



Politechnika Warszawska  
WYDZIAŁ ELEKTRONIKI  
I TECHNIK INFORMACYJNYCH

Instytut Radioelektroniki i Techniki Multimedialnych  
Zakład Elektroniki Jądrowej i Medycznej

# Praca dyplomowa inżynierska

na kierunku Elektronika  
w specjalności Inżynieria oprogramowania

Wieloplatformowa przeglądarka obrazów DICOM w C++

Adam Jędrzejowski  
nr albumu 277417

promotor  
prof. nzw. dr hab. inż. Waldemar Smolik

Warszawa 2019

# Wieloplatformowa przeglądarka obrazów DICOM w C++

Trza machnąć streszaenie

**Słowa kluczowe:** DICOM; DICOM viewer; C++; Qt; GDCM; programing

# Wieloplatformowa przeglądarka obrazów DICOM w C++

Po angielsku też xD

Słowa kluczowe: DICOM; DICOM viewer; C++; Qt; GDCM; programming

## Spis rysunków

2.1	Przykład narzędzia Lupa w przeglądarce MedDream DICOM Viewer. Zdjęcie użyte za zgodą Softnetu UAB. . . . .	9
2.2	Przykład wyświetlenia wielu obrazów na raz w jednym oknie w przeglądarce Sante DICOM Viewer 3D Pro. Zdjęcie użyte za zgodą Santesoft. . . . .	10
2.3	Przykład generowania obrazów 3D z wielu obrazów tomograficznych w przeglądarce Sante DICOM Viewer 3D Pro . . . . .	11
2.4	Przykład rekonstrukcji wielopłaszczyznowej w przeglądarce Athena DICOM Viewer. Zdjęcie użyte za zgodą Medical Harbour. . . . .	11
2.5	Elementy danych w zbiorze elementów danych. Zdjęcie ze standardu DICOM dostępne pod adresem <a href="http://dicom.nema.org/medical/dicom/2019a/output/chtml/part05/chapter_7.html">http://dicom.nema.org/medical/dicom/2019a/output/chtml/part05/chapter_7.html</a> . . . . .	14
3.1	Przykładowe okienko programu w Qt. Zdjęcie własne. . . . .	22
4.1	Okno przeglądarki tuż po uruchomieniu. Zdjęcie własne. . . . .	29
4.2	Okno przeglądarki z wczytanymi kilkoma obrazami. Zdjęcie własne. . . . .	30
4.3	Przykładowa scena z obrazem monochromatycznym. Zdjęcie własne. . . . .	31
4.4	Diagram klas UML globalnej struktury programu. . . . .	32
4.5	Diagram klas UML dziedziczenia klasy <i>Sokar::DicomScene</i> . . . . .	34
4.6	Diagram klas UML dziedziczenia klasy <i>Sokar::DicomScene</i> . . . . .	35
4.7	Diagram klas UML dziedziczenia klasy <i>Sokar::SceneIndicator</i> . . . . .	36
4.8	Diagram klas UML dziedziczenia klasy <i>Sokar::DicomSceneSet</i> . . . . .	40
4.9	Wygląd zakładki wraz z numeracją elementów interfejsu. Zdjęcie własne. . . . .	42
4.10	Paleta HotIron (po prawej) w porównaniu do palety w skali szarości (po lewej). Zdjęcie ze standardu DICOM dostępne pod adresem <a href="http://dicom.nema.org/medical/dicom/2019a/output/chtml/part06/chapter_B.html">http://dicom.nema.org/medical/dicom/2019a/output/chtml/part06/chapter_B.html</a> . . . . .	50
4.11	Wizualizacja układu osi współrzędnych kartezjańskich pacjenta. Zdjęcie własne. . . . .	58
4.12	Podział płaszczyzny sceny. Wyróżniono osiem części. Zdjęcie własne. . . . .	61
4.13	Przykładowy obraz medyczny (przekrój głowy MR) z oznaczeniem orientacji obrazu z apomocą liter A, P, R, L, F, H. Zdjęcie własne. . . . .	62

OŚWIADCZENIE

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa inżynierska pt. Wieloplatformowa przeglądarka obrazów DICOM w C++:

- została napisana przeze mnie samodzielnie,
- nie narusza niczych praw autorskich,
- nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam, że przedłożona do obrony praca dyplomowa nie była wcześniej podstawą postępowania związanego z uzyskaniem dyplomu lub tytułu zawodowego w uczelni wyższej. Jestem świadom, że praca zawiera również rezultaty stanowiące własności intelektualne Politechniki Warszawskiej, które nie mogą być udostępniane innym osobom i instytucjom bez zgody Władz Wydziału Elektrycznego.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Adam Jędrzejowski.....

## Rozdział 6 Podsumowanie

Celem pracy było napisanie aplikacji do obsługi obrazów DICOM w C++ z możliwością kompilacji na wiele platform. Cel udało się osiągnąć. Wieloplatformowość uzyskano poprzez użycie bibliotek dostępnych na różnych platformach: Qt i GDCM, które również zostały napisane w C++. Aplikacja działa w ten sam sposób na wszystkich testowanych platformach: Linux, MacOS i Windows. Zapewniono jednolity sposób kompilacji na platformach przy użyciu narzędzia CMake. Zaplansowano i dodano obsługę podstawowych operacji na obrazie ułatwiających jego oglądanie i ocenienie. 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 przekształceniami 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 ABI w różnych kompilatorach. Problem można rozwiązać kompilując bibliotekę GDCM własnoręcznie.

## Rozdział 5

### Kompilacja

- 5.1 Narzędzia potrzebne do kompilacji
- 5.2 Biblioteki potrzebne do kompilacji
- 5.3 Przygotowanie katalogów
- 5.4 Utworzenie plików budowania
- 5.5 Uruchomienie kompilacji
- 5.6 Przeniesienie plików do jednego folderu

# Spis treści

1	Wstęp	1
---	-------	---

3	Obrazowanie diagnostyczne w medycynie	3
---	---------------------------------------	---

3	2.1	Obrazowe techniki diagnostyczne	3
---	-----	---------------------------------	---

5	2.2	Parametry obrazów	5
---	-----	-------------------	---

5	2.2.1	Podstawowe parametry obrazu cyfrowego	5
---	-------	---------------------------------------	---

7	2.2.2	Kontrast	7
---	-------	----------	---

7	2.2.3	Rozdzielczość	7
---	-------	---------------	---

8	2.2.4	Stosunek sygnału do szumu (SNR)	8
---	-------	---------------------------------	---

8	2.2.5	Poziom artefaktów	8
---	-------	-------------------	---

8	2.2.6	Poziom zniekształceń przestrzennych	8
---	-------	-------------------------------------	---

8	2.3	Prezentacja obrazów medycznych	8
---	-----	--------------------------------	---

8	2.3.1	Przebiegi choroby	8
---	-------	-------------------	---

8	2.3.2	Funkcje przebiegi choroby	8
---	-------	---------------------------	---

12	2.3.3	Kryteria porównywania przebiegów choroby	12
----	-------	--	----

13	2.4	Format cyfrowych obrazów medycznych	13
----	-----	-------------------------------------	----

13	2.4.1	Standard DICOM v3.0	13
----	-------	---------------------	----

14	2.4.2	Sposób zapisu danych w pliku DICOM	14
----	-------	------------------------------------	----

17	2.4.3	DICOMDIR	17
----	-------	----------	----

17	2.4.4	Inne formaty zapisu	17
----	-------	---------------------	----

18	3	Biblioteki i narzędzia	18
----	---	------------------------	----

18	3.1	CMake	18
----	-----	-------	----

18	3.2	QT	18
----	-----	----	----

18	3.2.1	Wymowa	18
----	-------	--------	----

19	3.2.2	Licencja	19
----	-------	----------	----

19	3.2.3	Normy i certyfikaty	19
----	-------	---------------------	----

19	3.2.4	Globalne typy struktur	19
----	-------	------------------------	----

20	3.2.5	Klasa QObject	20
----	-------	---------------	----

22	3.2.6	Graficzny interfejs użytkownika	22
----	-------	---------------------------------	----

23	3.2.7	Oddzielenie od platformy	23
----	-------	--------------------------	----

23	3.3	GDCM	23
----	-----	------	----

23	3.3.1	Uzasadnienie wyboru	23
----	-------	---------------------	----

24	3.3.2	Opis	24
----	-------	------	----

24	3.3.3	Licencja	24
----	-------	----------	----

24	3.3.4	Podstawowe klasy	24
----	-------	------------------	----

25	3.3.5	Przykład użycia	25
----	-------	-----------------	----

27	3.4	Git	27
----	-----	-----	----

**Rysunek 4.13:** Przykładowy obraz medyczny (przekrój głowy MRI) z oznaczeniem orientacji obrazu z pomocą liter A, P, R, L, F, H. Zdjęcie własne.

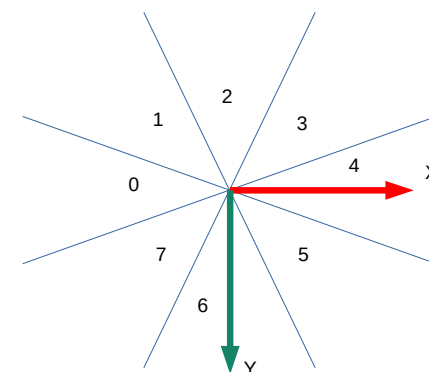


<b>4</b>	<b>Implementacja</b>	<b>28</b>
4.1	Zakres implementacji . . . . .	28
4.2	Wieloplatformowość . . . . .	29
4.3	Graficzny interfejs użytkownika . . . . .	29
4.4	Projekt struktury obiektowej programu . . . . .	31
4.5	Struktury danych . . . . .	32
4.5.1	Konwertowanie danych z znacznikach . . . . .	32
4.5.2	Scena . . . . .	33
4.5.3	Kolekcje scen . . . . .	39
4.5.4	Zakładka . . . . .	41
4.5.5	Obiekt zakładek . . . . .	45
4.5.6	Okno główne programu . . . . .	45
4.6	Algorytmy . . . . .	46
4.6.1	Cykl generowania obrazów . . . . .	46
4.6.2	Generowania obrazu monochromatycznego . . . . .	48
4.6.3	Tworzenie transformat i ich użycie na obrazie . . . . .	56
4.6.4	Ustalanie pozycji pacjenta względem sceny . . . . .	58
<b>5</b>	<b>Kompilacja</b>	<b>63</b>
5.1	Narzędzia potrzebne do kompilacji . . . . .	63
5.2	Biblioteki potrzebne do kompilacji . . . . .	63
5.3	Przegotowanie katalogów . . . . .	63
5.4	Utworzenie plików budowania . . . . .	63
5.5	Uruchomienie kompilacji . . . . .	63
5.6	Przeniesienie plików do jednego folderu . . . . .	63
<b>6</b>	<b>Podsumowanie</b>	<b>64</b>

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

Przykład:

Punkt „H”, czyli punkt reprezentujący kierunek głowy, został przypisany do części 1 i odpowiednio „L” do części 7, „R” do części 3 i „F” do części 5. Punkty „A” i „P”, zostały pominięte ponieważ znalazły się na środku. Do lewego pola wstawiany jest tekst „HL”, do górnego „HR”, do prawego „RF” i do dolnego „LF”.



**Rysunek 4.12:** Podział płaszczyzny sceny. Wyróżniono osiem części. Zdjęcie własne.

Przykład można zobaczyć na rysunku 4.13. Na obrazie widzimy, że lewa strona pacjenta znajduje się po prawej stronie obrazu, prawa strona pacjenta po lewej, góra pacjenta na górnej części obrazu. Wynika z tego, że obraz przedstawia pacjenta skierowanego twarzą do nas.



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*:

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

Powyższa macierz różni się od macierzy definiowanej w standardzie. Po pierwsze wartości z  $DicomPixelSpacing$  (0x0028, 0x0030) zostały pominięte, a nadano im wartość 1. Po drugie - pozycja z  $DicomImagePosition$  (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 star-

towej.

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 zwana 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 prze-

mnienie lewostronne *rotateTransform* i *ScenePosition*.

$$rotateTransform * ScenePosition$$

następujące teksty:

Wyznaczana jest w ten sposób pozycja sześciu punktów pacjenta na płaszczyźnie sceny wyświetlanej. Następnie określany jest na której z ośmiu części płaszczyzny jest umiesz- czony dany punkt. Podział płaszczyzny jest widoczny na rysunku 4.12. 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 ekr- nem, więc jest pomijany. Następnie do czterech pól wyświetlających zostają wstawione

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 zwiększona dostępność aparatury spowodowały, że badania obrazowe są coraz bardziej powszechne. Badania obrazowe pomagają lekarzom w diagnostyce i terapii w codziennej praktyce lekarskiej. Przekazywanie badań obrazowych pomiędzy lekarzami różnych specjalności zostały 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).

Wartość  $\text{Dicom}_{\text{Tag}}^{\text{Image Orientation}}$  (0x0020, 0x0037) składa się z sześciu liczb, odpowiednio oznaczanych dalej  $X_x, X_y, X_z, Y_x, Y_y, Y_z$ . Standard DICOM definiuje następujący sposób interpretowania danych:

$$\begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} = \begin{bmatrix} X_x \Delta_i & Y_x \Delta_j & 0 & S_x \\ X_y \Delta_i & Y_y \Delta_j & 0 & S_y \\ X_z \Delta_i & Y_z \Delta_j & 0 & S_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \\ 0 \\ 1 \end{bmatrix} = M \begin{bmatrix} i \\ j \\ 0 \\ 1 \end{bmatrix}$$

gdzie:

- $P_{xyz}$  — koordynaty woksela (i,j) we współrzędnych obrazu wyrażone w milimetrach
- $S_{xyz}$  — trzy wartości z elementu ze znacznikiem  $\text{Dicom}_{\text{Tag}}^{\text{Image Position}}$  (0x0020, 0x0032). Oznacza on punkt pozycji pacjenta wyrażony w milimetrach w stosunku do urządzenia wykonującego pomiar.
- $X_{xyz}$  — trzy pierwsze wartości ze znacznika  $\text{Dicom}_{\text{Tag}}^{\text{Image Orientation}}$  (0x0020, 0x0037)
- $Y_{xyz}$  — trzy ostatnie wartości ze znacznika  $\text{Dicom}_{\text{Tag}}^{\text{Image Orientation}}$  (0x0020, 0x0037)
- $i$  i  $j$  — oznaczają współrzędne na macierzy obrazu, odpowiednio kolumnę i wiersz. Zero oznacza początek.
- $\Delta_i$  i  $\Delta_j$  — rzeczywista wielkość piksela obrazu wyrażona w milimetrach. W algorytmie wyznaczania strony pacjenta ta wartość może wynosić 1, ponieważ odpowiada za skalę.

Praktycznie rzecz biorąc, pierwsza macierz to wektor reprezentujący pozycję pacjenta, druga jest to przekształcenie macierzowe we współrzędnych jednorodnych, trzecia to pozycja na obrazie.

### Wyznaczanie pozycji pacjenta

Interesuje nas wyznaczenie pozycji sześciu (punktów) na płaszczyźnie obrazu. Załóżmy, że pacjent znajduje się w środku układu współrzędnych i jest nieskończenie mały. Możemy więc zdefiniować sześć punktów o następujących współrzędnych, dalej używanych pod nazwą *PatientPosition*, które będą odpowiadały stronom pacjenta:

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

**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::getPixmapTr
2  qTransform transform;
3  transform = * centerTransform;
4  transform = * scaleTransform;
5  transform = * rotateTransform;
6  transform = * panTransform;
7  return transform;
8  }

```

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

partTransform \* rotateTransform \* scaleTransform \* centerTransform

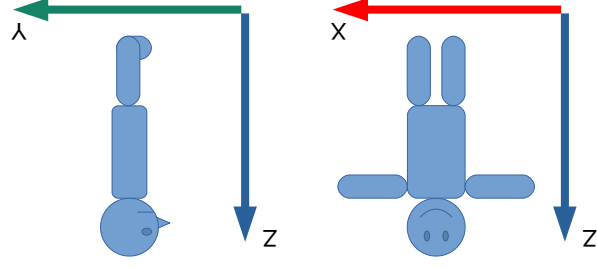
#### 4.6.4 Ustalenie pozycji pacjenta względem sceny

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 informacij o orijentaciji obrazu

Informacje o orientacji oraz pozycji względem pacjenta znajdują się w odpowiednio w tagach  $\text{DICOM}_{\text{Tag}}$  Image Orientation (0x0020, 0x0037) i  $\text{DICOM}_{\text{Tag}}$  Image Position (0x0020, 0x0032). Standard DICOM zdefiniował ułożenie osi we współrzędnych kartezjańskich następująco:

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



**Rysunek 4.11:** Wizualizacja układu osi współrzędnych kartezjańskich pacjenta. Zdjęcie własne.

## Rozdział 2

## 2.1 Obrazowe techniki diagnostyczne

Istnieje wiele technik obrazowania wykorzystujących różne zjawiska fizyczne zachodzące w materii. Podstawowe techniki obrazowania medycznego to:

- Radiografia — RTG

pod 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 biegnących pod różnym kątem i w różnych odległościach od badanego obiektu. Przekrój obiektu jest rekonstruowany numerycznie na podstawie zmierzonych projekcji wstecznej.

Obrazowany jest współczynnik przenikalności 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 rozdzielczość zależy od tych samych parametrów co w klasycznej radiografii.

W standardzie DICOM technika jest oznaczana skrótowcem „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 sekwencji pobudzeń polem elektromagnetycznym, wyróżniamy trzy typy obrazów: PD, T1 i T2. Kontrast zależy od gęstości protonów, czasu relaksacji podłużnej i poprzecznej, prędkości przepływu płynu. Rozdzielczość zależy od parametrów skanera (rozmiar wokseli).

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

- **Ultrasonografia**

Podczas 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 fali ultradźwiękowej pod różnymi kątami. Z każdego cyklu jest tworzona jedna linia, obraz jest tworzony z wielu linii, które następnie są układane pod różnymi kątami, odpowiadającym ich rzeczywistemu ułożeniu na głowicy. Wielkość obrazu jest zależna od algorytmu rekonstrukcji i jest z góry ustawiona przez producenta aparatu. Piksel w obrazie nie przedstawia żadnej wartości fizycznej, różnice pomiędzy pikselami definiują umowną różnicę gęstości zależną od aparatu. Kontrast zależy od częstotliwości fali, głębokości badanego obiektu, ilości piezoelektryków w głowicy, obrazowanej struktury. Rozdzielczość zależy od czasu trwania impulsu zaburzenia oraz od szerokości wiązki ultradźwiękowej (powierzchnia czynna przetworników).

W standardzie DICOM obraz ultrasonograficzny jest oznaczana 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 związku chemicznego zawierającego

- **FlipVertical** — na macierzy rotacji zostaje użyta funkcja *Qt::QTransform::scale()* z parametrami  $-1$  i  $1$ .
- **ClearRotate** — przywraca macierz rotacji do stanu zerowego

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

## Zmiany poprzez obsługę myszki

*Qt::QGraphicsScene* dostarcza możliwość obsługi myszki poprzez wirtualną funkcję *Qt::QGraphicsScene::mouseMoveEvent()*. Dzięki temu obsługa myszki może być rozszerzana przez wszystkie klasy dziedziczące po tej klasie. Dodatkowo funkcja ta dostarcza obiekt klasy *Qt::QGraphicsSceneMouseEvent*, w którym znajdują się informacje zarówno o obecnej pozycji myszki jak i poprzedniej pozycji myszki.

Jeżeli jest wykryty ruch myszki z wciśniętym lewym przyciskiem myszy, to w zależności od stanu paska narzędzi, wywoływana jest odpowiednia akcja. Akcje są obsługiwane przez klasy *Sokar::DicomScene* i *Sokar::Monochrome::Scene*. Każda z nich obsługuje pewną pulę stanów. Lista obsługiwanych stanów paska narzędzi:

- **Pan** — stan przesuwania, obsługiwany przez *Sokar::DicomScene*

Na macierzy przesuwania wywoływana jest funkcja przesunięcia *Qt::QTransform::translate()* z parametrami odpowiadającymi przesunięciu myszki.

- **Zoom** — stan skalowania, obsługiwany przez *Sokar::DicomScene*

Na macierzy skalowania wywoływana jest funkcja skalowania *Qt::QTransform::scale()* z parametrem **scale** wyliczanym podanym wzorem:

$$\begin{aligned} scale &= 1 \\ scale &= scale - \Delta y * 0.01 \\ scale &= scale - \Delta x * 0.001 \end{aligned}$$

Sprawia to, że ruch pionowy jest bardziej czuły na zmianę niż ruch poziomy. Teoretycznie jest możliwość implementacji odrębnego skalowania w dwóch osiach, jednakże jest to nieintuicyjne.

- **Rotate** — stan rotacji, obsługiwany przez *Sokar::DicomScene*

Na macierzy rotacji wywoływana jest funkcja rotacji *Qt::QTransform::rotate()* z parametrem **rotate** wyliczanym podanym wzorem:

$$\begin{aligned} rotate &= 0 \\ rotate &= rotate + \Delta y * 0.5; \\ rotate &= rotate + \Delta x * 0.1; \end{aligned}$$

Sprawia to, że ruch pionowy jest bardziej czuły na zmianę niż ruch poziomy.

- **Windowing** — stan okienkowania, obsługiwany przez *Sokar::Monochrome::Scene*

Do obiektu okienka są wysyłane zmiany poprzez funkcje: *Sokar::Window::mvVertical()* z parametrem  $\Delta y$  i *Sokar::Window::mvHorizontal()* z parametrem  $\Delta x$ . Następnie ponownie jest generowany obraz z uwzględnieniem zmiany okienka.

4.6.3 Tworzenie transformat i ich użycie na obrazie

Współrzedne jednorodne

Współrzedne jednorodne definiuje się jako sposób reprezentacji punktów  $n$ -wymiarowej przestrzeni rzutowej za pomocą układu  $n + 1$  współrzędnych. W bibliotece Qt jedna z implementacji współrzędnych jednorodnych jest klasa  $Qt::QTransform$ . Implementuje ona podstawowe zachowania macierzy 3 na 3, jak również wbudowane operacje takie jak: przesunięcie implementowane prze  $Qt::QTransform::translate()$ , obrót implementowany przez funkcję  $Qt::QTransform::rotate()$  i skalowanie implementowane przez  $Qt::QTransform::scale()$ .

Przykład działania:

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

Powyższe przekształcenie macierzowe skaluje obiekt na 50% szerokości, obraca o 45 stopni, przesunwa 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 przycisk.

Zmiany poprzez odebranie sygnału

Na pasku zadań, nad sceną, znajduje się szereg przycisków, które po wcisnięciu wysyła ją 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. Odbior wszystkich sygnałów jest implementowany przez wirtualną funkcję  $Sokar::DicomScene::toolBarActionSlot()$ , która jest słotem. Lista opisów reakcji na sygnały (stan zerowy macierzy, to stan w którym macierz nie wykonuje żadnych operacji):

- **ClearPan** — przywraca macierz przesunięcia do stanu zerowego

- **FitScreen** — 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  $Qt::QTransform::rotate(90)$  z parametrem 90.

- **RotateLeft90** — na macierzy rotacji zostaje użyta funkcja  $Qt::QTransform::rotate(-90)$  z parametrem -90.

- **FlipHorizontal** — na macierzy rotacji zostaje użyta funkcja  $Qt::QTransform::scale()$  z parametrami 1 i -1.

2.2 Parametry obrazów

2.2.1 Podstawowe parametry obrazu cyfrowego

izotop. Charakteryzuje się on krótkim czasem rozpadu i powinowactwem chemicznym z badanymi organami. Wykrywa się rozpad zachodzący w ciele poprzez rejestrację promieniowania wytwarzanego podczas tego rozpadu, a następnie przedstawia się go w formie graficznej.

Detekcja odbywa się za pomocą scyntylatora, fotopowielacza i układu liniowego su-mowania. Wielkość obrazu zależy od różniących się współrzędnych przez detektor. Píkseł reprezentuje ilość zliczeń na jednej współrzędnej. Kontrast zależy od czasu trwania pomiaru, oraz od aktywności wstrzykniętego radiofarmaceutyka. Rozdzielczość zależy od możliwości kamer scyntyacyjnych, zwanyimi także scyntykamerami, gammakamerami lub kamerami Angera.

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

- Tomografia SPECT

Jest to technika obrazowania z gałęzi medycyny nuklearnej, w której rejestruje się promieniowanie powstające rozpadu gamma. Zródłem promieniowania (fotonów) jest radiofarmaceutyk, którego izotop ulega rozpadowi z emisją promieniowania gamma. Kontrast zależy od wydajności detektorów, odległość detektora od obiektu oraz położenie obiektu. Na rozdzielczość ma wpływ przestrzenna rozdzielczość matrycy detektora oraz liczba detektorów.

W standardzie DICOM obraz jest oznaczany jako „PT”.

- Tomografia PET

Technika obrazowania z gałęzi medycyny nuklearnej, w której rejestruje się promieniowanie powstające podczas anihilacji pozytonów (antyelektronów). Zródłem anihilacji jest podana pacjentowi substancja promieniotwórcza, ulegająca rozpadowi beta plus. Rejestrujemy fotony powstające podczas anihilacji pozytonów. Kontrast zależy od wydajności detektorów, odległości detektora od obiektu oraz położenia obiektu. Na rozdzielczość ma wpływ przestrzenna rozdzielczość matrycy detektora oraz liczba detektorów.

W standardzie DICOM obraz jest oznaczana jako „PT”.

Istnieją badania łączące w sobie różne techniki, takie jak:

- PET-CT, PET/CT — połączenie PET z wielorzędowym tomografem komputerowym
- PET-MRI, PET/MRI — połączenie PET z rezonansem magnetycznym

Standard DICOM nazywa techniki obrazowania modalnościami (ang. *modality*).

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

$\text{Dicom}_{\text{Tag}}$  PatientID (0x0010, 0x0020)

Oznacza to, że jest to znacznik o słowie kluczowym „PatientID”, numerze grupy 10<sub>16</sub> i numerze elementu 20<sub>16</sub>.

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 matrycą pikseli o ustalonych rozmiarach. W przypadku standardu DICOM obrazy są matrycami woksela, posiadającymi wysokość (zapisaną w  $\text{Dicom}_{\text{Tag}}$  Rows (0x0028, 0x0010)) oraz szerokość (zapisaną w  $\text{Dicom}_{\text{Tag}}$  Columns (0x0028, 0x0011)). Do poprawnej interpretacji znaczenia macierzy służy znacznik  $\text{Dicom}_{\text{Tag}}$  Photometric Interpretation (0x0028, 0x0004), informujący o fotometrycznym znaczeniu woksela. 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 wksel jest trzy-kanałowym pikselem RGB (kanały: czerwony, zielony i niebieski).
- „HSV” (ang. *Hue Saturation Value*) — wksel 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 wksel 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 wksel to piksel w modelu przestrzeni barw nazwanej YC<sub>b</sub>C<sub>r</sub>. Dodatkowo standard zdefiniował pochodne tej wartości: „YBR\_FULL\_422”, „YBR\_PARTIAL\_422”, „YBR\_PARTIAL\_420”, „YBR\_ICT”, „YBR\_RCT”, 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 wstecznej kompatybilności, ale nie jest to wymagane.

```
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<quint8>();
14         break;
15     case gdcm::PixelFormat::INT16:
16         genQPixmapOfType<qint16>();
17         break;
18     case gdcm::PixelFormat::UINT16:
19         genQPixmapOfType<quint16>();
20         break;
21     case gdcm::PixelFormat::INT32:
22         genQPixmapOfType<qint32>();
23         break;
24     case gdcm::PixelFormat::UINT32:
25         genQPixmapOfType<quint32>();
26         break;
27     case gdcm::PixelFormat::INT64:
28         genQPixmapOfType<qint64>();
29         break;
30     case gdcm::PixelFormat::UINT64:
31         genQPixmapOfType<quint64>();
32         break;
33
34     default: /* W przypadku innych jest zwracany wyjątek */
35         throw Sokar::ImageTypeNotSupportedException();
36 }
37
38 pixmap.convertFromImage(qImage);
39 return true;
40 }
```

## Palety

Klasa *Sokar::Palette* reprezentuje palety kolorów używanych do pseudokolorowania obrazu. Paleta przerabia liczbę zmiennooprzecinkową od zera do jedynki na kolor RGB, zwracając *Sokar::Pixel*. Paleta nie ma zdefiniowanej długości minimalnej i maksymalnej.

Palety są wczytywane z plików XML w czasie uruchamiania programu i można definiować własne palety nie będące częścią standardu. Przykładowy wygląd pliku palety HotIron:

```
1 <palette display="Hot Iron" name="HOT_IRON">
2
3     <entry red="0" green="0" blue="0"/>
4     <entry red="2" green="0" blue="0"/>
5     <entry red="4" green="0" blue="0"/>
6
7     ...
8
9     <entry red="254" green="0" blue="0"/>
10    <entry red="255" green="0" blue="0"/>
11    <entry red="255" green="2" blue="0"/>
12
13    ...
14
15    <entry red="255" green="250" blue="248"/>
16    <entry red="255" green="252" blue="252"/>
17    <entry red="255" green="255" blue="255"/>
18 </palette>
```



Kwantyzacja obrazu, czyli informacja mówiąca o tym ile poziomów na obrazie jest zapisanych w czterech znacznikach:

- $\text{D}_{\text{icom}}^{\text{Tag}}$  Bits Allocated (0x0028, 0x0100) — informuje na jak wiele bitów zostało zaalokowanych do zapisania jednego piksela
- $\text{D}_{\text{icom}}^{\text{Tag}}$  Bits Stored (0x0028, 0x0101) — informuje jak wiele bitów z zaalokowanych posiada wartość piksela
- $\text{D}_{\text{icom}}^{\text{Tag}}$  High Bit (0x0028, 0x0102) — informuje gdzie znajduje się najstarszy bit
- $\text{D}_{\text{icom}}^{\text{Tag}}$  Pixel Representation (0x0028, 0x0103) — informuje czy poziomy są ze znakiem czy bez

Obraz DICOM również zawiera w sobie informacje o próbkowaniu. Z uwagi na to, że próbkowanie wygląda inaczej w każdej technice, standard posiada oddzielne tagi informujące o próbkowaniu dla każdej techniki. Próbkowanie poszczególnych technik opisałem w sekcji 2.1.

## 2.2.2 Kontrast

Jedną z wielu definicji kontrastu jest kontrast Michelsona wyrażony wzorem:

$$\frac{I_{\max} - I_{\min}}{I_{\max} + I_{\min}}$$

gdzie  $I_{\max}$  i  $I_{\min}$  to najwyższa i najniższa wartość luminancji.

## 2.2.3 Rozdzielczość

### Przestrzena

Rozdzielczość przestrzena obrazu to najmniejsza odległość między dwoma punktami obrazu, które można rozróżnić. Jest ona silnie związana z kontrastem obrazu za pomocą funkcji przenoszenia modulacji (MTF — Modulation Transfer Function). Jest to krzywa ukazująca degradację kontrastowości w miarę zwiększania częstotliwości przestrzennej okresowego wzorca. Funkcję MTF można wyznaczyć używając rozdzielnych tarcz rozdzielających przestrzenną rozdzielczość okresia się zazwyczaj jako liczbę równoległych linii, czarnych i białych, które można rozróżnić ma 1 milimetrze obrazu(paralinię na milimetr). Rozdzielczość przestrzenna jest zależna od kontrastu obrazu. Zależność ta jest inna dla każdej techniki.

### Czasowa

Każdy pomiar wymaga pewnego czasu pobierania danych. W nie których przypadkach ważna jest również zmiany zachodzące w czasie wykonywania badania. Rozdzielczość czasowa, jest istotna w obrazach dynamicznych, np. angioMR. Kiedy mamy pomiar dokonywany w określonym czasie i ustalone są markery czasowe. Rozdzielczość czasowa jest definiowana jako odległość w czasie od dwóch klatek obrazowania.

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

Jest to funkcja, która dzieli obraz na wątki, tworzy je i uruchamia. Ilość wątków jest ustalana za pomocą funkcji *Qt::QThread::idealThreadCount()*. Wątki działają na zakresach o długości ilości wokseli podzielonej przez ilość wątków. Kod funkcji:

```
1 template<typename IntType, typename WinClass>
2 void Monochrome::Scene::genPixmapOfTypeWidthWindow() {
3     /* Tworzenie wątków */
4     std::vector<std::thread> threads;
5     for (int i = 0; i < threads.size(); ++i) {
6         threads[i] = std::thread(
7             &Monochrome::Scene::genPixmapOfTypeWidthWindow,
8             this,
9             i,
10            &threads[i]);
11    }
12    for (auto &t: threads) t.join();
13    /* Czekanie na wszystkie wątki */
14    genPixmapOfTypeWidthWindow(IntType, WinClass, 0, step);
15    /* W celu zmniejszenia ilości wątków, wątek obecny też zostanie wykorzystany */
16    genPixmapOfTypeWidthWindow(IntType, WinClass, 0, step);
17    threads.push_back(std::move(t));
18    }
```

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

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

```
1 template<typename IntType>
2 void Monochrome::Scene::genPixmapOfType() {
3     switch (GetCurrentWindow()->type()) {
4         case Window::Interactive:
5             genPixmapOfTypeWidthWindow(IntType, WindowInteractive());
6             break;
7         case Window::Static:
8             genPixmapOfTypeWidthWindow(IntType, WindowStatic());
9             break;
10        default:
11            throw WrongScopeException(--FILE--, --LINE--);
12    }
```

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

Funkcja odwołuje się do *genQPixmapOfType()* i sprawdza, czy odwołanie obrazu jest konieczne, następnie sprawdza typ liczby wokseli i uruchamia *Sokar::Monochrome::Scene::genQPixmapOfType()*. Kod funkcji:

```
1 bool Monochrome::Scene::generatePixmap() {
2     /* Odwołamy okno i sprawdzamy czy odwołanie obrazu jest konieczne */
3     genPixmapOfType() ->genUT();
4 }
```

## 2.2.4 Stosunek sygnału do szumu (SNR)

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

## 2.2.5 Poziom artefaktów

Artefakty to zjawiska fałszujące obraz poprzez tworzenie nie istniejących struktur w obrazie. 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.

Standard DICOM pozwala na odczytanie badania i wyświetlanie go w postaci obrazów radiologicznych, scyntygraficznych, itd.

### 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 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.

```
12     qreal x = qreal(signedMove) * -1;
13
14     auto &background = isInversed() ? palette->getForeground() : palette->
getBackground();
15     auto &foreground = isInversed() ? palette->getBackground() : palette->
getForeground();
16
17     /* Interakcja */
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 {
26             *pixelArray = palette->getPixel(a * x + b);
27         }
28
29         x++;
30         pixelArray++;
31     }
32
33     pixelArray = &arrayVector[0];
34
35     updateLastChange();
36
37     return true;
38 }
39
40 return false;
41 }
```

Funkcja zmiany wartości obrazu na kolor prezentuje się następująco:

```
1 inline const Pixel &getPixel(quint64 value) override {
2     return *(pixelArray + signedMove + value);
3 }
```

#### Iterowanie po obrazie

Po wygenerowaniu obraz, należy przeiterować go przez funkcje „okna”. Do zokienkowania jednego piksela nie potrzeba innego piksela dlatego w celu przyspieszenia procesu okienkowania, iteracja po obrazie odbywa się w wielu wątkach.

W C++ typy zmiennych muszą być zdefiniowane przed kompilacją, co jest bardzo problematyczne. Mając dwa typy okienek, każde obsługujące 4 typy liczb całkowitych, musiało by zostać zaimplementowanych 8 prawie identycznych funkcji. Dlatego podział ten został zaimplementowany za pomocą 4 funkcji z szablonami:

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

Jest funkcją jednego wątku, który iteruje po obrazie. Jego parametrami są zakresy podane w indeksach woksela po których ma iterować. **IntType** to jest typ zmiennej woksela obrazu. **WinClass** klasa okienka. Nazewnictwo będzie kontynuowane w następnych punktach. Kod funkcji:

```
1 template<typename IntType, typename WinClass>
2 void Monochrome::Scene::genQPixmapOfTypeWidthWindowThread(quint64 from, quint64 to)
3 {
4     auto buffer = &targetBuffer[from];
5     auto origin = (IntType *) &originBuffer[0];
6     auto windowPtr = (WinClass *) getCurrentWindow();
7
8     origin += from;
9
10    for (quint64 i = from; i < to; i++, origin++) {
11        /* W tym miejscu jest dokonywana zamiana liczby na kolor */
```



- Przesuwanie (ang. *pan*), czyli możliwość przesuwania obrazu o dowolny wektor. Jest to przydatne, gdy powiększony obraz do takiego stopnia, że nie będzie mieścił się na ekranie lub w okienku programu.
- Lupa, skalowanie miejscowe. Jest to możliwość miejscowego powiększenia obrazu. Przykład użycia takiego narzędzia znajduje się na rysunku 2.1.



**Rysunek 2.1:** Przykład narzędzia Lupa w przeglądarce MedDream DICOM Viewer. Zdjęcie użyte za zgodą Softeta UAB.

- Rotacja i odbicia lustrzane, czyli możliwość obrócenia obrazu o zadany kąt oraz możliwość uzyskania odbicia lustrzanego obrazu w dwóch osiach X i Y.

### Analiza parametrów w celu lepszej informacji

- Okienkowanie. Termin odnosi się do używania funkcji okna cyfrowego w celu zamiany obrazu danych na obraz monochromatyczny możliwy do wyświetlenia. Okienkowanie jest szczegółowo opisane w sekcji 4.6.2 wraz z generowaniem obrazu monochromatycznego.
- Maski lub nakładki (ang. *overlay*). Jest możliwość nałożenia maski, elementu, który będzie przysłaniał fragment obrazu w celu lepszej wizualizacji bądź ukrycie mało wartościowych obiektów, np. tła. Standard DICOM umożliwia nałożenie wielu masek na jeden obraz.

### Obsługa wielu plików

- Obsługa DICOMDIR. Jest to możliwość wczytania pliku DICOMDIR i wyświetlenie struktury serii badań. Plik DICOMDIR to wiele zindeksowanych plików zawierających ich zbiór elementów danych, bez obrazów.
- Wczytanie wielu plików i ich połączenie w formie filmu, czyli możliwość wczytania wielu plików z tej samej serii, ułożenia ich według pozycji geometrycznej i wyświetlenie ich jako film. Innymi słowy jest periodyczna podmiiana obrazu na obraz następny w serii.

Posiadamy, teraz dwa punkty okienka odnoszące się do wartości obrazu. Wyznaczone parametry prostej przechodzącej przez dwa punkty:

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

$$b = y_1 - a * x_1$$

Teraz algorytm się rozpadła. Pobieranie wartości z okienka odbywa się za pomocą funkcji `Sokar::MonochromeWindow::getPixel()`.

### Implementacja dynamiczna bez tablicy LUT

W tej wersji funkcja `Sokar::MonochromeWindow::getPixel()` wygląda następująco:

```
inline const Pixel &getPixel(quint64 value) override {
    if (value < x0) {
        return background;
    } else if (value > x1) {
        return foreground;
    } else {
        return palette->getPixel(a * value + b);
    }
}
```

Widzimy tutaj, że funkcja najpierw sprawdza czy zakres okienka został przekroczony, 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.

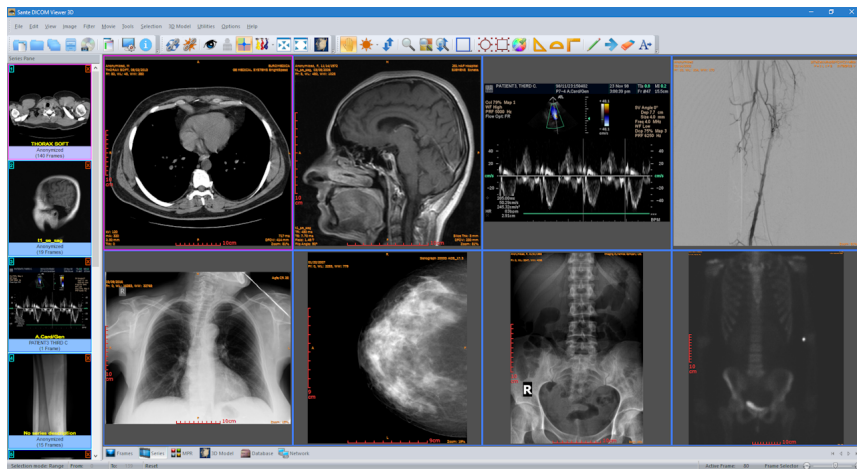
### Implementacja statyczna z tablicą LUT

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 `Tag` `DicomBitsStored` (0x0028, 0x0101). `signedMove` to liczba przesunięcia liczb, przyjmując wartość zero, gdy dane woksele są całkowite nieujemne lub wartość przeciąg do `maxValue`, gdy woksele są być ujemne. Długość wektora pikseli jest sumą `maxValue` i `signedMove`. A indeks woksele w wektorze ma wartość tego woksele zwiększoną o `signedMove`. Wyphienienie wektora wartościami odbywa się poprzez iterację 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:

```
bool genLUT() override {
    if (WindowInt::genLUT()) {
        /* Przekalowanie wektora, gdy jest to wymagane */
        if (arraySize != signedMove + maxValue) {
            arraySize = signedMove + maxValue;
            arrayVector.resize(arraySize);
        }
        /* Wyliczenie najmniejszej wartości */
    }
```

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

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



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

## Generowanie obrazów wolumetrycznych

Jeżeli mamy do dyspozycji wiele obrazów tomograficznych o znanych parametrach to możemy wczytać je, poseregować 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 obrazu, 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

*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.

## Wyznaczenie parametrów okna

Najpierw wyznaczam okienko, które zmienia wartości obrazu na skale od zera do jeden:

$$x_0 = center - width/2$$

$$x_1 = center + width/2$$

$$y_1 = 0.0$$

$$y_0 = 1.0$$

gdzie:

- *center* — środek okienka
- *width* — szerokość okienka
- *x0* i *y0* — współrzędne pierwszego punktu
- *x1* i *y1* — współrzędne drugiego punktu

Przeglądarka pozwala na inwersję okienka. Dlatego kiedy użytkownik zażyczy sobie inwersji, zmienne *y0* i *y1* zamieniają się wartościami.

Standard DICOM przewiduje, że wszystkie dane powinny być wyskalowane, za pomocą wzoru.

$$OutputUnits = m * SV + b$$

gdzie:

- *m* — wartość z  $^{Dicom}_{Tag}RescaleSlope$  (0x0028, 0x1053)
- *b* — wartość z  $^{Dicom}_{Tag}RescaleIntercept$  (0x0028, 0x1052)
- *SV* — stored values — wartość pixela z pliku
- *OutputUnits* — wartość wynikowa

Wartości okienka odnoszą się do wartości już wyskalowanej, a ponieważ skalowanie całego obrazu jest czasochłonne, przeskalowanie okienka da taki sam efekt:

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

więc:

$$x_0 = rescaleIntercept$$

$$x_1 = rescaleIntercept$$

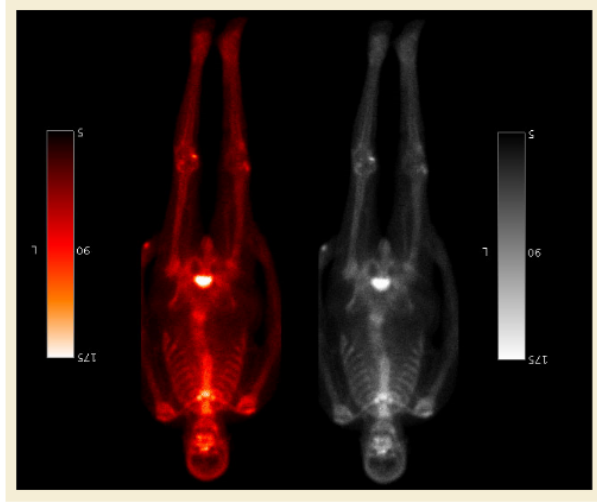
$$x_0 / = rescaleSlope$$

Z uwagi na konieczność osiągnięcia dużej szybkości wyświetlania obrazu, warto jest taksonować wartości funkcji  $f$ . Wartości tei funkcji należy przeliczyć, gdy zmienne zostanę 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 tan zwanym oknie, co jest bardzo kosztowne obliczeniowo. Dlatego dobrym pomysłem jest stworzenie mniejszej tablicy typu LookUpTable, wypelnienie jej wszystkimi możliwymi wartościami obrazu, a następnie przeobienie 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 eksabajtów ( $55 * 10^6$  terabajtów). Alokowanie dwóch największych wartości może być lekko problematyczne, 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 tablicy LUT, wielowątkowa iteracja po obrazie.

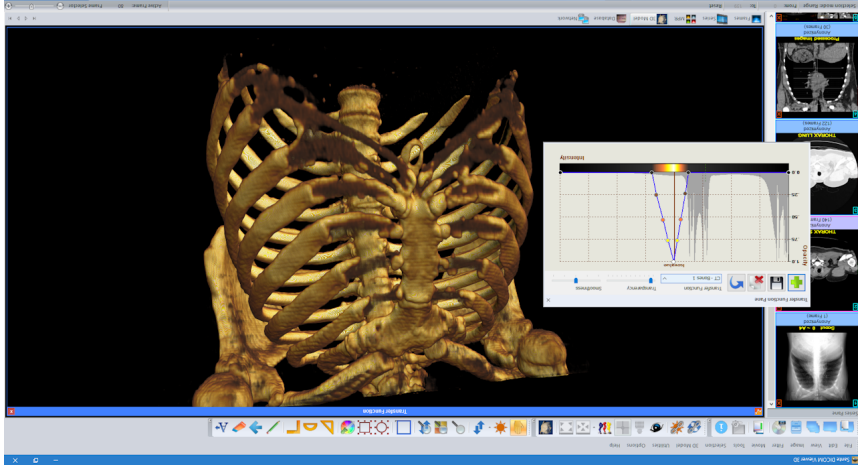
Okno z LUT jest implementowane przez Sokar::Monochrome::WindowInStatic. Okno bez LUT jest implementowane przez Sokar::Monochrome::WindowInDynamic. Obie klasy dziedziczą po abstrakcyjnej klasie Sokar::MonochromeWindow, która z kolei dziedziczy po

### Implementacja algorytmu

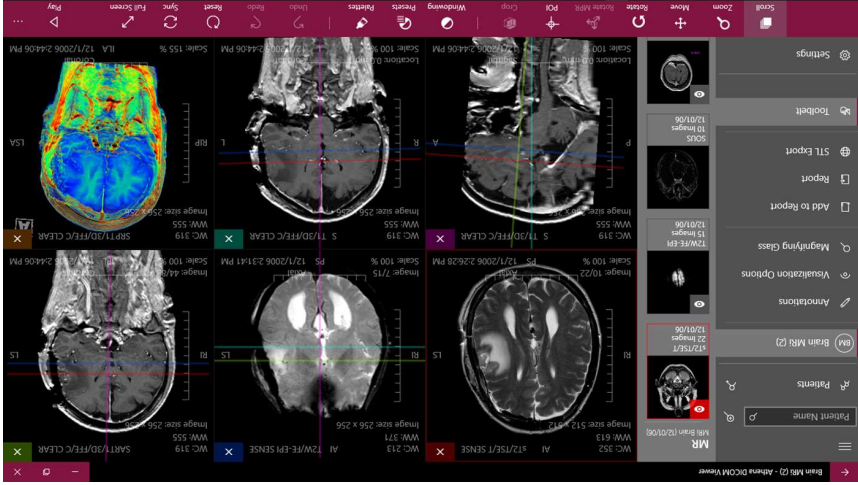
**Rysunek 4.10:** Paleta Holtron (po prawej) w porównaniu do palety w skali szarości (po lewej). Zdjęcie ze standardu DICOM dostępne pod adresem [http://dicom.nema.org/medical/dicom/2019a/output/chtml/part06/part06/chapter\\_B.html](http://dicom.nema.org/medical/dicom/2019a/output/chtml/part06/part06/chapter_B.html).



**Rysunek 2.3:** Przykład generowania obrazów 3D z wielu obrazów tomograficznych w przeglądarce Sante Medical Viewer 3D Pro



**Rysunek 2.4:** Przykład rekonstrukcji wielopłaszczyznowej w przeglądarce Athena DICOM Viewer. Zdjęcie użyte za zgodą Medical Harbourn.



## Edycja danych

- Dodawanie nowych obiektów. Pozwala na rysowanie, dodawanie figur geometrycznych lub tekstu przez lekarza i zapis tych informacji w pliku DICOM (lub nadpis). 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. 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 opinie i ich wymiana. Wiki, strona internetowa w formacie Wikipedii dostępna dla użytkownika.

#### Obrazowanie dwu-wymiarowe

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

## Pseudokolorowanie obrazu

Mamy obraz, którego piksele to n-bitowe liczby, na przykład 16 bitowa liczba całkowita. W takiej postaci wyświetlenie obrazu 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 kontraście stosuje się twór zwany okienkiem. Jest to funkcja, która mapuje n-bitowy obraz na 8-bitowy obraz w skali szarości. 8-bitów, ponieważ monitor RGB jest w stanie wyświetlić 256 odcieni szarości.

### Zwiększanie 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 okienka *center* i długości *width*.

$$x_0 = center - width/2$$

$$x_1 = center + width/2$$

Wyznaczamy parametry  $a$  i  $b$ , prostej przechodzącej przez dwa punkty  $(x_0, y_0)$  i  $(x_1, y_1)$ . Gdzie  $y_0$  jest równe 0, a  $y_1$  jest równe 255. Funkcja „okna” wygląda następująco:

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

gdzie  $v$  to wartość piksela danych obrazu.

Następnie iterujemy przez wszystkie woksele obrazu i używamy na nich funkcji „okna” i otrzymujemy obraz w skali od 0 do 255. Taki obraz w skali można już wyświetlić. Natomiast standard DICOM przewiduje, że obraz można jeszcze wyświetlić w wielokolorowej palecie barw. Przykład takiej palety HotIron w porównaniu do skali szarości można zobaczyć na rysunku. Taka paleta barw nie koniecznie musi mieć 256 odcieni, dlatego lepiej jest zrobić aby „okno”, mapowało na liczbę od 0 do 1, a później paleta mapowała na kolor RGB.

Teraz iterujemy po wszystkich możliwych wartościach wartościach obrazu i wykonujemy takie operacje.

- wyznaczenie wartości okienka.

$$y = a * x + b$$

- $y$  zostaje obcięte do 1.0 lub 0.0 jeżeli wyjdzie poza zakres od 1.0 do 0.0
- pobranie z palety piksel odpowiadający wartości
- wsadzenie piksela do tablicy, tak aby najmniejsza wartości obrazu miała indeks 0 a największy ostatni

Standard DICOM jest odpowiednią społecznością radiologów, radiolafarmaceutów, fizyków medycznych na potrzeby wyimiany danych pomiędzy różnymi systemami komputerów

### 2.4.1 Standard DICOM v3.0

Pierwsze tomografy komputerowe przeżyły swój rozkwit w latach siedemdziesiątych ubiegłego wieku. Obrazu medyczne nie były bezpośrednim wynikiem badania, a jedynie wynikiem obróbki danych pomiarowych przez komputer. Zwyczajne pliki graficzne (jak np. jpg, png, gif), nie nadawały się do zapisu takich obrazów, ponieważ zapisywały obraz w spektrum światła widzialnego w postaci składowych RGB. Każdy producent stosował własny format plików, który nie był upubliczniany.

## 2.4 Format cyfrowych obrazów medycznych

zastosować do poprawy wizualizacji niektórych struktur obrazu. generować powierźnie w postaci wokselów. Reprezentacje powierzchni można również stają się lepiej widoczne. Generowanie powierzchni, dzięki różnym algorytmom można Niewykorzystane szare wartości są wyświetlane jako przezroczyste. Specyficzne struktury nek (np. kości). Struktury obrazu pasujące do wzorców szarych wartości są podświetlone. służy do odwzorowania wartości szaroci obrazów wokseli na wartości krycia typów tkaluminem poprzez obtracanie lub skalowanie. Transter Function(nie znam polskiej nazwy), bezpośrednio wizualizowane jako objętość. Użytkownik może wchodzić w interakcje z wo-kroje są pokazane w osobnym oknie. Renderowanie objętościowym. Podczas tego prze-przeznich, strzałkowych lub czolowych) w modelu objętościowym. Później mogą być lepiej wyświetlane w określonej pozycji. Funkcjonalność *Volume*)), przekroje mogą być lepiej wyświetlane w określonej pozycji. Funkcjonalność mocniejszej na podstawie kierunku pierwotnego. Plasty objętości kostki((ang. *Slice Cube* niektórych struktur. W tym celu należy zapewnić funkcjonalność rekonstrukcji osi po-danych w innych kierunkach (np. strzałkowych lub czolowych), aby poprawić wizualizację wzdluz jednej osi ciała (np. poprzecznej). W wielu przypadkach ważne jest przeglądanie Rekonstrukcja włoma, zwykle dane dotyczące objętości medycznej są gromadzone

### Obrazowanie trój-wymiarowe

w odpowiedni sposób w pliku. notacje(opisy), które były wytworzone przez personel medyczny powinny być zapisywane DICOM zawierają parametry sprzętowe urządzenia (np. ilość pikseli na centymetr). Ad-ma odległości w jednostkach długości na obrazie. Jest to możliwe gdyż nagłówki pliku możliwości rysowania bądź zaznaczania linii lub innych kształtów do analizy i wyznacza-prawiają one czytelność obrazu. Histogram, histogramy wizualizują występowania i rozkład (LUT, (ang. *LookUpTable*)) odwzorowujące szare wartości obrazu na pseudo-kolory, po-skale szaroci, okienkowanie jest opisane w sekcji 7.2. Pseudo-kolorowanie obrazu, tabele miotu wykonującego badanie. Okienkowanie (okna cyfrowe), sposób zamiany danych na-oknie wyświetlacz jako nakładka na obraz. Na przykład aktywna pozycja lub nazwa pod-Warstwa informacyjna, najważniejsza informacja powinna być wizualizowane w DICOM specyficzne dla dostawcy (np. specjalne ustawienie urządzenia rejestrującego). Wyświetlanie rozdzielczości obrazu, badanie (np. identyfikator podmiotu) oraz znaczniki

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.

## 4.6.2 Generowania obrazu monochromatycznego

```
1 Sokar::Pixel yb2Pixel(uint8 y, uint8 b, uint8 r) {
2     qreal red, green, blue;
3     red = green = blue = (255.0 / 219.0) * (y - 16.0);
4     blue += 255.0 / 224 * 1.402 * (r - 128);
5     green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
6     blue += 255.0 / 224 * 1.772 * (b - 128);
7     green -= 255.0 / 224 * 1.772 * (b - 128) * (0.299 / 0.587);
8     blue += 255.0 / 224 * 1.772 * (b - 128);
9     qreal red, green, blue;
10     red = green = blue = (255.0 / 219.0) * (y - 16.0);
11     blue += 255.0 / 224 * 1.402 * (r - 128);
12     green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
13     blue += 255.0 / 224 * 1.772 * (b - 128);
14     qreal red, green, blue;
15     red = green = blue = (255.0 / 219.0) * (y - 16.0);
16     blue += 255.0 / 224 * 1.402 * (r - 128);
17     green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
18     blue += 255.0 / 224 * 1.772 * (b - 128);
19     qreal red, green, blue;
20     red = green = blue = (255.0 / 219.0) * (y - 16.0);
21     blue += 255.0 / 224 * 1.402 * (r - 128);
22     green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
23     blue += 255.0 / 224 * 1.772 * (b - 128);
24     qreal red, green, blue;
25     red = green = blue = (255.0 / 219.0) * (y - 16.0);
26     blue += 255.0 / 224 * 1.402 * (r - 128);
27     green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
28     blue += 255.0 / 224 * 1.772 * (b - 128);
29     qreal red, green, blue;
30     red = green = blue = (255.0 / 219.0) * (y - 16.0);
31     blue += 255.0 / 224 * 1.402 * (r - 128);
32     green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
33     blue += 255.0 / 224 * 1.772 * (b - 128);
34     qreal red, green, blue;
35     red = green = blue = (255.0 / 219.0) * (y - 16.0);
36     blue += 255.0 / 224 * 1.402 * (r - 128);
37     green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
38     blue += 255.0 / 224 * 1.772 * (b - 128);
39     qreal red, green, blue;
40     red = green = blue = (255.0 / 219.0) * (y - 16.0);
41     blue += 255.0 / 224 * 1.402 * (r - 128);
42     green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
43     blue += 255.0 / 224 * 1.772 * (b - 128);
44     qreal red, green, blue;
45     red = green = blue = (255.0 / 219.0) * (y - 16.0);
46     blue += 255.0 / 224 * 1.402 * (r - 128);
47     green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
48     blue += 255.0 / 224 * 1.772 * (b - 128);
49     qreal red, green, blue;
50     red = green = blue = (255.0 / 219.0) * (y - 16.0);
51     blue += 255.0 / 224 * 1.402 * (r - 128);
52     green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
53     blue += 255.0 / 224 * 1.772 * (b - 128);
54     qreal red, green, blue;
55     red = green = blue = (255.0 / 219.0) * (y - 16.0);
56     blue += 255.0 / 224 * 1.402 * (r - 128);
57     green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
58     blue += 255.0 / 224 * 1.772 * (b - 128);
59     qreal red, green, blue;
60     red = green = blue = (255.0 / 219.0) * (y - 16.0);
61     blue += 255.0 / 224 * 1.402 * (r - 128);
62     green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
63     blue += 255.0 / 224 * 1.772 * (b - 128);
64     qreal red, green, blue;
65     red = green = blue = (255.0 / 219.0) * (y - 16.0);
66     blue += 255.0 / 224 * 1.402 * (r - 128);
67     green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
68     blue += 255.0 / 224 * 1.772 * (b - 128);
69     qreal red, green, blue;
70     red = green = blue = (255.0 / 219.0) * (y - 16.0);
71     blue += 255.0 / 224 * 1.402 * (r - 128);
72     green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
73     blue += 255.0 / 224 * 1.772 * (b - 128);
74     qreal red, green, blue;
75     red = green = blue = (255.0 / 219.0) * (y - 16.0);
76     blue += 255.0 / 224 * 1.402 * (r - 128);
77     green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
78     blue += 255.0 / 224 * 1.772 * (b - 128);
79     qreal red, green, blue;
80     red = green = blue = (255.0 / 219.0) * (y - 16.0);
81     blue += 255.0 / 224 * 1.402 * (r - 128);
82     green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
83     blue += 255.0 / 224 * 1.772 * (b - 128);
84     qreal red, green, blue;
85     red = green = blue = (255.0 / 219.0) * (y - 16.0);
86     blue += 255.0 / 224 * 1.402 * (r - 128);
87     green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
88     blue += 255.0 / 224 * 1.772 * (b - 128);
89     qreal red, green, blue;
90     red = green = blue = (255.0 / 219.0) * (y - 16.0);
91     blue += 255.0 / 224 * 1.402 * (r - 128);
92     green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
93     blue += 255.0 / 224 * 1.772 * (b - 128);
94     qreal red, green, blue;
95     red = green = blue = (255.0 / 219.0) * (y - 16.0);
96     blue += 255.0 / 224 * 1.402 * (r - 128);
97     green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
98     blue += 255.0 / 224 * 1.772 * (b - 128);
99     qreal red, green, blue;
100    red = green = blue = (255.0 / 219.0) * (y - 16.0);
101    blue += 255.0 / 224 * 1.402 * (r - 128);
102    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
103    blue += 255.0 / 224 * 1.772 * (b - 128);
104    qreal red, green, blue;
105    red = green = blue = (255.0 / 219.0) * (y - 16.0);
106    blue += 255.0 / 224 * 1.402 * (r - 128);
107    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
108    blue += 255.0 / 224 * 1.772 * (b - 128);
109    qreal red, green, blue;
110    red = green = blue = (255.0 / 219.0) * (y - 16.0);
111    blue += 255.0 / 224 * 1.402 * (r - 128);
112    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
113    blue += 255.0 / 224 * 1.772 * (b - 128);
114    qreal red, green, blue;
115    red = green = blue = (255.0 / 219.0) * (y - 16.0);
116    blue += 255.0 / 224 * 1.402 * (r - 128);
117    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
118    blue += 255.0 / 224 * 1.772 * (b - 128);
119    qreal red, green, blue;
120    red = green = blue = (255.0 / 219.0) * (y - 16.0);
121    blue += 255.0 / 224 * 1.402 * (r - 128);
122    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
123    blue += 255.0 / 224 * 1.772 * (b - 128);
124    qreal red, green, blue;
125    red = green = blue = (255.0 / 219.0) * (y - 16.0);
126    blue += 255.0 / 224 * 1.402 * (r - 128);
127    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
128    blue += 255.0 / 224 * 1.772 * (b - 128);
129    qreal red, green, blue;
130    red = green = blue = (255.0 / 219.0) * (y - 16.0);
131    blue += 255.0 / 224 * 1.402 * (r - 128);
132    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
133    blue += 255.0 / 224 * 1.772 * (b - 128);
134    qreal red, green, blue;
135    red = green = blue = (255.0 / 219.0) * (y - 16.0);
136    blue += 255.0 / 224 * 1.402 * (r - 128);
137    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
138    blue += 255.0 / 224 * 1.772 * (b - 128);
139    qreal red, green, blue;
140    red = green = blue = (255.0 / 219.0) * (y - 16.0);
141    blue += 255.0 / 224 * 1.402 * (r - 128);
142    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
143    blue += 255.0 / 224 * 1.772 * (b - 128);
144    qreal red, green, blue;
145    red = green = blue = (255.0 / 219.0) * (y - 16.0);
146    blue += 255.0 / 224 * 1.402 * (r - 128);
147    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
148    blue += 255.0 / 224 * 1.772 * (b - 128);
149    qreal red, green, blue;
150    red = green = blue = (255.0 / 219.0) * (y - 16.0);
151    blue += 255.0 / 224 * 1.402 * (r - 128);
152    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
153    blue += 255.0 / 224 * 1.772 * (b - 128);
154    qreal red, green, blue;
155    red = green = blue = (255.0 / 219.0) * (y - 16.0);
156    blue += 255.0 / 224 * 1.402 * (r - 128);
157    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
158    blue += 255.0 / 224 * 1.772 * (b - 128);
159    qreal red, green, blue;
160    red = green = blue = (255.0 / 219.0) * (y - 16.0);
161    blue += 255.0 / 224 * 1.402 * (r - 128);
162    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
163    blue += 255.0 / 224 * 1.772 * (b - 128);
164    qreal red, green, blue;
165    red = green = blue = (255.0 / 219.0) * (y - 16.0);
166    blue += 255.0 / 224 * 1.402 * (r - 128);
167    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
168    blue += 255.0 / 224 * 1.772 * (b - 128);
169    qreal red, green, blue;
170    red = green = blue = (255.0 / 219.0) * (y - 16.0);
171    blue += 255.0 / 224 * 1.402 * (r - 128);
172    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
173    blue += 255.0 / 224 * 1.772 * (b - 128);
174    qreal red, green, blue;
175    red = green = blue = (255.0 / 219.0) * (y - 16.0);
176    blue += 255.0 / 224 * 1.402 * (r - 128);
177    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
178    blue += 255.0 / 224 * 1.772 * (b - 128);
179    qreal red, green, blue;
180    red = green = blue = (255.0 / 219.0) * (y - 16.0);
181    blue += 255.0 / 224 * 1.402 * (r - 128);
182    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
183    blue += 255.0 / 224 * 1.772 * (b - 128);
184    qreal red, green, blue;
185    red = green = blue = (255.0 / 219.0) * (y - 16.0);
186    blue += 255.0 / 224 * 1.402 * (r - 128);
187    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
188    blue += 255.0 / 224 * 1.772 * (b - 128);
189    qreal red, green, blue;
190    red = green = blue = (255.0 / 219.0) * (y - 16.0);
191    blue += 255.0 / 224 * 1.402 * (r - 128);
192    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
193    blue += 255.0 / 224 * 1.772 * (b - 128);
194    qreal red, green, blue;
195    red = green = blue = (255.0 / 219.0) * (y - 16.0);
196    blue += 255.0 / 224 * 1.402 * (r - 128);
197    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
198    blue += 255.0 / 224 * 1.772 * (b - 128);
199    qreal red, green, blue;
200    red = green = blue = (255.0 / 219.0) * (y - 16.0);
201    blue += 255.0 / 224 * 1.402 * (r - 128);
202    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
203    blue += 255.0 / 224 * 1.772 * (b - 128);
204    qreal red, green, blue;
205    red = green = blue = (255.0 / 219.0) * (y - 16.0);
206    blue += 255.0 / 224 * 1.402 * (r - 128);
207    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
208    blue += 255.0 / 224 * 1.772 * (b - 128);
209    qreal red, green, blue;
210    red = green = blue = (255.0 / 219.0) * (y - 16.0);
211    blue += 255.0 / 224 * 1.402 * (r - 128);
212    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
213    blue += 255.0 / 224 * 1.772 * (b - 128);
214    qreal red, green, blue;
215    red = green = blue = (255.0 / 219.0) * (y - 16.0);
216    blue += 255.0 / 224 * 1.402 * (r - 128);
217    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
218    blue += 255.0 / 224 * 1.772 * (b - 128);
219    qreal red, green, blue;
220    red = green = blue = (255.0 / 219.0) * (y - 16.0);
221    blue += 255.0 / 224 * 1.402 * (r - 128);
222    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
223    blue += 255.0 / 224 * 1.772 * (b - 128);
224    qreal red, green, blue;
225    red = green = blue = (255.0 / 219.0) * (y - 16.0);
226    blue += 255.0 / 224 * 1.402 * (r - 128);
227    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
228    blue += 255.0 / 224 * 1.772 * (b - 128);
229    qreal red, green, blue;
230    red = green = blue = (255.0 / 219.0) * (y - 16.0);
231    blue += 255.0 / 224 * 1.402 * (r - 128);
232    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
233    blue += 255.0 / 224 * 1.772 * (b - 128);
234    qreal red, green, blue;
235    red = green = blue = (255.0 / 219.0) * (y - 16.0);
236    blue += 255.0 / 224 * 1.402 * (r - 128);
237    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
238    blue += 255.0 / 224 * 1.772 * (b - 128);
239    qreal red, green, blue;
240    red = green = blue = (255.0 / 219.0) * (y - 16.0);
241    blue += 255.0 / 224 * 1.402 * (r - 128);
242    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
243    blue += 255.0 / 224 * 1.772 * (b - 128);
244    qreal red, green, blue;
245    red = green = blue = (255.0 / 219.0) * (y - 16.0);
246    blue += 255.0 / 224 * 1.402 * (r - 128);
247    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
248    blue += 255.0 / 224 * 1.772 * (b - 128);
249    qreal red, green, blue;
250    red = green = blue = (255.0 / 219.0) * (y - 16.0);
251    blue += 255.0 / 224 * 1.402 * (r - 128);
252    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
253    blue += 255.0 / 224 * 1.772 * (b - 128);
254    qreal red, green, blue;
255    red = green = blue = (255.0 / 219.0) * (y - 16.0);
256    blue += 255.0 / 224 * 1.402 * (r - 128);
257    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
258    blue += 255.0 / 224 * 1.772 * (b - 128);
259    qreal red, green, blue;
260    red = green = blue = (255.0 / 219.0) * (y - 16.0);
261    blue += 255.0 / 224 * 1.402 * (r - 128);
262    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
263    blue += 255.0 / 224 * 1.772 * (b - 128);
264    qreal red, green, blue;
265    red = green = blue = (255.0 / 219.0) * (y - 16.0);
266    blue += 255.0 / 224 * 1.402 * (r - 128);
267    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
268    blue += 255.0 / 224 * 1.772 * (b - 128);
269    qreal red, green, blue;
270    red = green = blue = (255.0 / 219.0) * (y - 16.0);
271    blue += 255.0 / 224 * 1.402 * (r - 128);
272    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
273    blue += 255.0 / 224 * 1.772 * (b - 128);
274    qreal red, green, blue;
275    red = green = blue = (255.0 / 219.0) * (y - 16.0);
276    blue += 255.0 / 224 * 1.402 * (r - 128);
277    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
278    blue += 255.0 / 224 * 1.772 * (b - 128);
279    qreal red, green, blue;
280    red = green = blue = (255.0 / 219.0) * (y - 16.0);
281    blue += 255.0 / 224 * 1.402 * (r - 128);
282    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
283    blue += 255.0 / 224 * 1.772 * (b - 128);
284    qreal red, green, blue;
285    red = green = blue = (255.0 / 219.0) * (y - 16.0);
286    blue += 255.0 / 224 * 1.402 * (r - 128);
287    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
288    blue += 255.0 / 224 * 1.772 * (b - 128);
289    qreal red, green, blue;
290    red = green = blue = (255.0 / 219.0) * (y - 16.0);
291    blue += 255.0 / 224 * 1.402 * (r - 128);
292    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
293    blue += 255.0 / 224 * 1.772 * (b - 128);
294    qreal red, green, blue;
295    red = green = blue = (255.0 / 219.0) * (y - 16.0);
296    blue += 255.0 / 224 * 1.402 * (r - 128);
297    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
298    blue += 255.0 / 224 * 1.772 * (b - 128);
299    qreal red, green, blue;
300    red = green = blue = (255.0 / 219.0) * (y - 16.0);
301    blue += 255.0 / 224 * 1.402 * (r - 128);
302    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
303    blue += 255.0 / 224 * 1.772 * (b - 128);
304    qreal red, green, blue;
305    red = green = blue = (255.0 / 219.0) * (y - 16.0);
306    blue += 255.0 / 224 * 1.402 * (r - 128);
307    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
308    blue += 255.0 / 224 * 1.772 * (b - 128);
309    qreal red, green, blue;
310    red = green = blue = (255.0 / 219.0) * (y - 16.0);
311    blue += 255.0 / 224 * 1.402 * (r - 128);
312    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
313    blue += 255.0 / 224 * 1.772 * (b - 128);
314    qreal red, green, blue;
315    red = green = blue = (255.0 / 219.0) * (y - 16.0);
316    blue += 255.0 / 224 * 1.402 * (r - 128);
317    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
318    blue += 255.0 / 224 * 1.772 * (b - 128);
319    qreal red, green, blue;
320    red = green = blue = (255.0 / 219.0) * (y - 16.0);
321    blue += 255.0 / 224 * 1.402 * (r - 128);
322    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
323    blue += 255.0 / 224 * 1.772 * (b - 128);
324    qreal red, green, blue;
325    red = green = blue = (255.0 / 219.0) * (y - 16.0);
326    blue += 255.0 / 224 * 1.402 * (r - 128);
327    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
328    blue += 255.0 / 224 * 1.772 * (b - 128);
329    qreal red, green, blue;
330    red = green = blue = (255.0 / 219.0) * (y - 16.0);
331    blue += 255.0 / 224 * 1.402 * (r - 128);
332    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
333    blue += 255.0 / 224 * 1.772 * (b - 128);
334    qreal red, green, blue;
335    red = green = blue = (255.0 / 219.0) * (y - 16.0);
336    blue += 255.0 / 224 * 1.402 * (r - 128);
337    green -= 255.0 / 224 * 1.402 * (r - 128) * (0.114 / 0.587);
338    blue += 255.0 / 224 * 1.772 * (b - 128);
339    qreal red, green, blue;
340    red = green = blue = (255.0 / 219.0) * (y - 16.0);
341    blue += 25
```



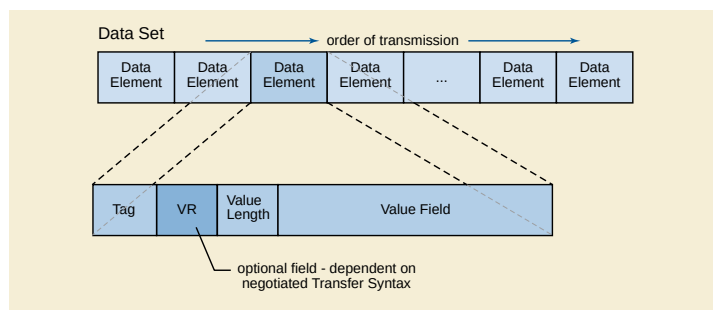
wymi, przeglądarek obrazów, stacji do przetwarzania i analizowania obrazów medycznych.

Standard DICOM wersji trzeciej to standard definiujący ujednolicony 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 wciąż 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 ułożone 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](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 czterech 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 znacznika przyjmą wartości odpowiednio wartość  $0010_{16}$  i  $0010_{16}$  to oznacza, że jest to znacznik  $\text{Dicom}_{\text{Tag}}^{\text{PatientName}}$  (0x0010, 0x0010), czyli zawiera w sobie parametr zawierający nazwę pacjenta.

Dokładne omówienie znaczników znajduje się w sekcji 2.4.2.

```
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  $\text{imgDimX} * \text{imgDimY}$  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**.
- **pixmap** obiekt obrazu do wyświetlania, klasy `Qt::QPixmap`.  
Obiektów klasy `Qt::QImage` nie da się wyświetlić, nie jest on przystosowany do wyświetlania. Natomiast klasa `Qt::QPixmap` to reprezentacja obrazu dostosowana do wyświetlania ekranie, która może być używana jako urządzenie do malowania w bibliotece Qt.
- **iconPixmap** obiekt obrazu ikonu, klasy `Qt::QPixmap`, docelowo powinien mieć 128 pikseli na 128 pikseli.

Generowanie obrazu jest robione przez czysto wirtualną funkcję `Sokar::DicomScene::generatePixmap()`. Po wywołaniu funkcji obiekt **targetBuffer** powinien zawierać obraz wygenerowany z obecnymi parametrami. Funkcja zwraca również wartość logiczną, który informuje nas czy **targetBuffer** rzeczywiście został zmieniony. Następnie obiekt **pixmap** jest na nowo generowany na bazie **qImage**.

Całe odświeżanie obrazu jest implementowane w funkcji `Sokar::DicomScene::reloadPixmap()`. Funkcja wywołuje `Sokar::DicomScene::generatePixmap()` i odświeża **pixmapItem** kiedy zajdzie taka potrzeba

Generowanie poszczególnych typów obrazów jest wyjaśnione poniżej.

### Obraz monochromatyczny

Obraz monochromatyczny to obraz w odcieniach szarości, od białego do czarnego lub od czarnego do białego. Generowanie takiego obrazu odbyła się poprzez pseudokolorowanie. Cały proces jest wyjaśniony w sekcji 4.6.2.

### RGB

Obrazów zapisanych w RGB nie trzeba w żaden sposób obrabiać, dane już są prawie gotowe do wyświetlenia. Należy je odpowiednio posortować, jeżeli zachodzi taka potrzeba. Sposób posortowania wartości w pliku określa znacznik  $\text{Dicom}_{\text{Tag}}^{\text{Planar Configuration}}$  (0x00028, 0x0006). Może on przyjąć dwie następujące wartości:

W środkowej części programu znajduje się obiekt z zakładcami, szczegółowo opisany w sekcji 4.5.5.

### Menu programu

W górnej części okna programu znajduje się menu, obiekt klasy *Sokar::QMenuBar*. Struktura Menu programu:

- File

– Open — otwiera okienko wyboru plików, implementowane przez *Qt::QFileDialog::getOpenFileName()*, następnie wczytuje plik

– Open Recent — program zapisuje ostatnio wczytane pliki i pozwala ja ich ponownie wczytać z tego menu

– Export as — zapisanie obrazu w formacie JPEG, BMP, GIF lub PNG. Zapisywanie jest zaimplementowane przez funkcję *Qt::QImage::save()*, która umożliwia zapisanie obrazu do pliku.

– Exit — wyjście z aplikacji

- Help

– 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*, muteks do zablokowania podczas generowania obrazu, aby parametry obrazu nie mogły być zmieniane podczas jego generowania.

• **imgDimX** zmienia typ *uint*, oznacza szerokość obrazu w pikselach.

• **imgDimY** zmienia typ *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.

- **Value Representation**, w skrócie **VR** – to dwa bajty w postaci tekstu, informujące o formacie w jakim parametr został zapisany.
- Dokładne omówienie **VR**-ów znajduje się w dalszej części sekcji.

- **Value Length**, w skrócie **VL** — 32-bitowa lub 16-bitowa liczba nieoznaczona, która informuje o długości pola danych (**Value Field**).

Wartość **VL** zwykle jest liczbą parzystą. Standard DICOM zakłada, że wszystkie dane powinny być dopełniane do parzystej liczby bajtów.

- **Value Field** (opcjonalne) — pole z parametrem o długości VL.

### Znacznik

**Tag** to unikalny znacznik pozwalający określać czego dotyczą dane zapisane w elemencie danych. Znacznik jest złożony z dwóch liczb: numeru grupy i numeru elementu. Obie liczby to 16-bitowe liczby całkowite zapisywane w postaci heksadecymalnej.

Istnieją dwa rodzaje znaczników: publiczne o parzystym numerze grupy i prywatne o nieparzystym numerze. Pierwsza grupa jest definiowana przez standard DICOM, zawiera ona podstawowe znaczniki. Publiczne znaczniki dzielę się na obowiązkowe, opcjonalne i warunkowe. Są określone przy definicji obiektów informacyjnych. Natomiast druga grupa to znaczniki, pozostawione do dyspozycji producentom sprzętu, tak by mogli zapisywać dodatkowe informacje, które nie zostały przewidziane w standardzie DICOM. Takie dodatkowe informacje zapisywane ogromnej liczby informacji standardyzowanej jak i informacji niestandardowej w sposób bezkonfliktowy oraz z możliwością odczytania danych przez aplikacje niepowiązane z producentem sprzętu.

Obecna odsłona DICOM definiuje znaczenie ponad 400 publicznych znaczników oraz określa jakie VR powinny mieć. Oto kilka przykładów:

- **Tag Patient Name** (0x0010, 0x0010) — nazwa pacjenta, czyli znacznik, który zawsze musi się pojawić. Może być pusty w przypadku kiedy pacjent jest bezimienny
- **Tag Patient ID** (0x0010, 0x0020) — id pacjenta, unikalny identyfikator pacjenta, najczęściej jest to numer HIS(Hospital Information System)
- **Tag Patient BirthDate** (0x0010, 0x0030) — data urodzenia pacjenta
- **Tag Patient Sex** (0x0010, 0x0040) — płeć pacjenta
- **Tag Patient Age** (0x0010, 0x1010) — wiek pacjenta w czasie badania
- **Tag Study Description** (0x0008, 0x1030) — opis badania, pole wypełniane przez technika lub lekarza
- **Tag Series Description** (0x0008, 0x103E) — opis serii, pole wypełniane przez technika lub lekarza

- **Tag Instance Number** (0x0020, 0x0013) — numer instancji ramki, używany w przypadku kiedy z jednego badania zostało utworzonych kilka plików DICOM

- $\text{Dicom}_{\text{Tag}}^{\text{Modality}}$  (0x0008, 0x0060) — modalność określająca rodzaj techniki diagnostycznej
- $\text{Dicom}_{\text{Tag}}^{\text{Study Date}}$  (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 to zawsze wynosi 4 bajty. Pierwsze trzy bajty to liczba całkowita zapisana za pomocą tekstu. Czwarty bajt to znaku 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  $\text{Dicom}_{\text{Tag}}^{\text{FrameIncrementPointer}}$  (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 zmiennoprzecinkowych 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 spacje na końcu.

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 czasowi 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 miniatur

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.

## Sposoby uzyskania nowych plików

Otworzenie nowego pliku może odbyć się z następujących źródeł: obiektu drzewa ze strukturą plików w systemie (opisanego w 4.5.4), menu programu (opisanego w 4.5.6), lub poprzez przeciągnięcie i upuszczenie. Z dwóch pierwszych można wczytać tylko po jednym pliku, natomiast trzecim sposobem można wczytać zarówno jedno jak i wiele plików. Wysyłanie prośby odbywa się za pomocą dwóch funkcji: *Sokar::DicomTabs::addDicomFile()* i *Sokar::DicomTabs::addDicomFiles()*. Każda z tych funkcji ma dwa przeciążenia, jedno z parametrem ścieżki a drugie z wczytanym plikiem, dodatkowo funkcje te są slotami.

## Wczytywanie plików

Po dostarczeniu ścieżek do obiektu, pliki zostają wczytane za pomocą funkcji *gdcmm::ImageReader*. W przypadku błędu proces wczytywania się kończy. Po wczytaniu wszystkich plików zostaje utworzony obiekt kolekcji ramek obrazu lub kolekcji plików DICOM za pomocą funkcji *Sokar::DicomFileSet::create()*, opisaną w sekcji 4.5.3.

## 4.5.6 Okno główne programu

Główne okno programu jest implementowane przez *Sokar::MainWindow*. Jest wywoływane od razu po uruchomieniu programu.

Zawiera w sobie 4 elementy: menu, drzewo ze strukturą plików, obiekt z zakładkami oraz w dolnej części okna sugestie, aby nie używać programu w celach medycznych.

## Drzewo katalogów i zakładki

W lewej części okna znajduje się element listy, implementowany przez *Sokar::FileTree*, zawiera on w sobie model drzewa plików systemu, który z kolei jest implementowany przez klasę *Qt::QFileSystemModel*. Po wybraniu pliku ścieżka jest przesyłana do obiektu z zakładkami.



– Hospital Data — Dane szpitala

Akcja: [HospitalData](#).

Po otrzymaniu sygnału obiekt klasy *Sokar::HospitalDataIndicator* znajdujący

się na scenie powinien pokazać lub ukryć się w zależności od stanu pozycji.

– Image Acquisition — Dane akwizycji

Akcja: [ModaliltyData](#).

Po otrzymaniu sygnału obiekt klasy *Sokar::ModalityIndicator* znajdujący się

na scenie powinien pokazać lub ukryć się w zależności od stanu pozycji.

- Tagi (5)

Akcja: [OpendataSet](#).

Kliknięcie tego przycisku wyśle prośbę o utworzenie okna ze zbiorem elementów

danych pliku obrazu, który jest obecnie wyświetlany na scenie.

### Miejsce na scene

Na środku znajduje kontrolka klasy *Sokar::DicomGraphics*, dziedziczącej po *Qt::GraphicsView*, która służy do wyświetlania sceny.

### Pasek filmu

Pasek filmu znajduje się w dolnej części zakłádki i jest implementowany prze klasę

*Sokar::MovieBar*. Ma dostęp do sekwencji scen i ukrywa swoją obecność przed użytkow-

nikiem, 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 prządką 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 wy-

ślanie sygnału *Sokar::SceneSequence::stepBackward()* do sekwencji. Wciśnięcie drugiego

przycisku (9) powoduje włączenie lub wyłączenie upływu sekwencji. Wciśnięcie trzeciego

przycisku (10) powoduje zatrzymanie upływu sekwencji i wysłanie sygnału *Sokar::Scene-*

*Sequence::stepForward()* do sekwencji.

Pasek (11) pokazujący postęp sekwencji jest obiektem klasy *Qt::QSlider*. Odswieżanie

paszka jest wrażliwe na sygnał *Sokar::SceneSequence::stepped()* od sekwencji.

Elementy po prawej stronie definiują parametry trybu filmowego. Prządką (12) jest

elementem do wprowadzania liczby zmniejszającej klasę *Qt::QDoubleSpinBox*. Im

większa wartość liczby, tym kładki filmu są dłużej wyświetlane. Drugi (13) przycisk po-

zwala zmienić sposób przemieszczania. Trzeci (14) przycisk wymusza tryb jednego okna dla

wszystkich klatek filmu. Jeżeli mamy załadowanych wiele obrazów tego samego badania,

to nie konieczne muszą mieć to samo okno. Dodatkowo ten tryb pozwala wprowadzić

jednolite okienko dla wszystkich klatek po zmianie parametrów tego okienka na jednej

klatce. Czwarty (15) i ostatni przycisk służy do użycia jednej macierzy transformaty na

### 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::Movie-*

*Mode*. Obiekt ten zapisuje wskazuje do obecnie wyświetlanej sceny, a także czy powinno

- IS — Integer String — liczba całkowita

Długość jednej liczby powinna maksymalnie wynosić 12 bajtów. Dostępne znaki to „0”-„9”, „+”, „-”, „.”. Biblioteka QT posiada wbudowany konwerter liczb całkowitych, dlatego mój konwerter używa konwertera z QT.

Przykład: „426” oznacza liczbę 426.

- PN — Person Name — nazwa osoby

Ponieważ pacjenta, bądź obiekt badany można nazwać w sposób dowolny i odbie-  
gający od polskiego standardu nazewnictwa, standard DICOM nie przewiduje roz-  
dzielania poszczególnych składowych nazwy na oznaczone fragmenty: „Person Name”  
dzieli nazwę na podane fragmenty, rozdzielony znakiem „.” (94 znak kodu ASCII):

– family name complex — nazwisko, np. Smolik

– given name complex — imię, np. Adam

– middle name — środkowe imię, brak odpowiednika w polskim nazewnictwie

– name prefix — prefiks przed imieniem, np: mgr. inż.

– name suffix — sufixs po imieniu, brak odpowiednika

Długość jednego fragmentu powinna maksymalnie wynosić 64 znaki. W przypadku

mnijszej ilości segmentów, mamy założyć, że są puste.

Przykład: „prof. dr. hab. inż. Waldemar Smolik pracownik ZEJIM” był by zapisany w

sposób następujący: „Smolik\_Waldemar\_prof. dr. hab. inż.\_pracownik\_ZEJIM”

- SS — Signed Short — 16 bitowa liczba całkowita bez znaku

- US — Unsigned Short — 16 bitowa liczba całkowita ze znakiem

- UT — Unlimited Text — tekst o nieograniczonej długości.

Zwyczajy tekst o długości maksymalnie 2<sup>32</sup> – 2 bajtów.

## 2.4.3 DICOMDIR

W przypadku większych instytucji pojawia się problem indeksowania plików i ich prze-

szukiwania. Wyszukiwanie konkretnego badania lub pliku w folderze, w którym znajduje się

kilkaset plików poprzez wczytanie każdego pliku do pamięci i analiza jego danych nie

jest rozwiązaniem optymalnym. Dlatego standard DICOM definiuje również pliki typu

DICOMDIR, który jest plikiem indeksującym pliki DICOM w folderze. Pozwala to na

efektywne przeglądanie wielu serii badań bez wczytywania plików badań.

## 2.4.4 Inne formaty zapisu

W tomografii komputerowej wynikiem rekonstrukcji jest macierz liczb opisujących

rozkład przetrzynny współczynnika osłabiania promieniowania. Ze względu na aspekty

prawne i medyczne, niezwykle istotną rzeczą jest zapis oryginalnych danych numerycz-

nych. Ze tego powodu producenci sprzętu wprowadzają własne formaty plików cyfrowych.

W plikach tych oprócz numerycznych danych obrazowych zapisane są parametry warun-

ków akwizycji itp.

## Rozdział 3

# Biblioteki i narzędzia

### 3.1 CMake

CMake to wieloplatformowe narzędzie do automatycznego zarządzania procesem kompilacji 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 kompilacji na 3 platformy CMake jest idealnym rozwiązaniem. Dodatkowo w pracy tej starano się wybrać biblioteki, które kompilują się za pomocą CMake.

### 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 kompilacją.

#### 3.2.1 Wymowa

Według autorów, Qt powinno się czytać jak angielskie słowo „cute”, po polsku „kiut”. Jednakże społeczność programistów nie jest co do tego zgodna. Ankiety zrobione na dwóch popularnych serwisach internetowych o tematyce programistycznej, pokazują, że najbardziej popularną wymową jest „Q.T.”, po polsku „ku te”.

Odnosiłki do przytoczonych ankiet:

- <https://ubuntuforums.org/showthread.php?t=1605716>
- <https://www.qtcentre.org/threads/11347-How-do-you-pronounce-Qt>

Stan: **Zoom**. Oznacza, że ruch myszki powinien skalować obraz kiedy jest wciśnięty 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.
- Rotate Left — Obróć w lewo  
Akcja: **RotateLeft90**.  
Po otrzymaniu sygnału obraz na scenie powinien obrócić się o 90 stopni w lewo.
- Flip Horizontal — Odbij lustrzanie poziomo  
Akcja: **FlipHorizontal**.  
Po otrzymaniu sygnału obraz na scenie powinien odbić się lustrzanie poziomo.
- Flip Vertical — Odbij lustrzanie pionowo  
Akcja: **FlipVertical**.  
Po otrzymaniu sygnału obraz na scenie powinien odbić się lustrzanie pionowo.
- Clear Transformation — Wyczyść przekształcenia obrotu  
Akcja: **ClearRotate**.  
Po otrzymaniu sygnału obraz na scenie powinien wyczyścić transformację obrotu.

#### • Informacje na obrazie (5)

Ten element potrafi wyłączyć wyświetlanie niektórych elementów na scenie. Kliknięcie go odznacza lub zaznacza wszystkie pozycje w menu kontekstowym. Wszystkie pozycje są pozycjami odznaczanymi.

Menu rozwijalne:

- Patient Data — Dane pacjenta  
Akcja: **PatientData**.  
Po otrzymaniu sygnału obiekt klasy *Sokar::PatientDataIndicator* znajdujący się na scenie powinien pokazać lub ukryć się w zależności od stanu pozycji.



- `quint64` — liczba całkowita, 64 bitowa, bez znaku
- `qreal` — największa dostępna liczba zmiennoprzecinkowa

### 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 wewnątrz klas są oznaczone następująco:

`Qt::QObject::connect()`

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

Biblioteka Qt implementuje 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 stacku 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 destruktor dzieci i w ten sposób całe drzewo obiektów zostanie zniszczone.

Mechanizm ten pozwala nam tworzyć nowe obiekty na stacku i nie martwić się o ich późniejsze sprzątnięcie. Jest to o tyle efektywne, że nie trzeba dla każdego obiektu tworzyć odrębnego wskaźnika lub wektora wskaźników w deklaracji klasy, a dzięki temu można mieć czystszy i czytelniejszy kod źródłowy. Przykładowe użycie:

```
1 int main() {
2
3     // Tworzymy obiekt przycisku
4     auto *quit = new QPushButton("Quit");
5     // Tworzymy obiekt okna
6     auto *window = new QWidget();
7
8     // Przypisujemy rodzica przyciskowi
9     quit->setParent(window);
10
11     ...
12 }
```

- `DicomTag` 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.
- `DicomTag` Cine Rate (0x0018, 0x0040) — zawiera ilość klatek wyświetlanych na sekundę, każdemu krokowi jest przypisywana wartość do niej odwrotna.

W przypadku braku znacznika lub gdy zostaje wskazany znacznik nieznanym, 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 wczytanych plików DICOM. Na początku pliki są sortowane na podstawie liczby zawartej w elemencie o znaczniku `DicomTag` 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 je rozdzielić na serie i uporządkować w odpowiedniej kolejności. Unikalny identyfikator serii jest zawarty w elemencie danych znaczniku `DicomTag` Series Instance UID (0x0020, 0x000E). Kolejności obrazów w serii to liczba zawarta w elemencie danych o znaczniku `DicomTag` Instance Number (0x0020, 0x0013).

Segregacja odbywa się za pomocą funkcji `Sokar::DicomFileSet::create()`. Do funkcji jest przesyłany wektor z wczytanymi plikami DICOM, następnie dzieli ona pliki na zbiory zawierające zdjęcia tej samej serii, tworzy obiekty zbiorów plików DICOM, ostatecznie zwraca ona wektor z gotowymi obiektami zbiorów plików DICOM. Sortowanie plików DICOM według ich kolejności odbywa się za pomocą funkcji `std::sort` wewnątrz konstruktora klasy `Sokar::DicomFileSet`, który nie jest publiczny.

### 4.5.4 Zakładka

Każda zakładka z obrazem lub obrazami jest implementowana przez klasę `Sokar::DicomView`.

Interfejs graficzny `Sokar::DicomView` wyświetla następujące elementy:

- pasek narzędzi znajdujący się na górze — implementowany za pomocą klasy `Sokar::DicomToolBar`, opisany w sekcji 4.5.4
- miejsce na scene z obrazem DICOM na środku — implementowany za pomocą klasy `Sokar::DicomGraphics`, opisany w sekcji 4.5.4
- suwak filmu w dolnej części — implementowany za pomocą klasy `Sokar::MovieBar`, opisany w sekcji 4.5.4
- podgląd miniatur obrazów w prawej części — implementowany za pomocą klasy `Sokar::FrameChooser`, opisany w sekcji 4.5.4

Dodatkowo posiada obiekt kolekcji scen opisaną w sekcji 4.5.3.

- $\text{DicomFrameTime}(0 \times 0018, 0 \times 1063)$  — element z tym znacznikiem zawiera czas trwania jednej ramki w milisekundach, każdemu krokowi jest przypisywana ta wartość trwania

Zawiera on wskaźnik do elementu o zadanym znaczniku. Została zaimplementowana obsługa poniższych znaczników:

Kolejność sekwencji scen jest taka sama jak kolejność ramek. Następnie jest tworzenie scen ile jest ramek. tania ramki może być zapisany w różnych znacznikach. To, w którym znaczniku został zapisany, informuje element o znaczniku  $\text{D}_{\text{Diagram}}^{\text{Diagram}}$  Frame Increment Pointer (0x0028, 0x0009).

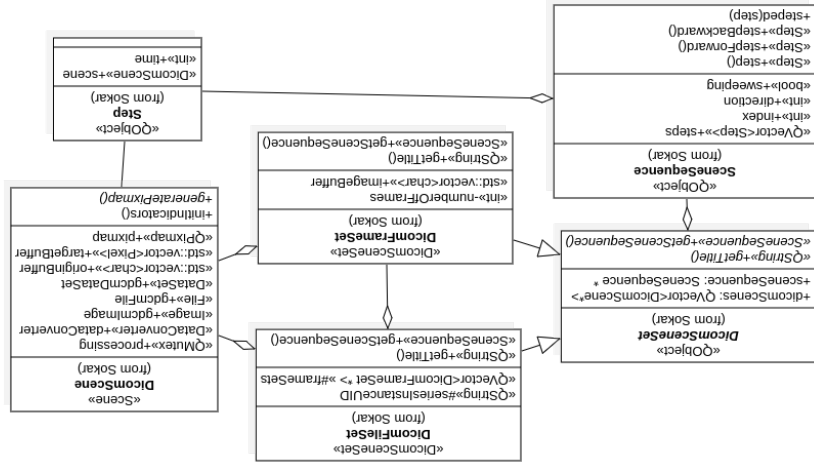
Zbiory ramek są implementowane przez *Sokarr::DicomFrameSet* i są tworzone z jednego wczytanego pliku DICOM. Klasa tworzy obiekt konwertera i pobiera liczbę ramek w obrazie. Tworzy jeden błądź na wszystkie ramki obrazów, a następnie dzieli go na ilość

## Kolekcja ramek DICOM

Wszystkie powyższe funkcje są zwracane slotami dla sygnałów oraz emitując sygnał *Sokar::SceneSequence::stepped()*.

- *Sokar::SceneSequence::stepForward()* — krok do przodu, zwiększając indeks tym samym wykonując krok w stronę końca sekwencji
- *Sokar::SceneSequence::stepBackward()* — krok do tyłu, zmniejszając indeks tym samym wykonując krok w stronę początku sekwencji
- *Sokar::SceneSequence::step()* — wykonuje krok w tył lub przód w zależności od kierunku sekwencji

**Rysunek 4.8:** Diagram klas UML dziedziczenia klasy *Sokar::DicomSceneSet*.



```

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         W przeciwnym razie sloty można zmieniac. */
15     public slots:
16         void setValue(int value) {
17             if (value != m_value) {
18                 m_value = value;
19             }
20         }
21     /* Podczas wywoływania sygnału należy

```

## Przykładowa klasa dziedzicząca po QObject

Pełna dokumentacja na temat sygnałów i slotów znajduje się na oficjalnej stronie Qt pod adresem <https://doc.qt.io/qt-5/signalsandslots.html>

```

1  /* Tworzymy dwa obiekty klasy Counter (definicja w następnym sekcji) */
2  Counter a, b;
3
4  /* Łączymy sygnał Counter::valueChanged obiektu "a",
5  do slotu Counter::setAValue obiektu "b" */
6  QObject::connect(&a, &Counter::valueChanged,
7                  &b, &Counter::setAValue);
8
9  /* Ustawiamy wartość licznika obiektu "a" na 12 */
10 a.setAValue(12);
11
12 /* W czasie ustawiania został wysłany sygnał z "a" do "b", więc:
13 a.value() == 12 b.value() == 12 */
14
15 /* Ustawiamy wartość licznika obiektu "b" na 48 */
16 b.setAValue(48);
17
18 /* Sygnał Counter::valueChanged obiektu "b" nie jest podłączony do
19 zadanego slotu, więc:
20 a.value() == 12 b.value() == 48 */

```

tworzenie programowania zdarzeniowego.  
Przykład użycia sygnałów do propagacji zdarzenia.

System sygnałów i słotów jest implementacją programowania zdarzeniowego. W odrozmieniu do sygnałów pozwalających się w C, sygnały w Qt są wstawiane przez argumenty definiowane przez programistę. Sygnały i słoty są implementowane przez funkcje definiowane w deklaracji klasy. Sygnał obiektu jest łączony do słotu obiektu dynamicznie w czasie działania programu. Do jednego sygnału można podłączyć wiele słotów, jak i do jednego słotu można wprowadzić wiele sygnałów. Taką implementację umożliwia to

## Sygnaly i sloty

```
12 // W tym momencie przycisk wraz z oknem zostaje usunięte
13 delete m;
14 }
```

```

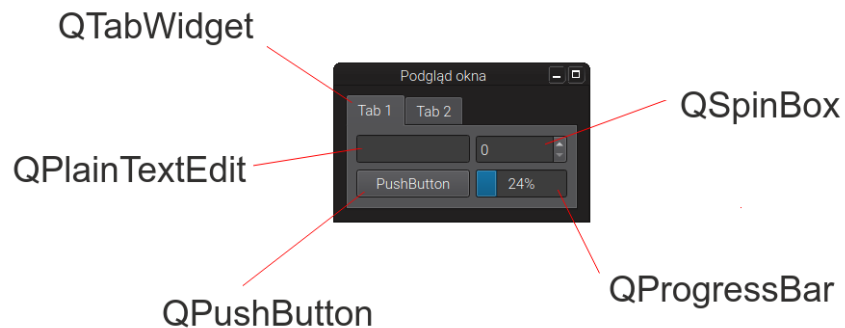
21         poprzedzić to makrem "emit". */
22         emit valueChanged(value);
23     }
24 }
25
26 /* Sygnały powinny być poprzedzone makrem "signals".
27    Wszystkie sygnały są publiczne. */
28 signals:
29     void valueChanged(int newValue);
30
31 private:
32     int m_value;
33 };

```

### 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.



Rysunek 3.1: Przykładowe okienko programu w Qt. Zdjęcie własne.

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::QAbstractButton*, która dziedziczy po *Qt::QWidget*. Obsługa zdarzenia wciśnięcia przycisku jest przez obsługę sygnału *Qt::QAbstractButton::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 zwy-

### 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 przechwycenie ruchów myszki, gdy wciśnięty jest lewy przycisk myszy.

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

### 4.5.3 Kolekcje scen

Abstrakcyjna klasa *Sokar::DicomSceneSet* implementuje wektor scen za pomocą klasy *Qt::QVector*. Jest to obiekt, który przechowuje sceny i tworzy sekwencje scen, która jest rzeczywistym ułożeniem ramek obrazów. Są dwie implementacje kolekcji scen: kolekcja plików i kolekcja ramek z jednego pliku. Diagram klas UML znajduje się na rysunku 4.8.

#### Sekwencja scen

Sekwencja scen implementuje strukturę danych informującą o przejściach pomiędzy scenami poprzez klasę *Sokar::SceneSequence*. Sekwencja to wektor zawierający kroki z dodatkowymi informacjami o stanie sekwencji:

- indeks, w którym obecnie znajduje się sekwencja
- kierunek sekwencji — sekwencja może iść w stronę początku lub końca
- rodzaj przemiatania — wartość logiczna informująca w jaki sposób ma zachować się, gdy sekwencja dojdzie do końca, lub początku

Po dojściu do końca sekwencja skoczy do pierwszego elementu lub może zmienić kierunek i zacząć iść do tyłu.

Kroki implementowane przez klasę *Sokar::Step* zawierają następujące informacje: wskaźnik do sceny oraz czas trwania sceny.

Sekwencja ma wbudowane funkcje zapewniające przesuwanie się po indeksie na wektorze:



- "A" — anterior — przód pacjenta
- "P" — posterior — tył pacjenta
- "F" — feet — część dolna
- "H" — head — część górna

Pelny opis implementacji algorytmu wyznaczania stron znajduje się w sekcji 4.6.4.

#### Podziałka

Jest implementowana przez *Sokar::PixelSpacingIndicator*. Obiekt wyświetla podziałkę informującą o rzeczywistych rozmiarach obiektu na obrazie. Pojawia się na dole i po prawej stronie sceny, gdy znacznik  $\text{DicomTag}_{\text{PixelSpacing}}$  (0x0028, 0x0030) jest obecny. Wygląd podziałki można zaobserwować na rysunku 4.13.

Podziałka dostosowuje swoją wielkość do obecnej sceny, jak i do innych elementów na scenie. Wartości wyświetlane biorą pod uwagę transformację skali i rotacji obrazu.

#### Dodatkowe informacje o modalności

Są implementowane przez *Sokar::ModalityIndicator*. Obiekt wyświetla informacje o akwizycji obrazu. Dane różnią się w zależności od modalności obrazu. Domyślnie zawierają następujące linie:

- bla bla bla
- bla bla bla
- bla bla bla
- bla bla bla

W przypadku następujących modalności zawierają również następujące informacje:

#### Generowanie obrazów z danych

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

### 3.3.1 Uzasadnienie wyboru

Znalezienie dobrej biblioteki do obsługi jest trudne, ponieważ jest ich bardzo dużo, a ich liczba wciąż rośnie. Powstał portal internetowy do ich indeksowania o nazwie „I DO IMAGING”, dostępny pod adresem <https://idoimaging.com/programs>. Biblioteka, której poszukiwano w tej pracy powinna:

- współpracować z językiem C++
- mieć licencję pozwalającą jej używać w potrzebnym zakresie
- darmowa, najlepiej otwarto źródłowa
- aktywnie rozwijana — znaczna większość bibliotek charakteryzowała się tym, że była porzucona i ostateńa była wprowadzona x lat temu, a proces jej rozwoju trwał od 2 do 5 miesięcy
- dostępna na Linux’a, MacOS i Microsoft Windows

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

## 3.3 GDCM

### 3.2.7 Oddzielenie od platformy

Biblioteka standardowa  
Własne wektory  
Własne wątki

- *Qt::QProgressBar* — implementuje prządek, czyli kontrolę przystosowaną do wprowadzania liczb przez użytkownika. Posiada dwa dodatkowe przyciski pozwalające w łatwy sposób zwiększyć lub zmniejszyć zawartość. Przykład można zobaczyć na rysunku 3.1.
- *Qt::QProgressDialog* — implementuje pasek postępu w dwóch wersjach poziomej i pionowej. Dziedziczy bezpośrednio po klasie *Qt::QWidget*. Przykład poziomego paska można zobaczyć na przykładowym rysunku 3.1.

- *Qt::QPlainTextEdit* — implementuje pole umożliwiające wprowadzanie tekstu przez użytkownika. Dziedziczy po klasie *Qt::QAbstractScrollArea*, które dziedziczy po *Qt::QFrame*, z kolei ta po *Qt::QWidget*. Przykład można zobaczyć na przykładowym rysunku 3.1.
- kłymi obiektami dziedziczącymi po *Qt::QWidget*. Przykład można zobaczyć na przykładowym rysunku 3.1.

### 3.3.2 Opis

Przetłumaczony opis biblioteki z oficjalnej strony prezentuje się następująco: Grasroots 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 badaczowi 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 kompilacji używa CMake. Dzięki temu w całym programie jest używany język C++ wraz z CMake, co ułatwia zarządzanie procesem kompilacji 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 używanie.

### 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 wewnątrz 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 w stanie wygenerować obiekt obrazu
- *gdcm::Image* — obiekt obrazu ułatwiający pobieranie informacji
- *gdcm::File* — obiekt pliku DICOM

- Opis wykonany przez instytucję lub klasyfikację badania (komponentu)  
Tekst brany z  $\text{Dicom}_{\text{Tag}}^{\text{Study Description}}$  (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  $\text{Dicom}_{\text{Tag}}^{\text{Series Description}}$  (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 0  
HIS/123456  
born 1996-07-16, 19 years  
Kregoslup 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  $\text{Dicom}_{\text{Tag}}^{\text{Institutional Department Name}}$  (0x0008, 0x1040) i wyświetlany bez ingerencji.
- Producent wyposażenia wraz z modelem urządzenia  
Tekst jest obierany z  $\text{Dicom}_{\text{Tag}}^{\text{Manufacturer}}$  (0x0008, 0x0070) i  $\text{Dicom}_{\text{Tag}}^{\text{Manufacturer Model Name}}$  (0x0008, 0x1070), oddzielony spacją i wyświetlany bez ingerencji.
- Nazwisko lekarza wykonującego badanie  
Tekst jest obierany z  $\text{Dicom}_{\text{Tag}}^{\text{Referring Physician Name}}$  (0x0008, 0x0090) i wyświetlany bez ingerencji.
- Nazwisko operatora wspierającego badanie  
Tekst jest obierany z  $\text{Dicom}_{\text{Tag}}^{\text{Operators Name}}$  (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.

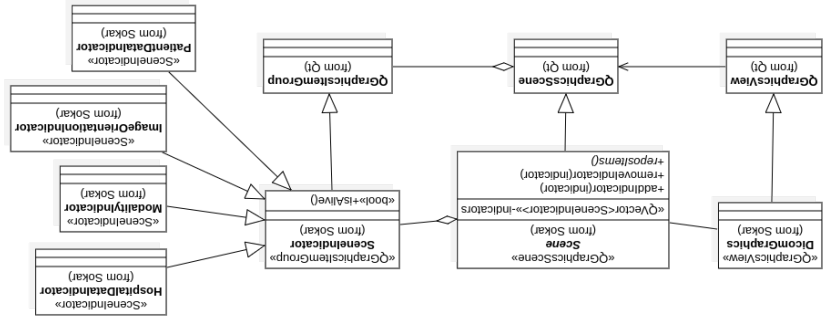
Każda z sześciu możliwych liter oznacza kierunek oraz zwrot w jakim jest ułożony pacjent:

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



**Informacje wyświetlane na scenie**

Wszystkie elementy wyświetlające dane z pliku DICOM dziedziczą po klasie *Sokar::SceneIndicator*. Diagram klas UML znajduje się na rysunku 4.7.



**Rysunek 4.7:** Diagram klas UML dziedziczenia klasy *Sokar::SceneIndicator*.

Domyślne obiekty wyświetlające informacje (tytuły punktów to nazwy klas):

## Dane pacjenta

Dane pacjenta są implementowane przez *Sokar::PatientDataIndicator* i pojawiają się zawsze na scenie w lewym górnym rogu. Zawierają następujące linie:

- Nazwa pacjenta oraz płeć  
Nazwa pacjenta znajduje się w *Tag* *Dicom* Patient Name (0x0010, 0x0010) o VR:PN. Płeć, zapisana jest w *Tag* *Dicom* Patient Sex (0x0010, 0x0040) i może mieć następujące wartości:

- „M” — oznacza mężczyznę, wyświetlana jako O
- „F” — oznacza kobietę, wyświetlana jako O
- „0” — oznacza inną płeć i nie jest wyświetlana

Przykład: „Adam Jędrzejowski 0”.

- Identyfikator pacjenta

Unikalny identyfikator pacjenta ze znacznika *Tag* *Dicom* Patient ID (0x0010, 0x0020) wyświetlany jest w takiej formie, w jakiej jest zapisany. W praktyce najczęściej jest to numer z systemu używanego w danym szpitalu, rzadziej numer PESEL.

Przykład: „HIS/000000”.

- Data urodzenia oraz wiek pacjenta w trakcie badania

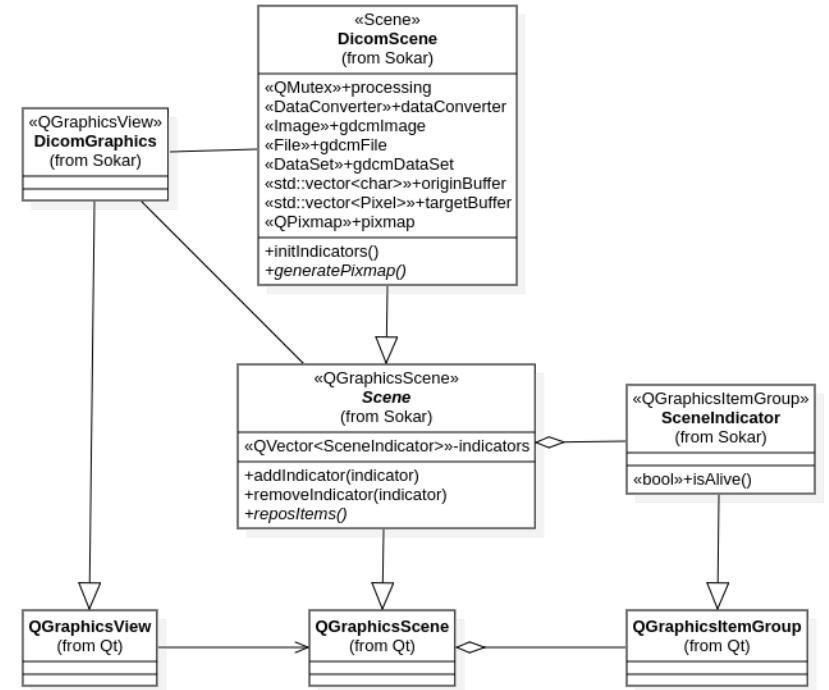
Data urodzenia znajduje się w *Tag* *Dicom* Patient Birth Date (0x0010, 0x0030) i jest zamieniana na format „YYYY-MM-DD”. Dodatkowo, jeżeli tag *Tag* *Dicom* Patient Age (0x0010, 0x010) jest obecny, wyświetlany jest także wiek pacjenta w czasie badania.

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

## 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        TagWindowWidth(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->GetDataElement(tag);
37        /* Pobieramy 16-bitowego inta */
38        quint16 center = ele.GetByteValue()->GetPointer();
39    }
40
41    if (dataset->FindDataElement(TagWindowWidth)){
42        const DataElement& ele = dataset->GetDataElement(tag);
43        quint16 width = ele.GetByteValue()->GetPointer();
44    }
45
46 }
```



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



## 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 władcy 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

- *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-bitową liczbę całkowitą ze znakiem (`qint16`).

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

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

Oprócz powyższych funkcji jest jeszcze kilka innych funkcji pobocznych oraz kilka aliasów. Ogólne zasady konwersji, które się dotyczą 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 to obiektem jednej ramki obrazu i jest odpowiedzialna za pośrednie wygenerowanie obrazu oraz jego wyświetlenie na ekranie. Implementowana jest ona przez klasę *Sokar::DicomScene*, dziedzicząca po *Sokar::Scene*, natomiast *Sokar::Scene* dziedziczy po *Qt::QGraphicsScene*. Diagram klas UML znajduje się na rysunku 4.5

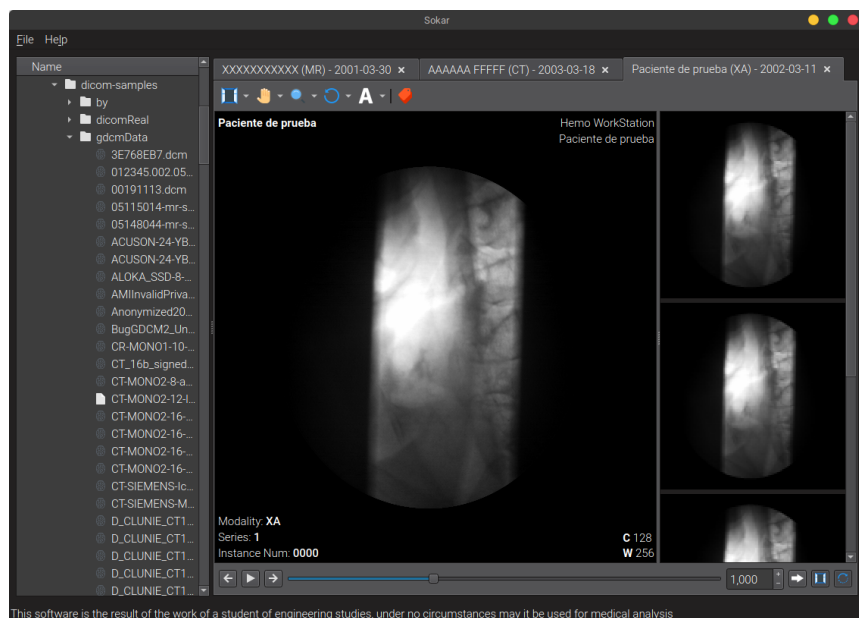
#### Wyświetlanie sceny

Qt zapewnia własny silnik graficzny, który pozwala na łatwą wizualizację przedmiotów, z obsługą obrotu 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



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



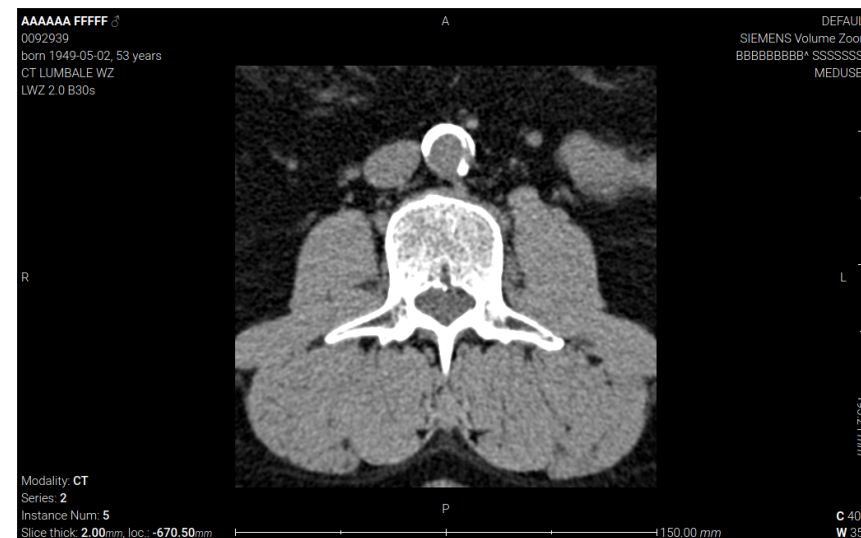
**Rysunek 4.2:** Okno przeglądarki z wczytanymi kilkoma obrazami. Zdjęcie własne.

Obiekt wewnątrz zakładek odpowiada za wyświetlanie wszystkich elementów umożliwiających interakcje 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 wczytany tylko jeden obraz.

Scena to obiekt wyświetlający i generujący obraz na ekranie. Dodatkowo na scenie znajdują 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

Przykładowa scena z obrazem monochromatycznym znajduje się na rysunku 4.3.



**Rysunek 4.3:** Przykładowa scena z obrazem monochromatycznym. Zdjęcie własne.

Możliwość wyświetlania animacji pojawia się wtedy, gdy w jednej zakładce będzie znajdowało się więcej niż jedna ramka obrazu. Można to osiągnąć wczytując wiele obrazów z tej samej serii lub wczytać obraz posiadający wiele ramek. Wówczas pod sceną pojawia się pasek, umożliwiający sterowanie animacją, a po prawej stronie obiekt z ikonami poszczególnych ramek obrazu. Dokładny opis przycisków i ich funkcji znajduje się w sekcji .

Pełna struktura menu programu znajdującego się na górze jest opisana w sekcji 4.5.6.

## 4.4 Projekt struktury obiektowej programu

W tej sekcji jest wyjaśniona ogólna struktura programu, z pominięciem dokładnych opisów poszczególnych elementów. Ich szczegółowy opis znajduje się w następnych sekcjach.

Obiekt okna, klasy *Sokar::MainWindow* posiada 3 elementy: menu (klasy *Qt::QMenuBar*), drzewa plików (klasy *Sokar::FileTree*), obiekt zakładek (klasy *Sokar::DicomTabs*). Zakładki obiektu zakładek są implementowane przez klasę *Sokar::DicomView*. Obiekt zakładki posiada abstrakcyjną kolekcję scen, implementowaną przez *Sokar::DicomSceneSet*. Kolekcja scen odpowiada za przechowywanie obrazów i scen, obiektów klasy *Sokar::DicomScene*. Sceny nie posiadają bezpośredniego dostępu do pliku, a jedynie wskaźniki do odpowiednich miejsc w pamięci gdzie obrazy są przechowywane. Ogólny diagram klas znajduje się na rysunku 4.4.