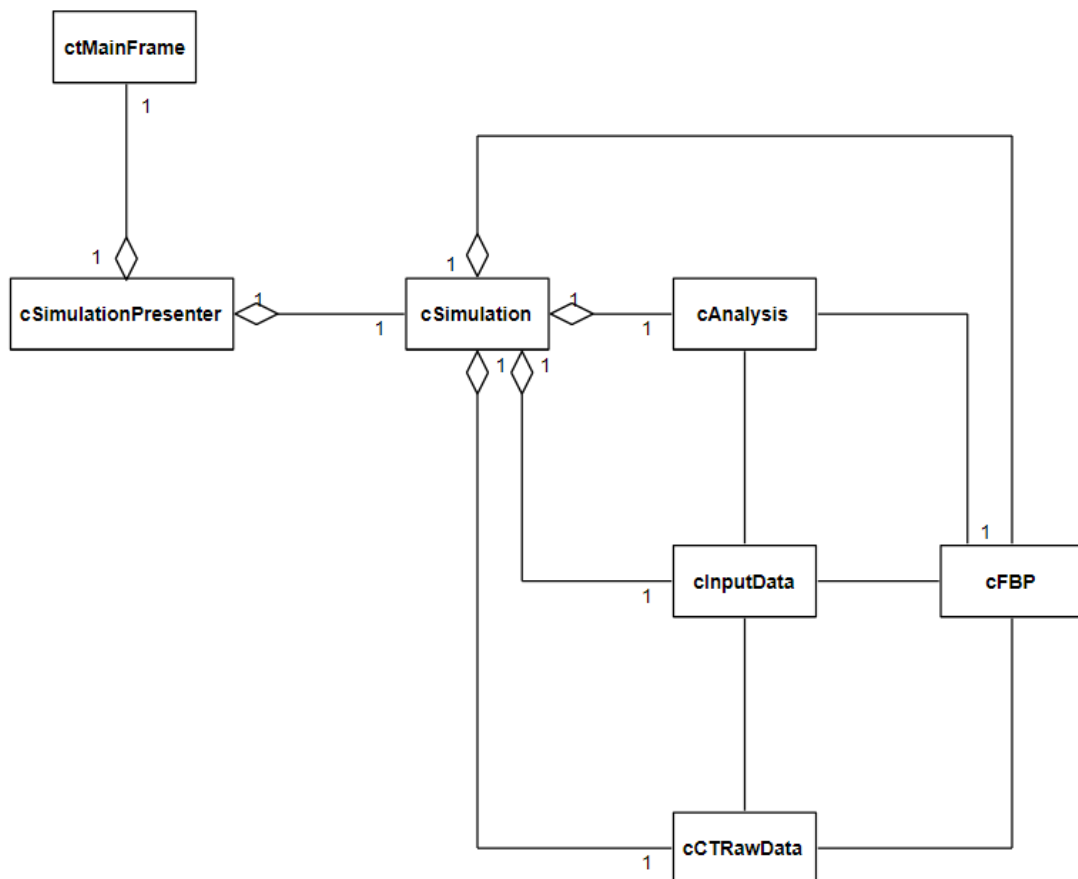


Figure 4.6 Class diagram with the main classes of MVP for the GUI application

5 Implementation Details

5.1 Graphical Layout

The navigation through the application is achieved by the usage of a top ribbon bar that is subdivided in four pages: *File, X-ray settings and materials, Reconstruction and analysis* and *Simulation control and view*. The results of each individual simulation stage are presented in the client area under the ribbon bar. Figure 5.1 shows the layout after a full run of a simulation with the client area displaying both the reconstructed image and a table with the results of the analysis for each region of interest.

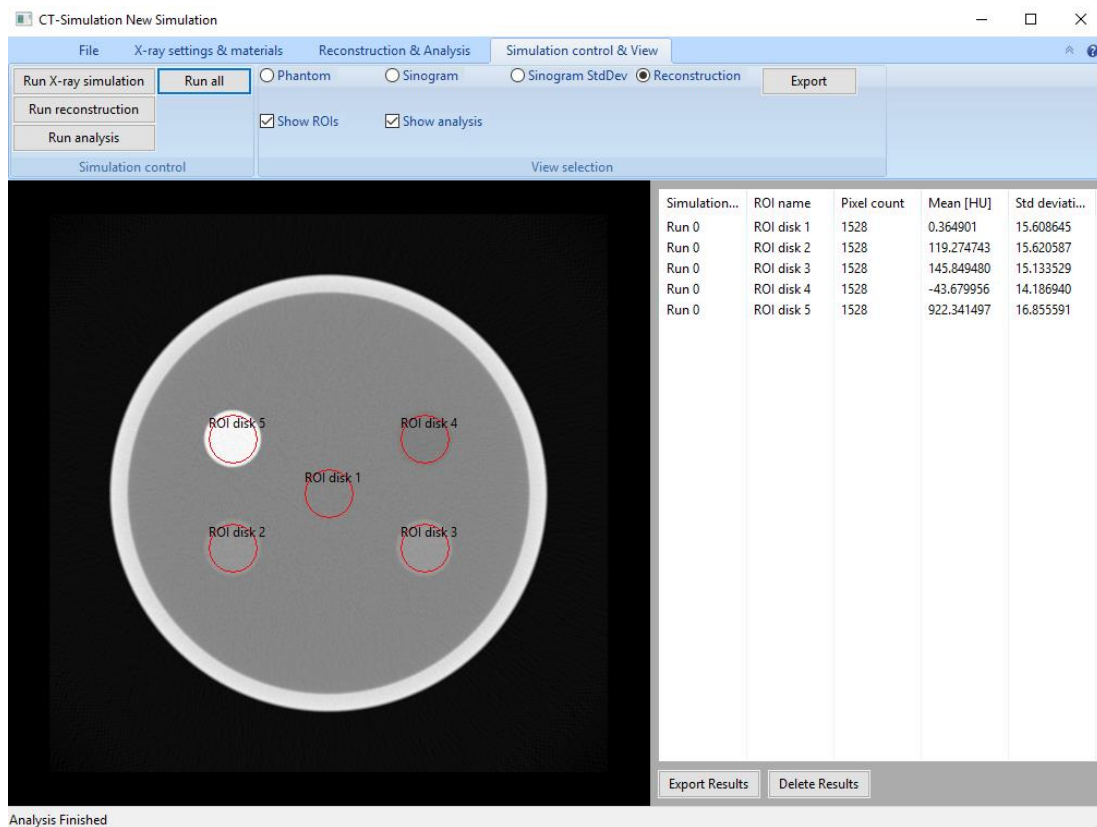


Figure 5.1 Layout of the GUI application after a full succesful run of a simulation

File page

This page will manage the storage and loading of simulation files, it consists of three buttons for opening, saving and saving copies of simulation files, as well as three checkboxes that enable the storage of intermediate results of the simulation whenever available. The transmission data stands for the two sinograms of the X-ray simulation, the reconstruction image is self-explanatory and the “last analysis” refers to the fact that more than one analysis can be run and exported eventually from the client area, however the analysis stored in a simulation file is only the one that is consistent with the settings of the simulation, therefore the last one that is successfully run.

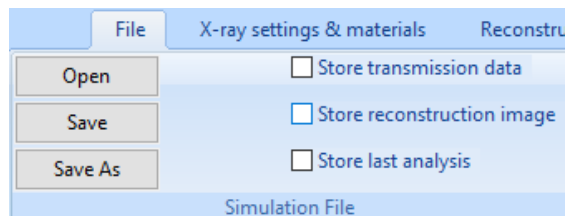


Figure 5.2 implemented layout of the file page of the ribbon bar

X-ray settings & materials page

This page is subdivided into 5 panels

Manage materials

Consisting of a button to add a new material, a combo-box to select an existing material, and two buttons for editing and deleting the selected material.

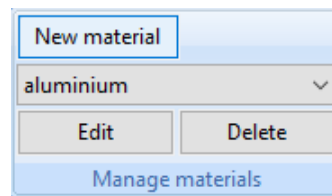


Figure 5.3 Manage materials panel of X-ray settings and materials page

Current phantom

This panel includes a combo-box to select among the existing test objects: the head phantom and the cylinder phantom. Additionally, there is a button to configure the selected test object.

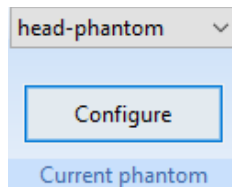


Figure 5.4 Current phantom panel of X-ray settings and materials page

Geometry

As per sub section 3.3.1 of this document, the geometry of the CT system is specified in this panel. All the parameters are implemented as labelled text controls.

Radius of focal point in mm	<input type="text" value="570.00"/>	Number of projections	<input type="text" value="400"/>
Fan angle in degrees	<input type="text" value="30.00"/>	Detector thickness in mm	<input type="text" value="1.00"/>
Number of detectors	<input type="text" value="400"/>		

Geometry

Figure 5.5 Geometry panel of X-ray settings and materials page

Incident beam data

In this panel, there are two combo boxes, one for selecting the tube voltage of the CT system and one for selecting the material for beam hardening correction. Additionally, there's a labelled text control to specify the Current-time product.

Tube voltage in kV	<input type="text" value="140"/>
Beam hardening material	<input type="text" value="water"/>
Current-time product [mAs]	<input type="text" value="100.00"/>

Incident beam data

Figure 5.6 Incident beam data panel of X-ray settings and materials page

Incident beam filters

To complete the parameter specification of the sub section 3.3.1 of this document, this panel implements the functionality for managing filters, this is done by the means of a button to add a new filter, a combo-box to select among the existing filters and two buttons to edit or delete the selected filter.

<input type="button" value="New filter"/>	
<input type="text" value="aluminum filter"/>	
<input type="button" value="Edit"/>	<input type="button" value="Delete"/>

Incident beam filters

Figure 5.7 Incident beam filters data panel of X-ray settings and materials page

Reconstruction & Analysis page

This page is subdivided into two panels.

Reconstruction data

This panel consolidates the settings relevant for the image reconstruction as per sub section 3.3.1 of this document. These parameters are implemented with labelled text controls.

Pixels in x direction	<input type="text" value="512"/>	Pixels in y direction	<input type="text" value="512"/>
Center in x direction in mm	<input type="text" value="0.00"/>	Center in y direction in mm	<input type="text" value="0.00"/>
Pixel size in mm	<input type="text" value="0.50"/>		
Reconstruction data			

Figure 5.8 Reconstruction data panel of Reconstruction and analysis page

Regions of interest

This panel consists of a button to add a new ROI, to select an existing one, and finally two buttons for editing or deleting the selected ROI.

New ROI	
ROI disk 1 ▾	
Edit	Delete
Regions of interest	

Figure 5.9 Regions of interest panel of Reconstruction and analysis page

Simulation control & View page

This page is subdivided into two panels

Simulation control

This panel consists of four buttons, three to run each of the individual simulation stages, and one that runs them sequentially one after the other. This panel is presented in Figure 5.10.

Run X-ray simulation	Run all
Run reconstruction	
Run analysis	
Simulation control	

Figure 5.10 Simulation control panel in Simulation control & View page

View selection

This panel encompasses the control of the results of the simulation that will be presented in the client area, namely:

- *Phantom* radio button: to display the current geometry and material color of the active configuration of the test object of the simulation.
- *Sinogram* radio button: to display the image representation of the Sinogram or Radon transform of the X-ray simulation.

- *Sinogram StdDev* radio button: to display the noise sinogram resulting as well from the X-ray simulation.
- *Reconstruction* radio button: to display in the client area the resulting image of the reconstruction.
- *Export* button: to save the two Sinograms and the reconstructed image in the file system.
- *Show ROIs* checkbox: to display an overlay of the geometry of the regions of interest over the phantom illustration or the reconstruction image.
- *Show Analysis*: To enable the table that accumulates the analysis results of all the analysis run over the session of the application.

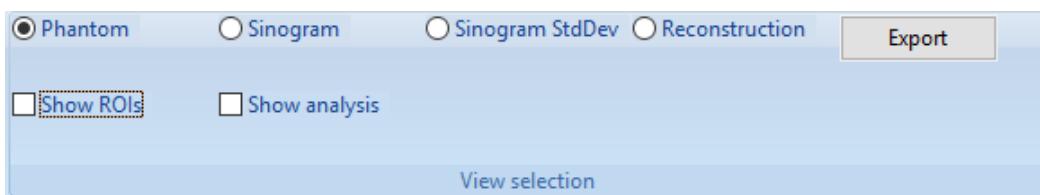


Figure 5.11 View selection panel in Simulation control & View page

Client Area

The client area mainly serves as a view port for the results of all the stages of the simulation, it is subdivided in two sections, the left part of the client area displays upon selection of the radio buttons of the “View selection” panel, either the geometry of the Phantom, any of the two sinograms or the reconstructed image. The right part of the client area whenever the “Show analysis” check-box is enabled will present a list with five columns: *simulation run*, meant for displaying an identifier for the analysis run, and the *name*, *pixel count*, *mean* and *standard deviation* of each ROI defined for the analysis. Figure 5.12 presents the client area showing the phantom geometry achieved after selecting the “Phantom” radio button of the View selection panel, and the analysis results displayed following the above mention checkbox interaction.

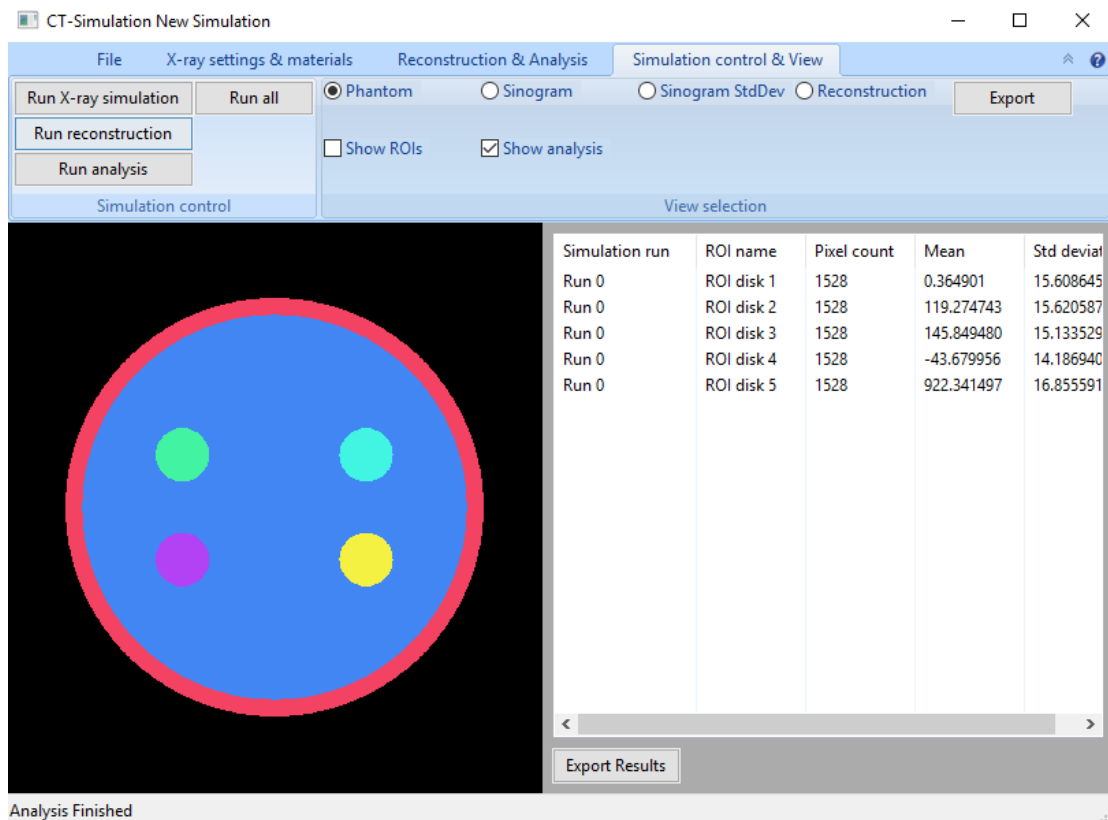


Figure 5.12 Client Area showing Head-phantom geometry and analysis results of a single run

Dialogs

Additionally, a set of dialogues were implemented to help with the management of the abstractions among the input parameters that involve an arbitrary number of instances. These are namely materials, elements, filters, bowtie samples and regions of interest. Furthermore, the dialogues for editing the parameters of the test objects were also implemented.

Material Dialog

This dialogue is invoked either by adding or editing a material, when adding a material intuitively the parameters contained within the dialogue are empty, when loading a material, the dialog is filled with the settings defining the material selected in the combo-box of the "Manage materials" panel.

As per 3.3.1 the parameters required to define a material can be found in this dialogue, the name and density are configurable through labeled text controls, the color property was implemented with the usage of a color picker, finally the elements composing the material can be visualized in a list control with two columns, one for an *atomic number* and the second for the *fraction*. The management of the elements composing the material is done by the *Add* and *Remove* buttons. The first will invoke another modal dialogue request for the two fields describing the element, and the latter will simply remove the entry selected in the list

control. Intuitively the changes of a material or an element will be either stored or discarded depending on whether the user chooses to click the *Save* or *Cancel* buttons, both close the dialogue. Figure 5.13 illustrates the material dialogue as well as the dialogue invoked for adding an element.

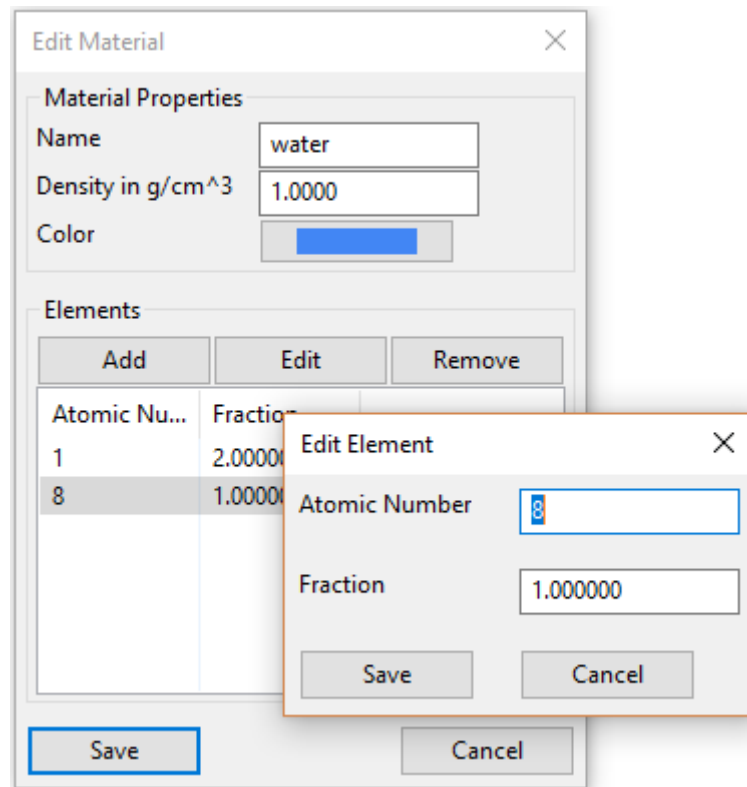


Figure 5.13 Sample material dialogue after while adding an element

Filter Dialog

This dialogue is triggered either by adding a new filter or modifying an existing one. The name and thickness parameters are implemented with labeled text controls, the material combo-box enables the user to select among the existing materials the one used for the filter. Finally, the bowtie samples can be visualized in a list control with their respective angle and thickness displayed as separate columns. Modifying the bowtie samples is achieved by using the *Add*, *Edit* and *Delete* buttons. The first two will trigger a dialogue for the input of the angle and thickness of the sample, and the latter will simply delete the sample selected from the list. The dialogues are closed using the *Save* or *Cancel* buttons which intuitively keep or discard the changes made to the filter or bowtie sample. Figure 5.14 illustrate the filter and bowtie sample dialogues.

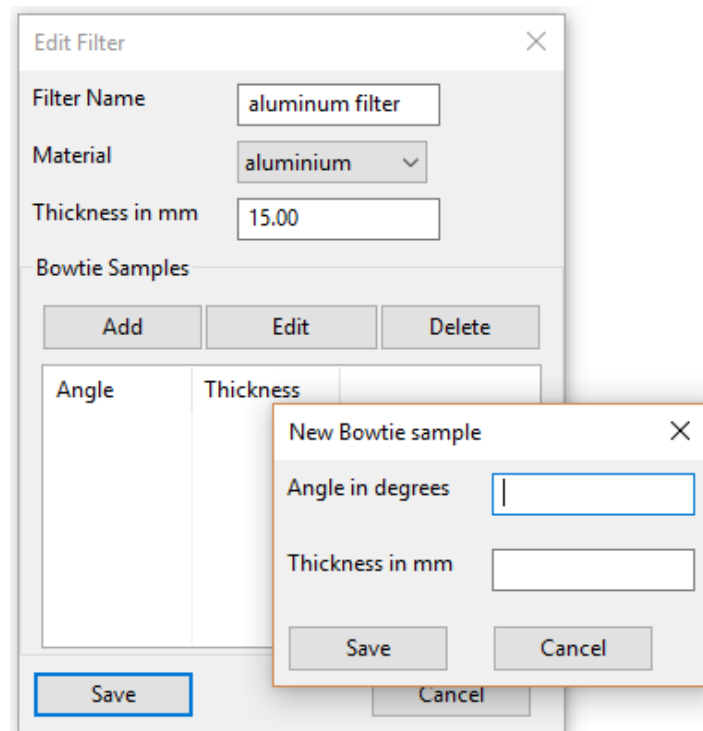
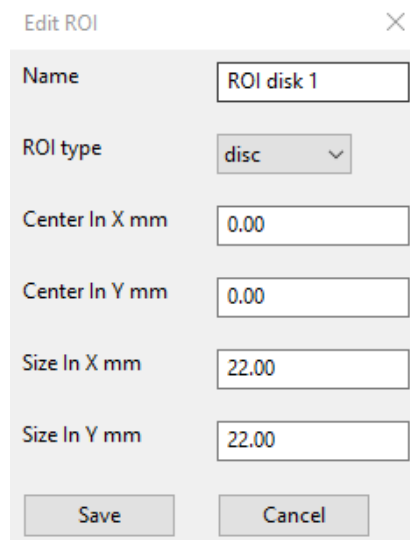


Figure 5.14 Filter and bowtie-sample dialogues

ROI Dialog

This dialog is invoked either by adding or editing an existing ROI from the *Reconstruction & Analysis* page of the main ribbon bar. This dialogue encapsulates the fields as per 3.3.1, namely the Name and geometry parameters are implemented with text controls and the *ROI type* can be selected from a combo-box defining either a disc or a rectangle. The changes to the ROI can be stored or discarded by the usage of the *Save* and *Cancel* buttons, these also yield in the closing of the dialog. Figure 5.15 shows the ROI dialog layout.



Name	ROI disk 1
ROI type	disc
Center In X mm	0.00
Center In Y mm	0.00
Size In X mm	22.00
Size In Y mm	22.00

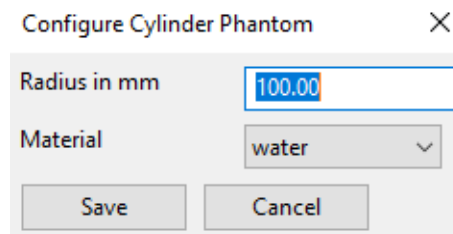
Save Cancel

Figure 5.15 ROI dialog layout

Phantom Dialogues

These dialogues are invoked once the *Configure* button of the *Current phantom* panel in the *X-ray settings & materials* page is clicked.

The cylinder phantom dialogue consists only of a labeled text controls corresponding to the cylinder radius as per 3.3.1, as well as the material of this cylinder which can be picked with the help of a combo-box listing all the available materials. The modifications done to the cylinder phantom can either be stored or discarded with the usage of the *Save* and *Cancel* buttons.



Radius in mm	100.00
Material	water

Save Cancel

Figure 5.16 Cylinder phantom dialog

The head-phantom dialog is slightly more complex. As per section 3.3.1 the head phantom's geometry is composed by a cover a filling and four cylinders. The cover and the filling have a radius that can be altered with the usage of a labeled text control. Also, their material is selected from a combo-box. The details implement the radius and material configuration in the same way as the cover and filling, additionally their position within the phantom geometry is specified by the two x and y coordinates which can be set with the help of two labeled text controls. The dialog is closed either by clicking the *Save* or *Cancel* buttons which store or discard the changes respectively.

Section	Parameter	Value
Cover	Radius in mm	100.00
	Material	PVC
Filling	Radius in mm	92.00
	Material	water
Detail 1	Radius in mm	13.00
	x position mm	44.00
	y position mm	25.00
	Material	PMMA (Acryl)
Detail 2	Radius in mm	13.00
	x position mm	-44.00
	y position mm	25.00
	Material	Lexan
Detail 3	Radius in mm	13.00
	x position mm	44.00
	y position mm	-25.00
	Material	Polyethylen
Detail 4	Radius in mm	13.00
	x position mm	-44.00
	y position mm	-25.00
	Material	Teflon

Figure 5.17 Head phantom dialog

5.2 Extended functionality on previously existing classes

5.2.1 Phantom related modifications

The class *cInputData* encompasses the parameters for the simulation, part of these parameters are the classes that define the geometries of the test objects. In the command line application provided as the basis for this work the test object classes *cHeadPhantom* and *cCylinderPhantom* implement an interface *cObject* which is then accessed mainly by *cCTRawData* for running the X-ray simulation. *cInputData* was modified to support an arbitrary number of references of *cObject* to guarantee that further phantoms can be implemented with minimum modifications to *cInputData*.

Furthermore, as stated the existing two classes *cHeadPhantom* and *cCylinderPhantom* and their *cObject* interface were modified to show a dialog that exposes their parameters. This dialogue is implemented in a single method with the signature:

```
int showDialog(wxWindow*parent, wxArrayString materials)
```


The return value of this function is the id of the control that triggered the end of the display of the dialogue, the first argument is a reference to its parent and the last argument is an array with the available materials that the phantom object can use.

An additional method implemented in the two test objects is the method that exposes their geometry by the means of a bitmap, this method's signature is as follows:

```
wxBitmap drawSelf(wxSize bitmapSize)
```

Finally, auxiliary setters and getters as well as the whole XML input/output functionality were implemented. Furthermore, the operators "=" and "==" were overloaded. Listing 5.1 shows the header file declaring the interface *cObject*, every test object implements this interface so naturally *cHeadPhantom* and *cCylinderPhantom* define these methods as well. The methods that were added to the class are highlighted.

```

3  #include <vector>
4  #include "TotalCrossSection.h"
5  #include "Material.h"
6  #include "spectrum.h"
7  #include <wx/xml/xml.h>
8  #include <wx/wx.h>
9
10 class cObject
11 {
12 private:
13 public:
14     cObject(void);
15     virtual ~cObject(void);
16     virtual int showDialog(wxWindow* parent, wxArrayString materials)=0;
17     virtual void writeWxXML(wxXmlNode* phantomNode) = 0;
18     virtual void readWxXML(wxXmlNode* phantomNode) = 0;
19     virtual void getMaterialIndices(std::vector<int> &index) = 0;
20     virtual void setMaterialIndices(std::vector<int> indices) = 0;
21     virtual void getLengths(double a, double b, double p,
22 std::vector<double> &length) = 0;
23     virtual std::string getName() = 0;
24     virtual wxBitmap drawSelf(wxSize bitmapSize) = 0;
25     virtual double getZoomFactor(wxSize bitmapSize) = 0;
26     virtual void setMaterials(std::vector<cMaterial> *materials) = 0;
27     virtual cObject& operator=(const cObject &x)=0;
28     virtual bool operator==(const cObject&x) = 0;
29
30 };

```

Listing 5.1 cObject.h file declaring the cObject interface.

5.2.2 cInputData modifications

This class groups the settings of all the simulation stages, the main changes done to this class are mainly the setters and getters needed to interact with the abstractions of materials, filters and ROIs as well as the XML reading and writing of the class. Additionally, the operators "=" and "==" were overloaded. This overload is necessary since within a session of the application there are two instances of *cInputData* available, one resides within the class

cCTMainFrame and the other within the *cSimulation* class. The purpose of the first is to receive the validated parameters coming from the user interface (view classes), the latter is connected to the actual simulation and its inherent state. Once the *cSimulationPresenter* instance is notified of a change in the parameters, it will compare these two instances of *cInputData* (hence using the “==” operator) and synchronize the changes in the GUI state and/or the *cSimulation* instance. Figure 5.18 depicts the sequence of method calls between *View* and *Presenter* when the user triggers a command that might rollback the state of the simulation, namely when a text control of the *mainFrame* object is altered. As an illustrative example changing the image reconstruction dimensions after the simulation has fully run should involve an update of the GUI such that it requires from the user to run the reconstruction again. In this context, the *presenter* object will implement the event handler of the command triggered by the user changing a parameter, lookup in a map linking user interface identifiers to simulation states and disable the corresponding views that are not expected to be available after the parameter change. This is achievable only if the two *cInputData* holding the parameters can be compared, in this case with the “==” operator. Since *cInputData* aggregates the parameters coming from other classes such as the test objects or materials, the “==” operator is also implemented in the classes *cMaterial*, *cCylinderPhantom* and *cHeadPhantom*. Consequently, this mechanism also enables the forward stage transitions of the simulation in the case when the changes to the parameters are reverted without having altered the *Model* state. i.e. without having run any of the algorithms.

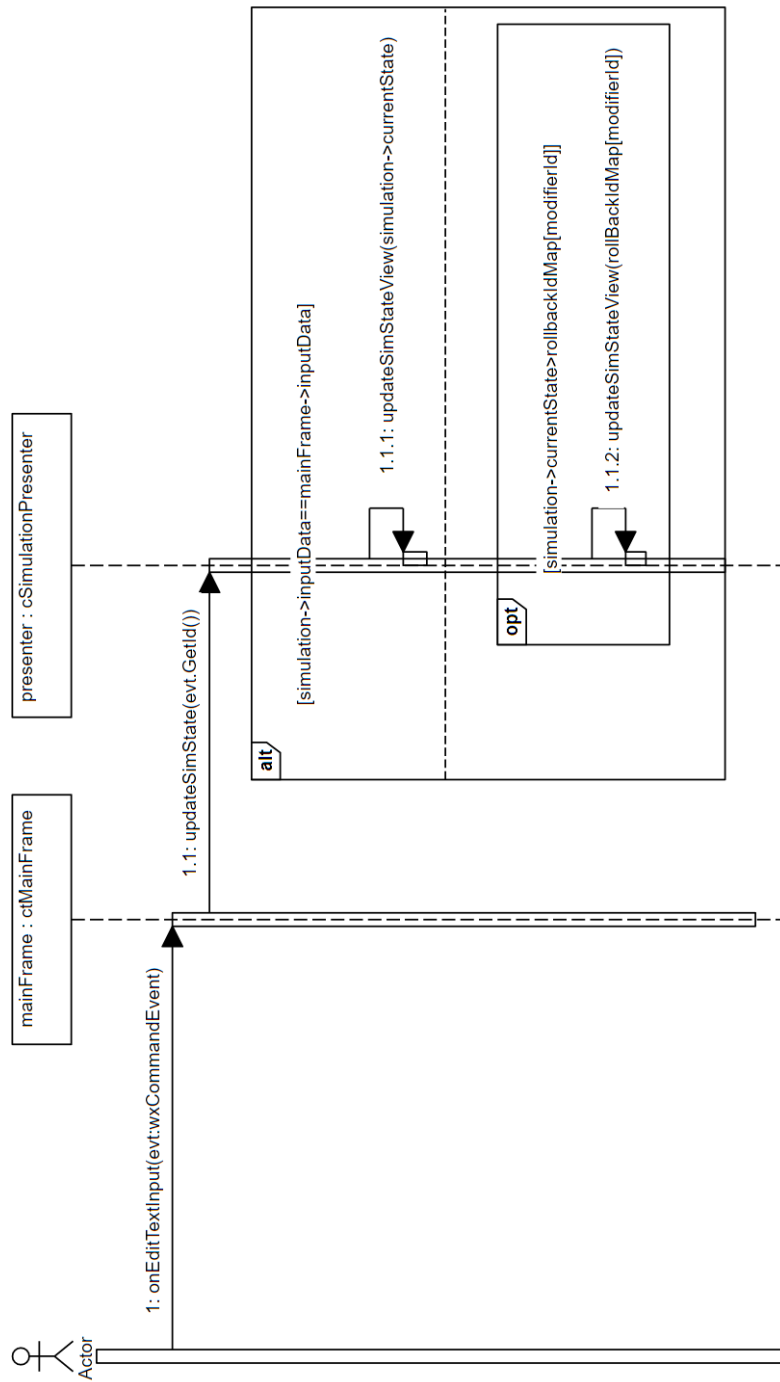


Figure 5.18 Rollback of the simulation state in the GUI

Another important operator overload of this class is the “=” or assignment operator. This is crucial for synchronizing the *Model* (*cSimulation*) *cInputData* instance right after the user has triggered a simulation stage run, namely clicked the “*Run X-ray Simulation*”, “*Run reconstruction*”, “*Run analysis*” and “*Run all*” buttons. Just as in the case of the “==” operator this change must also be implemented within the classes that are contained as parameters within *cInputData*.

5.2.3 Other modifications

The integration of the existing parameter classes with GUI controls needed some additional exposure of properties to be implemented by the means of getter and setter functions.

Additionally, the potential parallelism of the mathematical model and the implementation of the simulation algorithms provided made them able to exploit multithreading. Specifically, the X-ray simulation implemented in *cCTRawData* and the filtered back projection implemented in *cFBP* were modified. This was conveniently achieved using the OpenMP 2.0 pre-processor instructions that are supported by the VisualStudio compiler, mainly since the modifications to the source code are minimal and preserve the already intuitive definition.

5.3 Additional classes

5.3.1 Model Classes

As illustrated in Figure 4.6 the domain Model is comprised by a single *cSimulation* class that aggregates the *cCTRawData*, *cFBP* and *cAnalysis* class. The *cSimulation* class was implemented as a general abstraction of the whole process that offers state handling as well as the methods enabling the transitions between the states. The functionality of the *cAnalysis* class was originally included in the *cFBP* class, however following the notion of separating the simulation stages into different classes, this was also implemented. Figure 5.19 presents the definition of both class and their interdependence with the previously existing classes.

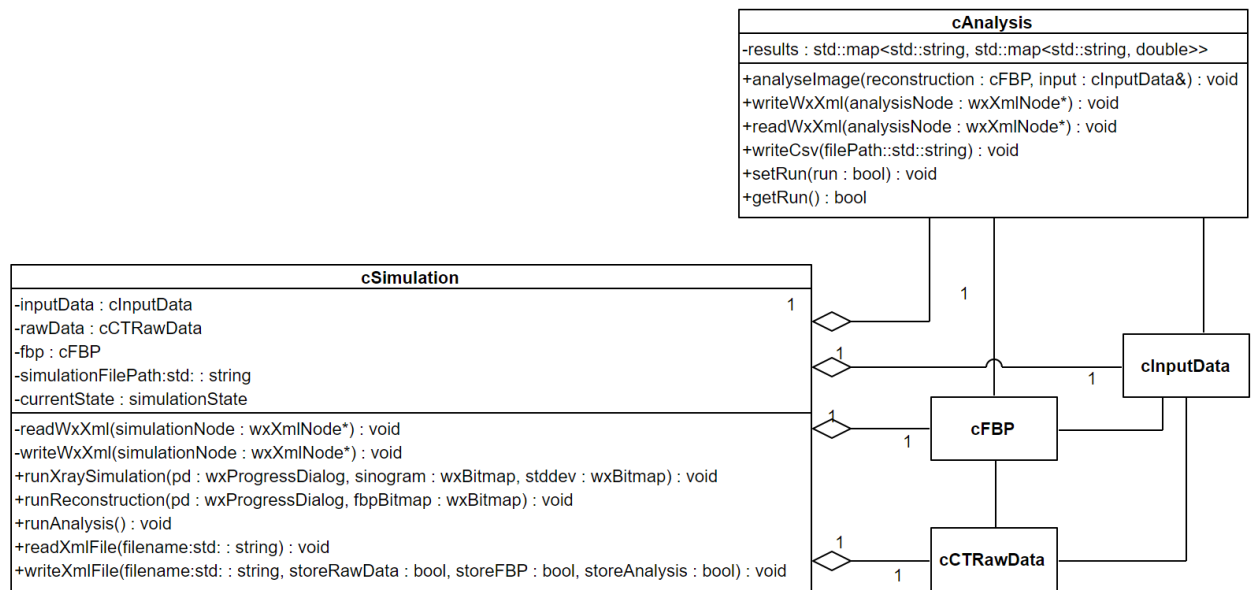


Figure 5.19 Specification of the Model classes detailing the additionally implemented

5.3.2 View Classes

Following the MVP approach the View classes follow mainly a declarative logic, the event handlers that were implemented within these were simply to update the *cInputData* instance that lives within the *ctMainFrame* object of the application (the main window). This decoupling is desired since it provides the convenience of validation of the parameters before they are used as input for the simulation stages.

ctMainFrame stands as the parent view of the application it includes the ribbon bar and the client area, instances of the dialogues are created and destroyed within handling of events of this class. E.g. creating a material, filter, ROI etc. This behaviour is convenient from the memory handling perspective since dialogues are used as stack variables hindering the likelihood of memory leaks. The only persistent views within a session are the *ctMainFrame* instance and a constituent *cViewPanel* object. The latter handles the display, zooming, and panning of the selected view within the ribbon bar, namely the phantom's geometry, the Sinogram bitmaps, the reconstructed image and an overlay of the geometries of the ROIs defined on top of either the phantom illustration or the reconstructed image. Figure 5.20 illustrates the structure of the View classes of the GUI, showing as well the extension of the wxWidgets framework classes.

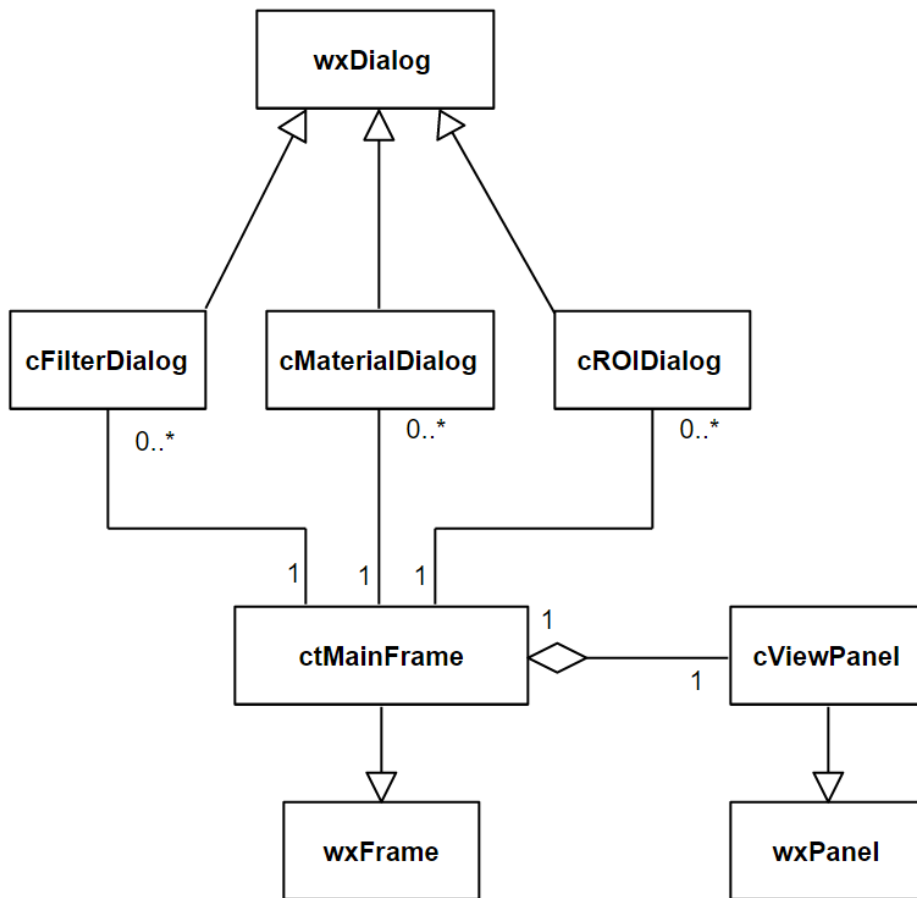


Figure 5.20 Class diagram with new classes and their generalized wxWidgets super classes

5.3.3 Presenter Classes

The `cSimulationPresenter` implements the presentation layer of the application and synchronizes the state of the `cSimulation` and the `ctMainFrame` classes. This class includes the event handlers of the GUI controls that directly trigger the state transition in the `cSimulation` object and the ones that would roll back the simulation state shown by the GUI whenever the input parameters are updated. This is consistent in the way that no result that is not coherent with the settings on the GUI at a given moment should be displayed. Figure 5.21 illustrates the way the `cSimulationPresenter` is implemented and how it relates to the view and model objects.

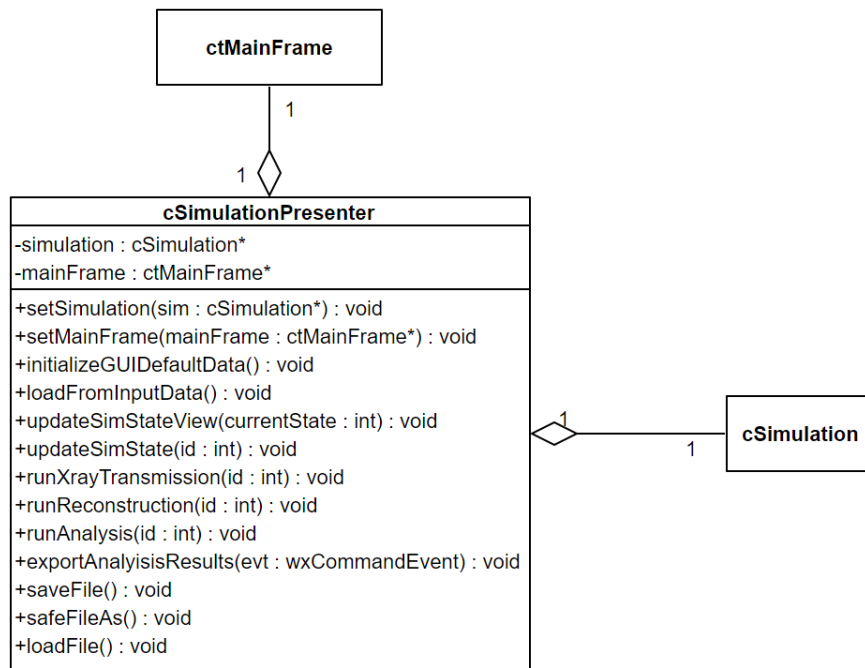


Figure 5.21 Presenter class cSimulationPresenter

6 Test and validation

The methodology used for testing the correct functionality of the application involves mainly a set of scenarios and verifications that are conducted manually. The verifications are specified depending on the features tested, these involve evaluating visually the GUI controls at runtime, the eventual output files and in some cases inspecting the state of critical variables while debugging the application.

6.1 Data persistence tests

This tests are designed to verify requirements from section 3.3.3.

Simulation file can be saved

The application is started in a default state; arbitrary settings are modified and a simulation file is saved, this file is later examined in a text editor to prove that the settings that were modified are consistent.

Simulation file can be open

The application is started and a simulation file with settings differing from the default state is loaded, the settings are checked to be consistent to the xml description of the loaded file.

Storage of intermediate state of the application

This scenario tests the functionality of the checkboxes "store transmission data", "store reconstruction image" and "store analysis" of the file page. For this purpose, the application is started, each of the simulation stages are run so that the required state of the simulation is achieved, subsequently a simulation file is saved by enabling all the combinations of the checkboxes. Finally, the application is restarted, the relevant files are loaded and the state of the application is verified to be consistent with these files, furthermore these three checkboxes are enabled automatically after loading the custom files. The state is verified successfully if the intermediate results are available, and the GUI state is coherent with these results.

Results can be exported

After a full run with the default settings of the application, all the results are exported namely three image files consisting of the transmission and noise sinograms as well as the reconstructed image displayed in the client area of the application. Lastly the analysis results are exported into a CSV file consistent with the data shown in the GUI. All the exported files are examined visually to be consistent with the selected view.

6.2 Parameters tests

This tests relate to requirements of section 3.3.1 and 3.3.2.

Materials can be added, edited and deleted:

For this test a material is added and its parameters are specified, subsequently this material is selected as part of the Head-phantom in any of the cylinders, then all the material settings are edited, the colour change triggers an update in the colour displayed for the phantom view of the relevant cylinder and when re-opening the material dialog the previously modified fields are persistent. Furthermore, this material is selected again and it is deleted, a dialogue is displayed prompting the fact that it can not be deleted because it is under use by the head-phantom, when this situation is changed and the material is successfully deleted.

Phantoms can be modified:

For this test both phantoms are configured, the geometry and material modifications are consistently updated in the phantom view in the client area and after running the X-ray simulation and reconstruction stages the specified geometry is verified to be consistent to the modifications.

Geometry, incident beam data and reconstruction data parameters modify the simulation input data

For this test the application is started and every setting from these panels is altered, subsequently a breakpoint is set within the event handler of the “Run X-ray simulation” button to inspect the application behaviour after this is pressed. The two instances of *clnputData* both in the *mainframe* and the *simulation* objects residing within *simulationPresenter* have the same modified values.

Filters can be added, edited and deleted

For the purpose of this test a filter with a single bowtie sample is created, a breakpoint is added to the “Run X-ray simulation” click handler and the state of the filter variables in the two *clnputData* instances of both the *mainFrame* and *simulation* objects of the *simulationPresenter* are consistent. The modification and deletion are evaluated in the same manner. The consistency of the values with the displayed controls is also verified.

ROIS can be added, edited and deleted

As in the case of the filters the creation, modification and deletion of the ROIs is verified with the usage of the debugger in the click handler of the “Run X-ray simulation” button.

Additionally, a full run of all the simulation stages is executed and the “*Analysis results*” from the client area are verified to be consistent with the definition of the size of the ROIs i.e. the geometry of the ROI and the pixel size matches the pixel count statistic. The number of ROIs and their respective names are also verified to be consistent with.

Additionally, the functionality of the “*show ROI*” checkbox from the “*View*” when enabled displays the ROIs consistently with the modifications.

6.3 GUI state behaviour tests

This tests relate to the requirements of section 3.3.4.

Default state

When the application is started none of the results of each simulation stage are available, furthermore the controls related uniquely to an already run stage are disabled namely the checkboxes related to intermediate result storage in the “*File*” page and the relevant buttons and radio buttons of the “*Simulation control and view*” page. Figure 6.1 presents both pages in their default state, none of the views of results are available.

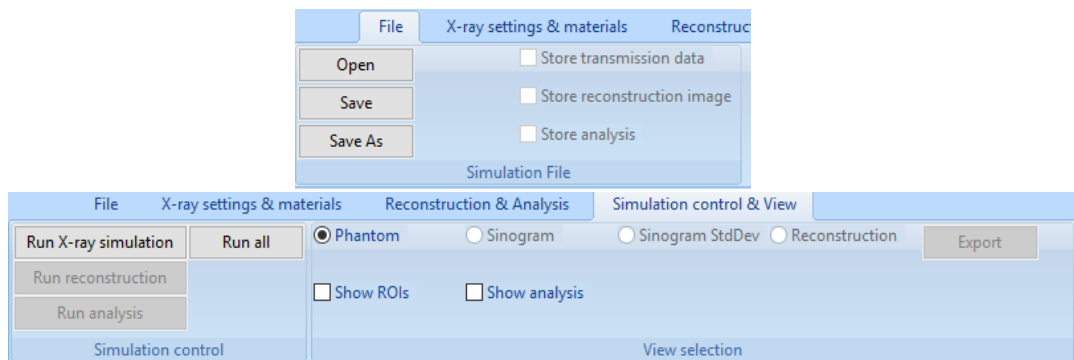


Figure 6.1 Default state of File and Simulation control & View pages of the ribbon bar

Run of x-ray simulation, Simulation control & View page behaviour

After the “*Run X-ray simulation*” button is pressed once the application has been freshly started, once the simulation is finished; the radio buttons related to the transmission and noise Sinograms are enabled and toggling them presents the corresponding image in the client area. The “*Export*” and “*Run reconstruction*” buttons are enabled as well.

Run of Reconstruction

After having run the reconstruction the resulting image is presented in the client area, and the “*Run analysis*” button is enabled.

Run of analysis

After running the analysis, the results are presented as a list in the client area alongside the reconstructed image is shown with the ROIs highlighted.

Successive full runs with different settings and accumulation of results

Results from the analysis stage of several runs are appended in the client area, Figure 6.2 illustrate four consecutive runs with the default settings of the application, testing all the possible configurations of the tube voltage in the incident beam data parameters.

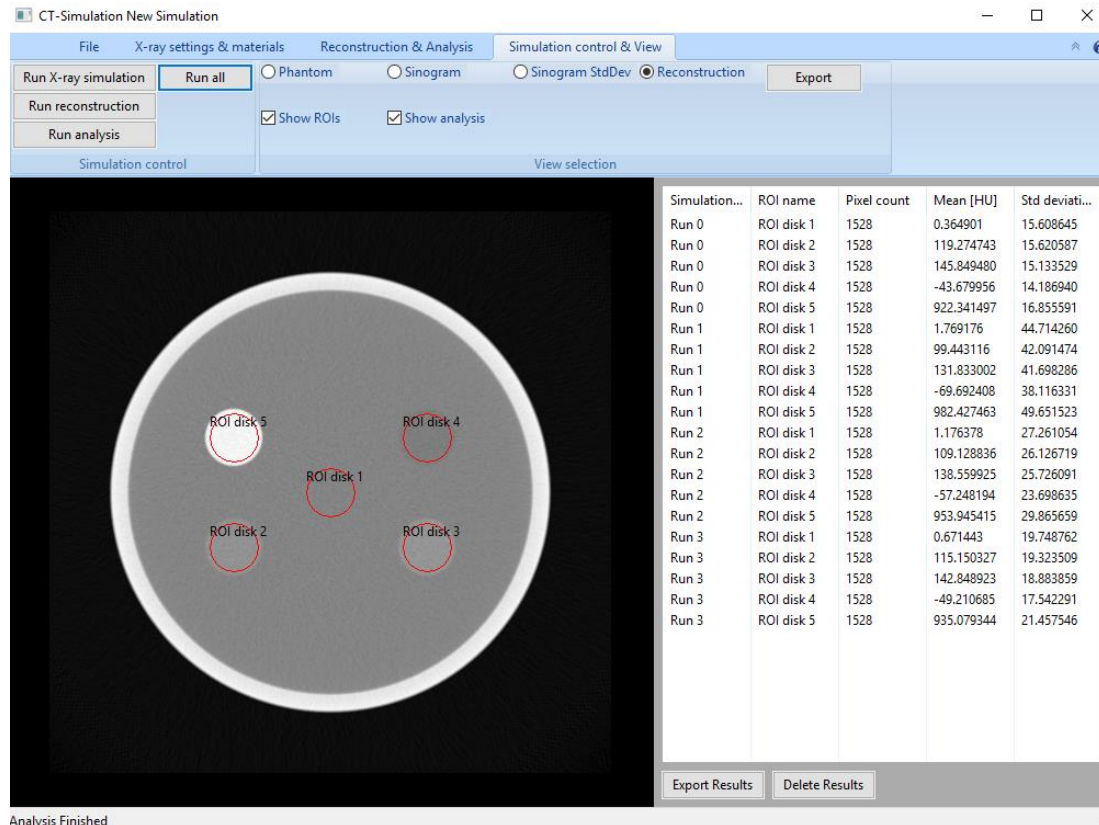


Figure 6.2 Application state after four consecutive runs for different values of tube voltage

Simulation state rollback after modification of parameters

For this test, all the simulation stages are incrementally run, after every stage is successfully run a parameter is chosen so that it reverts the state of the simulation as in Figure 4.3 disabling the corresponding GUI controls from the *Simulation control & View* ribbon page, as well as the related results from the client area. After the modification of the parameter is undone the results in the client area are available without the need to run the corresponding stage again. Figure 6.5 depicts the way the controls of the Simulation control & View page of the ribbon bar become enabled in coordination with the progress of the stages of the simulation.

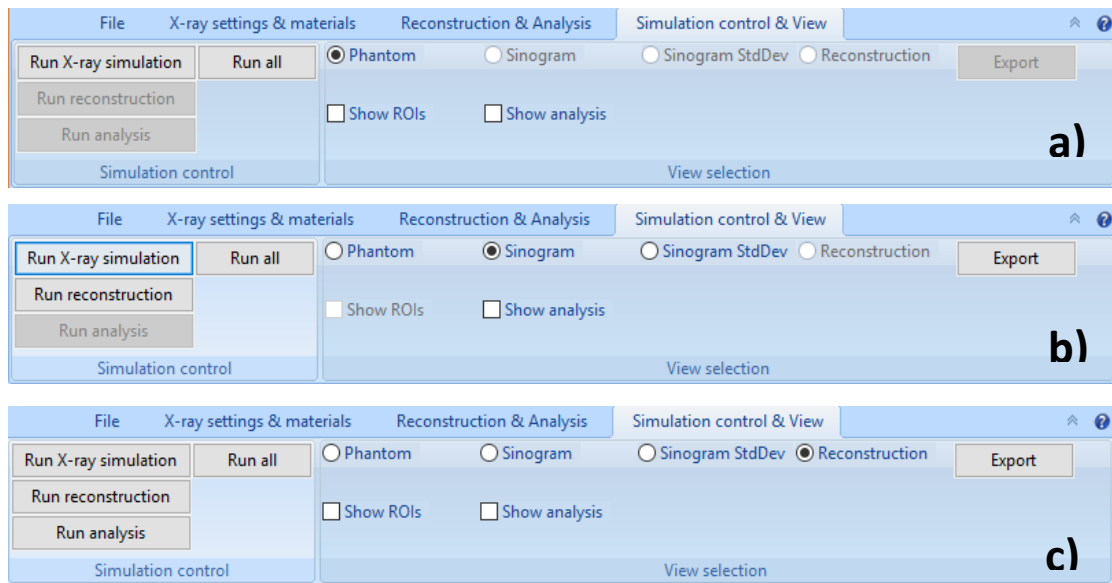


Figure 6.3 Simulation control & View states a) Nothing has run, b) X-ray simulation has been run and c) Reconstruction or Analysis have been run

6.4 Comparison test with command line application

For the purpose of validating that bugs were not introduced in the simulation algorithms, the results yielded by independent simulations with the same input settings of the command line application and the application presented in this work were compared.

After a full run of the simulation with the default settings of the GUI application with the ROIs placed as in Figure 6.3 the results of both analysis for every region of interest present significant differences.

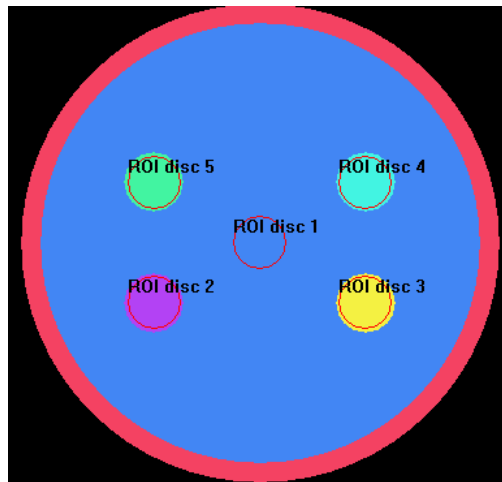


Figure 6.4 ROI positions in head phantom

ROI Name	Command line application			GUI Application		
	Pixel count [-]	Mean [HU]	Standard deviation [HU]	Pixel count [-]	Mean [HU]	Standard deviation [HU]
ROI disc 1	1528	0.102947	15.8081	1528	0.363553	15.610926
ROI disc 2	1528	144.957	15.057	1528	119.274743	15.620587
ROI disc 3	1528	119.872	15.4141	1528	145.849480	15.133529
ROI disc 4	1528	922.281	16.3024	1528	-43.679956	14.186940
ROI disc 5	1528	-44.0027	14.9103	1528	922.341497	16.855591

Table 6.1 Comparison of results of analysis of command line application and GUI application

The reconstructed images can be seen in Figure 6.4 they clearly depict differences in the details of the Head phantom, these differences come from the way these images are stored in each of the applications. The command line application stores the image following the axes directions: horizontal (x-axis) pointing to the right and vertical (y-axis) pointing up, conversely and following the requirements in section 3.3.2 the GUI application inverts the vertical axis to point down, furthermore an existing bug in the command line application reflects the horizontal axis in the X-ray simulation algorithm, this bug was corrected as part of this work.

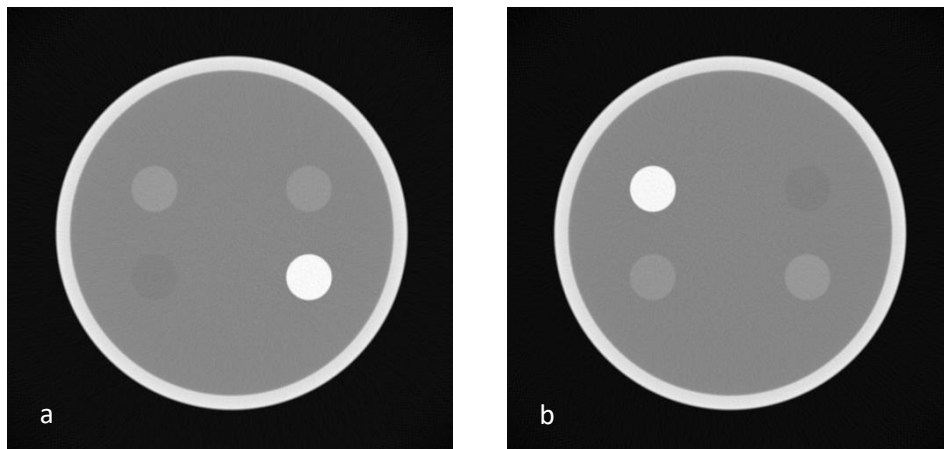


Figure 6.5 Reconstructed images a) Command line application b) GUI application

The difference in the definition of the vertical axis of both applications doesn't bring any consequence to the analysis results since this is only altering the layout of the way the image is stored hence the geometries and positions of the ROIs keep being consistent (on the y axis). On the other hand, the reflection of the horizontal axis in the command line application is not consistent with the position settings of the ROIs since it comes from a bug in the X-ray simulation algorithm i.e. the reconstruction uses an inconsistent Sinogram as an input that ends up in yielding a different (reflected) geometry of the phantom. When this is taken into consideration the ROIs 5 and 4 as well as 2 and 3 of the command line application would swap accordingly, narrowing down significantly the differences between the results of both applications. The remaining difference comes mainly from two sources: a known bug that was corrected by Prof. Dr. Robert Heß in the FBP algorithm in the reconstruction phase but not captured in the application that served as basis for this work, and the intermediate storage of the transmission data (Sinograms) to a file that is used as input for the reconstruction in the command line application. This storage alters the data by firstly cropping negative values of the results (equalizing them to 0) and casting them from *double* to *unsigned short int* for storage purposes.

Once these factors are taken into consideration and modified accordingly in the command line application the results of Table 6.2 are achieved, once again differences exist but are potentially negligible, nevertheless this is not further explored and remains as an open issue.

ROI Name	Command line application			GUI Application		
	Pixel count [-]	Mean [HU]	Standard deviation [HU]	Pixel count [-]	Mean [HU]	Standard deviation [HU]
ROI disc 1	1528	0.364901	15.6086	1528	0.363553	15.610926
ROI disc 2	1528	119.275	15.6206	1528	119.274743	15.620587
ROI disc 3	1528	145.849	15.1335	1528	145.849480	15.133529
ROI disc 4	1528	-43.68	14.1869	1528	-43.679956	14.186940
ROI disc 5	1528	922.341	16.8556	1528	922.341497	16.855591

Table 6.2 Comparison of results of analysis of adjusted command line application and GUI application

7 Summary

This work presented the adaptation of the existing work for X-ray simulation provided by Prof. Robert Heß to use a modern graphical user interface developed in the C++ language with the help of *wxWidgets*. An application supporting the complete simulation workflow including the presentation of intermediate results was designed, implemented and tested.

The actual implementation of the application follows the spirit of an open source development framework for further extension. The ideas of MVP and object oriented programming were a central aspect to support this nature.

The necessary modifications on the original code took place to support the development of the GUI, as well as additional classes were put in place.

Finally, in the process of building the GUI application the understanding of the mathematical concepts behind the X-ray simulation was reinforced, and modifications were made to resolve existing issues such as the reflection of the reconstructed image. Also, this understanding made it possible to improve the performance of the X-ray simulation and reconstruction algorithms by multithreading some of the workload with the OpenMP pre-processor directives with very simple modifications.

7.1 Further work

In terms of the implementation of the application provided with this work, an incremental refactoring cycle would be of interest. The fact that some of the functionality was adopted from an already existing command line application signalled the necessity of a different handling of some of the abstractions. An illustrative example to this case is how the objects of the materials are handled across the application, these are accessed via their positional arrangement in an array. Since such objects are now accessed through a GUI a better way of

management could be to relocate them into different data structures such as maps, and access them by their names instead of indices. Additional situations like this might require further work to refactor and possibly restructure the existing parameter organisation within the *cInputData* class. A more exhaustive exception handling can also be included within this refactoring cycle.

An open issue remains the differences highlighted in the chapter 6 with the comparison of the results between the command line application provided by Prof. Robert Heß and the application developed in this work.

In the sense of performance improvement, the iterative nature of the experimental process to investigate the relationship between the input parameters and the results of the simulation involves computational time that can not be disregarded. Both the Radon transform and the Filtered Back-Projection algorithm, can benefit from parallelisation. Current GPU and accelerator technologies might be exploited to reduce significantly the computation time of the whole simulation process.

List of tables

Table 3.1	Material parameters.....	16
Table 3.2	Geometry parameters	16
Table 3.3	Incident beam parameters	17
Table 3.4	Filter parameters	17
Table 3.5	parameters of the cylindrical geometry of head and cylinder phantoms	18
Table 3.6	parameters of the reconstructed image	19
Table 3.7	parameters of regions of interest ROIs	19
Table 4.1	Preconditions and results of the simulation stages	24
Table 4.2	Results available at each simulation state	26
Table 6.1	Comparison of results of analysis of command line application and GUI application.....	55
Table 6.2	Comparison of results of analysis of adjusted command line application and GUI application.....	57

List of figures

Figure 2.1 Schematic representation of the photoelectric effect [4]	4
Figure 2.2 Schematic representation of Compton scattering [4].....	5
Figure 2.3 Geometry of Fan-beam CT scanners of the third generation [3].....	6
Figure 2.4 Illustration of a first-generation pencil beam CT system [3].....	7
Figure 2.5 Cartesian Radon space, Sinogram of a synthetic image [3]	8
Figure 2.6 Activity diagram of a simulation workflow	11
Figure 2.7 MVC role description and interactions [7].....	12
Figure 2.8 Interactions between the roles of MVP [10].....	13
Figure 3.1 Home screen of the command line application for simulation of CT	15
Figure 3.2 Illustration of the geometry of the existing test objects	18
Figure 4.1 Layout of the main window of the application	23
Figure 4.2 Activity diagram of the simulation with intermediate validation points	25
Figure 4.3 State diagram of the simulation considering parameter modifications	27
Figure 4.4 MVC user parameter change interaction.....	29
Figure 4.5 MVP user parameter change interaction.....	30
Figure 4.6 Class diagram with the main classes of MVP for the GUI application	31
Figure 5.1 Layout of the GUI application after a full succesful run of a simulation.....	33
Figure 5.2 implemented layout of the file page of the ribbon bar	34
Figure 5.3 Manage materials panel of X-ray settings and materials page.....	34
Figure 5.4 Current phantom panel of X-ray settings and materials page.....	34
Figure 5.5 Geometry panel of X-ray settings and materials page.....	35
Figure 5.6 Incident beam data panel of X-ray settings and materials page.....	35
Figure 5.7 Incident beam filters data panel of X-ray settings and materials page	35
Figure 5.8 Reconstruction data panel of Reconstruction and analysis page	36
Figure 5.9 Regions of interest panel of Reconstruction and analysis page	36
Figure 5.10 Simulation control panel in Simulation control & View page	36
Figure 5.11 View selection panel in Simulation control & View page	37
Figure 5.12 Client Area showing Head-phantom geometry and analysis results of a single run	38
Figure 5.13 Sample material dialogue after while adding an element.....	39
Figure 5.14 Filter and bowtie-sample dialogues.....	40
Figure 5.15 ROI dialog layout	41

Figure 5.16 Cylinder phantom dialog.....	41
Figure 5.17 Head phantom dialog.....	42
Figure 5.18 Rollback of the simulation state in the GUI	45
Figure 5.19 Specification of the Model classes detailing the additionally implemented	47
Figure 5.20 Class diagram with new classes and their generalized wxWidgets super classes	48
Figure 5.21 Presenter class cSimulationPresenter.....	49
Figure 6.1 Default state of File and Simulation control & View pages of the ribbon bar	52
Figure 6.2 Application state after four consecutive runs for different values of tube voltage	53
Figure 6.3 Simulation control & View states a) Nothing has run, b) X-ray simulation has been run and c) Reconstruction or Analysis have been run	54
Figure 6.4 ROI positions in head phantom.....	55
Figure 6.5 Reconstructed images a) Command line application b) GUI application.....	56
Figure 0.1 Visual Studio project properties to modify	66
Figure 0.2 Folder structure of the digital material provided with this work	66

References

- [1] W. A. Kalender, Computed Tomography Fundamentals, System Technology, Image Quality, Applications 3rd Edition, Erlangen: Publicis Publishing, 2011.
- [2] N. Stahnmann, Bachelorthesis Vergleichende Analyse gemessener und simulierter computertomographischer Schnittbilder eines neuartigen Testobjektes, Hamburg, 2017.
- [3] T. M. Buzug, Computed Tomography From Photon Statistics to Modern Cone-Beam CT, Springer-Verlag Berlin Heidelberg, 2010.
- [4] S. R. Cherry, J. A. Sorenson and M. E. Phelps, Physics in Nuclear Medicine (Fourth Edition), ELSEVIER, 2012.
- [5] M. Fowler, Patterns of Enterprise Application Architecture, Addison-Wesley, 2003.
- [6] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad and M. Stal, Pattern-Oriented Software Architecture a System of Patterns, John Wiley & Sons, 2004.
- [7] H.-J. Hotop, Software Engineering lecture notes, HAW Hamburg, Hamburg, 2015.
- [8] M. Fowler, "GUI Architectures," 18 July 2006. [Online]. Available: <https://www.martinfowler.com/eaaDev/uiArchs.html>. [Accessed 04 September 2017].
- [9] M. Fowler, "Passive View," 18 July 2006. [Online]. Available: <https://martinfowler.com/eaaDev/PassiveScreen.html>. [Accessed 04 September 2017].
- [10] S. Chandel, "Testing ,GUI Design Patterns," March 2009. [Online]. Available: http://www.gwtproject.org/articles/testing_methodologies_using_gwt.html. [Accessed 04 September 2017].

-
- [11] J. Smart, K. Hock and S. Csomor, *Cross-Platform GUI programming with wxWidgets*, Prentice Hall, 2006.

Appendix A

This appendix shows basic considerations to build the source-code within the digital material attached to this document.

This procedure was tested in a Windows 10 installation using the Microsoft Visual Studio 2017 IDE and its compiler.

Building wxWidgets

1. Download and install or unzip the wxWidgets 3.1.0 source code from www.wxwidgets.org/downloads/ , this directory will contain the built dependencies which take considerable space. The software solution provided with this work assumes this path to be D:\wxWidgets-3.1.0 in case it is possible it is recommendable to use this path.
2. Within the wxWidgets installation path [wxWidgets-3.1.0 path]\build\msw open the wx_vc14.sln solution file with Visual Studio 2017.
3. From the main menu run *Build->Batch Build...*, select all build configurations and press *Build*. It might be needed to re-target the solution to use the corresponding Windows SDK version.

Building the software from this work

To build the software, the only steps needed are to modify the wxWidgets source path chosen above in the C++ and Linker properties of the solution for every build configuration:

In the C/C++ properties under the field *Additional Include Directories* the D:\wxwidgets-3.1.0 prefix of the two first values must be changed to the source installation of wxWidgets.

In the same way under the Linker properties, the *Additional Library Directories* field must be updated. Figure 0.1 highlights the root path of wxWidgets that must be changed.

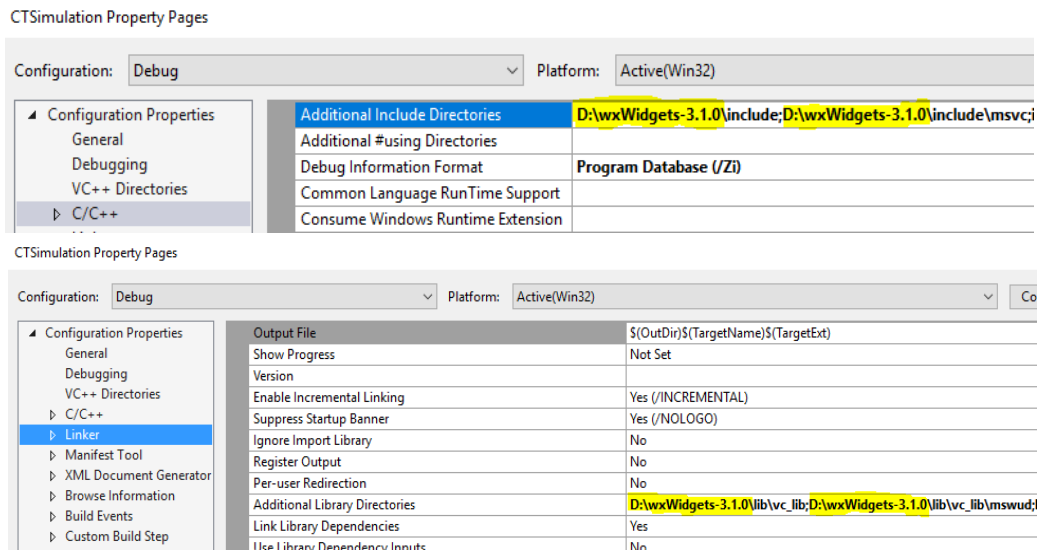


Figure 0.1 Visual Studio project properties to modify

The digital material attached with this document is delivered to Prof.Dr.Robert Heß and Prof.Dr. Marc Hensel. The volume is labelled Alexei_Figueroa_2169845_Bachelor_Thesis.

Name	Date modified	Type
CT-Simulation-CMD	07/09/2017 12:15	File folder
GUI Solution Source Code	07/09/2017 11:55	File folder
Portable Binaries Last build	07/09/2017 11:57	File folder
README.txt	07/09/2017 17:55	Text Document

Figure 0.2 Folder structure of the digital material provided with this work

The contents are mainly 3 folders and a digital version of this work:

- CT-Simulation-CMD: with the original command line application provided by Prof.Dr. Robert Heß with 3 modifications as per section 6.4:
 - CTRawData.cpp line 405 to correct the reflection bug
 - CTRawData.cpp line 527 cropping negative and casting to short int when writing to intermediate file.
 - cFBP.cpp line 59 added factor *30/input.fanAngle ,fixed by Prof.Dr. Robert Hess.

These original lines are kept commented in case the original results need to be reproduced.

- GUI Solution Source Code: With the Visual Studio project to build the software of this work.
- Portable Binaries Last build: With a working latest version of an x86 release build of the software presented in this work.

Declaration

I declare within the meaning of the section 16(5) of the General Examination and Study Regulations for Bachelor and Master Study Degree Programmes at the Faculty of Engineering and Computer Science and the Examination and Study Regulations of the International Degree Course Information Engineering that: This Bachelor Thesis has been completed by myself independently without outside help and only the defined sources and study aids were used. Sections that reflect the thoughts of others are made known through the definition of sources.

Hamburg, the _____