Aplikacja VTK 3D

Dokumentacja - Informatyka Medyczna
Michał Maksoń, Anna Banaszak

1. O aplikacji

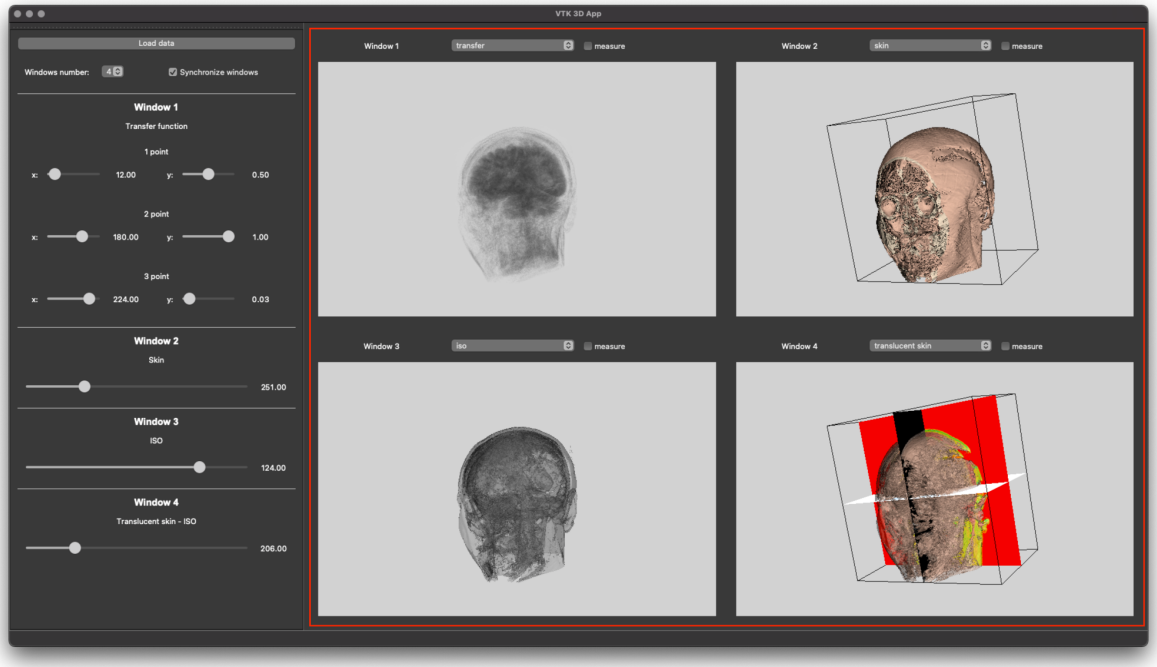
Aplikacja została wykonana jako aplikacja desktopowa przy wykorzystaniu PyQt5. Klasy należące do PyQt można poznać po charakterystycznej nazwie zaczynającej się od Q.

1.1. Vtk3dApp

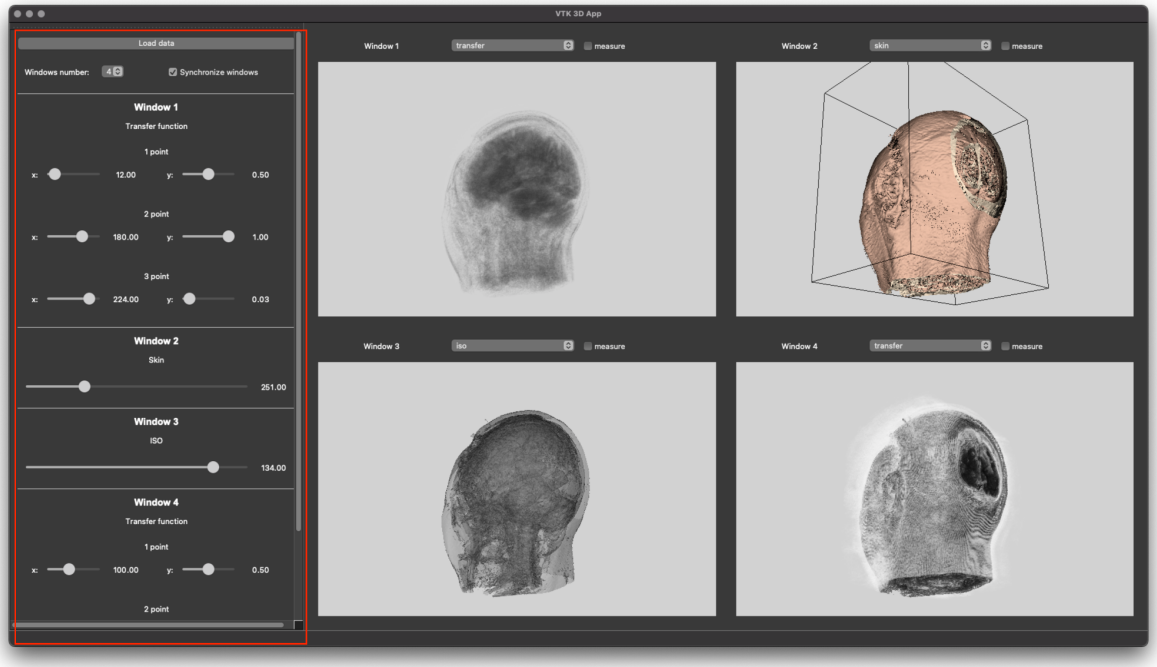
Główną klasą aplikacji jest `Vtk3dApp`, odpowiada ona za uruchomienie aplikacji i jest jej głównym oknem, co otrzymuje poprzez dziedziczenie z `QMainWindow`. Uruchomienie aplikacji odbywa się poprzez stworzenie obiektu tej klasy i wywołanie na nim metody `show()`.
To w tej klasie znajduje się logika odpowiedzialna za wygląd aplikacji oraz rozłożenie poszczególnych komponentów aplikacji.

W skład komponentów rozmieszczonych na głównym oknie można zaliczyć:

- podokna z wyświetlaną wizualizacją (subwindow)



- toolbar z przyciskiem do ładowania danych, wyboru liczby okien (1, 2, 4, 6, 8), opcją synchronizacji okien oraz sliderami obsługującymi poszczególne funkcjonalności w subwindow

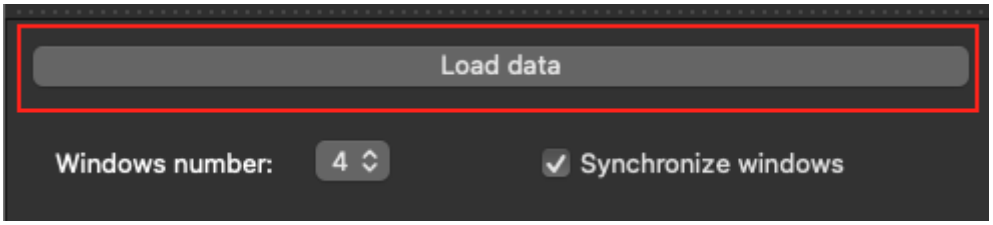


subwindow, stworzone jako `QGridLayout` umieszczone w oknie `QFrame` ustawiane są jako "CentralWidget", czyli główny widget aplikacji. W przypadku subwindow konieczna jest obsługa inicjalizacji początkowych danych oraz reakcji na zmianę danych. Implementacja znajduje się w metodach:
`Vtk3dApp.MainWindow.init_subwindows`
`Vtk3dApp.MainWindow.re_init_subwindows`

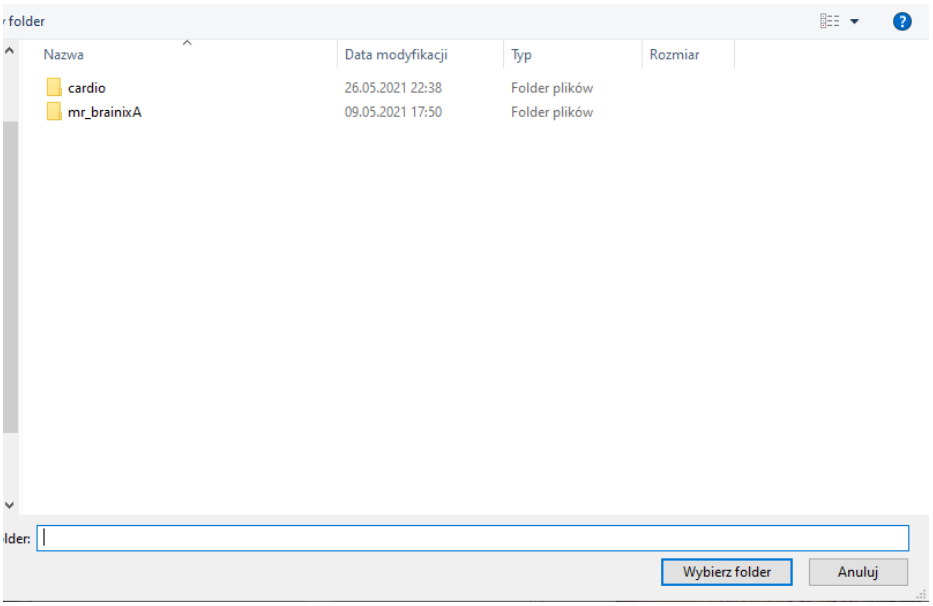
Toolbar natomiast jest gotową funkcjonalnością głównego okna aplikacji PyQt. Logika odpowiedzialna za obsługę tool bara znajduje się w metodach:
`Vtk3dApp.MainWindow.create_tool_bar`
`Vtk3dApp.MainWindow.refresh_tool_bar`
`Vtk3dApp.MainWindow.remove_tool_bar`

`refresh_tool_bar` – powinien być wywołany przy każdej zmianie akcji przypisanej do subwindow lub zmiany liczby subwindow aby wyrenderować odpowiednie slidery odpowiadające nowej akcji.

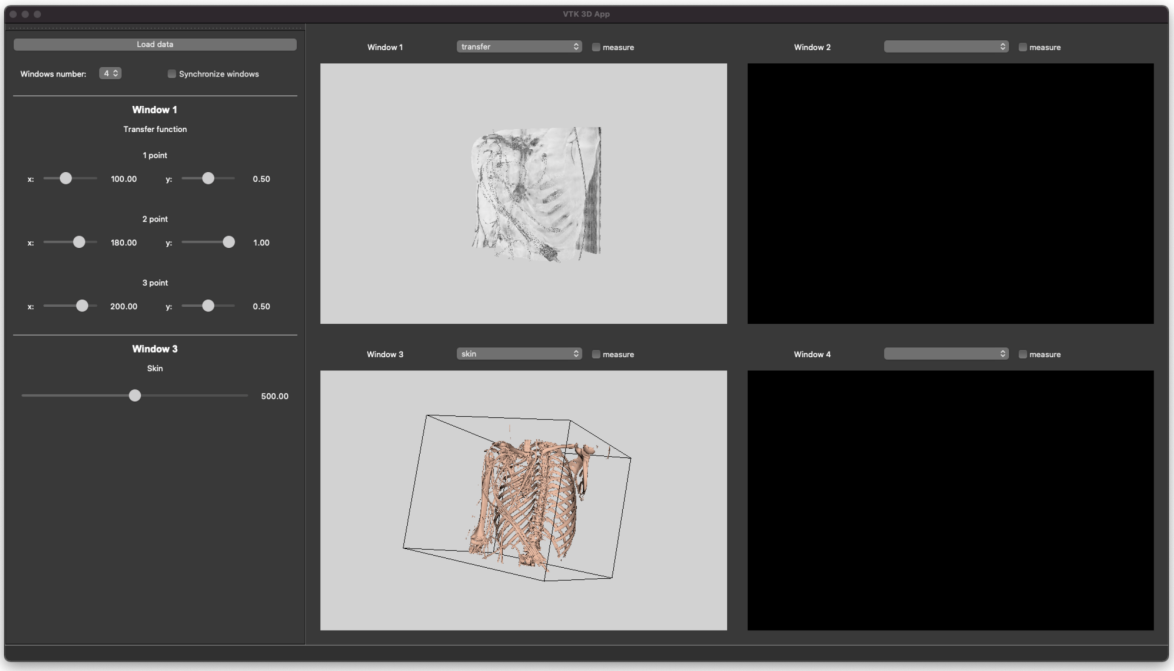
Ładowanie danych do aplikacji odbywa się poprzez wykorzystanie gotowej funkcjonalności PyQt o nazwie `QFileDialog`. Jest ona uruchamiana jako callback dla przycisku w metodzie:
`Vtk3dApp.MainWindow.open_file_dialog`



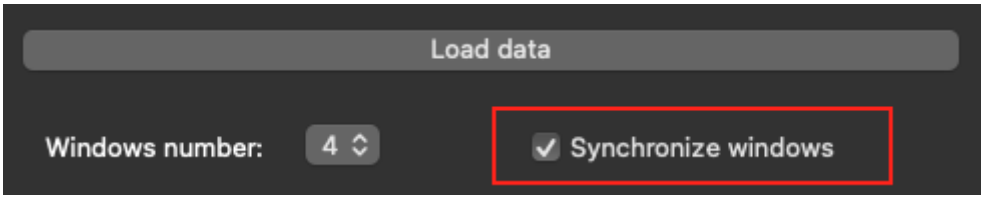
Po naciśnięciu przycisku pojawia się modal, w którym należy wybrać folder przechowujący serie DICOMowe.



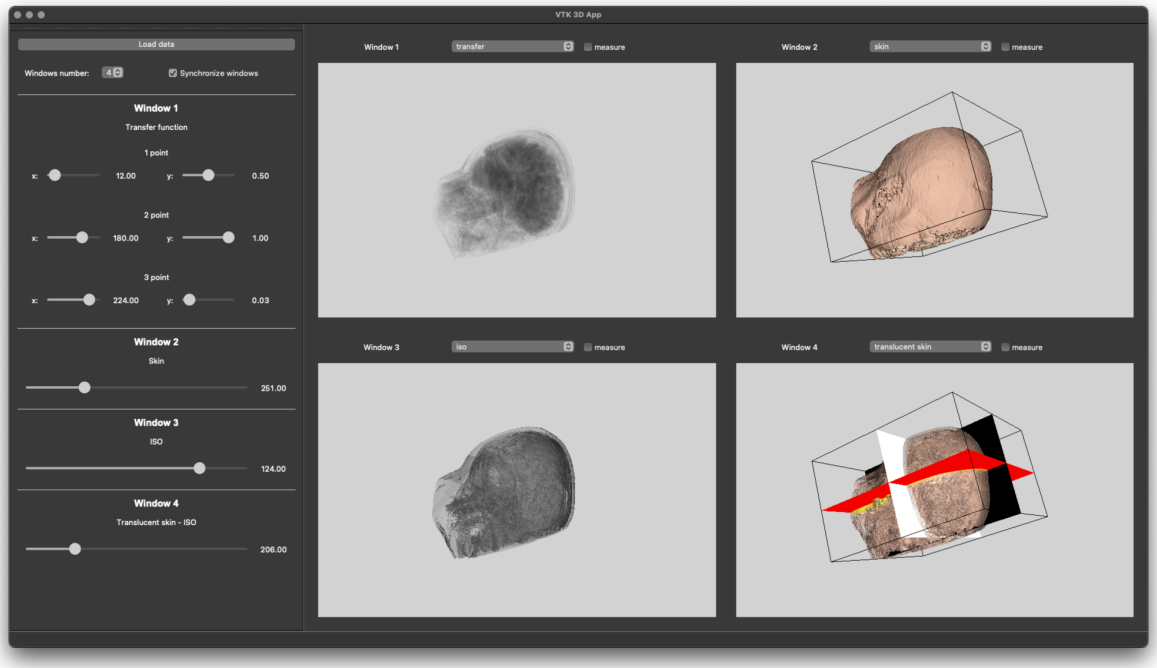
Po zmianie danych wszystkie subwindow powinny wrócić do stanu oczekiwania na wybranie funkcjonalności. To samo dotyczy znajdującego się po lewej stronie toolbaru.



Synchronizacja subwindow za pomocą checkboxu w toolbarze działa na zasadzie wzorca Observer i wysyłaniu notyfikacji o obrocie obiektu. Wykonane zostało to przy użyciu observera udostępnionego przez Vtk.
Implementacja znajduje się w metodzie
`Vtk3dApp.MainWindow.on_checkbox_change`



Zaznaczenie checkboxa skutkuje synchronizacją i ustawieniem identycznej perspektywy na każdym z okien.



2. vtk_utils

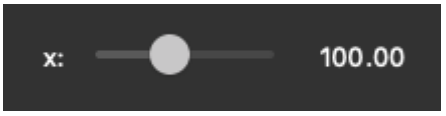
Moduł podzielony na mniejsze - dedykowane odpowiednim akcjom. Każdy z nich zawiera funkcje implementujące pojedyncze działania na bibliotece VTK.

3. Widgety

W skład zaimplementowanych przez nas widgetów wchodzą: Slider, SubWindow oraz ToolBarWidgets, ToolBar a także HWidgets.

3.1. Slider

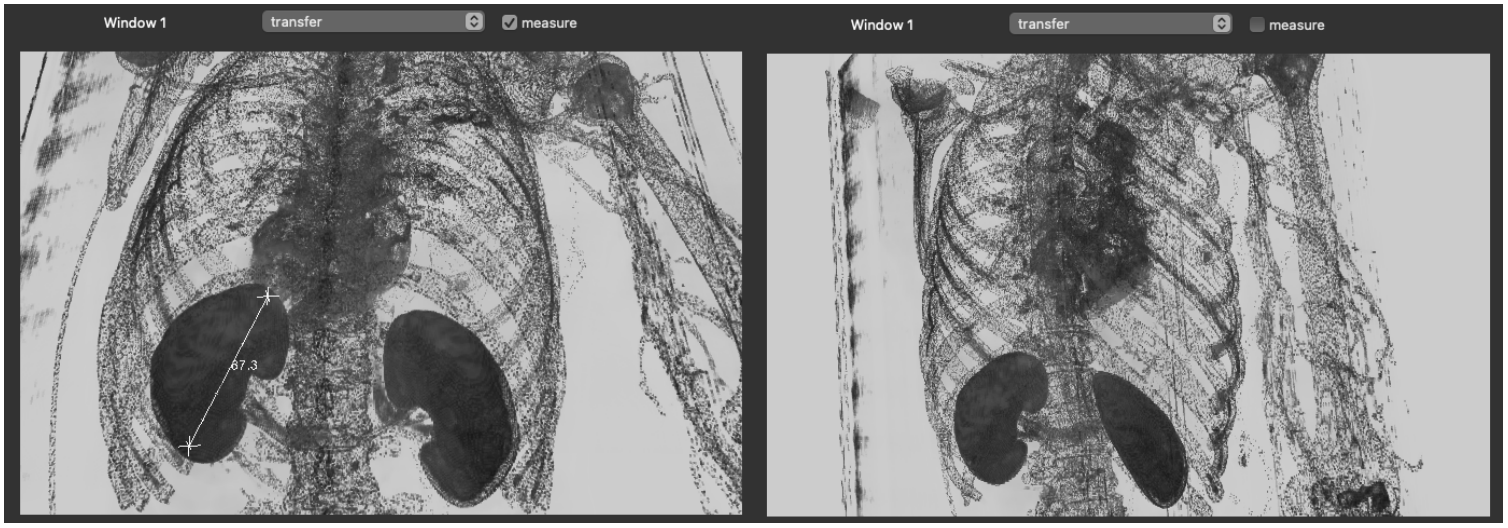
Do implementacji Slidera wykorzystaliśmy obiekt `QSlider`. Callback przypisany do zdarzenia upuszczenia slidera ustawiany jest przy użyciu funkcji:
`slider.valueChanged` i `slider.sliderReleased`.
`slider.valueChanged` – służy do ciągłego monitorowania wartości w labelce po prawej stronie.
`slider.sliderReleased` - służy zaktualizowania odpowiedniej wartości na renderowanym obrazie.
Zmienna `scale` służy skalowaniu wartości ustawionej na sliderze (`QSlider` nie pozwala na ustawienie zakresu wartości np 0-1 i niezbędne jest ustawienie przykładowo zakresu 0-100 i `scale` 0.01)
Możliwe jest także ustawienie labelki po lewej stronie slidera, która domyślnie nie występuje.



3.2. SubWindow

W tym module zdefiniowane są dostępne opcje wizualizacji danych umieszczone w słowniku `actions`. Kluczem jest nazwa pod jaką akcja będzie występować w aplikacji a wartością referencja klasy z implementacją.

Obiekt SubWindow zawiera:
`ToolBarWidgets` – kolekcję widgetów do interakcji w wyświetlanym obrazem,
`combo` – combobox to wyboru akcji przypisanej do subwindow,
`checkbox` – checkbox do możliwości włączenia pomiaru odległości między dwoma punktami,
`vtk_widget` czyli `QVTKRenderWindowInteractor` – obiekt udostępniający interaktor, który pozwala na komunikację VTK <-> PyQt.



Aby umożliwić wyświetlanie wybranej wizualizacji skojarzonej z akcjąj należy wybrać i zainicjalizować odpowiedni obiekt Action. Renderer obiektu Action powinien zostać ustawiony jako Renderer w Render Window danego subwindow (`self.vtk_widget.GetRenderWindow().AddRenderer(self.action.renderer)`).

3.3. ToolBarWidgets

Mowa tutaj o widgetach dedykowanym konkretnym akcjom uruchomionym wewnątrz subwindow. Jest to zwykły obiekt zawierający kolekcję widgetów PyQt, w szczególności `QLabel` i `QFrame` oraz odpowiednie slidery pochodzące z obiektu akcji. Każdorazowo przy zmianie akcji należy wywołać `ToolBarWidgets.set_up_action`.

Toolbar służy do wybrania funkcjonalności jaka ma być reprezentowana poprzez dane Subwindow. Udostępnia również funkcjonalność pomiaru - “measure”.

3.4. ToolBar

Obiekt implementujący LeftToolBar w MainWindow. Do jego inicjalizacji niezbędne są callbacki do widgetów, które są w nim umieszczone oraz lista subwindow.

3.5. HWidgets

Obiekt dziedziczący po `QWidget` służący do horyzontalnej kompozycji kilku widgetów.

4. Actions

Actions to obiekty odpowiadające za reprezentację funkcjonalności medycznych. Każdy z aktualnie obecnych obiektów Action być inicjalizowany zmienną `path` wskazującą na lokalizację danych, `iren` - interaktorem pozwalającym na komunikację z interfejsem użytkownika oraz `measurement_on` informującą o stanie checkboxa measurement w danym subwindow. Obiekt Action powinien implementować sposób wizualizacji danych oraz posiadać pola `renderer` - niezbędny do renderowania obrazu po stronie interfejsu oraz `widgets`, czyli kolekcję widgetów do umieszczenia w toolbarze.

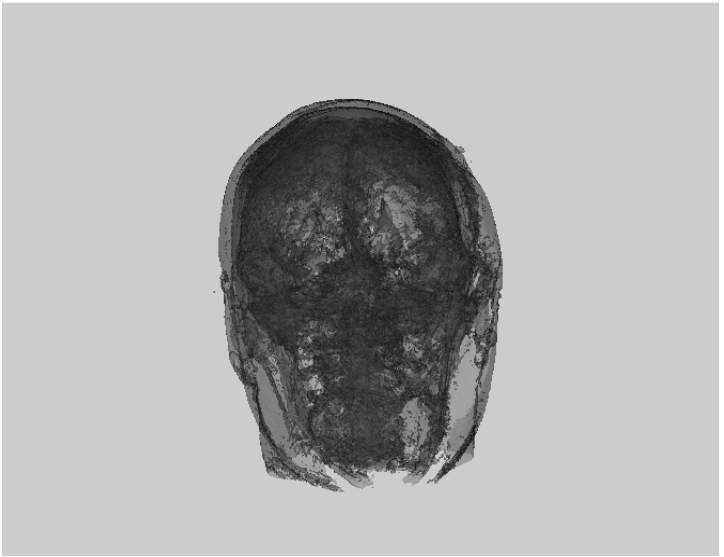
4.1. Iso

Wizualizacja obrazu za pomocą izopowierzchni.

Działanie bazuje na należącym do VTK aktorze.
Aktor do działania potrzebuje:

- `vtkContourFilter`
- `vtkPolyDataMapper`

Następnie należy połączyć aktora z rendererem, na przykład poprzez metodę `get_renderer`



4.2. transfer_fun

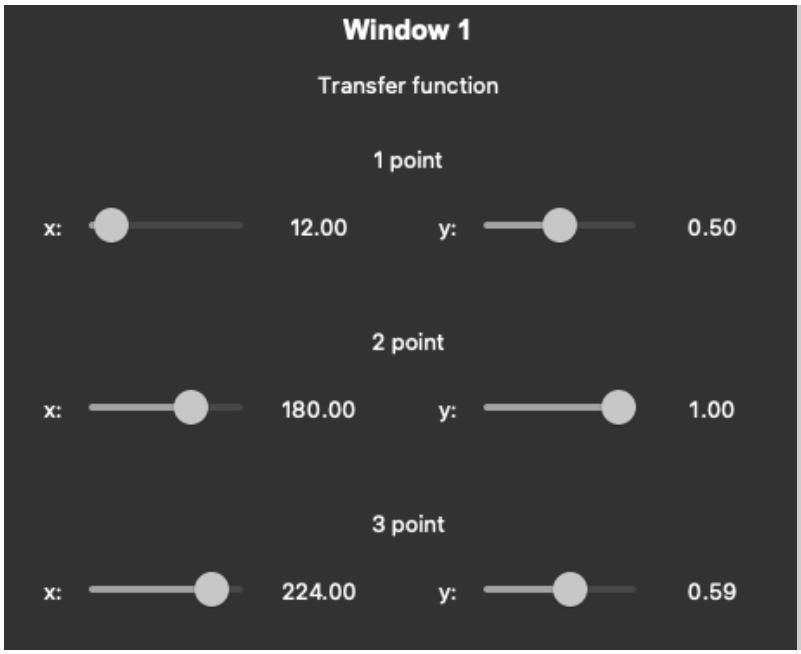
Implementacja funkcji transferu bazuje na aktorze `vtkVolume`.

Aktor operuje na piecewise function: `vtkPiecewiseFunction`
oraz `vtkSmartVolumeMapper`

4.3. transfer_mult_point

Bogatszy odpowiednik funkcji transferu:

Umożliwia konfigurację 3 punktów i osiąganych w nich wartości przez co umożliwia dokładniejsze odwzorowanie zachowania funkcji transferu.



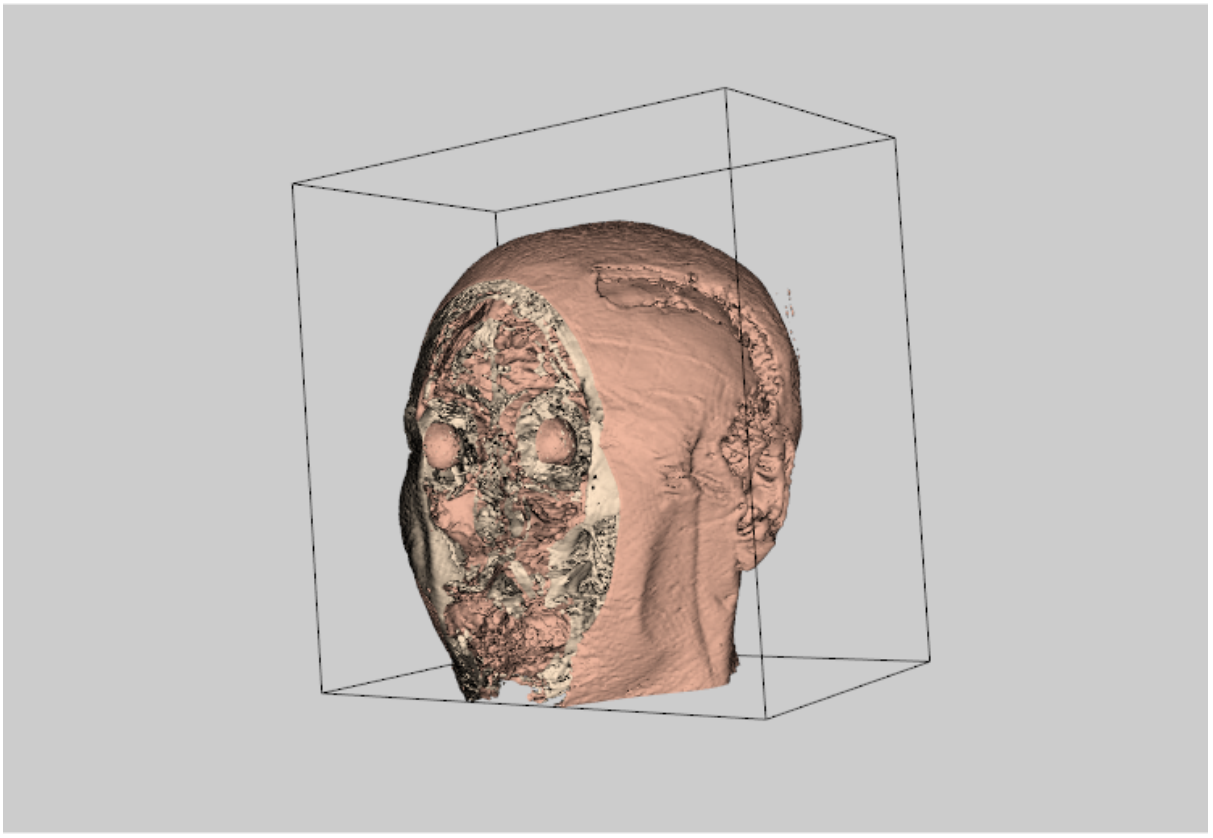
4.4. skin_cover

Funkcja służąca do obliczania obrysu skóry wybranych serii DICOM. Do działania wymagani są dwaj aktorzy:

- aktor odpowiedzialny za skórę(`skin_actor`)
- aktor odpowiedzialny za kontury (`outline_actor`)

Obaj aktorzy muszą zostać podłączeni do renderera, służy do tego funkcja.`get_renderer_with_multiple_actors`
Do wydzielenia skóry służy funkcjonalność VTK:
`vtkFlyingEdges3D`
Analogicznie jak w poprzednich przykładach potrzebny jest `vtkPolyDataMapper`
działający na otrzymanych danych.

Do wydzielenia konturów używane jest:
`vtkOutlineFilter`
którego rezultat przyjmuje również
`vtkPolyDataMapper`
Na bazie tych danych działa aktor do konturów będący instancją `vtkActor`



4.5. skin_display

Funkcjonalność polegająca na prześwietlaniu zdjęcia i rzutowaniu go na trzy płaszczyzny. Płaszczyzny tworzone są w połowie danego wymiaru serii DICOMowej. Jako baze obliczeniową traktowana jest zdefiniowane paleta kolorów, poprzez którą definiujemy interesujące nas wartości.`vtkNamedColors`
Do poprawnego działania wymagani są aktorzy:

- aktor do wykrycia konturów
- aktor do wykrycia skóry
- aktor do wykrycia kości
- 3 aktorów służących do tworzenia odpowiednich płaszczyzn.

Aktorzy do wykrywania skóry oraz do wykrywania konturów zostali już omówieni w poprzednich funkcjonalnościach.
Jedyna różnica polega na konieczności usuwania wykrywanych obiektów zamiast wyświetlania ich.
Służy ku temu `vtkStripper`
Aktor odpowiedzialny za wykrywanie kości działa na zasadzie identycznej do aktora wykrywającego skórę. Różni się jedynie konfiguracją tj. szukanymi wartościami oraz kolorem wykrywanych obiektów. VTK klasyfikuje kości jako kolor `'Ivory'`.

Aktorzy tworzący płaszczyny operują na zdefiniowanych `'lookup_table'` trzymających wartości konfigurujące płaszczyzny.
Aktor będący instancją `vtkImageActor` potrzebuje `vtkImageMapToColors` przechowującego odpowiedni lookup_table. Wymagane jest również skonfigurowanie wymiarów w każdej płaszczyźnie poprzez funkcję `SetDisplayExtent` przyjmującej początek oraz koniec płaszczyzny w każdym wymiarze.

Wartości dobierane są tak, aby płaszczyzny znajdowały się w środku poszczególnych wymiarów. Dwa wymiary można łatwo zdefiniować poprzez pobranie rozmiarów DICOMowych zdjęć 2D. Trzeci wymiar odnosi się do ilości zdjęć w danej serii DICOMowej.