

Warszawa 2019

prof. nzw. dr hab. inż. Włodzimierz Smoliak
Promotor

Adam Jędrzejowski
nr albumu 277417

Wieloplatformowa przeglądarka obrazów DICOM w C++

w specjalności Inżynieria oprogramowania
na kierunku Elektronika

Praça dyplomowa inżynierska

Zakład Elektronicji i Automatyki | Medycyny
Instytut Radiowejelektroniki i Technik Multimedialnych



I T E C H N I K I N F O R M A C Y J N Y C H
W Y D Z I A Ł E L E K T R O N I K I
P o l i t e c h n i k a W a r s z a w s k a

Wieloplatformowa przeglądarka obrazów DICOM w C++

Praca składa się z sześciu rozdziałów: wstęp; obrazowanie diagnostyczne w medycynie; biblioteki i narzędzia; implementacja; komplikacja; podsumowanie. Wstęp jest powierzchownym wprowadzeniem do tematu i celu pracy.

W drugim rozdziale jest opisane zagadnienie problemowe związane z obrazami w medycynie. Wymieniono techniki diagnostyczne oraz ich podstawowe różnice między sobą. Przedstawiono parametry jakie cyfrowy obraz medycyny posiada. Opisano prezentacje obrazów medycznych. Wyjaśniono czym są przeglądarki obrazów, jakie funkcje mogą posiadać i jakie kryteria wyróżniono do ich porównywania. Opisano format zapisu cyfrowych obrazów medycznych, standard DICOM.

Trzeci rozdział opisuje biblioteki i narzędzie użyte w czasie pisania pracy inżynierskiej. Wyjaśniono cele użycia narzędzia CMake i jego zalety. Opisano bibliotekę Qt, jej możliwości, drzewa obiektów implementowane przez nią i sposób konstrukcji programowania zdarzeniowego w niej zawartego. Przedstawiono i uzasadniono wybór biblioteki GDCM jako biblioteki do obsługi i wczytywania plików DICOM.

W czwartym rozdziale przedstawiono sposób implementacji pracy. Określono przewidywany zakres implementowanych funkcji oprogramowania. Opisano graficzny interfejs użytkownika i jego funkcje programu. Wyjaśniono projekt struktury obiektowej programu. Następnie szczegółowo opisano strukturę danych wraz z klasami C++. Tam gdzie była możliwość załączono diagram UML. Opisano wszystkie algorytmy przetwarzania danych w celu lepszej wizualizacji obrazu.

W piątym rozdziale opisano przebieg komplikacji kodu źródłowego.

Słowa kluczowe: DICOM; przeglądarka DICOM; obrazy; obrazowanie; C++; Qt; GDCM; programowanie, przeglądarka obrazów medycznych, medyczne diagnostyczne techniki obrazowe

Keywords: DICOM; DICOM viewer; images; imaging; C++; Qt; GDCM; programming

Na angielski przetłumacze jak będzie po polsku gotowe.

Wieloplatformowa przegladarka obrazów DICOM w C++

Bibliografia

[Daniel Haak(2016)] Thomas M. Deserno Daniel Haak, Charles-E. Page. A survey of dicom viewer software to integrate clinical research and medical imaging. *J Digit Imaging*, 29:206–215, 2016. doi: 10.1007/s10278-015-9833-1.

Spis rysunków

- 2.1 Nazwisko Lupy w przeglądarce MedDream DICOM Viewer. Zdjęcie użyte
za zgodą Software UAB.
- 2.2 Wykrotnie wielu obrazów na raz w jednym oknie w przeglądarce Sante
DICOM Viewer 3D Pro. Zdjęcie użyte za zgodą Santessoft.
- 2.3 Generowana obrazów 3D z wielu obrazów tonograficznych w przeglądarce
Sante DICOM Viewer 3D Pro. Zdjęcie użyte za zgodą Medical Harbou.
- 2.4 Rekonstrukcji wileopaszczyznowej w przeglądarce Athena DICOM Viewer.
Zdjęcie użyte za zgodą Medical Harbou.
- 2.5 Elementy danych wzbiorze elementów danych. Zdjęcie ze standardu DL
COM dostępnego pod adresem <http://dicom.nema.org/medical/dicom/>
- 2.6 Odmu przełączarki z wczesnym kilkoma obrazami. Zdjęcie własne.
- 2.7 Odmu przełączarki z wcześniejszym kilkoma obrazami. Zdjęcie własne.
- 2.8 Diagram klas UML dziedziczenia klas *Sokar::DicomSceneSet*. 40
- 2.9 Diagram klas UML dziedziczenia klas *Sokar::DicomScene*. 36
- 2.10 Diagram klas UML dziedziczenia klas *Sokar::DicomScene* z ujemnymi elementami interfejsu. Zdjęcie własne. 42
- 2.11 Palera Hot Iron (na stołku) i Hot Metal Blue (po prawej) w porowataniu
do palety w skali szarości (po lewej). Zdjęcie własne. 51
- 2.12 Wizualizacja układu osi współrzędnych kartezjańskich pacjenta. Zdjęcie
wzoru. 59
- 2.13 Podział płaszczyzny sceny. Wyrożnione osiem części. Zdjęcie własne. 62
- 2.14 Przykładowy obran medyczny (przekrój głowy MR) z oznaczeniem orienta-
cji obrazu z pomocą liter A, P, R, I, F, H. Zdjęcie własne. 63
- 2.15 Porównanie żadnego obran z trzech różnych elementów interfejsu. Zdjęcie
wzoru. 50
- 2.16 Palera Hot Iron (na stołku) i Hot Metal Blue (po prawej) w porowataniu
do palety w skali szarości (po lewej). Zdjęcie własne. 51
- 2.17 Diagram klas UML dziedziczenia klas *Sokar::DicomSceneIndicator*. 36
- 2.18 Diagram klas UML dziedziczenia klas *Sokar::DicomScene*. 35
- 2.19 Diagram klas UML dziedziczenia klas *Sokar::DicomScene* z ujemnymi elementami interfejsu. Zdjęcie własne. 34
- 2.20 Diagram klas UML globalnej struktury programu. Zdjęcie własne. 32
- 2.21 Odmu przełączarki z wcześniejszym kilkoma obrazami. Zdjęcie własne. 31
- 2.22 Odmu przełączarki z wcześniejszym kilkoma obrazami. Zdjęcie własne. 29
- 2.23 Przykładowe okienko programu w Qt. Zdjęcie własne. 23
- 3.1 Przykładowe okienko programu w Qt. Zdjęcie własne. 23
- 3.2 Odmu przełączarki z wcześniejszym kilkoma obrazami. Zdjęcie własne. 29
- 3.3 Diagram klas UML dziedziczenia klas *Sokar::DicomScene*. 31
- 3.4 Diagram klas UML dziedziczenia klas *Sokar::DicomScene* z ujemnymi elementami interfejsu. Zdjęcie własne. 32
- 3.5 Diagram klas UML dziedziczenia klas *Sokar::DicomScene* z ujemnymi elementami interfejsu. Zdjęcie własne. 34
- 3.6 Diagram klas UML dziedziczenia klas *Sokar::DicomScene* z ujemnymi elementami interfejsu. Zdjęcie własne. 35
- 3.7 Diagram klas UML dziedziczenia klas *Sokar::DicomScene* z ujemnymi elementami interfejsu. Zdjęcie własne. 36
- 3.8 Diagram klas UML dziedziczenia klas *Sokar::DicomScene* z ujemnymi elementami interfejsu. Zdjęcie własne. 37
- 3.9 Diagram klas UML dziedziczenia klas *Sokar::DicomScene* z ujemnymi elementami interfejsu. Zdjęcie własne. 38
- 3.10 Diagram klas UML dziedziczenia klas *Sokar::DicomScene* z ujemnymi elementami interfejsu. Zdjęcie własne. 39
- 3.11 Diagram klas UML dziedziczenia klas *Sokar::DicomScene* z ujemnymi elementami interfejsu. Zdjęcie własne. 40
- 3.12 Diagram klas UML dziedziczenia klas *Sokar::DicomScene* z ujemnymi elementami interfejsu. Zdjęcie własne. 41
- 3.13 Diagram klas UML dziedziczenia klas *Sokar::DicomScene* z ujemnymi elementami interfejsu. Zdjęcie własne. 42
- 3.14 Diagram klas UML dziedziczenia klas *Sokar::DicomScene* z ujemnymi elementami interfejsu. Zdjęcie własne. 43
- 3.15 Diagram klas UML dziedziczenia klas *Sokar::DicomScene* z ujemnymi elementami interfejsu. Zdjęcie własne. 44
- 3.16 Diagram klas UML dziedziczenia klas *Sokar::DicomScene* z ujemnymi elementami interfejsu. Zdjęcie własne. 45
- 3.17 Diagram klas UML dziedziczenia klas *Sokar::DicomScene* z ujemnymi elementami interfejsu. Zdjęcie własne. 46
- 3.18 Diagram klas UML dziedziczenia klas *Sokar::DicomScene* z ujemnymi elementami interfejsu. Zdjęcie własne. 47
- 3.19 Diagram klas UML dziedziczenia klas *Sokar::DicomScene* z ujemnymi elementami interfejsu. Zdjęcie własne. 48
- 3.20 Diagram klas UML dziedziczenia klas *Sokar::DicomScene* z ujemnymi elementami interfejsu. Zdjęcie własne. 49
- 3.21 Diagram klas UML dziedziczenia klas *Sokar::DicomScene* z ujemnymi elementami interfejsu. Zdjęcie własne. 50
- 3.22 Diagram klas UML dziedziczenia klas *Sokar::DicomScene* z ujemnymi elementami interfejsu. Zdjęcie własne. 51
- 3.23 Diagram klas UML dziedziczenia klas *Sokar::DicomScene* z ujemnymi elementami interfejsu. Zdjęcie własne. 52
- 3.24 Diagram klas UML dziedziczenia klas *Sokar::DicomScene* z ujemnymi elementami interfejsu. Zdjęcie własne. 53
- 3.25 Diagram klas UML dziedziczenia klas *Sokar::DicomScene* z ujemnymi elementami interfejsu. Zdjęcie własne. 54
- 3.26 Diagram klas UML dziedziczenia klas *Sokar::DicomScene* z ujemnymi elementami interfejsu. Zdjęcie własne. 55
- 3.27 Diagram klas UML dziedziczenia klas *Sokar::DicomScene* z ujemnymi elementami interfejsu. Zdjęcie własne. 56
- 3.28 Diagram klas UML dziedziczenia klas *Sokar::DicomScene* z ujemnymi elementami interfejsu. Zdjęcie własne. 57
- 3.29 Diagram klas UML dziedziczenia klas *Sokar::DicomScene* z ujemnymi elementami interfejsu. Zdjęcie własne. 58
- 3.30 Diagram klas UML dziedziczenia klas *Sokar::DicomScene* z ujemnymi elementami interfejsu. Zdjęcie własne. 59
- 3.31 Diagram klas UML dziedziczenia klas *Sokar::DicomScene* z ujemnymi elementami interfejsu. Zdjęcie własne. 60
- 3.32 Diagram klas UML dziedziczenia klas *Sokar::DicomScene* z ujemnymi elementami interfejsu. Zdjęcie własne. 61
- 3.33 Diagram klas UML dziedziczenia klas *Sokar::DicomScene* z ujemnymi elementami interfejsu. Zdjęcie własne. 62
- 3.34 Diagram klas UML dziedziczenia klas *Sokar::DicomScene* z ujemnymi elementami interfejsu. Zdjęcie własne. 63

- nie zawsze trudno w sposób niezwykły z obowiązkowym przedstawianiem
- zostala naprawiana przez mnie samodzielnie,
- nie naprawia się zawsze niszczycielsko,
- nie naprawia się zawsze zgodnie z normą.

Swiadom odpowiedzialności prawnie oświadczam, że niniejsza praca dyplomowa jest moją własnością.

Warszawa, 30 lutego 2019

POLITECHNIKA WARSZAWSKA

Wydział Elektrotechniki i Technik Informacyjnych Politechniki Warszawskiej

OSWIADCZENIE

Nazwisko pt. Wioletta Lupa w przeglądarce MedDream DICOM Viewer. Zdjęcie użyte

Warszawa, 30 lutego 2019

Rozdział 6

Podsumowanie

Celem pracy inżynierskiej było napisanie aplikacji do obsługi obrazów DICOM w C++ z możliwością komplikacji na wiele platform. Cel udało się osiągnąć. Zniesienie ograniczeń wirtualizacji kodu rozwiązano językiem programowania C++. Zastosowano biblioteki dostępne na różnych platformach: Qt i GDCM, które również zostały napisane w C++, dzięki czemu uzyskano jednolity program napisany w jednym języku. Zapewniono jednolity sposób komplikacji na platformach przy użyciu narzędzia CMake. Dzięki czemu aplikacja działa w ten sam sposób na wszystkich testowanych platformach: Linux, MacOs i Windows. Jednolity wygląd aplikacji zapewniała biblioteka Qt, dzięki czemu interfejs aplikacji jest prawie taki sam na każdym systemie.

Zaplanowano i dodano obsługę podstawowych operacji na obrazie ułatwiających jego oglądanie i ocenienie, takich jak: przenoszenie; skalowanie; obrót. Zaimplementowano kolorowe pseudokolorowanie obrazów monochromatycznych z możliwością dodawania nowych palet. Wprowadzono obsługę serii obrazów jako całości, włączając w to przegląd obrazów w serii, animacje, wspólne okna w skali barwnej oraz wspólne przekształceniemi macierzowymi.

Napotkano problem z biblioteką GDCM w postaci braku możliwości używania plików binarnych dostarczonych przez twórców. Te pliki binarne zostały skompilowane za pomocą innego kompilatora niż pliki binarne Qt. Co sprawia, że typ std::string z jednej biblioteki nie jest kompatybilny z std::string z drugiej biblioteki. Wynika to z użycia innych interfejsów binarnych aplikacji (ang. *application binary interface*) w różnych kompilatorach. Problem można rozwiązać komplikując bibliotekę GDCM własnoręcznie.

1	Wstęp
2	Obrazowniki diagnostyczne w medycynie
2.1	Obrzutki diagnostyczne
2.2	Parametry obrzutów
2.2.1	Podstawowe parametry obrzutu cyfrowego
2.2.2	Kontrast
2.2.3	Rozdzielcość
2.2.4	Słosunek sygnału do szumu (SNR)
2.2.5	Pozycja artefaktów
2.2.6	Pozycja znieksztalcenia przestrzennego
2.3	Prezentacja obrzutów medycznych
2.3.1	Przeglądarki obrzutów
2.3.2	Funkcje przeglądarki obrzutów
2.4	Format cyfrowych obrzutów medycznych
2.4.1	Standard DICOM V3.0
2.4.2	Spodób zapisu danych w pliku DICOM
2.4.3	DICOMDIR
2.4.4	Imię formaty zapisu
2.5	Formaty zapisu
2.6	Wykonanie
3.1	CMake
3.2	QT
3	Biblioteki i narzędzia
3.1	GDCM
3.2	Gratiany interfejsy użytkownika
3.2.1	Uzasadnienie wyboru
3.2.2	Opis
3.2.3	Licencja
3.2.4	Globale typy struktury
3.2.5	Klasa QObject
3.2.6	Gratiany interfejsy użytkownika
3.3	Uzasadnienie wyboru
3.3.1	Uzasadnienie wyboru
3.3.2	Opcje
3.3.3	Licencja
3.3.4	Podstawowe klasy
3.3.5	Przykłady użyćia
3.4	Git
23	Uzasadnienie wyboru
24	Opcje
24	Licencja
24	Podstawowe klasy
25	Przykłady użyćia
27	Uzasadnienie wyboru

Plik binarny znajduje się „/path/sokar-app-bin/Sokar”. Można go uruchomić klawiszem **lub** z terminala za pomocą komendy „/path/sokar-app-bin/Sokar”.

W przypadku systemu Mac OS X Windows niezbędne jest skopiowanie plików bibliotek do folderu „/path/sokar-app-bin”.

Spis treści

5.6 Przeniesienie plików do jednego folderu

W tym rozdziale omawiamy konfigurację systemu Mac OS X Windows niezbędne jest skopiowanie plików bibliotek w folderze „/path/sokar-app/”. W polu „Where to build the binaries:” wpisz „/path/sokar-app-bin”. Kliknij „Configure”. W oknie zatytułowanym „CMakeSetup” wybierz takiże same opcje na w GDCM. Kliknij „Finish”. Ustaw parametry wartości „QtDir” na ścieżkę do skompilowanej biblioteki Qt. Następnie kliknij przycisk „Generate”.

4 Implementacja	28
4.1 Zakres implementacji	28
4.2 Wieloplatformowość	29
4.3 Graficzny interfejs użytkownika	29
4.4 Projekt struktury obiektowej programu	31
4.5 Struktury danych	32
4.5.1 Konwertowanie danych ze znaczników	32
4.5.2 Scena	33
4.5.3 Kolekcje scen	40
4.5.4 Zakładka	42
4.5.5 Obiekt zakładek	45
4.5.6 Okno główne programu	46
4.6 Algorytmy	47
4.6.1 Cykl generowania obrazów	47
4.6.2 Generowania obrazu monochromatycznego	49
4.6.3 Tworzenie transformat i ich użycie na obrazie	56
4.6.4 Ustalanie pozycji pacjenta względem sceny	59
5 Kompilacja	64
5.1 Narzędzia potrzebne do komplikacji	64
5.2 Biblioteki potrzebne do komplikacji	64
5.2.1 Instalacja Qt	64
5.2.2 Pobranie kodu źródłowego GDCM	65
5.2.3 Pobranie kodu źródłowego Sokar	65
5.3 Przygotowanie katalogów	65
5.4 Kompilacja GDCM	65
5.5 Kompilacja Sokar	65
5.6 Przeniesienie plików do jednego folderu	66
5.7 Uruchomienie	66
6 Podsumowanie	67

Windows i MacOS

W celu instalacji Qt należy udać się na oficjalną stronę biblioteki Qt. W prawym górnym rogu kliknąć zielony przycisk „Download. Try. Buy.”. Na dole kolumny zatytułowanej „Open Source” kliknąć zielony przycisk „Go open source”. Zostanie automatycznie pobrany plik instalacyjny. Po pobraniu należy go otworzyć i postępować zgodnie z instalacją.

W pewnym momencie użytkownik może zostać poproszony o dane kontaktowe. Nie jest to wymagane i można kliknąć przycisk „Skip”.

Następnie należy wybrać komponenty do zainstalowania. W przypadku Windowsa należy zainstalować wersje „Qt 5.12.3 MSVC 2017 64-bit”. Z kolei na MacOS należy zainstalować „Qt 5.12.3 clang_x64”. Można odhaczyć wszystkie inne opcje, nie są one wymagane do komplikacji programu. Dalej należy postępować zgodnie z instrukcjami pojawiającymi się na ekranie.

5.2.2 Pobranie kodu źródłowego GDCM

Program był testowany na wersji 2.8.9. Dlatego jest zalecanym używanie tej wersji.

Należy udać się na stronę <https://github.com/malaterre/GDCM/releases/tag/v2.8.9> i pobrać plik „Source code (zip)”, a następnie go rozpakować.

5.2.3 Pobranie kodu źródłowego Sokar

Kod źródłowy aplikacji można pobrać repozytorium git znajdującego się pod adresem <https://gl.ire.edu.pl/ajedrzejowski/sokar-app>.

5.3 Przygotowanie katalogów

Należy utworzyć folder w którym będą znajdowały się wszystkie foldery z plikami, dalej ten folder będzie nazywany „/path/”. Kod źródłowy GDCM umieść w katalogu „/path/gdcm/”. Kod źródłowy Sokar umieść w katalogu „/path/sokar-app/”. Utwórz również foldery „/path/gdcm-bin/” i „/path/sokar-app-bin/”.

5.4 Kompilacja GDCM

Uruchom CMake z menu programów lub za pomocą polecenia „cmake-gui”. W polu „Where is the source code:” wpisz „/path/gdcm/”. W polu „Where to build the binaries:” wpisz „/path/gdcm-bin/”. Kliknij przycisk „Configure” znajdujący się w dolnej lewej części okna. W oknie zatytułowanym „CMakeSetup” wybierz opcje: „Unix Makefiles” i „Use default native compilers” dla systemu Linux i MacOS; „Visual Studio 15 2017”, „x64” i „Use default native compilers” dla systemu Windows. Zaznacz checkbox przy wartości „GDCM_BUILD_SHARED_LIBS”. Kliknij przycisk „Finish”. Następnie kliknij przycisk „Generate”.

5.5 Kompilacja Sokar

Następnie postępuj tak samo jak w przypadku GDCM. Uruchom CMake z menu programów lub za pomocą polecenia „cmake-gui”. W polu „Where is the source code:” wpisz

Rozdział 1

Wstęp

Medyczna diagnozistka odrzawała się pożółkawie medycznie i zaczęła rozmawiać z pacjentem. Pacjentka była zafascynowana i zainteresowana nową techniką diagnostyczną. Odrzawała ją z ciekawością i zainteresowaniem, a pacjentka odwzajemniła ją swoim zainteresowaniem. Wszystko to miało miejsce w szpitalu, gdzie pacjentka pracowała jako lekarz radiologiczny. Wszystko to miało miejsce w szpitalu, gdzie pacjentka pracowała jako lekarz radiologiczny.

Zaręczestwiane obrazy mogły być zapisane w formacie zdefiniowanym przez programy i techniki medyczne. Oba odrzawy miały dość czasu na rozmowę i rozmawiały o swoich doświadczeniach i umiejętnościach profesjonalnych. Wszystko to miało miejsce w szpitalu, gdzie pacjentka pracowała jako lekarz radiologiczny. Wszystko to miało miejsce w szpitalu, gdzie pacjentka pracowała jako lekarz radiologiczny.

Wszystko to miało miejsce w szpitalu, gdzie pacjentka pracowała jako lekarz radiologiczny. Wszystko to miało miejsce w szpitalu, gdzie pacjentka pracowała jako lekarz radiologiczny.

Nie istnieje dystrybucja Linuksa, w której repozytoria nie borykają się z problemem menadżera pakietów. Dlatego instalacja Qt sprawdza się poprawnie jeśli z repozitorium za pomocą menadżera pakietów. Linus program był testowany na wersji 5.12. Dlatego jest zalecanym używaniem tej wersji.

5.2.1 Instalacja Qt

Do kompilacji są potrzebne biblioteki Qt i GDCM.

5.2 Biblioteki potrzebne do kompilacji

Kod źródłowy został skompilowany za pomocą powyzszych narzędzi. Ale w kodzie pojawiły się problemy z użyciem nizszych wersji wspariących standard C++17. Kod źródłowy nie wykorzystał żadne elementy obiegające od standardu C++17, więc nie pojawiły się problemy z użyciem nizszych wersji wspariących standard C++17.

- macOS — Xcode w wersji 10 lub nowszej
- Linux — pakiet zawierający narzędzia komendy: make (w wersji 3.10 lub nowszej), g++ (w wersji 8 lub nowszej)
- Windows — Visual Studio w wersji 2017 lub nowszej

Do kompilacji wykorzystano narzędzia budowane dostosowane do systemu operacyjnego.

5.1 Narzędzia potrzebne do kompilacji

W przypadku pytania dotyczącej kompilacji, jest możliwość skontaktowania się z autorem pod adresem jedrzejowski@ gmail.com.

Kompilacja

Rozdział 5

Linus

Dla tego instalacja Qt sprawdza się poprawnie jeśli z repozitorium za pomocą menadżera pakietów. Linus

Do kompilacji są potrzebne biblioteki Qt i GDCM.

5.2.2 Kompilacja bibliotek

Kod źródłowy został skompilowany za pomocą powyzszych narzędzi. Ale w kodzie pojawiły się problemy z użyciem nizszych wersji wspariących standard C++17. Kod źródłowy nie wykorzystał żadne elementy obiegające od standardu C++17, więc nie pojawiły się problemy z użyciem nizszych wersji wspariących standard C++17.

- macOS — Xcode w wersji 10 lub nowszej
- Linux — pakiet zawierający narzędzia komendy: make (w wersji 3.10 lub nowszej), g++ (w wersji 8 lub nowszej)
- Windows — Visual Studio w wersji 2017 lub nowszej

Do kompilacji wykorzystano narzędzia budowane dostosowane do systemu operacyjnego.

5.2.3 Kompilacja aplikacji

Do kompilacji aplikacji są potrzebne biblioteki Qt i GDCM.

5.3 Kompilacja aplikacji

Do kompilacji aplikacji są potrzebne biblioteki Qt i GDCM.

Do kompilacji aplikacji są potrzebne biblioteki Qt i GDCM.

Do kompilacji aplikacji są potrzebne biblioteki Qt i GDCM.

Do kompilacji aplikacji są potrzebne biblioteki Qt i GDCM.

Do kompilacji aplikacji są potrzebne biblioteki Qt i GDCM.

Do kompilacji aplikacji są potrzebne biblioteki Qt i GDCM.

Do kompilacji aplikacji są potrzebne biblioteki Qt i GDCM.

Do kompilacji aplikacji są potrzebne biblioteki Qt i GDCM.

Do kompilacji aplikacji są potrzebne biblioteki Qt i GDCM.

Do kompilacji aplikacji są potrzebne biblioteki Qt i GDCM.

Do kompilacji aplikacji są potrzebne biblioteki Qt i GDCM.

Do kompilacji aplikacji są potrzebne biblioteki Qt i GDCM.

Do kompilacji aplikacji są potrzebne biblioteki Qt i GDCM.

Do kompilacji aplikacji są potrzebne biblioteki Qt i GDCM.

Do kompilacji aplikacji są potrzebne biblioteki Qt i GDCM.

Do kompilacji aplikacji są potrzebne biblioteki Qt i GDCM.

Do kompilacji aplikacji są potrzebne biblioteki Qt i GDCM.

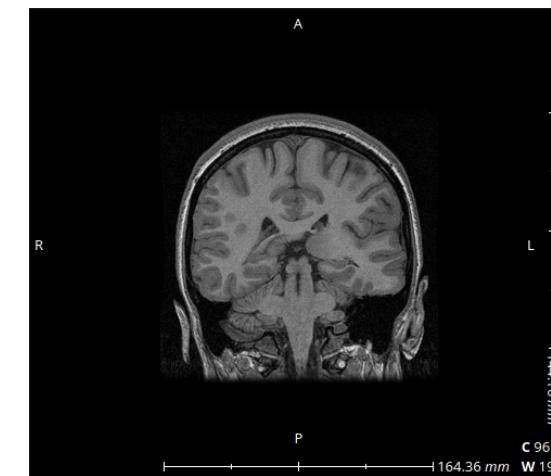
Do kompilacji aplikacji są potrzebne biblioteki Qt i GDCM.

znany jest sposób prezentacji danych obrazowych zawartych w pliku. Głównym aspektem tego procesu jest tak zwane pseudokolorowanie danych numerycznych.

Rozwój obrazowych technik diagnostycznych w medycynie oraz zwiększena dostępność aparatury spowodowały, że badania obrazowe są coraz bardziej powszechnne. Badania obrazowe pomagają lekarzom w diagnostyce i terapii w codziennej praktyce lekarskiej. Przekazywanie badań obrazowych pomiędzy lekarzami różnych specjalności zostało rozwiązane poprzez rozwój standardu DICOM, który przewiduje wymianę danych zarówno poprzez komunikację klient-serwer urządzeń medycznych jak i wymianę plików cyfrowych. Istnieje wiele narzędzi, komercyjnych i otwarto-źródłowych, do wizualizacji i analizy obrazów medycznych. Najczęściej jest to oprogramowanie dedykowane na jedną platformę systemową (system operacyjny). Innym rozwiązaniem jest zastosowanie środowiska, które pozwala na uruchomienie programu na wielu platformach. Takim środowiskiem jest Java firmy Oracle, która umożliwia uruchamianie programów napisanych w języku Java i skompilowanych do „kodu bajtowego” na dowolnej platformie, na której działa maszyna wirtualna Javy. Jednakże takie rozwiązanie sprawia, że nie jesteśmy w stanie osiągnąć pełnego potencjału obliczeniowego maszyny przez pewien dodatkowy poziom wirtualizacji.

Celem niniejszej pracy inżynierskiej było opracowanie przeglądarki obrazów medycznych działającej na różnych platformach i zapewniającej szybkość działania, która nie jest ograniczona wirtualizacją kodu. Założono, że cel ten zostanie zrealizowany poprzez opracowanie jednolitego kodu w języku C++ dla wizualizacji i przetwarzania obrazów, kompilowanego do kodu maszynowego na każdą z docelowych platform. Język C++ pozwala uzyskać kod maszynowy, który charakteryzuje się wysoką wydajnością z bezpośrednim dostępem do zasobów sprzętowych i funkcji systemowych. Przyjęto, że do obsługi zagadnień specyficznych dla danego systemu operacyjnego, w tym graficznego interfejsu użytkownika będzie wykorzystana biblioteka Qt. Biblioteka Qt jest wielo-platformowym zestawem narzędzi rozwijania oprogramowania. Zapewnia ona nie tylko obsługę interfejsu użytkownika ale również bogatą bibliotekę programowania aplikacji. Dodatkową zaletą wyboru biblioteki Qt w kontekście obrazowania medycznego jest to, że posiada ona certyfikaty zgodności z normą IEC 62304:2015 ułatwiający wprowadzanie przeglądarki obrazów na rynek Unii Europejskiej jako wyrobu medycznego klasy I z funkcją pomiarową, klasy II lub III.

W opracowanym kodzie przeglądarki obrazów do obsługi plików w formacie DICOM wykorzystano bibliotekę Grassroots (Grassroots DICOM library — GDCM).



Rysunek 4.14: Przykładowy obraz medyczny (przekrój głowy MR) z oznaczeniem orientacji obrazu z apomocą liter A, P, R, L, F, H. Zdjęcie własne.

Akwizycja w tomografii komputerowej jest podobna do badania RTG, ale w CT wykonywane jest pomiarów w różnych położeniach pozycjiach względem obiektu badanego i pod-

- Tomografia komputerowa (Computer Tomography — CT)

W standardzie DICOM radiografia cyfrowa jest oznaczana jako „RT”.

coś co jest liczbą rozróżniającą linią na jednostkę długosći.

położenia obiektu względem detektora a lampą i wieleoscią obiekktu. Miarą rozdziel-

niowością. Rozdzielenie zależy od rozdzielczości detektora, rozmiaru ognielska lampy,

(położenie optymalne), od napęcia动荡ego, filtracji, głąboczy okładki zmie-

trzycy detektora. Kontrast zależy od położenia obiektu między zasadą zasadą a detektorem

także formie zdjęcie jest analizowane przez lekarza. Wielkość dzięgię jest negatywem i w

i reprezentuje współczynnik promieniowania X, dla tego dzięgię jest negatywem w

badany obiekt. Plik w obrębie jest uzyskany przez liczanie liczby rozsyków

W radiografii rejestrowane jest natężenie promieniowania X przedziału przed-

złożonym z dwóch części, której rejestrowane są przez fotodiody krzemowe.

w konwersji pośredniej, w których wartości X konwertowane są w sygnały analogowe

w grubej warstwie odpowiadającej połowie głowy (np. elektro). Detektor

try z konwersją bezpośrednią, której wartości X konwertowane są elektronami

radiografii cyfrowej do detektuji się w zakresie 0-10000. Wszystkie

powoli w zapomnienie ze względem na koszt i uciążliwości wyołania filmu. W

Radiografia analogowa wykorzystująca swiatłoczuły odciski

droższe tego promieniowania. Wyrobiary dają tryb radiografii; analogowa cyfrowa

cyfrowa obsługa promieniowania rentgenowskiego dla materiału zbadanego za pomocą

transmisji promieniowania X przez badany obiekt, a następnie detektuji tego promie-

nioszczące analogowe zosłano wykorzystać Rontgena w 1896 roku. Polega na

Radiografia to najstarsza i najbardziej rozpoznawalna technika obrazowania. Pierw-

- Radiografia — RTG

Istnieje wiele technik obrazowania wykorzystujące różne zjawiska fizyczne zachodzące

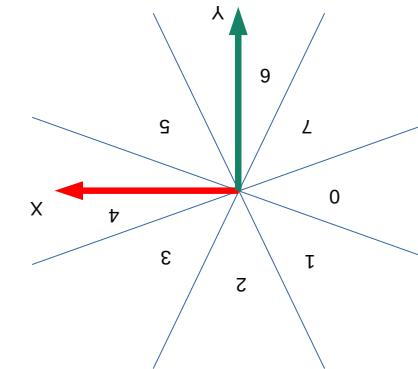
2.1 Obrzutowe techniki diagnostyczne

Obrzutowanie diagnostyczne w medycynie

Rozdział 2

Przykład mówiący o zasadach radiografii. W prawej stronie znajdują się cztery punkty, z których trzy są dobrze widoczne. W lewej stronie po-

Rysunek 4.13: Podział płaszczyzny sekcji. Wyrozumiano osiem czwótek. Zdjedecie własne.



Przykład „H”, czyli punkt reprezentujący kierunek głowy, został przyproszony do czwórki I i opowiedział, „L”, do czwórki 7, „W” do czwórki 3 i „P” do czwórki 5. Punkty „A” i „D”, zosłaly do czwórki 6, co oznacza, że na środku. Do lewego pola stawiany jest tekst „HL”, do góry pola „HF”, do prawego „RF” i do dolnego „LF”.

- górnego pole: tytuł czwórki 7, tytuł czwórki 6 i tytuł czwórki 5
- prawe pole: tytuł czwórki 3, tytuł czwórki 4 i tytuł czwórki 5
- górne pole: tytuł czwórki 1, tytuł czwórki 2 i tytuł czwórki 3
- lewe pole: tytuł czwórki 0 i tytuł czwórki 1

różnym kątem. W tomografii komputerowej podobnie jak w radiografii wykorzystuje się promieniowanie X do pomiaru projekcji (stąd inna nazwa tomografia rentgenowska). W wybranej płaszczyźnie dokonuje się pomiarów projekcji po liniach biegących pod różnym kątem i w różnych odległościach od badanego obiektu. Przekrój obiektu jest rekonstruowany numerycznie na podstawie zmierzonych projekcji.

Obrazowany jest współczynnik natężenia promieniowania X przez obiekt. Wielkość obrazu może być różna i jest zależna od ustawień tomografu, najczęściej jest to 512 na 512 wokseli. Piksel obrazu jest uzyskiwany podczas rekonstrukcji obrazu i reprezentuje przenikalność promieniowania X. Kontrast i rozdzielcość zależy od tych samych parametrów co w klasycznej radiografii.

W standardzie DICOM technika jest oznaczana skrótwcem „CT”.

- Obrazowanie metodą rezonansu magnetycznego — MRI

Sposób tworzenie obrazu MRI jest wysoce skomplikowanym procesem, którego szczegółowy opis przekracza zakres niniejszego opracowania. Obrazowana jest sumaryczna gęstość atomów wodoru (protonów) w badanym obiekcie. W zależności od sekwenacji pobudzeń polem elektromagnetycznym, wyróżniamy trzy typy obrazów: PD, T1 i T2. Kontrast zależy od gęstości protonów, czasu relaksacji podłużnej i poprzecznej, prędkości przepływu płynu. Rozdzielcość zależy od parametrów skanera (rozmiar woksela).

W standardzie DICOM modalność rezonansu magnetycznego jest oznaczana jako „MR”.

- Ultrasonografia

Podeczas badania ultrasonograficznego generujemy fale akustyczne o wysokich częstotliwości skierowane w stronę obiektu, następnie rejestrujemy fale odbite. Obrazowana jest różnica gęstości poszczególnych warstw znajdujących się w obiekcie.

Zbieranie danych odbywa się przez cyklicznie wysyłanie i odbieranie fal ultradźwiękowej pod różnymi kątami. Z każdego cyklu jest tworzona jedna linia, obraz jest tworzony z wielu linii, które następnie są układane pod różnymi kątami, odpowiadającymi ich rzeczywistemu ułożeniu na głowicy. Wielkość obrazu jest zależna od algorytmu rekonstrukcji i jest z góry ustaliona przez producenta aparatu. Różnice pomiędzy pikselami definiują umowną różnicę gęstości zależną od aparatu. Kontrast zależy od częstotliwości fali, głębokości badanego obiektu, liczby piezoelektryków w głowicy, obrazowanej struktury. Rozdzielcość zależy od czasu trwania impulsu zaburzenia oraz od szerokości wiązki ultradźwiękowej (powierzchnia czynna przetworników).

W standardzie DICOM obraz ultrasonograficzny jest oznaczano jako „US”. Obrazy dopplerowskie „Color flow Doppler(CD)” i „Duplex Doppler(DD)” były kiedyś w standardzie, ale zdecydowano się je wycofać.

- Scyntygrafia

Obrazowa technika diagnostyczna z gałęzi medycyny nuklearnej. Polega na wprowadzeniu do organizmu radiofarmaceutyku, czyli związków chemicznego zawierającego izotop. Charakteryzuje się on krótkim czasem rozpadu i powinowactwem chemicznym z badanymi organami. Wykrywa się rozpad zachodzący w ciele poprzez rejestrację:

Punkty *PatientPosition* odpowiadają punktom P_{xyz} z równania ze standardu DICOM.

UWAGA: Wszystkie obliczenia odbywają się we współrzędnych jednorodnych.

Na równaniu z poprzedniego punktu wykonuje takie przekształcenie:

$$PatientPosition = imgMatrix * ScenePosition$$

$$imgMatrix^{-1} * PatientPosition = imgMatrix^{-1} * imgMatrix * ScenePosition$$

$$imgMatrix^{-1} * PatientPosition = ScenePosition$$

$$ScenePosition = imgMatrix^{-1} * PatientPosition$$

gdzie:

- *imgMatrix* — macierz przekształcenia obrazu, o której będzie później opisana
- *ScenePosition* — pozycja na obrazie, która nas interesuje
- *PatientPosition* — jeden z punktów względem pacjenta.

Wygląd macierzy *imgMatrix*:

$$\begin{bmatrix} X_x & Y_x & 0 & 0 \\ X_y & Y_y & 0 & 0 \\ X_z & Y_z & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Powyzsza macierz różni się od macierzy definiowanej w standardzie. Po pierwsze wartości z $\overset{\text{Dicom}}{\text{Tag}}$ Pixel Spacing (0x0028, 0x0030) zostały pominięte, a nadano im wartość 1. Po drugie - pozycja z $\overset{\text{Dicom}}{\text{Tag}}$ Image Position (0x0020, 0x0032) została zrównana do punktu zerowego, dzięki temu, wynik też będzie względem punktu zero. Wyznaczenie macierzy *imgMatrix* jest jednorazowe.

Po wyznaczeniu sześciu punktów *ScenePosition*, po jednej dla każdego punktu *PatientPosition* są one zapisywane. *ScenePosition* odpowiada pozycji punktów na obrazie w pozycji startowej.

Na scenie, której jest wyświetlany obraz, użytkownik może obracać obraz o dowolny kąt, według własnego uznania. Te przekształcenia są realizowane za pomocą macierzy rotacji, dalej zwaną jako *rotateTransform*. Macierz *rotateTransform* jest przesyłana do naszego obiektu *Sokar::ImageOrientationIndicator* za każdym razem, kiedy zostanie zmieniona.

Ostateczne wyznaczenie pozycji punktów pacjenta na obrazie odbywa się przez mnożenie lewostronne *rotateTransform* i *ScenePosition*.

$$rotateTransform * ScenePosition$$

Wyznaczana jest w ten sposób pozycja sześciu punktów pacjenta na płaszczyźnie sceny wyświetlanej. Następnie określany jest, na której z ośmiu części płaszczyzny jest umieszczony dany punkt. Podział płaszczyzny jest widoczny na rysunku 4.13. Tej płaszczyźnie nadawany jest tytuł w postaci litery, która oznacza stronę pacjenta. Jeżeli punkt znajduje się w centrum, na przecięciu osi, to oznacza, że punkt znajduje się za lub przed ekranem, więc jest pomijany. Następnie do czterech pól wyświetlających zostają wstawione następujące teksty:

2.2.1 Podstawowe parametry obrazu cyfrowego

2.2 Parametry obrazów

Standard DICOM nazwana techniką obrazowania modalnością (ang. modality).

- PET-MRI — połączanie PET z rezonansem magnetycznym

• PET-CT — połączanie PET z wizualizacją tomograficzną komputerową

Istnieją badania fizyczne w swojej rożnej technice, takie jak:

W standardzie DICOM obraz jest oznaczany jako „PT”.

czość matricy detektora oraz liczba detektorów.
obiektyu oraz polozienia obiektu. Na rozdíl od cii pozycji bety plus. Rejestrujemy fotony powstające podczas absorpcja promieniowcza, ulegająca rozpadowi bety plus. Rejestrujemy fotony powstające podczas absorpcja promieniowcza, promieniowcza (pozitronów) jest dodatnia a negatywna (antylektronów). Zdjęcie

technika obrazowania z galęzi medyczny nuklearnej, w której rejestruję się pro-

- Tomografia PET

W standardzie DICOM obraz jest oznaczany jako „PT”.

detektorów.

rozdielcoče ma wypływ przestrzenia rozdzieliczoče matricy detektora oraz liczba rozdielcoče detektorek, odległość od obiektu oraz polozenie obiektu. Na wypłaszczonej izoparze ulegająca rozpadowi z emisją promieniowania gamma. Kontраст zalezły od kątowego izotopy ulegająca rozpadowi z emisją promieniowania (fotonów) jest radioterapeutyczny, promieniowcza gamma. Zdjęcie radioterapeutyczne, w której rejestruję się pro-

- Tomografia SPECT

W standardzie DICOM obraz scyntygraficzny jest oznaczany jako „NM”.

kamery, gąbkowa marmurami lub kamarami Angera. Rozdzielcoče zależy od mrozliwosci kamery scyntylacyjnej, zwanych marmurek. Detektor. Położenie obrazu zależy od rozłożenia liczby spłotek detektorów. Detekcja odbywa się za pomocą kolumnatora, scyntylatora, fotopowielacza i układu li-

się powiększenia.

ogół promieniowania wykorzystanego podczas tego rozpadu, a nastepnie przedstawia

- „H” — [0, 0, +1, 1]
- „F” — [0, 0, -1, 1]
- „P” — [0, +1, 0, 1]
- „A” — [0, -1, 0, 1]
- „L” — [+1, 0, 0, 1]
- „R” — [-1, 0, 0, 1]

na plazmę pozycji szesciu (punktów) na plazmę pozycji szesciu (punktów), które będą odpowiadaly trzonem pacjenta: wyciągnięte szescie punktów o nastepujących współrzędnych jest niezależne mody. Mody pacjentu są już w skrótu ujęte w spłotek detektorów, dalej uzyskany pod

szczególnie rzeź biologicznych makiety do wektora reprezentujących pozycje pacjenta, dłuższa jest to przekształcanie makiety we spłotek detektorów, tzweta to po- Praktyczne rzeź biologicznych makiety do wektora reprezentujących pozycje pacjenta, dłuższa jest to przekształcanie makiety we spłotek detektorów, tzweta to po- szkie na obrąbie.

nie wyznaczania stromy pacjenta ta warstwy moze wyhosić 1, poniewaz odpowiadają skale. • Δ_i^j — rzeczywista wielkość obrazu wyrażona w milimetrach. W algorytmie

Zero oznacza początek.
 i, j — oznacza ją spłotek na macierzy obrazu, odpowiadając kolumnie i wiersz.
 X_{xyz} — trzy ostaci w warstwach zaczynająca Tag Image Orientation (0x0020, 0x0037)
 S_{xyz} — trzy pierwsze wartości zaczynająca Tag Image Orientation (0x0020, 0x0037) do użyczenia wykonywacnego pomiaru.

S_{xyz} — trzy wartości z elementu zaczynająca Tag Image Position (0x0020, 0x0032). Oznacza on punkt pozycji pacjenta wyrażony w milimetrach w stosunku do użyczenia wykonywacnego pomiaru.

P_{xyz} — kordynaty wokselu (i, j) we spłotek obrazu wyrażone w milimetrach

$$\begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_{\Delta^i} \\ Y_{\Delta^i} \\ Z_{\Delta^i} \\ S_x \\ S_y \\ S_z \end{bmatrix} = M \begin{bmatrix} i \\ j \\ 1 \end{bmatrix}$$

Wartosc Tag Image Orientation (0x0020, 0x0037) skladajaca sie z szesciu liczb, opowied- sposob interpretowania danych:
które określają dalej $X_x, X_y, X_z, Y_x, Y_y, Y_z$. Standard DICOM definiuje nastepujacy

W dokumencie są wielokrotnie zawarte odniesienia do znaczników DICOM. Dlatego aby zwiększyć czytelność pracy, została zastosowana konwencja poprzedzania znaczników przedrostkiem ^{Dicom} Tag składającym się z numeru grupy i elementu grupy zapisanych heksadecymalnie. Przykład poniżej:

^{Dicom} Tag PatientID (0x0010, 0x0020)

Oznacza to, że jest to znacznik o słowie kluczowym „PatientID”, numerze grupy 10₁₆ i numerze elementu 20₁₆.

Wyrażenie „informacja ta zawarta w znaczniku ...” będzie oznaczało, że ta informacja znajduje się w elemencie danych o znaczniku.

Dodatkowo w dokumencie PDF można kliknąć na nazwę klasy i użytkownik zostanie przekierowany do strony <https://dicom.innolitics.com/ciods> poprzez wyszukiwarkę DuckDuckGo, na której znajduje się przeglądarka znaczników DICOM.

Każdy obraz cyfrowy jest maciązą pikseli o ustalonych rozmiarach. W przypadku standardu DICOM obrazy są matrycami wokseli, posiadającymi wysokość (zapisaną w ^{Dicom} Rows (0x0028, 0x0010)) oraz szerokość (zapisaną w ^{Dicom} Columns (0x0028, 0x0011)). Do poprawnej interpretacji znaczenia macierzy służą znacznik ^{Dicom} Tag Photometric Interpretation (0x0028, 0x0004), informujący o fotometrycznym znaczeniu wokseli. Standard DICOM definiuje następujące wartości tego tagu (wraz z wyjaśnieniem):

- „MONOCHROME1” i „MONOCHROME2” — ta wartość woksela odwzorowuje skale monochromatyczną, odpowiednio od jasnego do ciemnego i od ciemnego do jasnego.
- „PALETTE COLOR” — ta wartość woksela jest używana jako indeks w każdej z tabel wyszukiwania kolorów palety czerwonej, niebieskiej i zielonej. Palety mają swoje własne tagi. Wartość raczej rzadka i nie spotykana.
- „RGB” — oznacza, że woksel jest trzy-kanałowym pikselem RGB (kanaly: czerwony, zielony i niebieski).
- „HSV” (ang. *Hue Saturation Value*) — woksel reprezentuje piksel w modelu przestrzeni barw zaproponowany w 1978 roku przez Alveya Raya Smitha. Model ten nawiązuje do sposobu w jakim widzi oko człowieka. Wartość wycofana.
- „ARGB” — ta wartość woksela to piksel RGB z dodatkowym kanałem przezroczystości. Wartość wycofana.
- „CMYK” — ten woksel to piksel w modelu czterech podstawowych kolorów farb drukarskich stosowanych powszechnie w druku wielobarwnym w poligrafii: cyjan, magenta, żółty, czarny. Wartość wycofana.
- „YBR_FULL” — ten woksel to piksel w modelu przestrzeni barw nazwanej YCbCr. Dodatkowo standard zdefiniował pochodne tej wartości: „YBR_RCT”, „YBR_FULL_422”, „YBR_PARTIAL_422”, „YBR_PARTIAL_420”, „YBR_ICT”, ale wszystkie są już wycofane.

Wiele elementów danych lub wartości zostały wycofane ze standardu DICOM w wersji 3.0. Oznaczane są jako wycofane (ang. *retired*). Można dalej wspierać ich obsługę w celach wstępnej kompatybilności, ale nie jest to wymagane.

Połączenie macierzy

Ostatnim krokiem jest połączenie macierzy w jedną. Dlatego cztery macierze są mnożone za pomocą wirtualnej funkcji *Sokar::DicomScene::getPixmapTransformation()*. Kod funkcji:

```
1 QTransform DicomScene::getPixmapTransformation() {
2     QTransform transform;
3     transform *= centerTransform;
4     transform *= scaleTransform;
5     transform *= rotateTransform;
6     transform *= panTransform;
7     return transform;
8 }
```

Qt::QTransform posiada operator mnożenia, dlatego można mnożyć obiekty tej klasy jak liczby. Realizuje to następujące równanie:

*panTransform * rotateTransform * scaleTransform * centerTransform*

4.6.4 Ustalanie pozycji pacjenta względem sceny

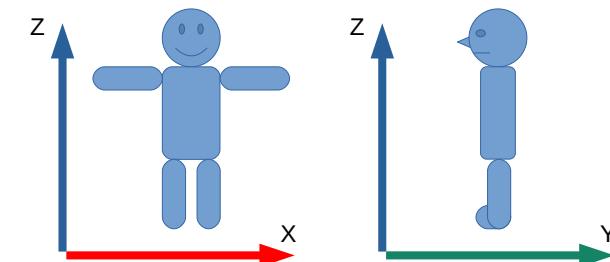
W obrazie DICOM jest pośrednio zapisana informacja o ułożeniu obrazu względem pacjenta. Celem algorytmu jest określenie jaką pozycję przyjmuje pacjent w stosunku do sceny tak, aby można było wyświetlić tą pozycję na scenie.

Format zapisu informacji o orientacji obrazu

Informacje o orientacji oraz pozycji względem pacjenta znajdują się w odpowiednio w tagach ^{Dicom} Tag Image Orientation (0x0020, 0x0037) i ^{Dicom} Tag Image Position (0x0020, 0x0032).

Standard DICOM zdefiniował ułożenie osi we współrzędnych kartezjańskich następująco:

- „x” — oś przechodząca od prawej do lewej strony pacjenta, „L” oznacza zwrot zgodny z osią, a „R” oznacza zwrot przeciwny
- „y” — oś przechodząca od przodu do tyłu pacjenta, „P” oznacza zwrot zgodny z osią, a „A” oznacza zwrot przeciwny
- „z” — oś przechodząca od dołu do góry pacjenta, „H” oznacza zwrot zgodny z osią, a „F” oznacza zwrot przeciwny



Rysunek 4.12: Wizualizacja układu osi współrzędnych kartezjańskich pacjenta. Zdjęcie własne.

2.2.4 Stosunek sygnału do szumu (SNR)

Rodzaj i poziom szumu zależy od techniki obrazowania. Stosunek sygnału do szumu ma decydujący wpływ na widoczność obiektów, kontrast oraz percepcję szczegółów w obrazie.

2.2.5 Poziom artefaktów

Artefakty to zjawiska fałszujące obraz poprzez tworzenie struktur w obrazie, nie istniejących w rzeczywistości. Jest to problem występujący w różnych technikach obrazowania. Najbardziej znanymi artefaktami są np. w badaniu USG tak zwany warkocz komety w przypadku obiektów o wysokiej różnicy impedancji w stosunku do otoczenia.

2.2.6 Poziom zniekształceń przestrzennych

Zniekształcenia przestrzenne powstają w wyniku geometrycznego ułożenia i kształtu obiektu badanego oraz aparatu pomiarowego. Przykładem takiego zniekształcenia mogą być różne powiększenia obiektów zależne od głębokości ich ułożenia w USG, zmiana pozycji pacjenta (przez ruchy klatki piersiowej w czasie badania), czy deformacja obrazu spowodowana zmianami rozkładu pola magnetycznego przez metalowe obiekty w znajdujące się w tym samym pomieszczeniu w przypadku badań MRI.

2.3 Prezentacja obrazów medycznych

W celu przeglądania i porównywania należy posiadać narzędzie do wyświetlenia w sposób poprawny, najlepiej jednym i tym samym programem.

2.3.1 Przeglądarki obrazów

Przeglądarki obrazów to programy należące do kategorii przeglądarki plików. Zwykle przeglądarki obrazów takich jak jpg, png lub gif wyświetlają obraz w takiej postaci jakiej jest zapisany, najpierw przeprowadzając dekompresję tego obrazu. W przypadku obrazów medycznych najczęściej nie mamy do czynienia z danymi reprezentującymi kolory w spektrum światła widzialnego. Przeglądarka obrazów DICOM musi wygenerować kolorowy obraz z danych na podstawie parametrów obrazu.

2.3.2 Funkcje przeglądarki obrazów

Obsługa wielu formatów danych

Standard DICOM przewidział możliwość zapisania wielu typów danych w różnych formatach, nie tylko obrazów, ale też nagrań nagrani audio i tekstów. Przeglądarka obrazów DICOM może mieć możliwość od czytania, wyświetlenia lub odsłuchania danych.

Podstawowe operacje na obrazie

- Skalowanie lub powiększenie, czyli możliwość powiększenia lub zmniejszenia wyświetlanego obrazu o pewny współczynnik skalujący.

```
1 QTransform transform;
2 transform.translate(50, 50);
3 transform.rotate(45);
4 transform.scale(0.5, 1.0);
```

Powyższe przekształcenie macierze skaluje obiekt na 50% szerokości, obraca o 45 stopni, przesuwa o 50 punktów na osi x i y.

Taką macierz można nałożyć na obiekty klasy *Qt::QGraphicsPixmapItem*.

Interakcja z użytkownikiem

Trzy macierze (bez wyśrodkowującej) są zmieniane w trakcie interakcji z użytkownikiem. Są zmieniane w dwóch przypadkach: po odebraniu sygnału od paska zadań, obiektu klasy *Sokar::DicomToolbar* lub podczas ruchu myszki, gdy wcisnięty jest prawy przycisk.

Zmiany poprzez oderanie sygnału

Na pasku zadań, nad sceną, znajduje się szereg przycisków, które po wcisnięciu wysyłają sygnał do obecnej sceny poprzez obiekt klasy *Sokar::DicomView*. Sposób wysyłania sygnałów jest szerzej opisany w sekcji 4.5.4.

Po otrzymaniu odpowiedniego sygnału jest wykonywana operacja na macierzy. Odbiór wszystkich sygnałów jest implementowany przez wirtualną funkcję *Sokar::DicomScene::toolBarActionSlot()*, która jest slotem.

Lista opisów reakcji na sygnały (stan zerowy macierzy, to stan w którym macierz nie wykonuje żadnych operacji):

- **ClearPan** — przywraca macierz przesunięcia do stanu zerowego
- **Fit2Screen** — przywraca macierz skali do stanu zerowego, następnie wylicza nową skalę w zależności od wymiarów obrazu i sceny
- **OriginalResolution** — przywraca macierz skali do stanu zerowego
- **RotateRight90** — na macierzy rotacji zostaje użyta funkcja *Qt::QTransform::rotate()* z parametrem 90.
- **RotateLeft90** — na macierzy rotacji zostaje użyta funkcja *Qt::QTransform::rotate()* z parametrem -90.
- **FlipHorizontal** — na macierzy rotacji zostaje użyta funkcja *Qt::QTransform::scale()* z parametrami 1 i -1.
- **FlipVertical** — na macierzy rotacji zostaje użyta funkcja *Qt::QTransform::scale()* z parametrami -1 i 1.
- **ClearRotate** — przywraca macierz rotacji do stanu zerowego

Po jakiekolwiek zmianie macierzy jest wywoływana funkcja *Sokar::DicomScene::updatePixmapTransformation()*, która odświeża macierz przekształcenia na obiekcie *pixmapItem*.

- Przypadkowa uzywania mikroskopu, jest to mozaikowe mikroskopowe powielkszenie obrazu na odcialu.
- Lupa, skallowanie mikroskopu, jest to mozaikowe mikroskopowe powielkszenie obrazu.
- Przesuwanie (ang. pan), czyl mozaikowe przesuwania obrazu o dowolny wektor. Jest to przesuniecie lupy w określonej kierunku.

- Obsługa DICOMDIR. Jest to mozaikowe wczytawanie pliku DICOMDIR i wyswietlenie struktury sepii bąblach. Plik DICOMDIR to wiele zindeksowanych plików zawierających
- Wczytanie wejtu pliku i ich połączenie w formie filmu, czyl mozaikowe wczytanie wielu plików z tej samej sepii, ulozenie ich wzdłuż pozytywnej osi.

Obsługa wejtu pliku

- Przyszanialna fragmentacja obrazu w celu lepszej vizualizacji bąblów wodnych i jednorodnych.
- Maski (ang. overlay). Jest to mozaikowe natokenia maski, elementu, który będzie tyczącego.

- Okienkowanie. Termin odnosi się do uzywania funkcji okna cyfrowego w celu za-
- Miany obrazu danych na obraz monochromatyczny mimoży do wyselekcji. Okienkowanie jest szczególnie opisane w sekci 4.6.2 wraz z generowaniem obrazu monochromatu.

- Rotacja i obrótka lustrzane, czyl mozaikowe obrócenia obrazu o zadany kąt oraz mozaikowe uzywanie obrócenia odbicia lustrzane do dwu osiach X i Y.

Rysunek 2.1: Narzędzie Lupa w programie MedDream DICOM Viewer. Zdjēcie uzyte za zgoda Softneta UAB.



- Przypadkowa uzywania mikroskopu, jest to mozaikowe mikroskopowe powielkszenie obrazu.
- Lupa, skallowanie mikroskopu, jest to mozaikowe mikroskopowe powielkszenie obrazu.
- Przesuwanie (ang. pan), czyl mozaikowe przesuwania obrazu o dowolny wektor. Jest to przesuniecie lupy w określonej kierunku.

Prykład dzialania:
scale()

funckje Qt::Transform::rotate() i skallowanie implementowane przez Qt::Transform::translate() obrot implementowany przez Qt::Transform::translate(jakie jakie jest klasa Qt::Transform. Implementowane założowana macierz 3 na 3, jak rownież wbudowane operacje takie jak: przerobianie implementowane przez Qt::Transform::translate(), obrot implementowany przez Qt::Transform::rotate()

Współczesne jednorodne

4.6.3 Tworzenie transformat i ich użycie na obrazie

```

1 <pallete display="Hot Iron" name="HOTIRON">
2
3 <entry red="0" green="0" blue="0"/>
4 <entry red="255" green="0" blue="0"/>
5 <entry red="0" green="255" blue="0"/>
6 <entry red="0" green="0" blue="255"/>
7 ...
8 <entry red="254" green="0" blue="0"/>
9 <entry red="255" green="0" blue="0"/>
10 <entry red="255" green="2" blue="0"/>
11 <entry red="255" green="2" blue="0"/>
12 <entry red="255" green="0" blue="0"/>
13 ...
14 <entry red="255" green="0" blue="255"/>
15 <entry red="255" green="252" blue="248"/>
16 <entry red="255" green="255" blue="255"/>
17 <entry red="255" green="255" blue="255"/>
18 </pallete>
```

Hotbar: Klasa Sołka: Pixel Pallete ma dziedziczącą ją klasę PixelFormat. Pallete ma dziedziczącą ją klasę PixelFormat, która ma dziedziczącą ją klasę ImageFormat. ImageFormat ma dziedziczącą ją klasę ImageFormatSupportException. Przykładowy wygląd pliku Pallete

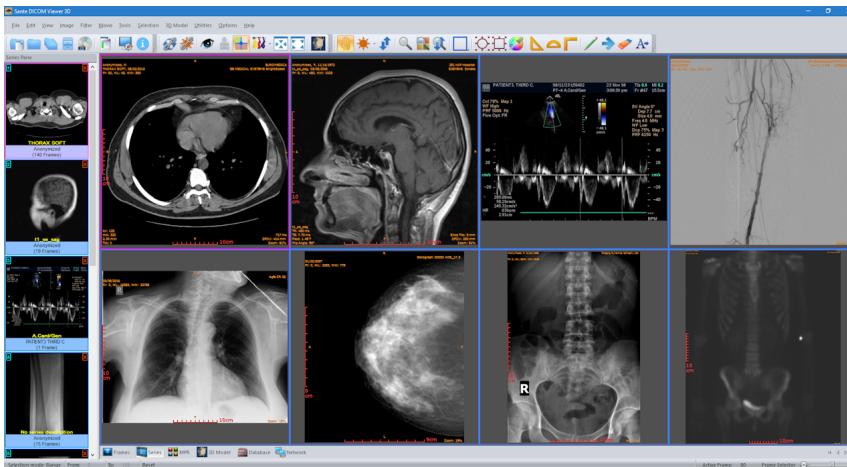
```

40 }
41 return true;
42 pixmap, convertFromImage(qImage);
43 }
44 default: /* * przypadek inny niż jest zwracany wyjątek */
45 break;
46 genQPixmapOfType<qint64>());
47 case qdm: :PixelFormat : :UNIT64:
48 genQPixmapOfType<qint64>();
49 break;
50 case qdm: :PixelFormat : :UNIT32:
51 genQPixmapOfType<qint32>();
52 break;
53 break;
54 genQPixmapOfType<qint32>();
55 break;
56 genQPixmapOfType<qint32>();
57 case qdm: :PixelFormat : :UNIT32:
58 genQPixmapOfType<qint64>();
59 break;
60 case qdm: :PixelFormat : :UNIT64:
61 genQPixmapOfType<qint64>();
62 break;
63 throw Sołka: :ImageFormatNotSupportedException();
64 default: /* przypadek inny niż jest zwracany wyjątek */
65 break;
66 }
```

Pallete

- Wyświetlanie wielu obrazów jednocześnie. Jest to możliwość wyświetlania kilku obrazów w postaci tabelki, w której każda komórka była by innym obrazem.

Przykład wyświetlania wielu obrazów na raz w jednym oknie znajduje się na rysunku 2.2



Rysunek 2.2: Wyświetlenie wielu obrazów na raz w jednym oknie w przeglądarce Sante DICOM Viewer 3D Pro. Zdjęcie użyte za zgodą Santesoft.

Generowanie obrazów woliumetycznych

Jeżeli mamy do dyspozycji wiele obrazów tomograficznych o znanych parametrach to możemy wczytać je, poszgregować a następnie wygenerować trójwymiarowy obiekt, który wyświetlany jest ekranie komputera za pomocą trójwymiarowej grafiki komputerowej.

Przykład takiego obrazu znajduje się na rysunku 2.3.

Analiza i przetwarzanie danych

- Histogram, czyli możliwość wygenerowania histogramu obrazu.

Histogram to wykres przedstawiający dystrybucję wartości numerycznych obrazu.

- Mierzenie i wykonywanie pomiarów. Pozwala na określenie odległości pomiędzy dwoma punktami przez lekarza lub zmierzenie wielkości/pola zadanego kształtu.

- Rekonstrukcja wielopłaszczyznowa. Obrazy tomograficzne przedstawiają przekroje. Jeżeli parametry wielkości woksela są dostępne to istnieje możliwość wygenerowania nowego obrazu, który byłby przekrojem poprzecznym.

Przykład generowania rekonstrukcji wielopłaszczyznowej jest pokazany na rysunku 2.4

```

5     std::vector<std::thread> threads;
6
7     quint64 max = imgDimX * imgDimY;
8     quint64 step = max / QThread::idealThreadCount();
9
10    for (int i = 1; i < QThread::idealThreadCount(); i++) {
11        std::thread t(&Scene::genQPixmapOfTypeWidthWindowThread<IntType, WinClass>,
12                      this,
13                      i * step,
14                      std::min((i + 1) * step, max));
15
16        threads.push_back(std::move(t));
17    }
18
19    /* W celu zmniejszenia ilości wątków, wątek obecny też zostanie wykorzystany */
20    genQPixmapOfTypeWidthWindowThread<IntType, WinClass>(0, step);
21
22    /* Czekanie na wszystkie wątki */
23    for (auto &t: threads) t.join();
24 }
```

- Sokar::Monochrome::Scene::genQPixmapOfType()*

Jest to funkcja pomocnicza ustalająca obecną klasę obecnego „okna”, aby móc wykonać funkcje *Sokar::Monochrome::Scene::genQPixmapOfTypeWidthWindow()*. Kod funkcji:

```

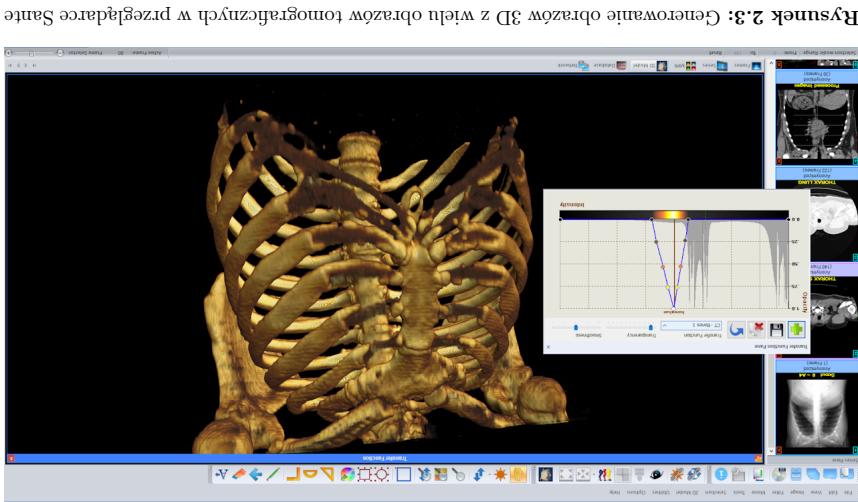
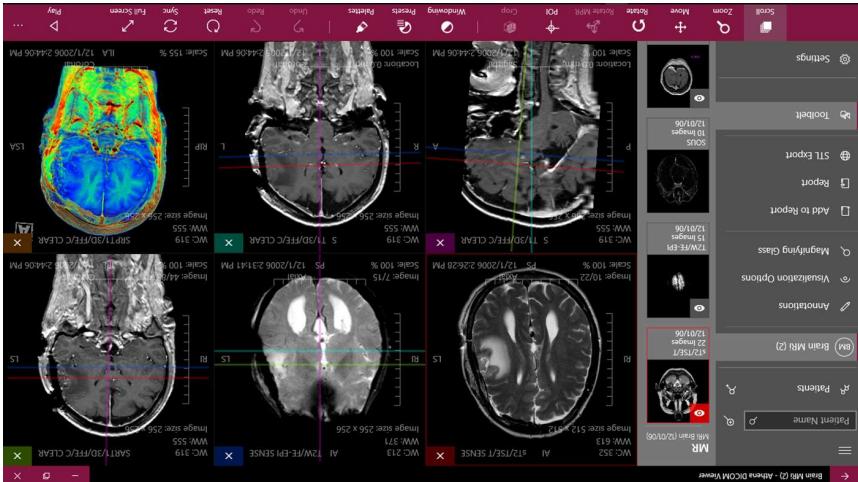
1 template<typename IntType>
2 void Monochrome::Scene::genQPixmapOfType() {
3
4     switch (getCurrentWindow()->type()) {
5         case Window::IntDynamic:
6             genQPixmapOfTypeWidthWindow<IntType, WindowIntDynamic>();
7             break;
8
9         case Window::IntStatic:
10            genQPixmapOfTypeWidthWindow<IntType, WindowIntStatic>();
11            break;
12
13         default:
14             throw WrongScopeException(__FILE__, __LINE__);
15     }
16 }
```

- Sokar::Monochrome::Scene::generatePixmap()*

Funkcja odświeża okienko i sprawdza, czy odświeżenie obrazu jest konieczne, następnie sprawdza typ liczby woksela i uruchamia *Sokar::Monochrome::Scene::genQPixmapOfType()*. Kod funkcji:

```

1 bool Monochrome::Scene::generatePixmap() {
2
3     /* Odświeżamy okno i sprawdzamy czy odświeżenie obrazu jest konieczne */
4     getCurrentWindow()->genLUT();
5     if (lastPixmapChange >= getCurrentWindow()->getLastChange()) return false;
6
7     /* Sprawdzamy typ liczby woksela obrazu */
8     switch (gdcmImage.GetPixelFormat()) {
9         case gdcm::PixelFormat::INT8:
10            genQPixmapOfType<qint8>();
11            break;
12        case gdcm::PixelFormat::UINT8:
13            genQPixmapOfType<qint8>();
14            break;
15        case gdcm::PixelFormat::INT16:
16            genQPixmapOfType<qint16>();
17            break;
18        case gdcm::PixelFormat::UINT16:
19            genQPixmapOfType<qint16>();
20            break;
21    }
22 }
```



- **Sokar::Monochrome::Scene::genQPixmapOfTypeWidthWindow()**

```

1 template<typename MessageType> void Monochrome::Scene::genQPixmapOfTypeWidthWindow() {
2     void* type = static_cast<void*>(this);
3     /* Wyszczególnione metody */
4     template<typename MessageType> void Monochrome::Scene::tempalte<MessageType>::genQPixmapOfTypeWidthWindow();
5     template<typename MessageType> void Monochrome::Scene::tempalte<MessageType>::genQPixmapOfTypeWidthWindow();
6     auto buffer = targetBuffer<MessageType>();
7     auto buffer = targetBuffer<MessageType>();
8     origIn += from;
9     for (int i = to; i < to + offsetIn; ++i) {
10        auto buffer = targetBuffer<MessageType>();
11        /* W tym miejscu jest dokonywana liczbą na kolor */
12        *buffer++ = windowPtr->getPixel(i);
13    }
14 }
```
- **Sokar::Monochrome::Scene::genQPixmapOfTypeWidthThread()**

```

1 template<typename MessageType> void Monochrome::Scene::genQPixmapOfTypeWidthThread(int classIndex) {
2     void* type = static_cast<void*>(this);
3     /* Wyszczególnione metody */
4     template<typename MessageType> void Monochrome::Scene::tempalte<MessageType>::genQPixmapOfTypeWidthThread();
5     auto buffer = targetBuffer<MessageType>();
6     auto buffer = targetBuffer<MessageType>();
7     auto buffer = targetBuffer<MessageType>();
8     origIn += from;
9     auto buffer = targetBuffer<MessageType>();
10    auto buffer = targetBuffer<MessageType>();
11    auto buffer = targetBuffer<MessageType>();
12    auto buffer = targetBuffer<MessageType>();
13    auto buffer = targetBuffer<MessageType>();
14 }
```

Podane w indeksach wskaz po kolejnych ma iterowac. **IntType** to jest typ zmiennej określającej kolejno, który iteruje po obrazie. Jego parametrym są zakresy kolejnych indeksów, których wartości po kolejnych obrazach. Następnie dodać kolejne dwa indeksy, aby określić, ile zdjęć chodzi o. Dla tego podział musi stać się prostą zasadą, aby działały identycznie dla każdego. Mając dwa typy obiektów, każdej dostępującą typy liczb całkowitych, problematyczne. Mając dwa typy obiektów, każdej dostępującą typy liczb całkowitych, W C++ typy zmienią się, aby działały właściwie. Przykładem może być, że dla tego samego indeksu, iteracja po obrazie odwróci się w wierszach. W任何时候, jeśli obraz ma rozmiar $m \times n$, to dla indeksu i , który odnosi się do kolejnego obrazu, iteracja po obrazie odwróci się w wierszach. W任何时候, jeśli obraz ma rozmiar $m \times n$, to dla indeksu i , który odnosi się do kolejnego obrazu, iteracja po obrazie odwróci się w wierszach.

Funkcja zmiany wartości obrazu na kolor przedstawia się następująco:

```

1 int line const PixelGetPixel(quint64 value) override {
2     return *(PixelArray + signedValue + value);
3 }
```

Przykładem może być funkcja "okna". Do zaktualizowania wartości w określonym obszarze, należy przekazać go przez funkcję "okna". Po wygenerowaniu obrazu, należy przekreślić go przez funkcję "okna".

Edycja danych

- Dodawanie nowych obiektów. Pozwala na rysowanie, dodawanie figur geometrycznych lub tekstu przez lekarza i zapis tych informacji w pliku DICOM. Chodzi tu głównie o szkice i notatki tworzone podczas analizy obrazu przez personel medyczny.
- Edycja parametrów oraz anonimizacja danych. Jest to możliwość edycji parametrów w pliku DICOM w różnych celach. Funkcja jest używana do usuwania danych osobowych pacjenta w celu późniejszej publikacji obrazu.

2.3.3 Kryteria porównywania przeglądarek obrazów

Porównanie aplikacji posiadających tak wiele parametrów jak przeglądarki DICOM jest bardzo skomplikowanym procesem. Dlatego wyróżniono 26 kryteriów do ich porównywania w postaci logicznej: „tak” lub „nie”, podzielonych na 5 grup, platformy, interfejsu, wsparcia, obrazowania dwu i trójwymiarowego [Daniel Haak(2016)]. Kryteria te w jasny sposób pozwalają na ocenę praktycznych aspektów użytkowania przeglądarki.

Platforma

Grupa platforma zawiera kryterium samodzielności. Aplikacje samodzielne są zaprojektowane tak, aby nie wymagały żadnego dodatkowego sprzętu fizycznego bądź infrastruktury do poprawnego działania. Rozwiązania sieciowe określają czy aplikacja jest usługą sieciową i czy można z aplikacji korzystać jak ze strony WWW. Aplikacje są wieloplatformowe, czyli mają możliwość uruchomienia ich na różnych systemach operacyjnych Linux/MacOS/Windows oraz możliwość używania ich na urządzeniach mobilnych takich jak telefon.

Interfejs

Przeglądarka powinna mieć możliwość komunikacji z interfejsami innych systemów. Podstawowe interfejsy sieciowe to: C-STORE SCP DICOM C-STORE, C-STORE SCU, Query-Retrieve, WADO, Parameter Transfer.

Wsparcie techniczne

Aplikacja powinna mieć dostępną pisemną dokumentację oprogramowania (np. podręczniki lub strony internetowej), wsparcie przez pocztę internetową, możliwość porozumienia się z twórcą lub opiekunem oprogramowania. Forum, możliwość pytania się społeczności o opinię i ich wymiana. Wiki, strona internetowa w formacie Wikipedii dostępna dla użytkownika.

Obrazowanie dwuwymiarowe

Przewijanie(ang. *scroll*), proces wyświetlania obrazów, można poprawić dzięki zmniejszeniu interakcji z klawiaturą oraz myszką. Można to osiągnąć na przykład, oferując możliwość przejścia do następnego lub poprzedniego obrazu przez przesunięcie kółkiem myszy lub używając przycisków góra/dół na klawiaturze. Metadane, przeglądania powinna obejmować analizowanie i wyświetlanie metadanych obiektów DICOM, powinna obejmować wyświetlanie rozdzielczości obrazu, badanie (np. identyfikator podmiotu) oraz znaczniki

Implementacja dynamiczna bez tablicy LUT

W tej wersji funkcja *Sokar::Monochrome::Window::getPixel()* wygląda następująco:

```
1 inline const Pixel &getPixel(quint64 value) override {
2     if (value < x0) {
3         return background;
4     } else if (value > x1) {
5         return foreground;
6     } else {
7         return palette->getPixel(a * value + b);
8     }
9 }
```

Widzimy tutaj, że funkcja najpierw sprawdza czy zakres okienka został przekroczyony, następnie wylicza wartość obrazu i pobiera kolor z palety.

UWAGA: ponieważ nie istnieją rzeczywiste obrazy o wokselu 32-bitowym lub 64-bitowym, implementacja dynamiczna nie była testowana w warunkach rzeczywistych.

Implementacja statyczna z tablicą LUT

W wersji z LUT, podczas tworzenia okienka jest alokowany wektor obiektów *Sokar::Pixel* klasy std::vector. Standard DICOM przewiduje, że woksele mogą mieć wartości ujemne, więc tablica powinna mieć możliwość posiadania takich wartości indeksów, ale C++ nie przewiduje takiej możliwości. Dlatego wprowadzono dwie zmienne pomocnicze *maxValue* i *signedMove*. *maxValue* jest to maksymalna wartość jaką dane mogą przyjąć, jest ona równa 2^N , gdzie N to liczba bitów brana z *Dicom BitsStored* (0x0028, 0x0101). A *signedMove* to liczba przesunięcia liczb, przyjmuje wartość zero, gdy dane wokseli są całkowite nieujemne lub wartość przeciwną do *maxValue*, gdy woksele są być ujemne. Długość wektora pikseli jest sumą *maxValue* i *signedMove*. A indeks wokselu w wektorze ma wartość tego woksela zwiększoną o *signedMove*.

Wypełnienie wektora wartościami odbywa się poprzez iteracje po wszystkich możliwych wartościach, przeliczenie ich przez funkcję „okna”, a następnie wstawienie ich do wektora. W celu poprawy szybkości, zastosowano sprawdzanie czy wartości są w zakresie „okna”. Poniżej kod funkcji:

```
1 bool genLUT() override {
2     if (WindowInt::genLUT()) {
3         /* Przeskalowanie wektora, gdy jest to wymagane */
4         if (arraySize != signedMove + maxValue) {
5             arraySize = signedMove + maxValue;
6             arrayVector.resize(arraySize);
7         }
8
9         /* Wyliczenie najmniejszej wartości */
10        qreal x = qreal(signedMove) * -1;
11
12        auto &background = isInversed() ? palette->getForeground() : palette->
13            getBackground();
14        auto &foreground = isInversed() ? palette->getBackground() : palette->
15            getForeground();
16
17        /* Iteracja */
18        pixelArray = &arrayVector[0];
19        for (int i = 0; i <= arraySize; i++) {
20
21            if (x < x0) {
22                *pixelArray = background;
23            } else if (x > x1) {
24                *pixelArray = foreground;
25            } else {
```

DICOM specifies a data structure (up. specific implementation) that defines the way in which a medical image is stored in a digital format. DICOM consists of a header and a body. The header contains information about the image, such as the patient's name, date of birth, and the type of image. The body contains the image data itself.

2.4.1 Standard DICOM v3.0

Standard DICOM uses a standard file format for medical images. It supports various file types, including JPEG, PNG, and GIF. The DICOM standard defines a specific file structure, including a header and a body. The header contains information such as the patient's name, date of birth, and the type of image. The body contains the image data itself.

2.4 Format cyfrowych obrazów medycznych

Format cyfrowych obrazów medycznych (DICOM) jest odpowiedzią na potrzeby przemysłu medycznego. Jego zadaniem jest zapewnienie jednolitej i bezproblemowej wymiany obrazów medycznych między różnymi systemami informatycznymi. Format ten posiada specjalny zestaw znaków, który pozwala na przechowywanie i odczytywanie danych medycznych w sposób bezwzględnie dokładny i niezależny od technologii i platformy, na której obraz został stworzony. Format DICOM opiera się na standardach ISO i IEC, co gwarantuje jego międzynarodową zgodność i łatwość integracji z innymi systemami.

Obrazowanie trójwymiarowe

Obrazowanie trójwymiarowe (3D rendering) to proces tworzenia pełnowymiarowych obrazów medycznych, które przedstawiają struktury organizmu w przestrzeni. W tym celu do wykorzystywanych są różnorodne techniki, takie jak tomografia komputerowa (CT), rezonans magnetyczny (MRI), ultrasonografia (USG) oraz rentgenografia komputerowa (RKG). Wykorzystywane są również specjalne algorytmy, takie jak segmentacja, rekombinacja i interpolacja, aby stworzyć obraz złożony z wielu pojedynczych klatek.

Teraz algorytm służy rozdrożni. Próbujemy warstwy z określonej głębokości obrazu. Wybrane

$$b = y_1 - a * x_1$$

$$a = (y_1 - y_0)/(x_1 - x_0)$$

parametry prostego przekroju przedstawiają głębokość warstwy.

$$x_1/ = \text{rescaleSlope}$$

$$x_0/ = \text{rescaleSlope}$$

$$x_1 - = \text{rescaleIntercept}$$

$$x_0 - = \text{rescaleIntercept}$$

więc:

$$(OutputUnits - b)/m = SV$$

Wartości określone przez skalowanie, przechowywane określonej głębokości warstwy.

- *OutputUnits* — wartości wykładowe

$$\bullet SV - \text{strored values} - \text{wartości wokół obrazu}$$

$$\bullet b - \text{wartość z Tag RescaleIntercept (0x0028, 0x1052)}$$

$$\bullet m - \text{wartość z Tag RescaleSlope (0x0028, 0x1053)}$$

$$\text{gdzie: } OutputUnits = m * SV + b$$

Standard DICOM posiada wiele warstw, które mają różne funkcje. Warstwa najdalej od obiekta ma funkcję przekształcania danych medycznych na format, który jest łatwy do zrozumienia i łatwy do przetwarzania.

$$\bullet x_1 \text{ i } y_1 - \text{współrzędne punktu drążego obrazu}$$

$$\bullet width - szerokość określona$$

$$\bullet center - środki określone$$

$$y_0 = 1.0$$

$$y_1 = 0.0$$

$$x_1 = center + width/2$$

$$x_0 = center - width/2$$

Najpiękniejszym okresemDICOM jest jego zdolność do tworzenia obrazów medycznych, które przedstawiają struktury organizmu w pełnej rozdzielcości i szczegółowości.

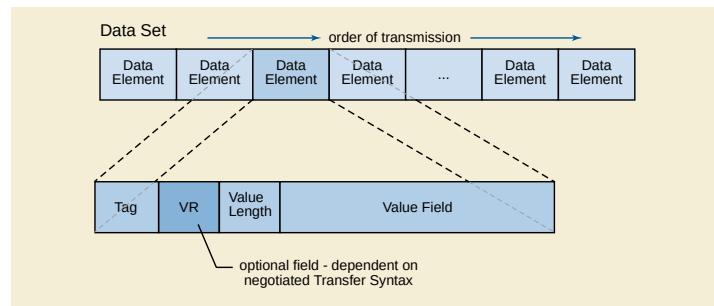
Wyznaczenie parametrów okna

Standard DICOM wersji trzeciej to standard definiujący ujednoliciony sposób zapisu i przekazywania danych medycznych reprezentujących lub związanych z obrazami diagnostycznymi w medycynie. Standard został wydany w 1993 przez dwie agencje ACR (American College of Radiology) i NEMA (National Electrical Manufacturers Association). Wcześniejsze wersje nazywały się ACR/NEMA v1.0, wydana w 1983 roku i ACR/NEMA v2.0, wydana w 1990 roku, stąd wersja trzecia. Od wydania wersji trzeciej w 1993, standard jest ciągle rozwijany i uzupełniany o nowe elementy. W obecnej chwili standard DICOM definiuje 81 różnych typów badań.

UWAGA: Za każdym razem kiedy jest odniesienie do obecnego standardu DICOM, w domyśle jest to odsłona numer 2019a.

2.4.2 Sposób zapisu danych w pliku DICOM

Plik w formacie DICOM przypomina zbiór elementów danych z rekordami. Zbiór nazywa się **Data Set** i składa się z rekordów, które nazywają się **Data Element**. Elementy danych są ulozone w postaci listy. Element danych może zawierać w sobie listę elementów danych.



Rysunek 2.5: Elementy danych w zbiorze elementów danych. Zdjęcie ze standardu DICOM dostępne pod adresem http://dicom.nema.org/medical/dicom/2019a/output/chtml/part05/chapter_7.html.

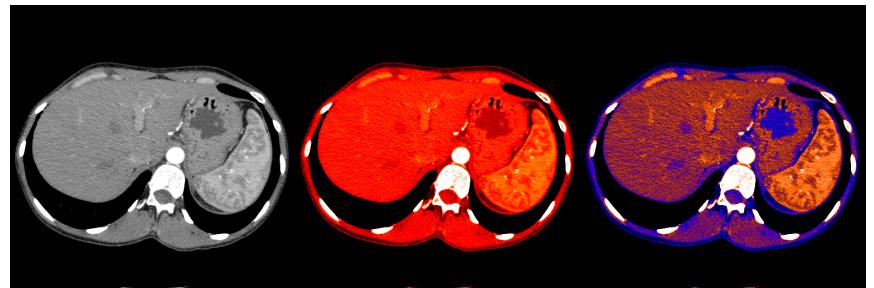
Element danych

Element danych, zwany przez standard DICOM **Data Element** jest rekordem, który przechowuje pojedynczą informację o obiekcie. Składa się z czterem elementów:

- **Tag** — to unikalny identyfikator, dalej zwany znacznikiem, jest złożony z dwóch liczb: numer grupy (`uint16`) i numer elementu (`uint16`) grupy. Informuje o tym co dany rekord w sobie zawiera. W jednym zbiorze elementów nie mogą się pojawić dwa elementy posiadających ten sam znacznik.

Na przykład: jeżeli liczby znaczniku przyjmą wartości odpowiednio wartość 0010_{16} i 0010_{16} to oznacza, że jest to znacznik ^{Dicom}_{Tag} PatientName (0x0010, 0x0010), czyli zawiera w sobie parametr zawierający nazwę pacjenta.

Dokładne omówienie znaczników znajduje się w sekcji 2.4.2.



Rysunek 4.11: Paleta Hot Iron (na środku) i Hot Metal Blue (po prawej) w porównaniu do palety w skali szarości (po lewej). Zdjęcie własne.

Implementacja algorytmu

Opis

Z uwagi na konieczność osiągnięcia dużej szybkości wyświetlania obrazu, warto jest szacować wartości funkcji f . Wartości tej funkcji należy przeliczyć, gdy zmienione zostaną parametry tak zwanego „okna”. Indeks koloru wyznaczany jest wtedy poprzez pobieranie wartości z tabeli o indeksie równym wartości numerycznej w obrazie. Unikamy w ten sposób wielokrotnego wyznaczania wartości funkcji, która wymaga sprawdzenia warunku, czy dana wartość mieści się w wybranym przedziale wartości, w tak zwanym „oknie”, co jest bardzo kosztowne obliczeniowo. Dlatego dobrym pomysłem jest stworzenie mniejszej tablicy typu LookUpTable, wypełnienie jej wszystkimi możliwymi wartościami obrazu, a następnie przerobienie obrazu z tablicą LUT. Ponieważ tablica LUT posiada wszystkie możliwe kombinacje wartości, jej rozmiar można wyznaczyć wzorem: $2^N * 3$, gdzie N to liczba bitów liczby. Standard DICOM definiuje, że liczby mogą mieć 8, 12, 16, 32 i 64 bity, jednakże, 12 bitowe i tak się zapisuje w postaci 16-bitowych w pamięci RAM. Dlatego możliwe wartości wielkości tablicy LUT to w przybliżeniu: 768 bajtów, 196 kilobajtów, 12,5 gigabajtów i 56 eksabajta ($55 * 10^6$ terabajtów). Alokowanie dwóch największych wartości może być niemożliwe, dlatego w pracy wykonano dwie implementacje algorytmu: z tablicą LUT (dla 8 i 16 bitowych obrazów) i bez tablicy LUT (dla 32 i 64 bitowych obrazów). Algorytm składa się z 3 części: wyznaczenie parametrów „okna”, przygotowanie „okna” (tylko gdy jest tablica LUT), wielowątkowa iteracja po obrazie.

Okno z LUT jest implementowane przez *Sokar::Monochrome::WindowIntStatic*. Okno bez LUT jest implementowane przez *Sokar::Monochrome::WindowIntDynamic*. Obie klasy dziedziczą po abstrakcyjnej klasie *Sokar::MonochromeWindow*, która z kolei dziedziczy po *Sokar::SceneIndicator*, dlatego od razu może wyświetlać obecne wartości „okna”. Decyzja o używanym „oknie” jest podejmowana podczas wczytywania obrazu przez klasę *Sokar::Monochrome::Scene*.

UWAGA: Standard DICOM zakłada, że danymi mogą być liczby całkowite (`int`) oraz zmiennoprzecinkowe (`float` lub `double`), ale praktycznie, nie ma takich aparatów medycznych, które zapisywały by takie obrazy, gdzie dane to liczby zmiennoprzecinkowe. Dlatego w pracy założono, że takie obrazy nie będą obsługiwane.

- Tag to unikalny znacznik pozwalający określić czego dotyczy dane zapisane w le-
- Tag to znacznik znacznikowi który zakłada, że wszystkie wartości oznaczonej parametry **VL** - 32-bitowa liczba binarna, która informuje o dłużosci pola danych (**Value Field**).
- Tag **Value Length**, w której **VL** - 32-bitowa liczba binarna ilość znaczników, które mająte być doppelowane do parametru liczbę bajtów.
- Tag **Value Representation**, w której **VR** - to dwa bajty w postaci tekstu, informujące o formacie w jakim parametr zostanie zapisany.
- Tag **Value** oznawcza VR-wą znajduje się w dalszej części sekcji.

Znacznik

- **Value Field** (ogólna) — pole z parametrem o długosci **VL**.
- **Value** powinno być doppelowane do parametru liczbą bajtów.
- **Value** **VL** zwykle jest liczbą parzystą. Standard DICOM zakłada, że wszystkie wartości **VL** zawsze będą parzyste.
- **Value** **Length**, w której **VL** - 32-bitowa liczba binarna ilością znaczników, które mające być doppelowane do parametru ilością znaczników.
- **Value Representation**, w której **VR** - to dwa bajty w postaci tekstu, informujące o formacie w jakim parametr zostanie zapisany.

Rysunek 4.10: Porównanie jednego obrazu z trzech różnych „funkcji okna”.



- wpisanej plików do tablicy, tak aby najmniejsza wartość obrazu miała indeks 0 a największa jest to numer HIS(Hospital Information System).
- Tag **Patient ID** (0x0010, 0x0020) — id pacjenta, unikalny identyfikator pacjenta, który zawrze-
- Tag **Patient Name** (0x0010, 0x0030) — nazwa pacjenta, czyli nazwisko, imię i nazwisko pacjenta, unikalny identyfikator pacjenta.
- Tag **Patient BirthDate** (0x0010, 0x0030) — data urodzenia pacjenta
- Tag **Patient Sex** (0x0010, 0x0040) — plec pacjenta
- Tag **Patient Age** (0x0010, 0x1010) — wiek pacjenta w czasie badania
- Tag **Study Description** (0x0008, 0x1030) — opis badania, pole wypełniające przes-
- Tag **Series Description** (0x0008, 0x103E) — opis serii, pole wypełniające przes-
- Tag **Series Instance UID** (0x0020, 0x000E) — unikalny numer serii, który jest
- Tag **Series Number** (0x0020, 0x0013) — numer instancji razy.

- y zapisie obiektu do 1.0 lub 0.0 jeżeli wyjdzie poza zakres od 1.0 do 0.0
- poprawie z pełnej plików odczytajacych wartości
- wpisanej plików do tablicy, tak aby najmniejsza wartość obrazu miała indeks 0 a największa jest to numer HIS(Hospital Information System).
- y zapisie obiektu do 1.0 lub 0.0 jeżeli wyjdzie poza zakres od 1.0 do 0.0
- wyznaczanie wartości określka.

$$y = a * x + b$$

Trzy iteracyjny po wszystkich możliwych wartościach obrazu i wykony-

256 odcieni, dla tego ląpieli jedyne z nich aby „okno”, mapowano na liczbę od 0 do 1, a pozostałe

do skali szarości moźna zobaczyć na rysunku. Taka palota barw nie koniecznie musi mieć

wyswietlone w wielokolorowej paletce barw. Przykład takiej palety Holiion w porównaniu

z inną na obrazie RGB. Na ostatnim standard DICOM przewidziane, że obraz moźna jeszcze

zapisywać do skali od 0 do 255. Obraz w 256 pozyciach jest mniej wygodny do wyswie-

Następnie iteracyjny obraz w skali 0 do 255, który jest znacznie mniej wygodny do wyswie-

glienie u do wartości określającej obrazu.

$$f(a) = \begin{cases} 255 & \text{gdy } x_1 \leq a \vee a \leq 1 \\ a * x + b & \text{gdy } x_0 < a > a > x_1 \\ 0 & \text{gdy } 0 \leq a \vee a \leq x_0 \end{cases}$$

Gdzie y_0 jest rowne 0, a y_1 jest rowne 255. Funkcja „okna” wygląda następująco:

$$x_1 = \text{center} + \frac{\text{width}}{2}$$

$$x_0 = \text{center} - \frac{\text{width}}{2}$$

okienka center i długosci width.

- **Dicom Modality** (0x0008, 0x0060) — modalność określająca rodzaj techniki diagnostycznej
- **Dicom Study Date** (0x0008, 0x0020) — data wykonania badania

Reprezentacja wartości

VR to reprezentacja wartości, który informuje w jakim formacie jest zapisany parametr obrazu. Składa się z dwóch bajtów.

Przykładowe VR:

- **AS** — Age String — wiek lub długość życia

Długość danych wynosi 4 bajty. Pierwsze trzy bajty to liczba całkowita zapisana za pomocą tekstu. Czwarty bajt to znak określający jednostkę czasu. Standard definiuje cztery możliwe jednostki czasu: „D” jako dzień, „W” jako tydzień, „M” jako miesiąc, oraz „Y” jako jeden rok.

Przykład: „018M” oznacza 18 miesięcy, „123D” oznacza 123 dni.

- **AT** — Attribute Tag — inny znacznik

Długość danych to zawsze 32 bity, są to dwie 16 bitowe liczby, odpowiednio grupa i element grupy. Ten VR jest używany kiedy wskazujemy na inny znacznik. Wartość nie jest nigdy pokazywana użytkownikowi, a jedynie używana w interpretacji przez inne algorytmy do analizy obrazu.

Przykład: znacznik **FrameIncrementPointer** (0x0028, 0x0009) jest używany kiedy w pliku jest zapisana sekwencja kilku obrazów. Wskazuje on na inny znacznik zawierający informacje, w jaki sposób ta sekwencja ma być wyświetlona.

- **DA** — Date — data lub dzień

Długość danych zawsze wynosi 8 bajtów. Data zapisana w formacie „YYYYMMDD”, gdzie: „YYYY” cztery cyfry roku, „MM” dwie cyfry miesiąca, „DD” dwie cyfry dnia w kalendarzu Gregoriańskim.

Przykład: „19800716” oznacza 16 lipca 1980

UWAGA: Standard „ACR-NEMA Standard 300”, czyli poprzednik DICOM definiował datę w sposób „YYYY.MM.DD”, według standardu DICOM, taki zapis jest nie poprawny, ale zdarzają się stare obrazy z takimi datami i *Sokar::DataConverter* obsługuje taki format.

- **DS** — Decimal String — liczba zmiennoprzecinkowa lub ciąg kilku liczb zmienno-przecinkowych zapisanych za pomocą tekstu w notacji wykładniczej

Długość jednej liczby powinna maksymalnie wynosić 16 bajtów. Dostępne znaki to „0”, „9”, „+”, „-”, „E”, „e”, „.”. Biblioteka QT posiada wbudowany konwerter liczb zapisanych w formacie wykładniczym, dlatego mój konwerter dzieli tekst i konwertuje za pomocą QT.

Przykład: „426\468 ” oznacza dwie liczby 426 i 468. Proszę zwrócić uwagę na spację na końcu.

YBR

YBR albo YC_bC_r to model przestrzeni kolorów do przechowywania obrazów i wideo. Wykorzystuje do tego trzy typy danych: Y – składową luminancji, B lub C_b – składową różnicową chrominancji Y-B, stanowiącą różnicę między luminancją a niebieskim, oraz R lub C_r – składową chrominancji Y-R, stanowiącą różnicę między luminancją a czerwonym. Kolor zielony jest uzyskiwany na podstawie tych trzech wartości. YBR nie pokrywa w całości RGB, tak jak RGB nie pokrywa YBR. Posiadają one część wspólną, a część która nie jest wspólna ulega zniekształceniu.

Wartości w pliku DICOM są ulożone w taki sposób.

$$Y_1, B_1, R_1, Y_2, B_2, R_2, Y_3, B_3, R_3, Y_4, B_4, R_4, \dots$$

Ponieważ wartości te reprezentują kolory, są już formą obrazu, ale nie można jeszcze wyświetlić na monitorze RGB. Dlatego należy przekonwertować kolor YBR na kolor RGB, iterując po wszystkich wartościach obrazu.

Poniżej przedstawiono kod źródłowy funkcji zamiany kolorów YBR na RGB.

```

1 Sokar::Pixel ybr2Pixel(quint8 y, quint8 b, quint8 r) {
2     qreal red, green, blue;
3
4     red = green = blue = (255.0 / 219.0) * (y - 16.0);
5
6     red += 255.0 / 224 * 1.402 * (r - 128);
7     green -= 255.0 / 224 * 1.772 * (b - 128) * (0.114 / 0.587);
8     green -= 255.0 / 224 * 1.402 * (r - 128) * (0.299 / 0.587);
9     blue += 255.0 / 224 * 1.772 * (b - 128);
10
11    /* W tym miejscu jest dokonywana utrata danych */
12    red = qBound(0.0, red, 255.0);
13    green = qBound(0.0, green, 255.0);
14    blue = qBound(0.0, blue, 255.0);
15
16    return Sokar::Pixel(quint8(red), quint8(green), quint8(blue));
17 }
```

4.6.2 Generowanie obrazu monochromatycznego

Obraz monochromatyczny to obraz w odcieniach szarości, od białego do czarnego lub od czarnego do białego. Dane są zapisane w sposób ciągły wartość po wartości.

Pseudokolorowanie obrazu

Mamy obraz, którego piksele to n-bitowe liczby, na przykład 16 bitowa liczba całkowita. W takiej postaci wyświetlemo obraz na monitorze RGB lub nawet na profesjonalnym 10-bitowym jest niemożliwe. Należy taką liczbę przerobić na trzy liczby, reprezentujące 3 kanały RGB, czerwony, zielony i niebieski. Dlatego do wyświetlania obrazów monochromatycznych o dużym kontraste stosuję się twór zwany okienkiem. Jest to funkcja, która mapuje n-bitwy obraz na 8-bitowy obraz w skali szarości. 8-bitów, ponieważ monitor RGB jest w stanie wyświetlić 256 odcieni szarości.

Zwiększenie kontrastu za pomocą „funkcji okna”

Jest przyjęte, że „okno” definiuje się dwoma liczbami: środkiem, oznaczanym jako *center* i długością, oznaczaną jako *width*. Wyznaczamy zakres okienka x_0 i x_1 ze środka

Rozdział 3

Biblioteki i narzędzia

3.1 CMake

CMake to wieloplatformowe narzędzie do automatycznego zarządzania procesem komplikacji programu. Jest to niezależne od kompilatora narzędzie pozwalające napisać jeden plik, z którego można wygenerować odpowiednie pliki budowania dla dowolnej platformy.

Z uwagi na to, że projekt musi mieć możliwość komplikacji na 3 platformy CMake jest idealnym rozwiązańiem. Dodatkowo w pracy tej starano się wybrać biblioteki, które komplikują się za pomocą CMake.

Licencja

CMake został opublikowany na licencji BSD, zgodnej z zasadami wolnego oprogramowania. Powstał początkowo na Uniwersytecie Kalifornijskim w Berkeley. Licencje BSD skupiają się na prawach użytkownika. Są bardzo liberalne, zezwalają nie tylko na modyfikacje kodu źródłowego i jego rozpowszechnianie w takiej postaci, ale także na rozpowszechnianie produktu bez postaci źródłowej czy włączenia do zamkniętego oprogramowania, pod warunkiem załączenia do produktu informacji o autorach oryginalnego kodu i treści licencji. W programie została załączona informacja o użyciu CMake, więc jest możliwość użycia jej w pracy.

3.2 QT

Biblioteka Qt, rozwijana przez organizację Qt Project, jest zbiorem bibliotek i narzędzi programistycznych dedykowanych dla języków C++, QML i Java.

Qt jest głównie znana jako biblioteka do tworzenia interfejsu graficznego, jednakże posiada ona wiele innych rozwiązań ułatwiających programowanie obiektowe i zdarzeniowe.

W tej pracy wybrano biblioteki Qt z uwagi na to, że posiada interfejs w C++. Kompilacja oprogramowania używającego Qt może odbywać się za pomocą dwóch narzędzi: CMake oraz dedykowanego narzędzia qmake, zrobionego specjalnie na potrzeby biblioteki Qt. Dzięki czemu cały projekt przeglądarki używa tego samego języka oraz tego samego narzędzia zarządzania komplikacją.

- About Qt — otwiera okno informacji o bibliotece Qt. Biblioteka Qt ma wbudowane takie okno w postaci `Qt::QMessageBox::aboutQt()`
- About GDCM — otwiera okno z informacjami o bibliotece GDCM, implementowane przez funkcję `Sokar::About::GDCM()`
- About Sokar — otwiera okno z informacjami o aplikacji, implementowane przez funkcję `Sokar::About::Sokar()`

4.6 Algorytmy

4.6.1 Cykl generowania obrazów

Klasa `Sokar::DicomScene` dostarcza następujące obiekty do generowania obrazu:

- `processing`, obiekt klasy `Qt::QMutex`, mutex do zablokowania podczas generowania obrazu, aby parametry obrazu nie mogły być zmieniane podczas jego generowania.
- `imgDimX` zmienna typu `uint`, oznacza szerokość obrazu w pikselach.
- `imgDimY` zmienna typu `uint`, oznacza wysokość obrazu w pikselach.
- `targetBuffer` wektor docelowego obrazu RGB o długości `imgDimX * imgDimY`, typu `std::vector<Pixel>`.

`Sokar::Pixel` to struktura reprezentująca piksel. Nie jest to w żadnym wypadku obiekt, a jedynie twór ułatwiający zarządzanie kodem.

```
1 struct Pixel {  
2     quint8 red = 0;  
3     quint8 green = 0;  
4     quint8 blue = 0;  
5 }
```

C++ od standardu C++03 przewiduje, że elementy znajdujące się w `std::vector` są ułożone ciągiem, jeden za drugim. Dlatego odwołując się do wskaźnika pierwszego elementu w ten sposób `&targetBuffer[0]`, mogę potraktować to jako tablicę.

- `originBuffer` wektor danych wypełniona danymi z jednej ramki o długości iloczynu `imgDimX * imgDimY` i ilości bajtów jednego piksela obrazu.
- `qImage` obiekt obrazu klasy `Qt::QImage`.

`Qt::QImage` można zrobić z istniejącego bufora, w tym przypadku jest to `targetBuffer`. Format obrazu to `Qt::QImage::Format_RGB888`, czyli trzy bajty, każdy na jeden kanał. Proszę zwrócić uwagę, że struktura `Sokar::Pixel` odpowiada temu formatowi. Według dokumentacji Qt obiekt ten po utworzeniu z istniejącego bufora powinien z niego dalej korzystać, dlatego zmiany `targetBuffer` nie wymagają odświeżania `qImage`.

• Help

- Exit — wyjście z aplikacji

wia zapisań obranu do pliku.

walne jest zaimplementowane przez funkcję `Qt::QImage::save()`, która umożliwia

- Export as — zapisać obrazu w formacie JPEG, BMP, GIF lub PNG, Zapisy-

powane wczytanie z tego menu

- Open Recent — program zapisać ostatnio wczytane pliki i pozwala na ich

`log::getOpenFileName()`, następuje wczytanie pliku

- Open — otwiera okienko wyboru pliku, implementowane przez `Qt::QFileDialog::`

• File

W gorniczej części okna programu znajdują się menu, obiekt klasy `Sokar::QMenuBar`.

Menu programu

w sekcji 4.5.5.

W środkowej części programu znajdują się obiekty z zakładkami, szczególnie opisany z zakładkami.

zawiera on w sobie model dzewa pliku systemu, który z kolei jest implementowany przez klasę `Qt::QFileSystemModel`. Po wybór pliku siezka jest przesyłana do obiektu zawierającego model dzewa pliku systemu, który z kolei jest implementowany

Drzewo katalogów i zakładki

Zawiera w sobie 4 elementy: menu, drzewo ze strukturą pliku; obiekt z zakładkami oraz w dolnej części okna znajdują się obiekty sugerujące, aby nie używać programu w celach medycznych.

Program działa po użyciu menu programu.

Główne okno programu jest implementowane przez `Sokar::MainWindow`. Jest wywo-

luje się za pomocą funkcji `Sokar::QMenuBar::MainWindow`.

4.5.6 Okno główne programu

WCZYTYWANIE PLIKÓW

parametrem sciezki a drugie z wczytanym plikiem.

i `Sokar::DicomTables::addDicomFiles()`. Kazała z tych funkcji ma dwa parametry

syfumie prosty obrywa się za pomocą obiektu katalogu funkcji: `Sokar::DicomTables::addDicomFile()`

pliku, natomiast trzecim parametrem może być żadny. Wysyłać może jedynie plików. Wy-

popularność serwisach internetowych o tematyce programistyycznej, pozażycie, że nasi bar-

strukturna pliku w systemie (opisane w 4.5.4), menu programu (opisane w 4.5.6), lub

otworzenie nowego pliku może odbyć się z nastąpiącymi zasadami: obiektu dzewa ze

poprzecznego programu (opisane w 4.5.3).

Spis treści nowych plików

3.2.1 Wyjmo

3.2.1.1 Wyjmo

Wyjmo to nazwa sięczyka, jak angielskie słowo „out”, po polsku „kut”.

Wydrukowanie wyjścia programistów nie jest co do tego zgodna. Alkietry zrobione na dwoch

jednakże spłoszące programistów nie jest co do tego zgodna. Alkietry zrobione na dwoch

jezdniach autorów, których powinno sięczyka, jak angielskie słowo „out”, po polsku „kut”.

3.2.2 Licencja

Licencja — wydrukowanie wyjścia programistów.

Wydrukowanie wyjścia programistów nie jest co do tego zgodna. Alkietry zrobione na dwoch

jednakże spłoszące programistów nie jest co do tego zgodna. Alkietry zrobione na dwoch

jezdniach autorów, których powinno sięczyka, jak angielskie słowo „out”, po polsku „kut”.

3.2.3 Normy i certyfikaty

Normy i certyfikaty — wydrukowanie wyjścia programistów.

Wydrukowanie wyjścia programistów nie jest co do tego zgodna. Alkietry zrobione na dwoch

jednakże spłoszące programistów nie jest co do tego zgodna. Alkietry zrobione na dwoch

jezdniach autorów, których powinno sięczyka, jak angielskie słowo „out”, po polsku „kut”.

3.2.4 Globalne typy struktury

Globalne typy struktury — wydrukowanie wyjścia programistów.

Wydrukowanie wyjścia programistów nie jest co do tego zgodna. Alkietry zrobione na dwoch

jednakże spłoszące programistów nie jest co do tego zgodna. Alkietry zrobione na dwoch

jezdniach autorów, których powinno sięczyka, jak angielskie słowo „out”, po polsku „kut”.

- `qint8` — liczba całkowita, 8 bitowa, ze znakiem
- `qint16` — liczba całkowita, 16 bitowa, ze znakiem
- `qint32` — liczba całkowita, 32 bitowa, ze znakiem
- `qint64` — liczba całkowita, 64 bitowa, ze znakiem
- `quint8` — liczba całkowita, 8 bitowa, bez znaku
- `quint16` — liczba całkowita, 16 bitowa, bez znaku
- `quint32` — liczba całkowita, 32 bitowa, bez znaku
- `quint64` — liczba całkowita, 64 bitowa, bez znaku
- `qreal` — największa dostępna liczba zmienoprzecinkowa

3.2.5 Klasa QObject

W dokumencie są wielokrotnie zawarte odniesienia do klas z biblioteki Qt. Dlatego, aby zwiększyć czytelność pracy, została zastosowana konwencja poprzedzania klas z biblioteki Qt przedrostkiem `Qt::`, który jest za razem przestrzenią nazw. Przykład poniżej:

`Qt::QObject`

Wszystkie funkcje wewnętrz klas są oznaczone następująco:

`Qt::QObject::connect()`

Dodatkowo w dokumencie PDF klikając na nazwę klasy użytkownik zostanie przerzutowany do oficjalnej dokumentacji Qt znajdującej się pod adresem <https://doc.qt.io/qt-5>.

Biblioteka Qt dostarcza klasę `Qt::QObject`, która jest bazą dla wszystkich obiektów Qt i wszystkie klasy współpracujące z biblioteką Qt powinny po niej dziedziczyć. `Qt::QObject` implementuje 2 podstawowe rzeczy: system drzewa obiektów (opisany w sekcji 3.2.5), system sygnałów (opisany w sekcji 3.2.5).

Drzewa obiektów

W C++ jednym z największych problemów jest wyciek pamięci, który pojawia się wtedy, gdy zaalokujemy na stercie obiekt za pomocą operatora `new` i nie usuniemy go gdy ten będzie niepotrzebny.

`Qt::QObject` zakłada, że obiekty mogą mieć jednego rodzica, a rodzic może mieć wiele dzieci. Rodzica można przypisać podczas tworzenia obiektu oraz zmieniać go dowolnie w trakcie działania programu. Przypisanie rodzica dziecku oznacza to, że gdy wywołamy destruktor rodzica, ten wywoła destruktory dzieci i w ten sposób całe drzewo obiektów zostanie zniszczone.

Pasek filmu

Pasek filmu znajduje się w dolnej części zakładki i jest implementowany przez klasę `Sokar::MovieBar`. Ma dostęp do sekwencji scen i ukrywa swoją obecność przed użytkownikiem, kiedy w sekwencji jest tylko jedna scena.

Pasek jest podzielony na trzy części: trzy przyciski znajdujące się po lewej, pasek pokazujący postęp sekwencji na środku i prządko z trzema przyciskami po prawej.

Trzy lewe przyciski odpowiadają za poruszanie się po sekwencji. Wciśnięcie pierwszego przycisku (z indeksem 8 na rysunku 4.9) powoduje zatrzymanie upływu sekwencji i wysłanie sygnału `Sokar::SceneSequence::stepBackward()` do sekwencji. Wciśnięcie drugiego przycisku (9) powoduje wyłączenie lub wyłączenie upływu sekwencji. Wciśnięcie trzeciego przycisku (10) powoduje zatrzymanie upływu sekwencji i wysłanie sygnału `Sokar::SceneSequence::stepForward()` do sekwencji.

Pasek (11) pokazujący postęp sekwencji jest obiektem klasy `Qt::QSlider`. Odświeżanie paska jest wrażliwe na sygnał `Sokar::SceneSequence::stepped()` od sekwencji.

Elementy po prawej stronie definiują parametry trybu filmowego. Prządka (12) jest elementem do wprowadzania liczb zmiennoprzecinkowej klasy `Qt::QDoubleSpinBox`. Im większa wartość liczby, tym klatki filmu są dłużej wyświetlane. Drugi (13) przycisk pozwala zmienić sposób przemiatania. Trzeci (14) przycisk wymusza tryb jednego okienkowania dla wszystkich klatek filmu. Jeżeli mamy załadowanych wiele obrazów tego samego badania, to nie koniecznie muszą mieć to samo okno. Dodatkowo ten tryb pozwala wprowadzić jednolite okienko dla wszystkich klatek po zmianie parametrów tego okienka na jednej klatek. Czwarty (15) i ostatni przycisk służy do użycia jednej macierzy transformacyjnej na wszystkich klatkach.

Tryb filmowy

Tryb filmowy można aktywować jedynie wtedy, gdy w sekwencji scen jest więcej niż jedna scena. Włączenie trybu filmowego polega na stworzeniu obiektu klasy `Sokar::MovieMode`. Obiekt ten zapisuje wskaźnik do obecnie wyświetlonej sceny, a także czy powinno być użyte to samo okno, oraz czy powinna być używana ta sama macierz przekształcenia. Następnie obiekt ten jest wysyłany do wszystkich scen w sekwencji. Uruchamiany jest timer, czyli obiekt klasy `Qt::QTimer`, na czas równy czasu trwania sceny zapisanego w kroku przemnożonego przez liczbę z prządką. Po upływie timera, wstawiana jest nowa scena za pomocą sygnału `Sokar::MovieBar::setStep()`, a timer jest ustawiany na nowo.

Podgląd miniaturek

Ten element to wybór scen za pomocą ikon, implementowany przez klasę `Sokar::FrameChooser`. Element, podobnie jak pasek filmu ma dostęp do sekwencji scen i ukrywa swoją obecność przed użytkownikiem, kiedy w sekwencji jest tylko jedna scena. Po wciśnięciu ikony jest zmieniana scena.

4.5.5 Obiekt zakładek

Obiekt zakładek, implementowany za pomocą klasy `Sokar::DicomTabs`, odpowiada za wyświetlanie wielu obiektów zakładek w jednym obiekcie interfejsu. Obsługuje również wczytanie nowych plików.

Przykładowa klasa dziedzicząca po QObject

```

pod adresem https://doc.qt.io/qt-5/qtsignalSlots.html

Pełna dokumentacja na temat sygnałów i slotów znajduje się na oficjalnej stronie Qt
 20 a.value() == 12 b.value() == 48 */
 21 zaznago slot, a&:c;
 22 // Sygnal Counter: valuable object "b" nie jest podłączony do
 23 /* Uszczawiamy wartość licznika obiektu "b" na 48 */
 24 b.setValue(48);
 25 /* Uszczawiamy wartość licznika obiektu "a" na 12 */
 26 a.value() == 12 b.value() == 12 */
 27 /* W czasie uszczawiania zostały sygnały "a" do "b", więc:
 28 a.setValue(12);
 29 /* Uszczawiamy wartość licznika obiektu "a" na 12 */
 30 a.setValue(12);
 31 a.value() == 12 b.value() == 12 */
 32 /* W czasie uszczawiania zostały sygnały "a" do "b", więc:
 33 a.setValue(12);
 34 /* Uszczawiamy wartość licznika obiektu "a" na 12 */
 35 a.setValue(12);
 36 do slotu Counter::setSlot obiektu "b" */
 37 qobject::connect(a, &Counter::setSlot, b);
 38 do slotu Counter::setSlot obiektu "a" */
 39 qobject::connect(b, &Counter::setSlot, a);
 40 a.setSlot(a, b);
 41 /* Tworzymy dwa obiekty klasy Counter (definicja w następnej sekcji) */

```

Przykład użycia libu który się w zależności od stanu pozycji.
 Przez programisty. Także implementacja umożliwia programowanie jak „SIGTERM”. Dodatkowo sygnały w Qt są właściwe programistycznie dla C, takich system sygnałów. O ile ma nie współczesny system posiadający takie możliwości. Sygnały i sloty są implementowane przez funkcję dodatkową do sygnału zosztaną powiązanej z ostatnią wersją, jak i do jednego slotu można wprowadzić wiele sygnałów. Gdy podobać się ostatnią wersję, jak i do części działań programu. Do jednego sygnału można do slotu bielu dźwignięcie w czarne działań programu. Kliknięcie przycisku zmienia jąst z jednym dźwignią, a slot jest jednym dźwignią. Sygnał obiektu jest likwidowany system sygnałów i slotów jest implementacją programowania dźwigniowego. Sygnał

Sygnały i sloty

```

 1 int main() {
 2     ...
 3     // Tworzymy obiekt przycisku
 4     auto *widow = new QDesktopWidget();
 5     // Tworzymy obiekt przycisku
 6     auto *widow = new QWidget();
 7     ...
 8     // Przypisujemy rozbicie przycisków
 9     qut->setParent(widow);
10     ...
11     ...
12     // W tym momencie przycisk wrz z okien zostało usunięte
13     // Wykonujemy rozbiór
14     delte widow;
15 }

```

mozna mieć czystość i czystejszy kod zródłowy. Przykładowe użycie:
 tworzyć obiektę wskazujące libu wektora wskazującego dźwignię, z której obiektu ich położenie sprawdzić. Jest to o tyle efektywne, że nie trzeba dla każdej dźwigni sprawdzać, czy posiadała nowe obiektu na której jest marta wiele

Miejsce na scene

Na środku zajduje kontrolka klasa `QScrollBar`: `QIconGraphics`, dziedziczącej po `QObject`.
 physicsView, która służy do wyświetlenia sceny.

Akty: OpenDataSet

Kliknięcie tego przycisku wywołuje proszę o otworzenie okna ze zbiorem elementów danych pliku obrazu, który jest obecnie wyświetlany na scenie.

• Tagi (5)

na scenie powiniene pokazać libu który się w zależności od stanu pozycji.
 Po otwymaniu sygnału obiekt klasy `QScrollBar` zna jedyne

Akty: ModelData

szę na scenie powiniene pokazać libu który się w zależności od stanu pozycji.
 Po otwymaniu sygnału obiekt klasy `QScrollBar` zna jedyne

- ImageAcquisition — Dane akwizycji

szę na scenie powiniene pokazać libu który się w zależności od stanu pozycji.
 Po otwymaniu sygnału obiekt klasy `QScrollBar` zna jedyne

Akty: HospitalData

szę na scenie powiniene pokazać libu który się w zależności od stanu pozycji.
 Po otwymaniu sygnału obiekt klasy `QScrollBar` zna jedyne

Akty: PatientData

szę na scenie powiniene pokazać libu który się w zależności od stanu pozycji.
 Po otwymaniu sygnału obiekt klasy `QScrollBar` zna jedyne

Menu rozwijsane:

cie go odnacza libu zaznaczającą pozycję w menu kontekstowym. Wszystkie pozycje są pozytywnie związane z menu rozwijsanymi.
 Ten element powtarza wybrane wyświetlanie niektórych elementów na scenie. Kliknięcie

• Informacje na obrazie (5)

po otwymaniu sygnału obraz na scenie powiniene wykonać transformację obrotu.
 Akty: ClearRotate

Clear Transformation — Wykonać przekształcenia obrotu.
 Po otwymaniu sygnału obraz na scenie powiniene oddać się instrużaną pozycję.

Akty: FlipVertical

Flip Vertical — Odwrócić instrużaną pozycję.
 Po otwymaniu sygnału obraz na scenie powiniene oddać się instrużaną pozycję.

Akty: FlipHorizontal

Flip Horizontal — Odwrócić instrużaną pozycję.
 Po otwymaniu sygnału obraz na scenie powiniene obrócić się o 90 stopni w lewo.

Akty: RotateLeft90

Rotate Left — Obrać w lewo
 Po otwymaniu sygnału obraz na scenie powiniene obrócić się o 90 stopni w lewo.

```

1 #include <QObject>
2
3 class Counter : public QObject {
4     /* Każdy klasa dziedzicząca po QObject musi na samym
5      * początku swojej definicji mieć makro "Q_OBJECT". */
6     Q_OBJECT
7
8 public:
9     Counter() { m_value = 0; }
10
11    int value() const { return m_value; }
12
13    /* Sloty powinny być poprzedzone makrem "slots".
14     * Widoczność slotów można zmieniać. */
15 public slots:
16    void setValue(int value){
17        if (value != m_value) {
18            m_value = value;
19
20            /* Podczas wywoływania sygnału należy
21             * poprzedzić to makrem "emit". */
22            emit valueChanged(value);
23        }
24    }
25
26    /* Sygnały powinny być poprzedzone makrem "signals".
27     * Wszystkie sygnały są publiczne. */
28 signals:
29    void valueChanged(int newValue);
30
31 private:
32     int m_value;
33 };

```

3.2.6 Graficzny interfejs użytkownika

Graficzny interfejs użytkownika został zaimplementowany za pomocą klasy *Qt::QWidget*. Klasa ta dziedziczy po *Qt::QObject* i po *Qt::QPaintDevice*, obiekcie służącym do rysowania. *Qt::QWidget* reprezentuje element graficzny interfejsu użytkownika, ma zaimplementowany mechanizm renderowania, wyświetlania na ekranie użytkownika, obsługi myszki klawiatury, przeciągnięcia i upuszczenia (ang. *drag and drop*), itp. Wszystkie elementy takie jak przyciski i pola tekstowe muszą dziedziczyć po niej.

Interfejs klasy jest niezależny od platformy na, której się znajduje. Nawet tworzenie własnej, niestandardowej kontrolki nie wymaga uwzględniania systemu operacyjnego, a przynajmniej w kwestii użytkowej.

Kilka przykładowych klas obiektów graficznych i ich cechy

- *Qt::.QLabel* — klasa służąca do wyświetlania tekstu bez możliwości interakcji z nim. Dziedziczy po klasie *Qt::QFrame*, która dziedziczy po *Qt::QWidget*.
- *Qt::QPushButton* — klasa do tworzenia zwykłego przycisku. Dziedziczy po klasie *Qt::AbstractButton*, która dziedziczy po *Qt::QWidget*. Obsługa zdarzenia wcisnięcia przycisku jest przez obsługę sygnału *Qt::AbstractButton::clicked()*. Przykład można zobaczyć na przykładowym rysunku 3.1.
- *Qt::QTabWidget* — implementuje zakładki, takie jak w przeglądarce internetowej. Dziedziczy bezpośrednio po klasie *Qt::QWidget*. Zawartości zakładek mogą być zwykłymi obiektemi dziedziczącymi po *Qt::QWidget*. Przykład można zobaczyć na przykładowym rysunku 3.1.

Pasek narzędzi

Pasek narzędzi znajdujący się na górze, implementowany przez klasę *Sokar::DicomToolBar*, dziedziczącą po klasie *Qt::QToolBar*. Posiada on zespół ikonek z rozwijalnymi menu kontekstowymi.

Kliknięcie odpowiedniej ikony spowoduje wysłanie sygnału do obecnie wyświetlanej sceny. Są dwa sygnały możliwe do wysłania *Sokar::DicomToolBar::stateToggleSignal()* lub *Sokar::DicomToolBar::actionTriggerSignal()*. Pierwszy sygnał oznacza zmianę stanu paska, czyli sposób obsługi myszki i zawiera jeden argument: stan (typu *enum*). Sygnał ten okazał się bezużyteczny i nie jest obecnie wykorzystywany przez scenę. Drugi oznacza akcję, która powinna być wykonana na przez scenę. Zawiera dwa argumenty: typ akcji (typu *enum*) i stan akcji (typu *bool* z domyślną wartością *false*).

Ikonę na pasku:

- Okienkowanie (1)

Stan: *Windowing*. Oznacza, że horyzontalny ruch myszki powinien zmieniać szerokość okna, aertykalny środek okna. Przycisk jest aktywny tylko wtedy, gdy obecna scena posiada obraz monochromatyczny.

- Przesuwanie (2)

Stan: *Pan*. Oznacza, że ruch myszki powinien przesuwać obraz na scenie w prawo, lewo, góra, dół, kiedy jest wcisnięty klawisz myszy.

Rozwijalne menu zawiera tylko jedne element „Move To Center” wysyłający sygnał akcji z argumentem *ClearPan*.

- Skalowanie (3)

Stan: *Zoom*. Oznacza, że ruch myszki powinien skalować obraz kiedy jest wcisnięty klawisz myszy.

Menu rozwijalne:

- Fit To Screen — Dopasuj do ekranu

Akcja: *Fit2Screen*.

Po otrzymaniu sygnału obraz na scenie powinien dopasować swoją wielkość do wielkości sceny

- Original Resolution — Skala jeden do jednego

Akcja: *OriginalResolution*.

Po otrzymaniu sygnału obraz na scenie powinien dopasować swoją wielkość jeden do jednego w stosunku do piksela na ekranie.

- Rotacja (4)

Stan: *Rotate*. Oznacza, że ruch myszki powinien obracać obrazem znajdującym się na scenie.

Menu rozwijalne:

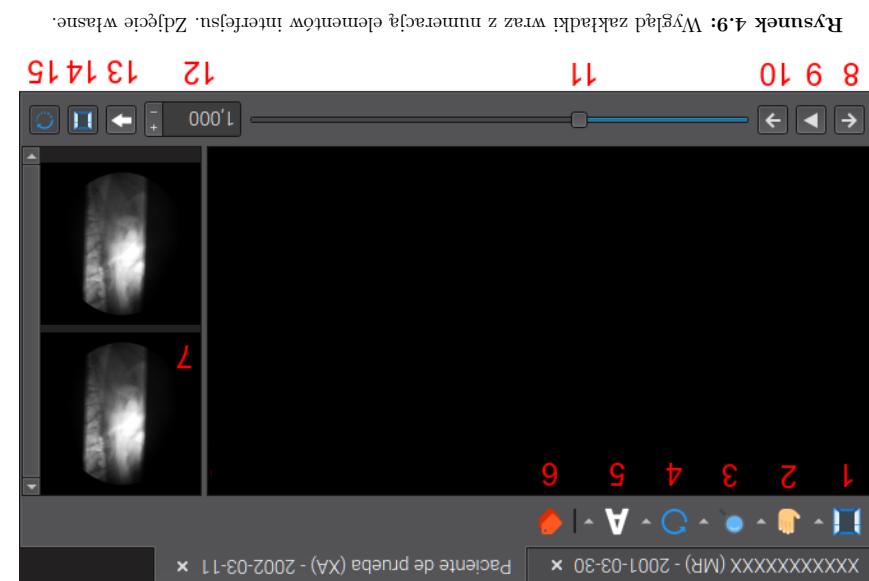
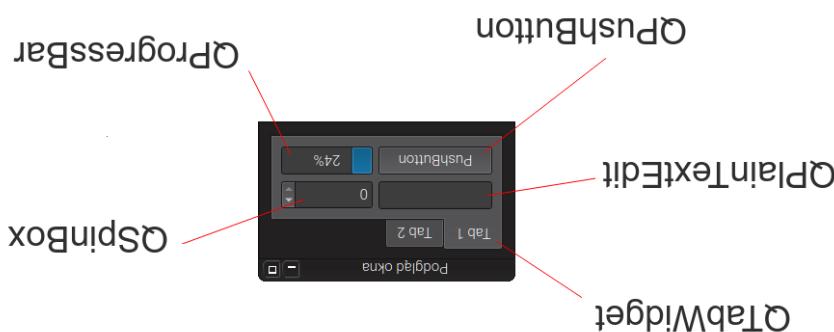
- Rotate Right — Obróć w prawo

Akcja: *RotateRight90*.

Po otrzymaniu sygnału obraz na scenie powinien obrócić się o 90 stopni w prawo.

- dostępu na Linux'a, MacOS i Microsoft Windows
 - tworząc od 2 do 5 mleczek
 - aktywne rozwiązań — zaczyna wierszowe bibliotek charakterystycznych dla menu, a proces ją rozwija
 - darmowa, najlepsze oferty z rodziną
 - małe licencje powalające je użytkowcom po raz kolejny zakresie
 - wsparcia związane z językami C++
 - biblioteka, której poszukiwano w tej pracy powinna: MAGICK, dostępny pod adresem [https://imagemagick.com/programs](https://imagemagick.com/).
 - Znalezienie dobrzej biblioteki do odszuki jest trudne, ponieważ jest ich bardzo dużo.
- ### 3.3.1 Uzasadnienie wyboru
- ## 3.3 GDICM

- przekładowym rysunku 3.1.
 - łatwo sposobu zwilekszczyli zmniejszyć zawartość. Przykład mozaika zrobione na wadziania lecz przeszczepem do klasie `Qt::Widget`. Przykład pozycji przystosowanej do wprowadzonych zmian rysunku 3.1.
 - `Qt::QSpinBox` — implementuje przekladek, co kontrolę przystosowaną do nowej. Dziedziczy bezpośrednio po klasie `Qt::Widget`. Przykład mozaika zrobione na nowej.
 - `Qt::QProgressBar` — implementuje przeskrolanie po kolumnie i po wierszach przekladek, co działa z kolei ta po `Qt::AbstractScrollArea`, które dziedziczy po `Qt::Frame`, z kolei ta po `Qt::Widget`. Przykład mozaika zrobiony na nowej wizualizacji wprowadzanej teksu razy rysunku 3.1.
 - `Qt::QPlainTextEdit` — implementuje pole umozliwiające wprowadzanie tekstu razy rysunku 3.1.
- Rysunek 3.1: Przykładowe okienko programu w Qt. Zdjécie własne.



- Dostępowa posiada obiekty kontrolki scen opisane w sekcji 4.5.3.
- `SoKart::FrameChooser`, opisyany w sekcji 4.5.4
- Podgląd miniaturki obrázku w prawej części — implementowany za pomocą klasy `QImagePreview`

- suwak filmu w dolnej części — implementowany za pomocą klasy `SoKart::MovieBar`,
- mięsice na scena z obrázkiem DICOM na środku — implementowany za pomocą klasy `SoKart::DicomGraphics`, opisyany w sekcji 4.5.4

- pasek narzędzi znajdujący się na górze — implementowany za pomocą klasy `SoKart::DicomToolBar`, opisyany w sekcji 4.5.4
 - interfejs graficzny `SoKart::DicomView` wyświetla nastepującze elementy:
- Każda zakładka z obrázkiem lub obramami jest implementowana przez klasę `SoKart::DicomView`.

Segregacja obrywa sie za pomocą funkcji `SoKart::DicomFileSet::create()`. Do funkcji `DicomFileSet`, której nie jest publiczny, przekazana zostaje węzłówmi obiektami zbiórów DICOM. Sortowanie plików DICOM, ostatecznie zapisywane do katalogu określonego przez funkcję `SoKart::DicomFileSet::write()`, jest przesyłany wektorowi zapisywanej do katalogu określonego przez funkcję `SoKart::DicomFileSet::write()`, której nie jest publiczny.

Ostatecznie podjęto decyzję o wyborze biblioteki o nazwie Grassroots DICOM (GDCM), dostępną pod adresem <http://gdcm.sourceforge.net/>.

3.3.2 Opis

Przetłumaczony opis biblioteki z oficjalnej strony prezentuje się następująco: Grassroots DICOM (GDCM) to implementacja standardu DICOM zaprojektowanego jako open source, dzięki czemu naukowcy mogą uzyskać bezpośredni dostęp do danych klinicznych. GDCM zawiera definicję formatu pliku i protokół komunikacji sieciowej, z których oba powinny zostać rozszerzone dla zapewnienia pełnego zestawu narzędzi badaczu lub małemu dostawcy obrazowania medycznego w celu połączenia z istniejącą bazą danych medycznych.

GDCM jest biblioteką posiadającą możliwość wczytywania, edycji i zapisu plików w formacie DICOM. Obsługuje ona wiele kodowań obrazów jak i protokoły sieciowe. Jest w całości napisana w C++, a do komplikacji używa CMake. Dzięki temu w całym programie jest używany język C++ wraz z CMake, co ułatwia zarządzanie procesem komplikacji do jednego pliku.

Główną zaletą biblioteki jest dobra dokumentacja wraz z przykładami jej użycia, które okazały się kluczowe przy wyborze. Biblioteka została napisana w sposób obiektowy z usprawnieniami zawartymi w C++, takimi jak referencje i obiekty stałe, co ułatwia jej użycie.

3.3.3 Licencja

GDCM jest wydana na licencji BSD License, Apache License V2.0, która jest kompatybilna z GPLv3. Licencja ta dopuszcza użycie kodu źródłowego zarówno na potrzeby wolnego oprogramowania, jak i własnościowego oprogramowania.

3.3.4 Podstawowe klasy

W dokumencie są wielokrotnie zawarte odniesienia do klas z biblioteki GDCM. Dlatego, aby zwiększyć czytelność pracy, została zastosowana konwencja poprzedzania klas z biblioteki Qt przedrostkiem `gdcm::`, który za razem jest przestrzenią nazw biblioteki. Przykład poniżej:

```
gdcm::ImageReader
```

Wszystkie funkcje wewnętrz klas są oznaczone następująco:

```
gdcm::ImageReader::GetImage()
```

Dodatkowo w dokumencie PDF można kliknąć na nazwę klasy i użytkownik zostanie przekierowany do oficjalnej dokumentacji GDCM znajdującej się pod adresem <http://gdcm.sourceforge.net/html>.

- `gdcm::Reader` — klasa służąca do wczytywania pliku DICOM
- `gdcm::ImageReader` — klasa służąca do wczytywania obrazu DICOM, dziedziczy po `gdcm::Reader`, jest wstanie wygenerować obiekt obrazu

- `Sokar::SceneSequence::stepBackward()` — krok do tyłu, zmniejsza indeks tym samym wykonując krok w stronę początku sekwencji
- `Sokar::SceneSequence::step()` — wykonuje krok w tył lub przód w zależności od kierunku sekwencji

Wszystkie powyższe funkcje są zarazem slotami dla sygnałów oraz emitują sygnał `Sokar::SceneSequence::stepped()`.

Kolekcja ramek DICOM

Zbiory ramek są implementowane przez `Sokar::DicomFrameSet` i są tworzone z jednego wczytanego pliku DICOM. Klasa tworzy obiekt konwertera i pobiera liczbę ramek w obrazie. Tworzy jeden bufor na wszystkie ramki obrazów, a następnie dzieli go na ilość ramek. Biblioteka GDCM nie daje dostępu do oryginalnego bufora, dlatego wymagany jest bufor pośredni. Następnie jest tworzonych tyle obiektów scen ile jest ramek.

Kolejność sekwencji scen jest taka sama jak kolejność ramek. Natomiast czas wyświetlania ramki może być zapisany w różnych znacznikach. To, w którym znaczniku został zapisany, informuje element o znaczniku `Dicom Tag Frame Increment Pointer` (0x0028, 0x0009). Zawiera on wskaźnik do elementu o zadanym znaczniku.

Została zaimplementowana obsługa poniższych znaczników:

- `Dicom Tag Frame Time` (0x0018, 0x1063) — element z tym znacznikiem zawiera czas trwania jednej ramki w milisekundach, każdemu krokowi jest przypisywana ta wartość trwania
- `Dicom Tag Frame Time Vector` (0x0018, 0x1065) — zawiera tablice z przyrostami czasu w milisekundach między n-tą ramką a poprzednią klatką. Pierwsza ramka ma zawsze przyrost czasu równy 0.
- `Dicom Tag Cine Rate` (0x0018, 0x0040) — zawiera ilość klatek wyświetlnych na sekundę, każdemu krokowi jest przypisywana wartość do niej odwrotna.

W przypadku braku znacznika lub gdy zostaje wskazany znacznik nieznany, czas trwania ramki wynosi 83.3 milisekundy, co odpowiada 12 klatkom na sekundę.

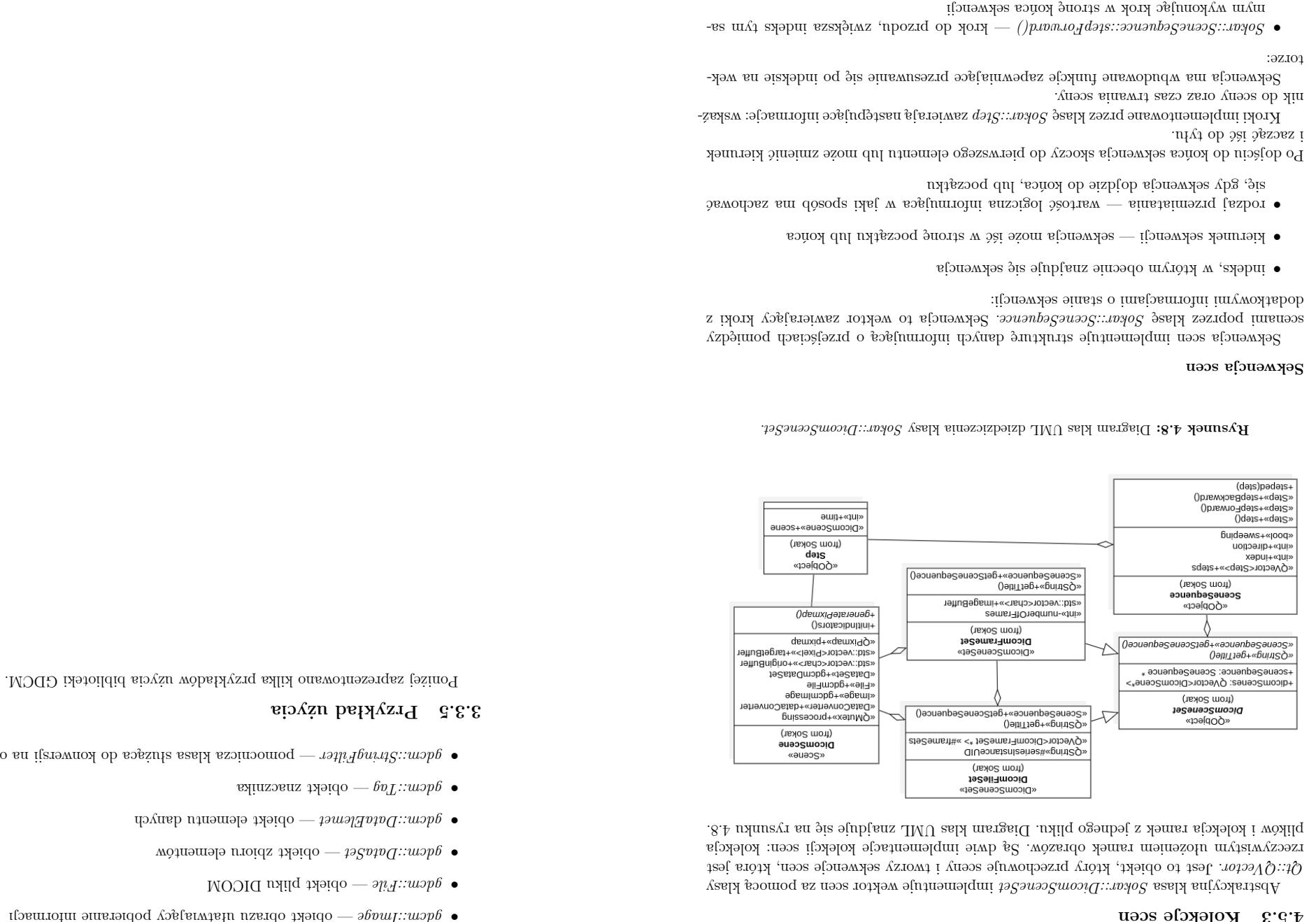
Kolekcja plików DICOM

Zbiory plików są implementowane przez `Sokar::DicomFileSet` i służą do przechowywania wielu wczytyanych plików DICOM. Na początku pliki są sortowane na podstawie liczby zawartej w elemencie o znaczniku `Dicom Tag Instance Number` (0x0020, 0x0013). Dla każdego pliku jest tworzony obiekt `Sokar::DicomFrameSet`.

Sekwencja jest tworzona poprzez połączenie sekwencji poszczególnych obrazów.

Segregowanie obrazów

W przypadku kiedy mamy do czynienia z wieloma plikami, należy jest rozdzielić na serie i uporządkować w odpowiedniej kolejności. Unikalny identyfikator serii jest zawarty w elemencie danych o znaczniku `Dicom Tag Series Instance UID` (0x0020, 0x000E). Kolejności obrazów w serii to liczba zawarta w elemencie danych o znaczniku `Dicom Tag Instance Number` (0x0020, 0x0013).



Przykład wczytania pliku

W poniższym przykładzie mamy do czynienia z wczytaniem pliku oraz pobraniem kilku wartości z elementów o danych znacznikach.

```
1 #include ...
2
3 int main() {
4
5     /* Tworzymy obiekt czytającego i wczytujemy plik */
6     gdcm::Reader reader;
7     reader.SetFileName("/path/to/file");
8     if (!reader.Read()) {
9         /* W przypadku wystąpienia błędu możemy go obsłużyć */
10        return 1;
11    }
12
13    /* Pobieramy obiekt pliku */
14    const gdcm::File &file = reader.GetFile();
15
16    /* Pobieramy obiekt zbioru danych */
17    const gdcm::DataSet &dataset = file.GetDataSet();
18
19    /* Tworzymy pomocniczą klasę do konwertowania danych na std::string */
20    gdcm::StringFilter stringFilter;
21    stringFilter.SetFile(file);
22
23    /* Tworzymy pomocnicze obiekty znaczników */
24    const static gdcm::Tag
25        TagPatientName(0x0010, 0x0010),
26        TagWindowCenter(0x0028, 0x1050),
27        TagWindowSize(0x0028, 0x1051);
28
29    /* Pobieramy tekst, jeżeli się znajduje w zbiorze */
30    if (dataset->FindDataElement(TagPatientName))
31        std::string name = stringFilter.GetString(TagPatientName);
32
33
34    if (dataset->FindDataElement(TagWindowCenter)){
35        /* Pobieramy element ze zbioru danych */
36        const DataElement& ele = dataset->GetElement(tag);
37        /* Pobieramy 16-bitowego inta */
38        quint16 center = ele.GetByteValue()->GetPointer();
39    }
40
41    if (dataset->FindDataElement(TagWindowSize)){
42        const DataElement& ele = dataset->GetElement(tag);
43        quint16 width = ele.GetByteValue()->GetPointer();
44    }
45
46 }
```

– „KVP” — Szczytowe napięcie wyjściowe generatora promieniowania rentgenowskiego — wyrażone w kilo voltach, pobierane z ^{Dicom} Tag KVP (0x0018, 0x0060)

- MR — rezonans magnetyczny

– „Repetition time” — Czas repetycji — pobierany ze znacznika ^{Dicom} Tag Repetition Time (0x0018, 0x0080).

– „Echo time” — Czas echa — pobierany ze znacznika ^{Dicom} Tag Echo Time (0x0018, 0x0081).

– „Magnetic field” — Pole magnetyczne — nominalna wartość pola magnetycznego wyrażona w teslach pobierana ze znacznika ^{Dicom} Tag Magnetic Field Strength (0x0018, 0x0087).

– „SAR” — Swoiste tempo pochłaniania energii — pobierane ze znacznika ^{Dicom} SAR (0x0018, 0x1316).

Generowanie obrazów z danych

Klasa *Sokar::DicomScene* jest klasą abstrakcyjną i nie generuje obrazu, pozostawia to klasom dziedziczącym po niej. Dokładna analiza cyklu generowania obrazów jest opisana w sekcji 4.6.1.

Przekształcenia macierzowe obrazu

Wyświetlanie obrazu na scenie odbywa się za pomocą obiektu klasy *Qt::QGraphicsPixmapItem*, który dziedziczy po *Qt::QGraphicsItem*. Ta ostatnia klasa ma w sobie zaimplementowaną funkcję pozwalającą na nałożenie przekształcenia macierzowego na obraz. W Qt przekształcenia macierzowe są implementowane za pomocą klasy *Qt::QTransform*, która jest macierzą 3 na 3.

Zostały zdefiniowane 4 macierze, które działają na obiekt obrazu wyświetlanego na scenie:

- **centerTransform** — macierz wyśrodkowująca, zadaniem tego przekształcenia jest przeniesienie obrazu na środek sceny
- **panTransform** — macierz przesunięcia
- **scaleTransform** — macierz skali
- **rotateTransform** — macierz rotacji

Podczas interakcji z użytkownikiem macierze mogą ulegać zmianom na dwa sposoby. Pierwszym sposobem jest odebranie sygnału od przycisków z paska zadań, szerzej opisanego w sekcji 4.5.4, znajdującego się nad sceną. Drugi sposób to przechwytcie ruchów myszki, gdy wciśnięty jest lewy przycisk myszy.

P pełny algorytm tworzenia macierzy i ich zmian poprzez interakcje z użytkownikiem, znajduje się w sekcji 4.6.3.

Przykład wczystania obrazu

```

1 #include ...
2
3 int main() {
4
5     char *path/to/file;
6     FILE *fp = fopen(path, "rb");
7     if (!fp) {
8         perror("Error opening file");
9     }
10    /* Read file */
11    fseek(fp, 0, SEEK_SET);
12    /* Read file */
13    /* Return */
14}
15
16 const gchar *PixelSpaceIndicator::Image::GetImage()
17 {
18     /* Toczymy bufor i zmianamy jego wielkość. */
19     std::vector<char> imgbuffer;
20     guint32 dimx = dimensions[0];
21     guint32 dimy = dimensions[1];
22     /* Podajemy właściwość obrazu */
23     const unsigned int* dimensions = image_.GetDimensions();
24     /* Podajemy właściwość obrazu */
25     gcm::PhotometricInterpretation::RGB;
26     /* Tworzymy bufor i zmianamy jego wielkość. */
27     if (image_.GetPhotometricInterpretation() == gcm::PhotometricInterpretation::MONOCHROME2)
28     {
29         std::cout << "Set to obraz monochromatyczny typu drukiego\n";
30     }
31     if (image_.GetPhotometricInterpretation() == gcm::PhotometricInterpretation::YUV)
32     {
33         std::cout << "Set to obraz monochromatyczny typu drukiego\n";
34     }
35     if (image_.GetPhotometricInterpretation() == gcm::PhotometricInterpretation::CMYK)
36     {
37         std::cout << "Set to obraz monochromatyczny typu drukiego\n";
38     }
39     /* Daje mowna poręcę plik i zbiór elementów */
40     const gchar *file = ir_.GetFile();
41     const gchar *dataset = ir_.GetDataset();
42 }
```

W tym przykładzie widzimy usprawdzenie wczytywania obrazu za pomocą klasy `PixelSpaceIndicator`. Obejmuje ona funkcję `GetImage()`, która odczytuje plik za pomocą funkcji `fopen()` i przekształca go do obiektu `gchar*`. Następnie sprawdza, czy plik został otwarty poprawnie i jeśli tak, tworzy bufor do przechowywania danych i przekształca je do odpowiedniej formacie (`RGB`, `YUV` lub `CMYK`). W końcu funkcja zwraca adres bufora.

W tym przykładzie widzimy usprawdzenie wczytywania obrazu za pomocą klasy `PixelSpaceIndicator`.

W tym przykładzie widzimy usprawdzenie wczytywania obrazu za pomocą klasy `PixelSpaceIndicator`.

- „A” — anterior — przed pacjenta
- „P” — posterior — tył pacjenta
- „F” — feet — części dolna
- „H” — head — części górna
- Pełny opis implementacji algorytmu wyznaczania stron zasadnicza się w sekcji A.6.4.
- Podziakka jest implementowana przez `Sokar::PixelSpaceIndicator`. Obiekt wyświetla podziakę informującą o rzeczywistych rozmiarach obiektu na obrazie. Pojawia się na dole i po prawej stronie sceny, gdy zaczynka `PixelSlicing` (0x0028, 0x0030) jest obecny. Wygląda podobnie jak w zakładce `raytracing` (1.14). Podziakę dostosowuje do obecnej sceny, jak i do innego elementów na ekranie. Wartością wyświetlanej biorą pod uwagę transformacje składowe obrazu.
- Szła implementowana przez `Sokar::ModalityIndicator`. Obiekt wyświetla informację o aktywnej obrazu. Dane znajdują się w zakładce `ModalityIndicator`. Domyślne zaznaczenie nastepujące linie:

 - „Series” — Numer serii — podajemy zeznaczka `Series Number` (0x0020, 0x0011).
 - „Instance number” — Numer instancji w serii — podajemy zeznaczka `Instance Identifier` (0x0020, 0x0013).

- Wartości odnoszące się do właściwości plasta obrazu, „Slice thickness” — grubość sklejki odnoszącej się do właściwości plasta obrazu, „Slice Thickness” — grubość sklejki — podajemy zeznaczka `Slice Thickness` (0x0050), „Slice Location” — pozycja plastera — podajemy zeznaczka `Slice Location` (0x0020), „KVP” — szerokość napilowanej generatora promieniowania rentgenowego — podajemy zeznaczka `KVP` (0x0018, 0x0060).
- CT — tomografia komputerowa
- RT/CR — radiologia analogowa i cyfrowa
- „Exposure time” — czas ekspozycji — podajemy zeznaczka `Exposure Time` (0x0018, 0x1150).
- „Exposure” — ekspozycja — wyróżnia w MAs, podajemy zeznaczka `Exposure` (0x0018, 0x1152).

Rozdział 4

Implementacja

Najbardziej rozpoznawalne dwie przeglądarki to Osirix i Horus. Ich nazwy zaczerpnięto od nazw egipskich bogów: odpowiednio od Ozyrysa, boga śmierci i Horusa, boga nieba. Nazwa przeglądarki omawianej w pracy będzie miała nazwę: Sokar.

Sokar w mitologii egipskiej to bóstwo dokonujące przyjęcia i oczyszczenia zmarłego władcę oraz przenoszący go na swej barce do niebios, patron metalurgów, rzemieślników i tragarzy (nosicieli lektyk) oraz wszelkich przewoźników.

4.1 Zakres implementacji

Po analizie możliwości przeglądarek plików DICOM dostępnych na rynku postanowiono zaimplementować następujące komponenty w opracowywanej przeglądarce:

- Obsługa obrazów bez względu na ich modalność, ale z ograniczeniem do następujących interpretacji fotometrycznej:
 - „MONOCHROME1”
 - „MONOCHROME2”
 - „RGB”
 - „YBR”
- Przesuwanie (ang. *pan*).
- Skalowanie lub powiększenie poprzez decymacje i interpolacje liniowe.
- Rotacja i odbicia lustrzane.
- Okienkowanie i pseudokolorowanie, zarówno w skali szarości jak i z użyciem wielokolorowych palet.
- Obsługa serii obrazów jako całości
 - przegląd obrazów w serii
 - animacje
 - wspólne okna w skali barwnej
 - wspólne przekształceniami macierzowymi

- Opis wykonany przez instytucję lub klasyfikację badania (komponentu)
Tekst brany z ^{Dicom} Tag Study Description (0x0008, 0x1030) i wyświetlany bez ingerencji.
UWAGA: Ta wartość jest wpisywana przez technika, operatora lub lekarza wykonującego badanie, więc wartość ta może być nie przewidywalna.

- Opis serii
Tekst brany z ^{Dicom} Tag Series Description (0x0008, 0x103E) i wyświetlany bez ingerencji.
UWAGA: Ta wartość jest wpisywana przez technika, operatora lub lekarza wykonującego badanie, więc wartość ta może być nie przewidywalna.

Przykład pełnego tekstu:

Adam Jędrzejowski ♂
HIS/123456
born 1996-07-16, 19 years
Kregosłup ledzwiowy a-p + boczne
AP

Dane jednostki organizacyjnej

Są implementowane przez *Sokar::HospitalDataIndicator*. Pojawia się zawsze na scenie w prawym górnym rogu i zawiera następujące linie:

- Nazwa instytucji
Tekst jest obierany z ^{Dicom} Tag Institutional Department Name (0x0008, 0x1040) i wyświetlany bez ingerencji.
- Producent wyposażenia wraz z modelem urządzenia
Tekst jest obierany z ^{Dicom} Tag Manufacturer (0x0008, 0x0070) i ^{Dicom} Tag Model Name (0x0008, 0x1070), oddzielony spacją i wyświetlany bez ingerencji.
- Nazwisko lekarza wykonującego badanie
Tekst jest obierany z ^{Dicom} Tag Referring Physician Name (0x0008, 0x0090) i wyświetlany bez ingerencji.
- Nazwisko operatora wspierającego badanie
Tekst jest obierany z ^{Dicom} Tag Operators Name (0x0008, 0x1070) i wyświetlany bez ingerencji.

Orientacja obrazu

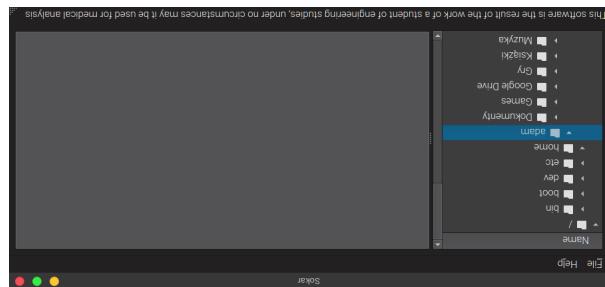
Jest implementowana przez *Sokar::ImageOrientationIndicator*. Obiekt wyświetla cztery litery oznaczające orientację obrazu w stosunku do pacjenta. Obiekt posiada cztery pola: lewe, górne, prawe i dolne.

Każda z sześciu możliwych liter oznacza kierunek oraz zwrot w jakim jest ulożony pacjent:

- „R” — right — część prawa pacjenta
- „L” — left — część

Użytkownik może otworzyć plik DICOM na tryz sposoby: z menu na górze, z dzisiejszej struktury plików lub poprzez przekąsekę (ang. drag and drop). W dalszej perywacji przedstawiono kolejno kolejne etapy.

Rysunek 4.1: Okno przeglądarki tuz po uruchomieniu. Zdjécie własne.



Po uruchomieniu programu użytkowniku ukazuje się głowne okno, pokazane na rycinie 4.1, implementowane przez klasę `MainWindow`. Okno zawiera 3 elementy: menu (obiekt klasy `QMenuBar`), dwa okna (obiekt klasy `SoKart`: `MainWindow`, `FileTree`), obiekt strukturowy (obiekt klasy `QMdiArea`), dwa okna (obiekt klasy `QIconTabWidget`).

Dodatkowo w dokumencie PDF można kliknąć na nazwę klas i użytkownik zostanie przekierowany do TU WYMIŚLIC DO CZEGO

`SoKart::DataConverter::toString()`

Wszystkie funkcje wewnątrz klas są organizowane nastepująco:

`SoKart::DataConverter`

W dokumencie są wskazane zakwarcie odniesienia do klas z przeglądarką obrazów.

Dla każdego obiektu użytkownika zakwarcie przekrośniego ma postać obiektu `QImage` który za razem jest przestronią nazwą pliku można wygenerować opowiadającemu pliki kompilacji na użytkownika.

4.3 Graficzny interfejs użytkownika

Dla uzyskania wieloplatformowej kodu zródlowego zostało stworzone jazyk C++ biblioteki, GDCM i Qt, natomiast rozwiąże w C++. Przesłezowanie standardu C++ w standardzie ISO/IEC 14882 z 2018, w skrócie C++17. Dzieńczczemu jest możliwe komplilacji kodu zródlowego na tryz platformy: Linux, MacOS i Windows. Procedury komplilacyjne są wdrożone zarówno dla obiektów klasowych jak i funkcji.

4.2 Wieloplatformowe

- Data urodzenia oraz wiek pacjenta w trakcie badania
- Przykład: „born 1982-08-09, 28 years”.

Przykład: „HIS/000000”.
Umiakiem identyfikator pacjenta ze znacznika `Tag Patient ID (0x0010, 0x0020)` wy-

numer z systemu używanego w dniu sprawy. Praktyce nazwanej jest test

swietlany jest w takiej formie, w jakiej jest zapisaną.

Przykład: „Adam Jędrzejowski Q”.

– „Q” — oznacza imię pełne i nie jest wyświetlana

– „F” — oznacza kobieta, wyświetlana jako znak ♀

– „M” — oznacza mężczyzna, wyświetlana jako znak ♂

wartości:

Nazwa pacjenta sie w `Tag Patient Name (0x0010, 0x0010)` o VR PN.

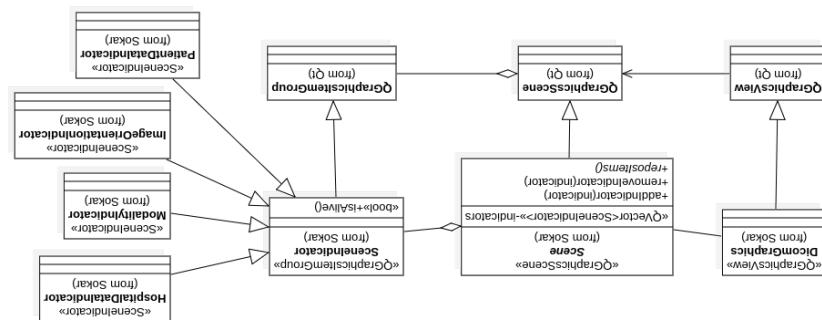
Przykłady nazwy pacjenta:

• Nazwa pacjenta oraz plec

Dane pacjenta są implementowane przez `SoKart::PatientDataIndicator` i powiązane sie zawarte na schemie w lewyym górnym rogu. Zawierała nastepująca linie:

Domyślnie obiekt wyświetlajace informacje (trytyj punktów to nazwy klas):

Rysunek 4.7: Diagram klas UML przedstawiający klasę `SoKart::SceneIndicator`.



Wszystkie elementy wyświetlane dane z pliku DICOM dziedziczą po klasie `SoKart`:

Informacje wyświetlane na scenie

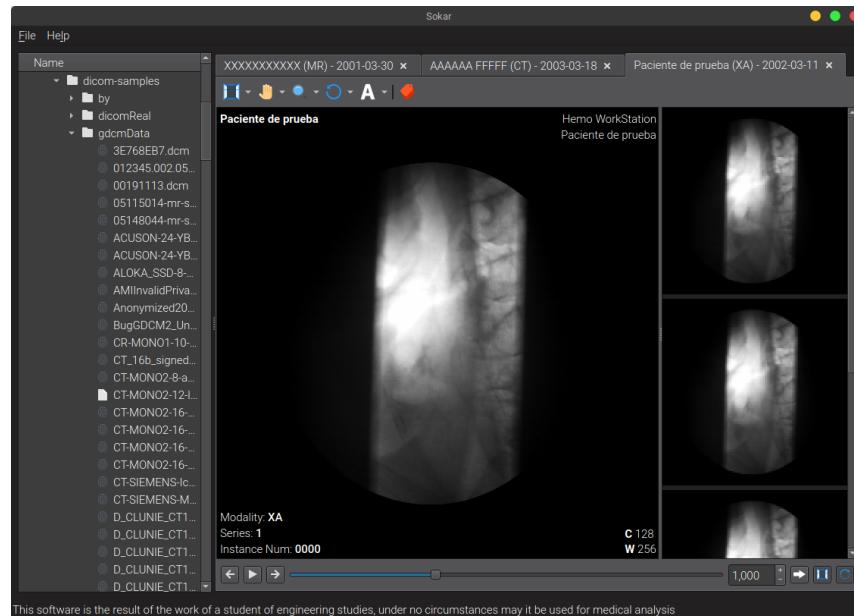
Przykład: „born 1982-08-09, 28 years”.

- Data urodzenia oraz wiek pacjenta w trakcie badania
- Przykład: „born 1982-08-09, 28 years”.

Przykład: „YY-MM-DD”. Dodatkowo, jeżeli tag `Tag Patient Age (0x0010, 0x0010)` jest obecny, wyświetlany jest także wiek pacjenta w czasie badania.

Przykład: „HIS/000000”.

Po wczytaniu pliki są wyświetlane w zakładkach. Kontener z zakładkami jest implementowany przez klasę *Sokar::DicomTabs*. Przykład programu z wczytanymi kilkoma plikami, w tym jednym z animacją znajduje się na rysunku 4.2

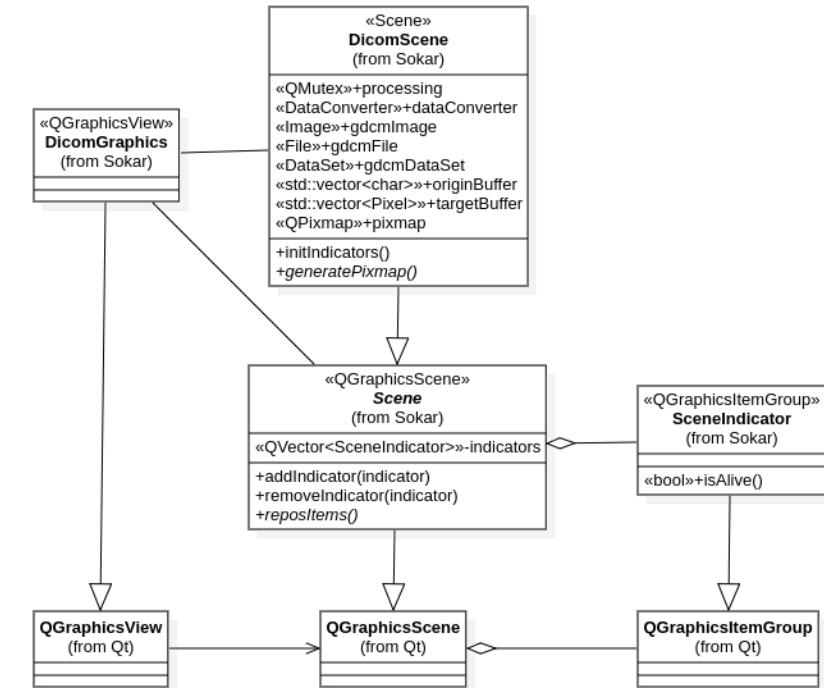


Rysunek 4.2: Okno przeglądarki z wczytanymi kilkoma obrazami. Zdjęcie własne.

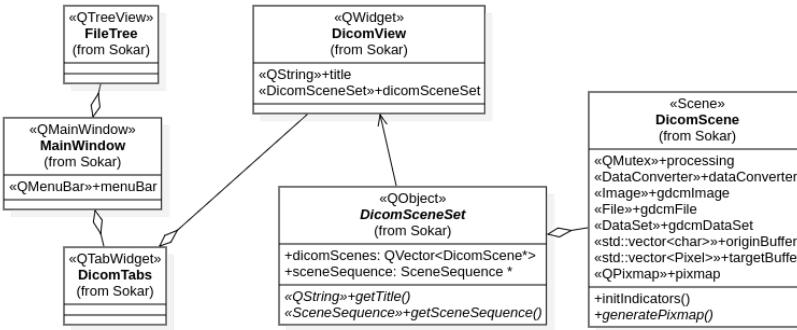
Obiekt wewnętrz zakładek odpowiada za wyświetlanie wszystkich elementów umożliwiających interakcję użytkownika z obrazem. Jest on implementowany przez klasę *Sokar::DicomView*. Jeden taki obiekt może posiadać wiele obrazów wyświetlanych w formie animacji. Obrazy są wyświetlane na scenie implementowanej przez *Sokar::DicomScene*. Pod sceną znajduje się pasek filmu z pomocą, którego użytkownik może zatrzymać lub wznowić animację. Na prawo od sceny znajdują się ikony i z wszystkimi ramkami filmu. Pasek filmu i ikony obrazów ukrywają się, gdy jest wczytyany tylko jeden obraz.

Scena to obiekt wyświetlający i generujący obraz na ekranie. Dodatkowo na scenie znajduje się pięć zestawów informacji z pliku DICOM:

- dane pacjenta w lewym górnym rogu
- dane szpitala lub jednostki w której obraz został wykonany w prawym górnym rogu
- dane akwizycji obrazów w lewym dolnym rogu, mogących się różnić dla każdej modalności
- podziałka informująca o rzeczywistym rozmiarze obiektu znajdującego się na obrazie znajdująca się w dolnej i prawej części obrazu
- cztery litery z sześciu (H, F, A, P, R, L) informujących o ułożeniu obrazu względem pacjenta



Rysunek 4.6: Diagram klas UML dziedziczenia klasy *Sokar::DicomScene*.



Rysunek 4.4: Diagram klas UML globalnej struktury programu.

4.5 Struktury danych

4.5.1 Konwertowanie danych ze znaczników

Każdy plik DICOM posiada zbiór elementów danych. Zapisane elementy danych należy przekonwertować na obiekty danych odpowiadające potrzebom programu. Dlatego został zaimplementowany obiekt klasy *Sokar::DataConverter* zajmujący się konwersją danych z pliku DICOM na dane w formacie odpowiadającym programowi.

Obiekt konwertera jest tworzony na podstawie pliku DICOM i przy wywoływaniu konwersji należy podać tylko znacznik, który nas interesuje. Takie rozwiążanie pozwala na przesyłanie do wszystkich obiektów jednego względnie małego obiektu konwertera, co ułatwia zarządzanie dostępem do pliku DICOM.

Klasa *Sokar::DataConverter* posiada następujące funkcje, pozwalające na konwertowanie danych:

- *Sokar::DataConverter::toString()*

Funkcja konwertuje element na obiekt tekstu *Qt::QString*.

- *Sokar::DataConverter::toAttributeTag()*

Funkcja konwertuje element o znaczniku typu VR:AT na obiekt znacznika *gdcm::Tag*.

- *Sokar::DataConverter::toAgeString()*

Funkcja konwertuje element o znaczniku typu VR:AS na tekst w postaci czytelnej, np: „18 weeks” lub „3 years”.

- *Sokar::DataConverter::toDate()*

Funkcja konwertuje element o znacznik typu VR:DA na obiekt klasy *Qt::QDate*, który ma w sobie wbudowaną konwersję na tekst zależny od ustawień językowych aplikacji.

- *Sokar::DataConverter::toDecimalString()*

Funkcja konwertuje element o znacznik typu VR:DS na obiekt wektora posiadającego liczby rzeczywiste. *qreal* jest aliasem do typu zmiennoprzecinkowego, na systemach 64-bitowy jest to *double*.

- *Sokar::DataConverter::toIntegerString()*

Funkcja konwertuje element o znacznik typu VR:IS na 32-bitową liczbę całkowitą (*qint32*).

- *Sokar::DataConverter::toPersonName()*

Funkcja konwertuje element o znacznik typu VR:PN na obiekt tekst zawierający imię w formie pisanej.

- *Sokar::DataConverter::toShort()*

Funkcja konwertuje element o znacznik typu VR:SS na 16-bitowa liczbę całkowitą ze znakiem (*qint16*).

- *Sokar::DataConverter::toUShort()*

Funkcja konwertuje element o znacznik typu VR:US na 16-bitowa liczbę całkowitą bez znaku (*quint16*).

Oprócz powyższych funkcji jest jeszcze kilka innych funkcji pobocznych oraz kilka aliasów. Ogólne zasady konwersji, które się tyczą wszystkich danych:

- Większość VR jest to zapisanych jako tekst, kodowanie i dekodowanie tekstu jest zapewniane przez bibliotekę.
- Większość danych może mieć kilka wartości oddzielonych backslashem „\”, dlatego konwerter dla VR, w których standard przewiduje wiele wartości, zawsze zwraca wektor z tymi wartościami.
- Wszystkie dane są zapisane parzystą ilością bajtów, w przypadku tekstu dodaje się znak spacji na końcu danych. Taka spacja jest pomijana w analizie danych.

4.5.2 Scena

Scena jest obiektem jednej ramki obrazu i jest odpowiedzialna za pośrednie wygenerowanie obrazu oraz jego wyświetlenie na ekranie. Implementowana jest ona przez klasę *Sokar::DicomScene*, dziedziczącą po *Sokar::Scene*, natomiast *Sokar::Scene* dziedziczy po *Qt::QGraphicsScene*. Diagram klas UML znajduje się na rysunku 4.5

Wyświetlanie sceny

Qt zapewnia własny silnik graficzny, który pozwala na łatwą wizualizację przedmiotów, z obsługą obrótu i powiększania. Silnik ten jest implementowany w postaci scen za pomocą *Qt::QGraphicsScene*. Natomiast klasa *Qt::QGraphicsView* dostarcza element interfejsu graficznego, który jest miejscem do wyświetlania scen.

Na scenie mogą być wyświetlane obiekty dziedziczące po *Qt::QGraphicsItem*. Obiekty te mogą być dodawane, usuwane i przesuwane ze sceny w czasie rzeczywistym. Dodatkowo