Projekt zaliczeniowy

Z przedmiotu „Wizualizacja danych w systemach biomedycznych” realizowanego dla kierunku Elektrotechnika – specjalność Pomiary Techniczne i Biomedyczne.

„Przeglądarka obiektów siatkowych 3D z widgetem do sterowania współczynnikami wyglądu powierzchni 3D”

Autorzy: Michał Kluska nr albumu 258240
Maciej Kucharski nr albumu 258116

Kraków, AGH, 1.07.2016r.

Przeglądarka obiektów polydate "GUT"

1.0

Generated by Doxygen 1.8.6

Sat Jul 2 2016 14:10:19

Contents

Chapter 1

Przeglądarka obiektów polydate "GUT"

Author

Michał Kluska & Maciek Kucharski

Date

29.06.16

Version

1.0

1.1 Cel

Temat projektu: "Przeglądarka obiektów siatkowych 3D z widgetem do sterowania współczynnikami wyglądu powierzchni 3D"

Celem projektu była realizacja w pełni działającej aplikacji w framwork-u Qt z wykorzystaniem bibliteki do wizualizacji danych VTK. Realizacja miała obejmować założenia podstawowe oraz dodatkowe z uwagi na realizację projektu w zespole dwuosobowym.

Aplikacja miała mieć możliwość pracy z obiektami typu polydata. Użykownik ma posiadać pełną kontrolę nad parametrami renderu powierzchni zgodnie z modelem Phonga. Ponadto aplikacja ma umożliwiać krojenie obiektu na plastry o zadanej przez użytkownika grubości oraz zapis do pliku wyników wytworzonych przekrojów.

Poza częścią zwiazaną z tematem projektu, tworzony program powinien umożliwiać wykonywanie zrzutów ekranu aplikacji oraz zapis ich na dysku. Inne wymagane elementy to: zakładka "O programie", menu do obsługi operacji na plikach (np. otwieranie plików wejściowych, zapisywanie wizualizacji, pomoc), działające automatyczne rozmieszczanie widgetów w oknie programu (musi on dostosowywać rozmieszczenie i rozmiar elementów GUI przy zmianie rozmiarów okna), "dymki" (podpowiedzi) objaśniające przeznaczenie elementów interfejsu użytkownika, wymyślone, narysowane w programie Inkscape i umieszczone w aplikacji logo programu, a także wymyślona i zaimplementowana nazwę programu.

Tworzona aplikacja w przyszłości może być podstawą do wytworzenia narzędzia przeznaczonego dla artystów, które umożliwi drukowanie wycinków zeskannownych obiektów w technologi 3D. Byłoby to znaczne uproszczenie ich pracy, szczególnie w pracy rzeźbiarskiej.

1.2 Założenia projektu:

1. program wczytuje obiekty typu PolyDate z plików STL, OBJ lub innych danych typu PolyData
2. opracowanie widgetu Qt do sterowania współczynnikami modelu oświetlenia Phonga:

- współczynnik rozproszenia, rozbłysku, mocy rozbłysku
- zmiana koloru obiektu
- podgląd nastaw
- możliwość zapisu i odczytu nastaw

3. opracowanie widgetu Qt umożliwiającego wizualizację i krojenie obiektów 3D w sposób określony przez użytkownika.
    - automatyczne krojenie obiketu
    - zapis wycinków do pliku
    - wizualizacja krojenia
4. Dodatkowo
    - wykonywanie i zapis zrzutów ekranu
    - zakładka "O programie"
    - menu do obsługi funkcji programu (otwieranie plików wejściowych, zapisywanie wizualizacji, pomoc)
    - dymki z podpowiedziami objasniającymi przeznaczenie elementów interfejsu
    - wykonanie loga programu
    - wymyślona nazwa programu

1.3 Sposób realizacji:

W celu lepszego zobrazowania osiągnętych efektów, na końcu dokumentacji załączono "Dodatek" zawierający zestawienie zrzutów ekranu. Przedstawiają one poszczególne etapy wykorzystania programu oraz wycinek z serii wygenerowanych przekrojów.

Zwarzając na chęć dodatkowego rozwinięcia zdolności programisycznych, jak i sposobów kontroli kodu, zdecydowano się na skorzystanie z rozproszonego systemu kontroli wersji Git, w celu podziału prac i równoległego rozwoju aplikacji. Równocześnie skorzystano z serwisu Bitbucket, który pozwala na darmowe korzystanie z systemu kontroli wersji, a zarazem jest profesjonalnym rozwiazaniem nawet dla dużych zespołów.

Podstawową kwestą okazało się wgrywanie modeli 3D. Wczytywanie obiektów typu polydate zostało rozwiązane za pomocą szablonu funkcji do tworzenia odpowiednich obiektów do czytania plików .stl i .vtl (vtkSTLReader oraz vtkXMLPolyDataReader). Cały proces, od kliknięcia przez użytkownika przycisku w menu programu, po wyświetlenie wybranego obiektu w oknie programu, obsługiwany jest przez kilka metod do analizy danych o pliku oraz tworzenia elementów i wyświetlania ich na wirtualnej scenie.

Inną sprawą, którą uznano za podstawową, była wizualizacja wgranych danych. W tym celu, w początkowej fazie, spróbowano wykorzystać biblioteke QVTKWidget. Niestety, problemy z wersją Qt, uniemożliwiły jej zastosowanie wprost. Wersja 5.2 nie wspera tego widgetu, dlatego też, w finalnej wersji użyto Qt 4.8. Sam widget został stworzony poprzez transformowanie obektu QWidget, dziedzicząc po wspominanej bibliotece.

Wyświetlane obiekty stanowią typowy przykład przeływu danych wykorzystujący strukturę biblioteki VTK. Zawarte zostały one w szablonie readObject(), służącym do otwierania plików polydata. Wczytany obiekt jest przypisywany do źródła, opisanego jako reader. Następnie dane przechodzą do mappera, aktora oraz rendererów obiektu i okana. Ostatnie dwa są szczególnie potrzebne do powiązania wyświetlanego obrazu z interfejsem urzytkownika. W górnej belce przygotowano także przyciski do wyboru sposobu reprezentacji samych danych. Domyślnie załadowany model renderowany jest w postaci bryłowej, z bazowymi ustawieniami interpolacji powierzchni według modelu Phonga. Przyciski pozwalają na zmianę sposobu wyświetlania danych umożliwiając przedstawienie w postaci: chmury punktów - przycisk "Points View"(Zrzut 5); siatki - przycisk "Wireframe View"(Zrzut 4); bryłowy - "Surface View"(Zrzut 3); lub poprostu wyłaczenie podglądu obiektu - przycisk "On/Off object". Interakcję z wizualizowanym modelem, za pomocą myszy, zapewnia obiekt-interaktor powiązany z QVTKWidget-em.

Manipulację parametrami renderu wgranych brył zapewnia widget "Sterowanie Model Phonga"(Zrzut 6). Odwołuje się on do podklasy vtkProperty obiektu aktora, a dzięki indykatorom umiejscowionym przy odpowiednich suwakach, na bieżąco wskazuje wartość poszczególnych nastaw. Użytkownik ma możliwość zmainy podstawowych parametrów renderu według modelu Phonga, tj. rozproszenia(diffuse), rozbłysku(specular), mocy

rozbłysku(specular power) oraz koloru. Podstawowe warości tych parametrów to kolejno: 1, 0, 1, biały. Wyświetlane wartosci podawane są w procentach i są przeskalowywane do wymiaru przyjmowanego przez metody klasy vtkProperty. W widgecie zaimplementowano także możliwość zapisu nastaw. Jest ona szczególnie przydatna, przy wielokrotnych zmianach wgrywanego obiektu, gdyż program resetuje ustaweina renderu do nastaw podstawowych(typowych dla surowego wyświetlania wizualizacji w VTK). Mając na względzie multipatformowość rozwijanej aplikacji zdecydowano się na wykorzystanie biblioteki QSettings. Wytwarza ona obiekt zdolny do gromadzenia dowolnych rodzajów danych, zapisując je w dostosowanym do systemu operacyjnego pliku konfiguracyjnym. Wspomniane dostosowanie odbywa się automatycznie, przy tworzeniu pliku z ustawieniami.

Krojenie obiektów wykonane zostało przy użyciu klasy vtkCutter, dzięki której było możliwe stworzenie płaszczyzn krojących oraz obliczenie przekroju obiektu. W oknie aplikacji dokonywana jest wizualizacja krojenia o zadanych parametrach przez użytkownika. Użytkownik, za pomocą wyszczególnionego widgetu "Sterowanie krojenie obiektu" (Zrzut 8), może dostosować krok krojenia (grubość plastrów) oraz grubość noża. Dodatkowo, ma on pełną kontrolę nad ustwaieniem płaszczyzny tnącej, poprzez podawania współrzędnych składowych x,y,z. Aplikacja została wyposarzona w zestaw funkcji do automatycznego krojenia całego obiektu typu polydata. W skład zestawu wchodzi metoda do tworzenia poszczególnych przekrojów oraz wizualizacji swojego działania, a także metoda do zapisu stanu widgetu do pliku .png w lokalizacji, z której został wczytany model bazowy. W Zrzutach 9-19 przedstawiono przykładową sekwencję uzyskanych przekrojów.

Obydwa widgety zostały zaprojektowane w taki sposób, by były intuicyjne dla użytkownika. Równocześnie zabezpieczono je tak, aby uaktywaniały się dopiero w momencie poprawnego załadowania modelu do programu (Zrzut 1,2). Dodatkowo, w celu zwiększenia konfortu użytkowania, widgety można odpiąć od głównego okna programu za pomocą odpowiedniego przycisku, znajdującego się po prawej, górnej stronie każdego z nich.

1.4 Instrukcja obsługi:

W nijenszym rodziale zostanie przedstawione przykładowe wykorzystanie aplikacji w celu uzyskania przekrojów w płaszczyźnie strzałkowej, gotowych do wydruku.

1. Uruchom aplikację GUT.
2. W menu głównym programu wybierz: Plik $>$ Wczytaj obiekt(Zrzut 1).
3. W okienku kontekstowym wybierz lokalizacje i plik z modelem, z którego chcesz wygenerować przekroje.
    - Pamiętaj, że przekroje będą zapisywane w tym samym folderze.
4. Jeżeli poprawnie wybrałeś plik, w aplikacji uaktywnią się 2 widgety (Zrzut 2):
    - Widget do sterowania parametrami renderu powierzchni,
    - Widget do tworzenia przekrojów.
5. W celu manipulacji ustawienia orientacji obiektu , kliknij kursorem na wyświetlony obraz i przesuń mysz w taki sposób, aby uzyskać zadowalający kąt.
6. Możesz zmienić paramety renderu za pomocą widgetu "Sterowanie Model Phonga" (Zrzut 7).
7. Zaznacz "Włącz/Wyłacz krojenie obiektu" w widgecie "Sterowanie krojenie obiektu".
    - W tym momencie uaktywnią się pozostałe kontrolki w tym widgecie.
8. Ustaw pożądzane wartości grubości ostrza, grubość plastra oraz współrzędne płaszczyzny tnącej.
    - W celu uzyskania płaszczyzny sztrzałkowej należy ustawić wspołrzedną x w maksymalnej wartosci, a pozostałe na minimum.
9. Zwizualizuj efekt wybranych nastaw klikając przycisk "Podgląd".
    - W tym momencie na ekranie podglądu obiektu wyrysowane zostaną żółte linie, reprezentujące wspólną granicę wgranego modelu oraz płaszczyzny tnącej.
    - Jeżeli osiągnięty rezultat nie jest zadowalający, zmodyfikuj nastawy powtarzając kroki 8 i 9.

10. Uruchom algorytm generujący przekroje, klikając przycisk "Krój".
    - Na ekranie wizualizacji wyświetlona zostanie animacja cięcia obiektu. Równocześnie aplikacja zostanie zablokowana do końca działania algorytmu.
11. Po zakończeniu generowania przekrojów, w dolnym pasku aplikacji zostanie wyświetlona informacja o końcu obliczeń. Utworzone przekroje znajdyją się w folderze z wgranym modelem.

1.5 Wnioski:

Zadanie postawione w temacie pracy zostało wykonane. Udało się zrealizować w satysfakcjonujący sposób wszystkie cele i założenia, a także poboczne elementy, jak np. możliwość zapisywania zrzutów ekranowych czy logo programu. Stworzona apliakcja jest łatwa w obsłudze i posiada przyjazny, intuicyjny dla użytkownika interfejs.

Z uwagi na to, iż jest to pierwsza wersja programu, posiada ona duże możliwości rozwoju, w zależności od zapotrzebowania użytkownika. Kolejnym etapem projektu powinna być weryfikacja założeń, przeprowadzona poprzez udostępnienie aplikacji artystom-rzeźbiarzom, stanowiących docelową grupę użytkowników. Jako zespół jesteśmy otwarci na propozycje rozwoju aplikacji i dostosowawania jej do bardziej szczególowych zastosowań.

Równocześnie starano się dopracować jak najbardziej zaprezentowane rozwiązania i wyeliminować wszystkie wady. Kontynułacja pracy nad aplikacją zależy od zainteresowania osób widzących jej zastosowanie w konkretnej pracy.

Relacjonując samą pracę nad programem, wyciągnięto szereg wniosków. Jako pierwszą należy omówić kwestie środowiska Git. Praca z rozproszonym systemem kontroli wersji w przedstawionej wcześniej konfiguracji przebiegała sprawnie, aż do momentu łaczenia branch-y. Na zaawansowanym etapie projektu Git zupełnie nie radził sobie z merge-owaniem różnych części kodu. Niejednokrotnie zgłaszał konflikty w miejscach projektu, w których go nie było. Skutkowało to tym, iż połączenie ich w ten sposób stało się niemożliwe. Do samego merge-owania zostało zastosowane narzędzie graficzne DiffMerge. Być może konfiguracja i wykorzystanie narzędzi wbudowanych w Qt Creator, w czasie z pracy z projektem, umożliwiłoby uniknięcie takiego problemu.

Drugim problemem na jaki natroafiono były kłopoty z działaniem klasy vtkOBJReader, która nie chciała współpracować z Qt. Mimo implementacji wczytywania obiektu, zgodnie ze specyfikacją w dokumentacji klasy, w trakcie wczytywania obiektów .obj aplikacja przerwywała niespodziewanie prace. Po przyjrzeniu się dokładniej temu zagadnieniu okazało się, iż problem występuje w strukturze klasy i może być spowodowany barkiem zgodności z użytą wersją Qt. Biorąc pod uwagę ten fakt oraz brak rozwiązań problemu wśród społeczności korzystajacej z Qt i VTK zaniechano wykorzystania powyższej klasy.

Następnym problemem okazała się kwestia zapamiętywania ustawień renderu modelu. Zastosowanie wspominanej wcześniej klasy QSettings zapewniło wbudowaną multiplatformowość, jednakże utrudniło odczyt warości, zwłaszcza w przypadku próby odtworzenia wybranego koloru. Problem ten rozwiązano stosując dodatkową klase QVariant. Zaliczają się do niej obiektey, które w rzeczyywistości są wytwarzane w klasie QSettings. Do tego, zastosowano rzutowanie typów wbudwanych, co pozwoliło na przesłanie odpowiednich wartosci do panelu sterowania nastawami, a w przypadku samego koloru, zastosowano rozdzielenie barwy na zestaw składowych RGB.

Nadal nie roziwiązanym problemem pozostaje kwestia "przeszkadzania" aplikacji. W momenci uruchomienia automatycznego algorytmu krojącego, należy pozostawić aplikacię w takim stanie w jakim sie znajduje(nie przesuwać okna). Jest to powiązane ze sposobem generowania przekroju. W tym czasie aplikacja dokonuje zrzutu ekranowanego prezentowanego przekroju. Jeżeli w jakiś sposób ją przesuniemy, to przesunięcie zostanie zarejesteowane na przekroju. Nie udało się zaimplementować innego sposobu eksportowania wizualizacji.

Aplikacja została przetestowana na dwóch różnych ustawieniach systemu operacyjnego Linux. Pierszym z nich, była dystrybucja Mint, stanowiąca normalny system operacyjny komputera. Aplikacja zachowywała pełną sprawność i funcjonalność. Drugą z nich, była dystrybucja Ubuntu 14.04, działajaca jako maszyna wirtualna na komputezrze z systemem operacyjnym Windows 7. W tym przypadku, odnotowano problemy ze skalowaniem okienek - czasami aplikcja nie mieściła się na ekranie, co uniemożliwiało np. dokonanie podglądu ustawień tworzenia przekrojów. Prawdopodobnie, problemy te sa powiązane z za małą rozdzielczością ekranu wirtualnej maszyny, w stosunku do rzeczywistej rozdzielczości komputera. Dlatego też, by w pewien sposób obejść ten dość niespodziewany problem, zdecydowano się właśnie na dokowane widgety. Rozłączenie niektórych z nich, pozwala na zochowanie pełnej funkcjonalności. Rozwiązanie to stanowi swoiste, dodatkowe zabezpieczenie, przy wystąpieniu takiego problemu.

W kolejnych rozdziałch została zaprezentowana dokumentacja kodu, zawierająca opisy działania poszczególnych metod i elementów zawrtych w kodzie. Została ona wygenrowania autoamatycznie w programie Doxygen, w oparciu

o zamieszczony kod programu GUT.

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 Ui Namespace Reference

6.1.1 Detailed Description

Klasa MainWindow - odpowiada za połączenie widżetów z metodami oraz działanie całej aplikacji

W obrębie klasy wykonywane są wszystkie działania związane z działaniami w głównym oknie programu. Skonfigurowany jest również wygląd podpowiedzi oraz działania podstawowych opcji w menu.

Klasa posiada: chronione sloty - pozwalające na otrzymywanie inforamcji signały - do przesyłania danych do metod i innych klas zmienne oraz wskaźniki prywatne - potrzebne do poprawnego działania metod klasy oraz wywoływania chronione funkcje - odpowiadające za akcje wewnątrz klasy konstruktor jawny

Michał Kluska & Maciek Kucharski, czerwiec 2016

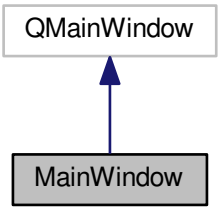
Chapter 7

Class Documentation

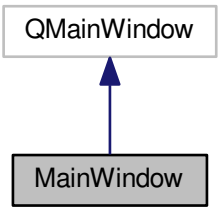
7.1 MainWindow Class Reference

`#include <mainwindow.h>`

Inheritance diagram for MainWindow:



Collaboration diagram for MainWindow:



Signals

- void setStatusText (QString)

metoda do wystanie tekstu na StatusBar

- void wellOpen ()

  flaga, że plik został wgrany poprawnie

Public Member Functions

- MainWindow (QWidget *parent=0)

  MainWindow::MainWindow - konstruktor klasy.

- ~MainWindow ()

  MainWindow::~MainWindow() - destruktor klasy MainWindow.

Protected Slots

- void updateCoords ()

  metoda do aktualizacji współrzędnych kamery

- void openFile ()

  metoda do otwierania pliku

- void takeScreen ()

  metoda wykonywanie screenshot-a

- void takeScreenWidgetArea ()

  metoda screenshot-a samego widgetu

- void takeAutoScreenWidgetArea (QString)

  metoda do wykonywania autmatycznych screenshot-ów

- void saveScreen ()

  metoda do zapisywania screen-ów

- void aboutApp ()

  metoda do wyświetlania inforamcji o aplikacji

- void objectToVisualization ()

  metoda do wywoływania szablonu wizualizacji cięcia

- void objectToSlice ()

  metoda do wywoływania szablonu klasy do cięcia automatycznego obiektu

- void changeCheckBox ()

  metoda do aktywacji cześci okna odpowiadającego za ustawianie parametrów krojenia

- void setLabel (int)

  metoda do ustawiania wartości na labelach

- void hideActor ()

  metoda do ukrywania aktora

- void setDiff (int)

  metoda do ustawiania wartości rozproszenia renderu modelu

- void setSpec (int)

  metoda do ustawiania wartości rozbłysku renderu modelu

- void setSpecPow (int)

  metoda do ustawiania wartości mocy rozbłysku renderu modelu

- void setColor ()

  metoda do ustawienia koloru modelu

- void phongEnabled ()

  metoda ustawiająca dostęp do kontrolek renderu modelu

- void saveRenderParam ()

  metoda zapisująca nastawy parametrów renderu

- void loadRenderParam ()

metoda wgrywająca nastawy parametrów renderu

- void setPointsObjectView ()

  metoda do zmiany sposobu reprezentacji obiektu na punktowy
- void setSurfaceObjectView ()

  metoda do zmiany sposobu reprezentacji obiektu na powierzchniowy
- void setWireframeObjectView ()

  metoda do zmiany sposobu reprezentacji obiektu na szkieletowy

Protected Member Functions

- void viewObject ()

  metoda do inicjalizacji strumienia VTK
- void setConnections ()

  metoda do inicjalizacji połoczeń SIGNAL-SLOT
- void phongSliderReset ()

  metoda do resetowanie ustawień Slider'ów do stanu poczatkowego
- template<class T >
  void readObject ()

  szablon klasy odowiadajacy za wczytanie wybranego pliku
- template<class T >
  void sliceObject ()

  szablon klasy odpowiadajacy za wizualizację krojenia
- template<class T >
  void sliceObject (float, float, float, float, float)

  szablon klasy odpowiadający za automatyczne krojenie obiektu 3D
- void showOnStatusBar ()

  metoda odpowiadająca za wyświetlanie wartości nastaw na pasku status bar
- void showMessageOnStatusBar ()

  metoda odpowiadająca za wyświetlanie inforacji na pasku status bar
- void hideCutter ()

  metoda odpowiadająca za ukrywanie wizualizacji cięcia

Protected Attributes

- QString fileExt

  pole klasy zawierające rozszerzenie otwartego pliku
- QString fileName

  pole klasy zawierajace nazwę otwawrtego pliku
- QString filePath

  pole klasy zawierające ścieżkę otwartego pliku
- QFileInfo fileInfo

  pole klasy zawierające informacje o pliku
- bool OnOffobject = true

  pole kalsy zawierające informacje o widzialności obiketu

7.1.1 Constructor & Destructor Documentation

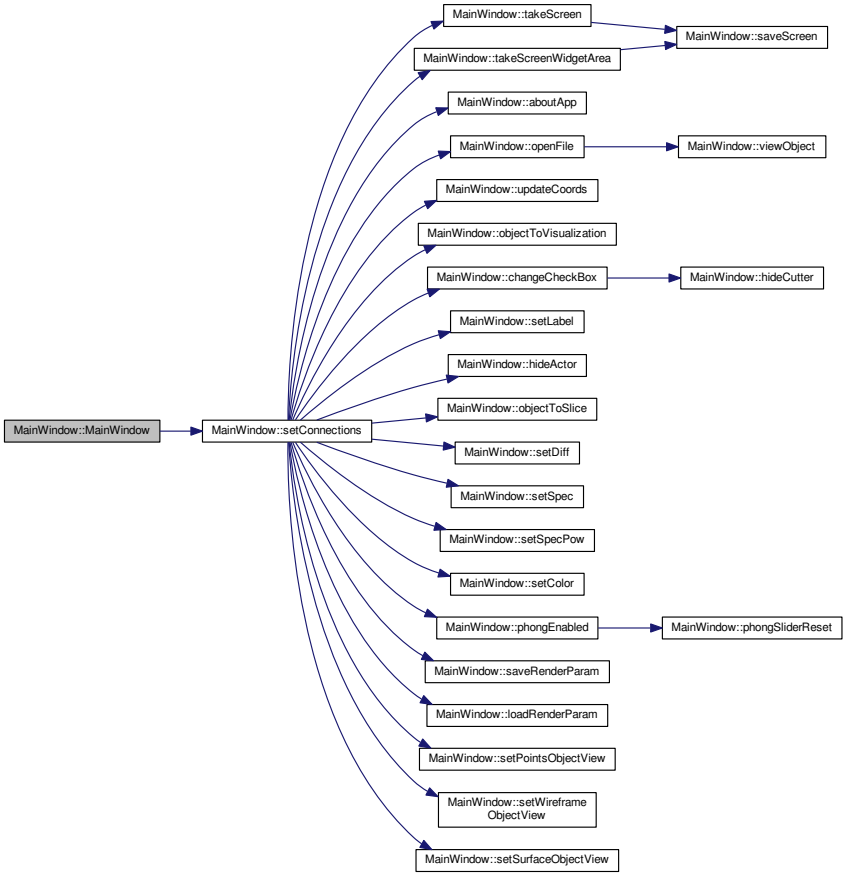
7.1.1.1 MainWindow::MainWindow (QWidget * *parent* = 0)

MainWindow::MainWindow - konstruktor klasy.

Parameters

parent	- wskaznik na obiekt nadrzędny
ui	- wskaźnik na interfejs aplikacji

Here is the call graph for this function:



7.1.1.2 MainWindow::~MainWindow ( )

MainWindow::~MainWindow() - destruktor klasy MainWindow.

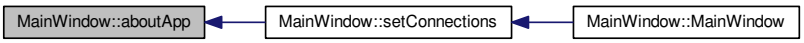
7.1.2 Member Function Documentation

7.1.2.1 void MainWindow::aboutApp ( ) [protected], [slot]

metoda do wyświetlania inforamcji o aplikacji

MainWindow::aboutApp() - funkcja wyświetlająca okno "O aplikacji".

Here is the caller graph for this function:

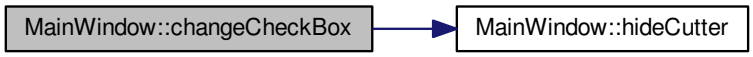


7.1.2.2 void MainWindow::changeCheckBox ( ) [protected], [slot]

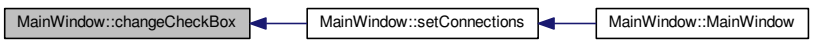
metoda do aktywacji cześci okna odpowiadającego za ustawianie parametrów krojenia

MainWindow::changeCheckBox() - funkcja do aktywacji groupBoxSlider części formularza odpowiadającego za ustalanie parametrów i krojenie obiektu.

Here is the call graph for this function:



Here is the caller graph for this function:

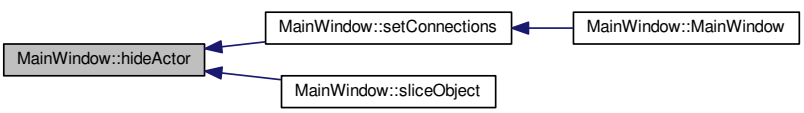


7.1.2.3 void MainWindow::hideActor ( ) [protected], [slot]

metoda do ukrywania aktora

MainWindow::hideActor() - funkcja do ukrywania aktora.

Here is the caller graph for this function:

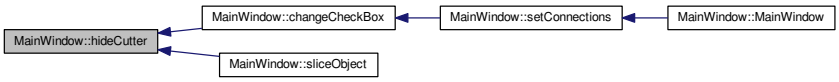


7.1.2.4 void MainWindow::hideCutter ( ) [protected]

metoda odpowiadająca za ukrywanie wizualizacji cięcia

MainWindow::hideCutter() - funkcja do ukrywania ostrza.

Here is the caller graph for this function:

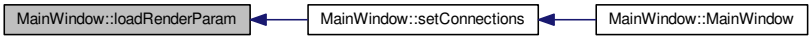


7.1.2.5 void MainWindow::loadRenderParam ( ) [protected], [slot]

metoda wgrywająca nastawy parametrów renderu

MainWindow::loadRenderParam() - metoda odpowiadająca za wczytywanie nastaw renderu modelu.

Here is the caller graph for this function:

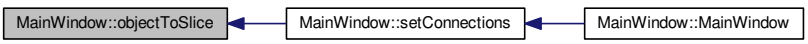


7.1.2.6 void MainWindow::objectToSlice ( ) [protected], [slot]

metoda do wywoływania szablonu klasy do cięcia automatycznego obiektu

MainWindow::objectToSlice() - funkcja wywołująca odpowiedni szablon funkcji do krojenia obiektu oraz odbiera parametry ustawione w oknie aplikacji, przeskalowywuje je i wysyłą do funkcji tnącej.

Here is the caller graph for this function:

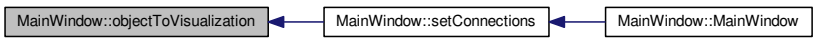


7.1.2.7 void MainWindow::objectToVisualization ( ) [protected], [slot]

metoda do wywoływania szablonu wizualizacji cięcia

MainWindow::objectToVisualization() - funkcja wywołująca odpowiedni szablon funkcji do wizualiacji.

Here is the caller graph for this function:



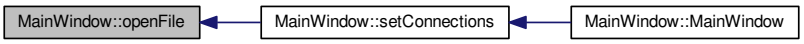
7.1.2.8 void MainWindow::openFile ( ) [protected], [slot]

metoda do otwierania pliku

MainWindow::openFile() - funkcja otwarcia pliku .obj lub .stl przy wykorzystaniu dodatkowego okna dialogowego.

Here is the call graph for this function:

Here is the caller graph for this function:

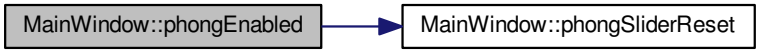


7.1.2.9 void MainWindow::phongEnabled ( ) [protected], [slot]

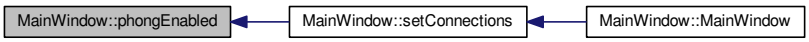
metoda ustawiająca dostęp do kontrolek renderu modelu

MainWindow::phongEnabled() - metoda aktywująca interfejs renderu modelu w momencie poprawnego wgrania modelu do przeglądarki.

Here is the call graph for this function:



Here is the caller graph for this function:



7.1.2.10 void MainWindow::phongSliderReset ( ) [protected]

metoda do resetowanie ustawień Slider'ów do stanu poczatkowego

MainWindow::phongSliderReset() - metoda resetująca ustawienia suwaktków interfejsu renderu modelu.

Here is the caller graph for this function:

7.1.2.11 template<class T > void MainWindow::readObject ( ) [protected]

szablon klasy odowiadajacy za wczytanie wybranego pliku

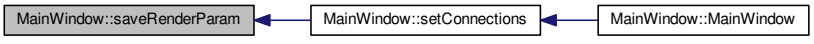
MainWindow::readObject() - szablon funkcji do otwarcia obiektów .stl i .obj. Dzięki wykorzystaniu szablonu można było rozwiazać problem tworzenia obiektów o różnym typie. Wewnątrz funkcji tworzone są wszystkie elementy niezbędne do pokazania wczytanego obiektu w oknie aplikacji.

7.1.2.12 void MainWindow::saveRenderParam ( ) [protected], [slot]

metoda zapisująca nastawy parametrów renderu

MainWindow::saveRenderParam() - metoda odpowiadająca za zapisywanie nastaw renderu modelu.

Here is the caller graph for this function:

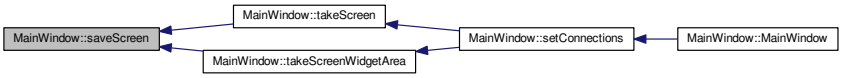


7.1.2.13 void MainWindow::saveScreen ( ) [protected], [slot]

metoda do zapisywania screen-ów

MainWindow::saveScreen() - funkcja zapisuje wykonany zrzut ekranu do lokalizacji wybranej przez użytkownika.

Here is the caller graph for this function:

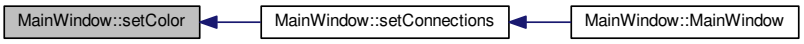


7.1.2.14 void MainWindow::setColor ( ) [protected], [slot]

metoda do ustawienia koloru modelu

MainWindow::setColor() - metoda odpowiadająca za ustawienie koloru renderu modelu.

Here is the caller graph for this function:

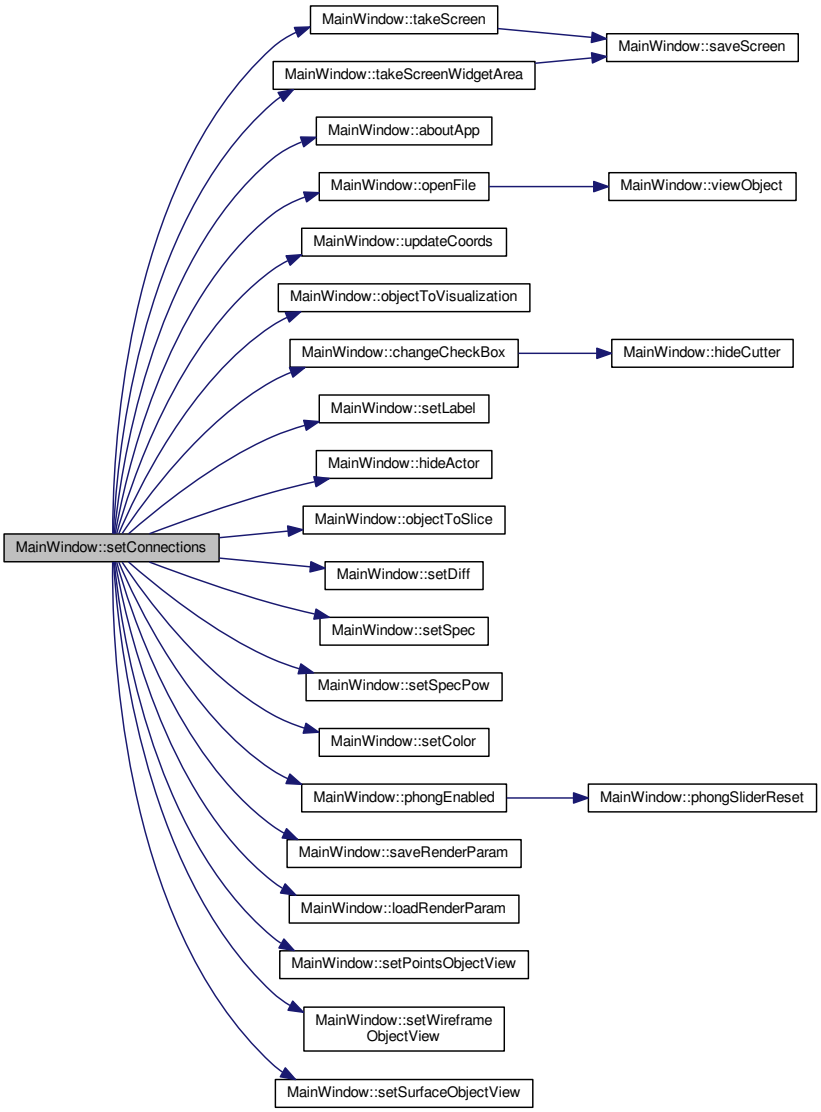


7.1.2.15 void MainWindow::setConnections ( ) [protected]

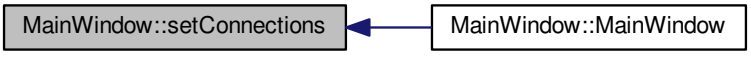
metoda do inicjalizacji połoczeń SIGNAL-SLOT

MainWindow::setConnections() - zestawianie połączeń sygnałów i slotów.

Here is the call graph for this function:



Here is the caller graph for this function:



7.1.2.16 void MainWindow::setDiff (int *value*) [protected], [slot]

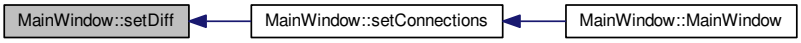
metoda do ustawiania wartości rozproszenia renderu modelu

MainWindow::setDiff(int value) - metoda odpowiadająca za ustawienie rozproszenia w renderu modelu.

Parameters

value	- wartość rozproszenia

Here is the caller graph for this function:



7.1.2.17 void MainWindow::setLabel (int *actualValue*) [protected], [slot]

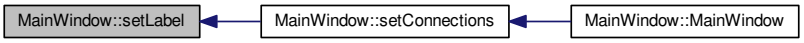
metoda do ustawiania wartości na labelach

MainWindow::setLabel(int actualValue) - funkcja do wyświetlania przekalowanej wartości na label-ach.

Parameters

actualValue	- wartość aktualnie ustawiona na slajderze

Here is the caller graph for this function:

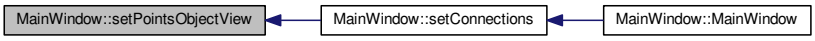


7.1.2.18 void MainWindow::setPointsObjectView ( ) [protected], [slot]

metoda do zmiany sposobu reprezentacji obiektu na punktowy

MainWindow::setPointsObjectView() - funkcja do zmiany sposobu reprezentacji obiektu na punktowy.

Here is the caller graph for this function:



7.1.2.19 void MainWindow::setSpec (int *value*) [protected], [slot]

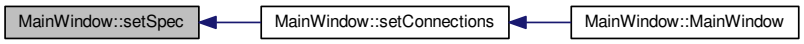
metoda do ustawiania wartości rozbłysku renderu modelu

MainWindow::setSpec(int value) - metoda odpowiadająca za ustawienie rozbłysku w renderu modelu.

Parameters

value	- wartość rozbłysku

Here is the caller graph for this function:



7.1.2.20 void MainWindow::setSpecPow (int *value*) [protected],[slot]

metoda do ustawiania wartości mocy rozbłysku renderu modelu

MainWindow::setSpecPow(int value) - metoda odpowiadająca za ustawienie mocy rozbłysku renderu modelu.

Parameters

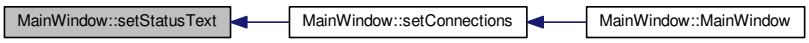
value	- wartość mocy rozbłysku

Here is the caller graph for this function:

7.1.2.21 void MainWindow::setStatusText (QString) [signal]

metoda do wystanie tekstu na StatusBar

Here is the caller graph for this function:

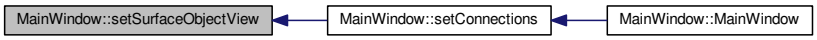


7.1.2.22 void MainWindow::setSurfaceObjectView ( ) [protected],[slot]

metoda do zmiany sposobu reprezentacji obiektu na powierzchniowy

MainWindow::setSurfaceObjectView() - funkcja do zmiany sposobu reprezentacji obiektu na płaszczyznowy.

Here is the caller graph for this function:

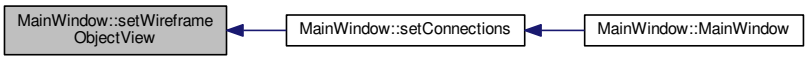


7.1.2.23 void MainWindow::setWireframeObjectView ( ) [protected], [slot]

metoda do zmiany sposobu reprezentacji obiektu na szkieletowy

MainWindow::setWireframeObjectView() - funkcja do zmiany sposobu reprezentacji obiektu na szkieletowy.

Here is the caller graph for this function:

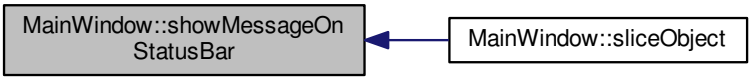


7.1.2.24 void MainWindow::showMessageOnStatusBar ( ) [protected]

metoda odpowiadająca za wyświetlanie inforacji na pasku status bar

MainWindow::showMessageOnStatusBar() - metoda odpowiadająca za wyświetlenie informacji na status bar o zakończeniu krojenia obiektu.

Here is the caller graph for this function:

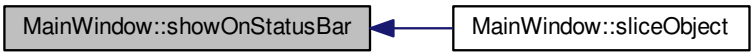


7.1.2.25 void MainWindow::showOnStatusBar ( ) [protected]

metoda odpowiadająca za wyświetlanie wartości nastaw na pasku status bar

MainWindow::showOnStatusBar - metoda odpowiadająca za wyświetlanie wartości parametrów ustawionych do krojenia.

Here is the caller graph for this function:

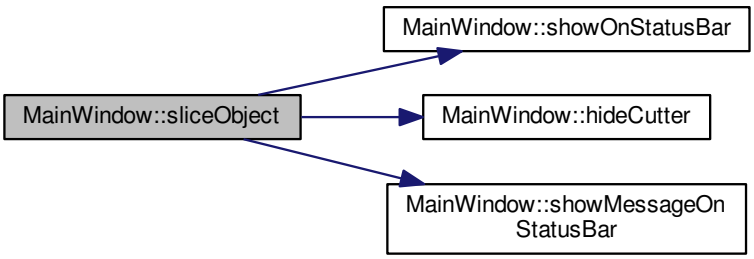


7.1.2.26 template<class T > void MainWindow::sliceObject ( ) [protected]

szablon klasy odpowiadajacy za wizualizację krojenia

MainWindow::sliceObject() - funkcja do krojenia obiektów wykorzystywana do wizualizacji sposobu krojenia. Szablon został zastosowany w celu obejścia problemu tworzenia obiektów różnego typu dla wczytywanych plików polydate. Efektem działania fukcji jest dorysowywanie do obiektu dwóch "obrysów" o ustalonej grubości obrazujących górną i dolną powierzchnię wycinanego plastra obiektu 3D.

Here is the call graph for this function:



7.1.2.27 template<class T > void MainWindow::sliceObject (float *coordinateX*, float *coordinateY*, float *coordinateZ*, float *sliceWidth*, float *thicknessKnife*) [protected]

szablon klasy odpowiadający za automatyczne krojenie obiektu 3D

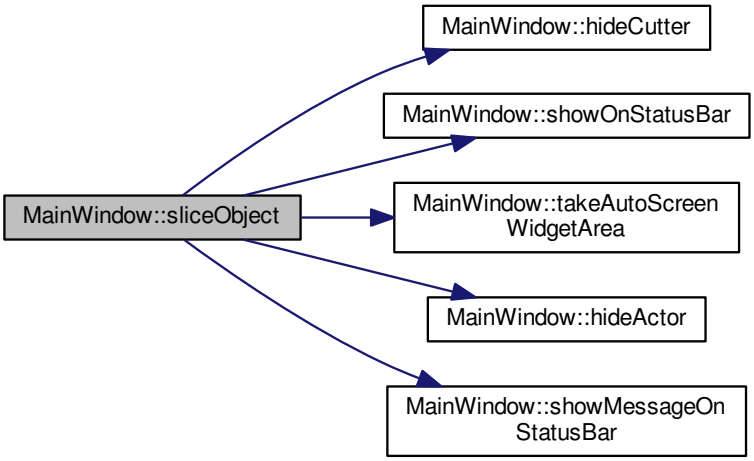
MainWindow::sliceObject(float coordinateX,float coordinateY,float coordinateZ,float sliceWidth,float thicknessKnife) - przeciążony szablon funkcji do krojenia obiektów odbierająca parametry potrzebne do cięcia. Szablon został zastosowany w celu obejścia problemu tworzenia obiektów różnego typu dla wczytywanych plików polydate. Wewnątrz funkcji dokonywane jest krojenie na plastry, wystawianie kamery i zapis do pliku poprzez wykożystanie funkcji zewnętrznych. O zakończeniu operacji użytkownik informowany jest komunikatem na dolnym pasku status bar.

Parameters

coordinateX	- współrzędna X płaszczyzny tnącej odbierana ze slajdera
coordinateY	- współrzędna Y płaszczyzny tnącej odbierana ze slajdera

coordinateZ	- współrzędna Z płaszczyzny tnącej odbierana ze slajdera
sliceWidth	- grubość wycinanego plastra odbierana ze spinbox-a
thicknessKnife	- grubość noża tnącego odbierana ze spinbox-a

Here is the call graph for this function:



7.1.2.28 void MainWindow::takeAutoScreenWidgetArea (QString *screenName*) [protected],[slot]

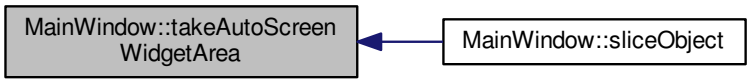
metoda do wykonywania autmatycznych screenshot-ów

MainWindow::takeScreenWidgetArea(QString screenName) - funkcja wykonująca zrzut obszaru widgetu automatycznie do lokalizacji z której otwierany był obiekt typu polydate.

Parameters

screenName	- zmienna przekazująca spreparowaną nazwę aktualnie zapisywanego przekroju

Here is the caller graph for this function:



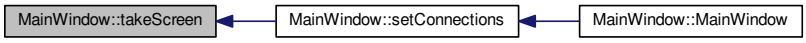
7.1.2.29 void MainWindow::takeScreen ( ) [protected],[slot]

metoda wykonywanie screenshot-a

MainWindow::takeScreen() - funkcja wykonująca zrzut całego okna aplikcji.

Here is the call graph for this function:

Here is the caller graph for this function:

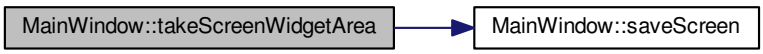


7.1.2.30 void MainWindow::takeScreenWidgetArea ( ) [protected], [slot]

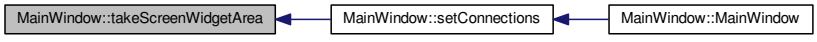
metoda screenshot-a samego widgetu

MainWindow::takeScreenWidgetArea() - funkcja wykonująca zrzut obszaru widgetu i wywołuje fukcję odpowiadającą za zapis.

Here is the call graph for this function:



Here is the caller graph for this function:

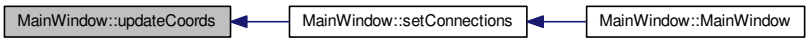


7.1.2.31 void MainWindow::updateCoords ( ) [protected], [slot]

metoda do aktualizacji współrzędnych kamery

MainWindow::updateCoords() - metoda odpowiadająca za pobranie współrzędnych kamery zmienionych poprzez przekręcanie obiektu za pomocą myszki w oknie programu.

Here is the caller graph for this function:

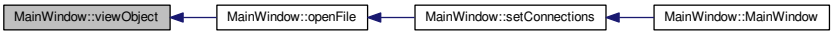


7.1.2.32 void MainWindow::viewObject ( ) [protected]

metoda do inicjalizacji strumienia VTK

MainWindow::viewObject() - metoda analizująca informacje o pliku, wyłuskująca informacje o roszerzeniu, lokalizacji, nazwie oraz wywołująca odpowiedni szablon klasy czytajacej obiekt.

Here is the caller graph for this function:



7.1.2.33 void MainWindow::wellOpen ( ) [signal]

flaga, że plik został wgrany poprawnie

Here is the caller graph for this function:

7.1.3 Member Data Documentation

7.1.3.1 QString MainWindow::fileExt [protected]

pole klasy zawierające rozszerzenie otwartego pliku

7.1.3.2 QFileInfo MainWindow::fileInfo [protected]

pole klasy zawierające informacje o pliku

7.1.3.3 QString MainWindow::fileName [protected]

pole klasy zawierajace nazwę otwawrtego pliku

7.1.3.4 QString MainWindow::filePath [protected]

pole klasy zawierające ścieżkę otwartego pliku

7.1.3.5 bool MainWindow::OnOffobject = true [protected]

pole kalsy zawierające informacje o widzialności obiketu

The documentation for this class was generated from the following files:

- mainwindow.h
- mainwindow.cpp

Chapter 8

File Documentation

8.1 main.cpp File Reference

```
#include "mainwindow.h"
#include <QApplication>
```

Include dependency graph for main.cpp:

Functions

- int main (int argc, char ∗argv[])

8.1.1 Function Documentation

8.1.1.1 int main (int *argc,* char ∗ *argv[]*)

8.2 mainwindow.cpp File Reference

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
```

Include dependency graph for mainwindow.cpp:

Macros

- #define NORMALIZACJA /255.0
- #define INIT_DIFF_POW 100
- #define INIT_SPEC 0

8.2.1 Macro Definition Documentation

8.2.1.1 #define INIT_DIFF_POW 100

8.2.1.2 #define INIT_SPEC 0

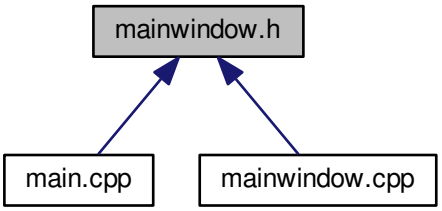
8.2.1.3 #define NORMALIZACJA /255.0

8.3 mainwindow.h File Reference

```
#include <QMainWindow>
#include <QDebug>
#include <QVTKWidget.h>
#include <vtkSmartPointer.h>
#include <vtkRenderWindow.h>
#include <vtkRenderer.h>
#include <vtkRenderWindowInteractor.h>
#include <vtkInteractorStyleUnicam.h>
#include <vtkEventQtSlotConnect.h>
#include <vtkConeSource.h>
#include <vtkPolyDataMapper.h>
#include <vtkActor.h>
#include <vtkCamera.h>
#include <vtkObject.h>
#include <vtkPolyData.h>
#include <vtkSTLReader.h>
#include <QMessageBox>
#include <QPixmap>
#include <QDir>
#include <QFileDialog>
#include <QDesktopWidget>
#include <QApplication>
#include <QFileInfo>
#include <vtkXMLPolyDataReader.h>
#include <vtkPlane.h>
#include <vtkCutter.h>
#include <vtkProperty.h>
#include <vtkPropCollection.h>
#include <QPainter>
#include <QSettings>
#include <QVariant>
#include <QColorDialog>
```

Include dependency graph for mainwindow.h:

This graph shows which files directly or indirectly include this file:



Classes

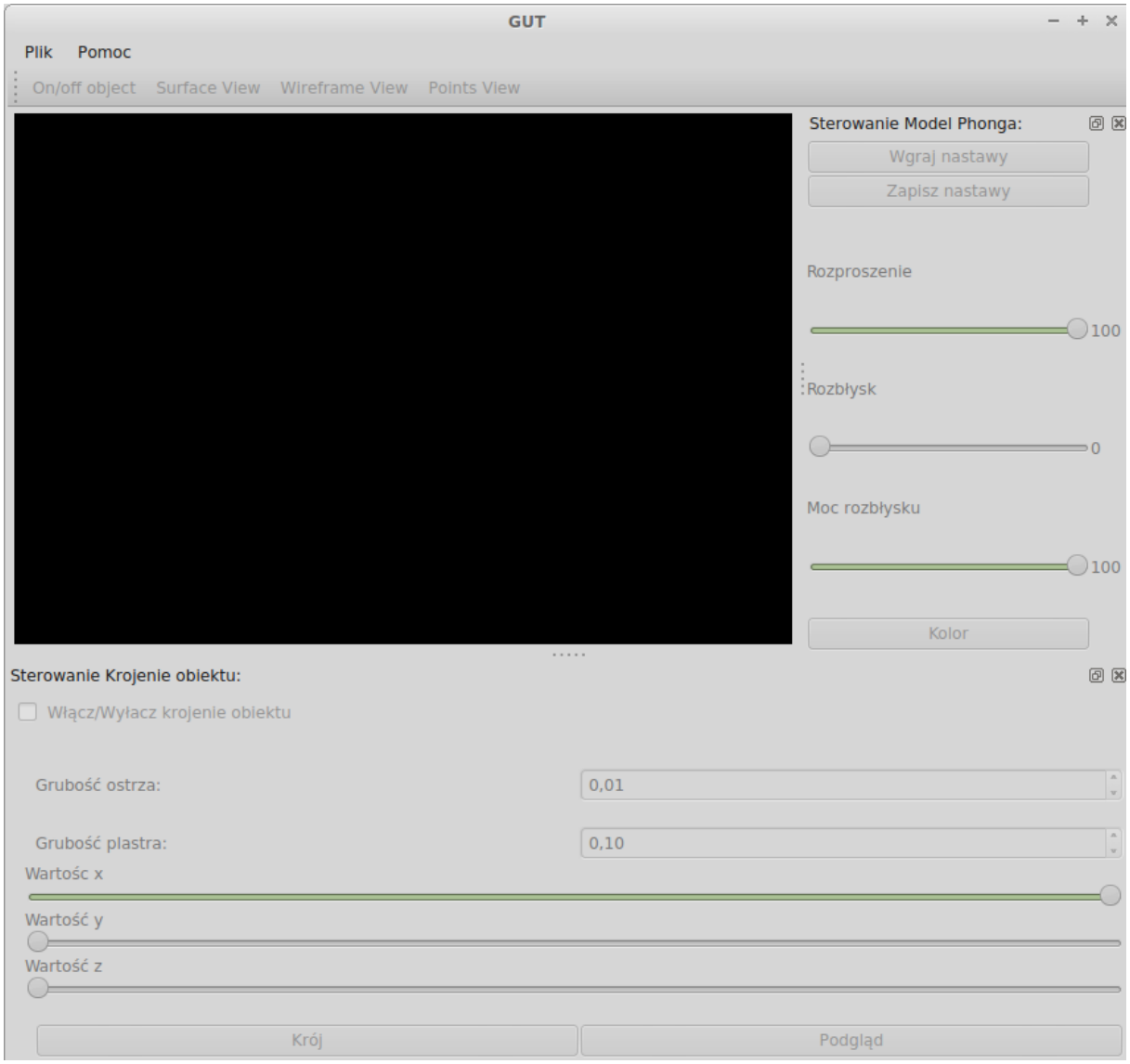
- class MainWindow

Namespaces

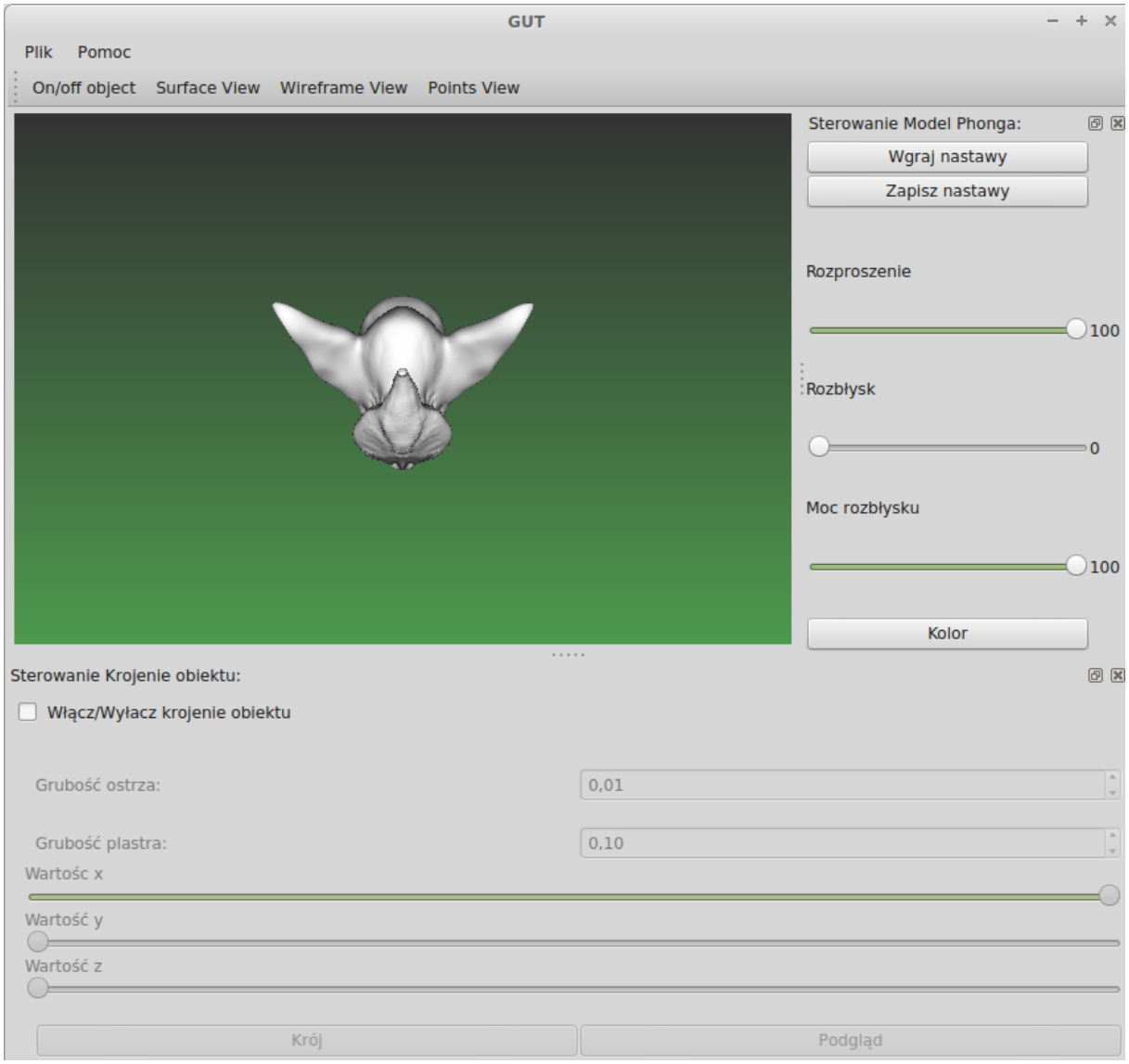
- Ui

Index

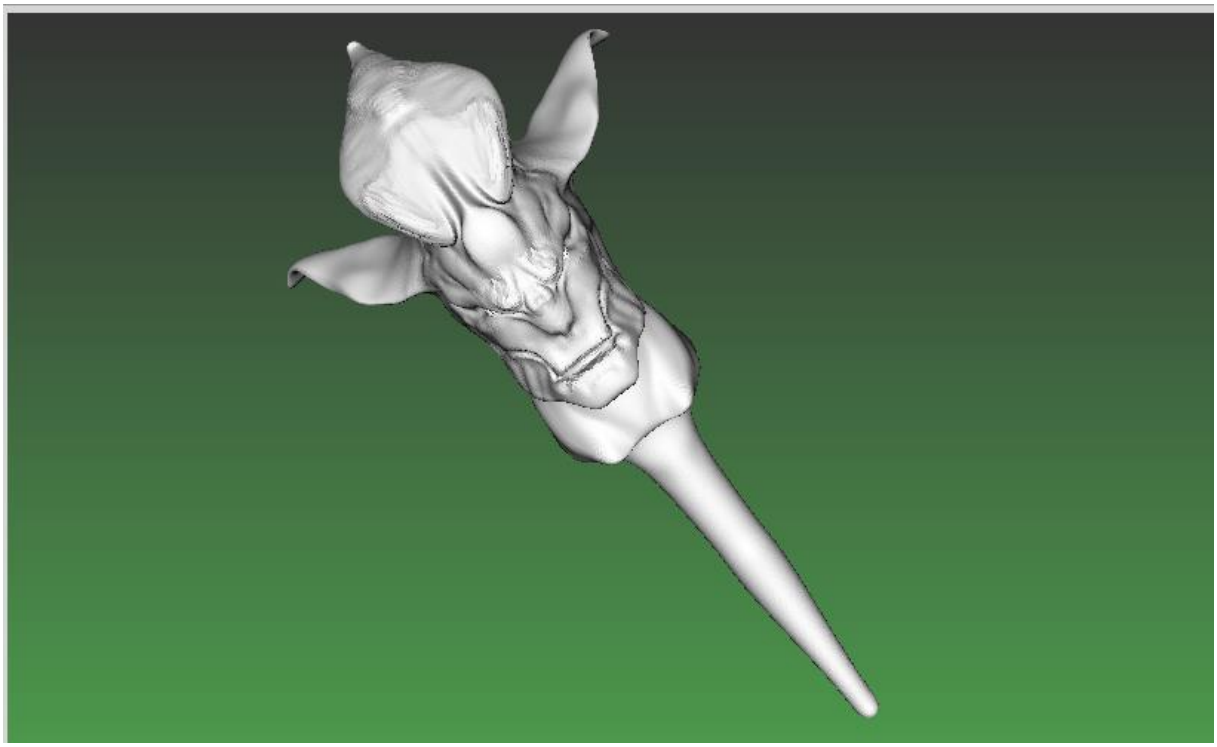
Zrzuty ekranu prezentujące aplikacje „GUT”



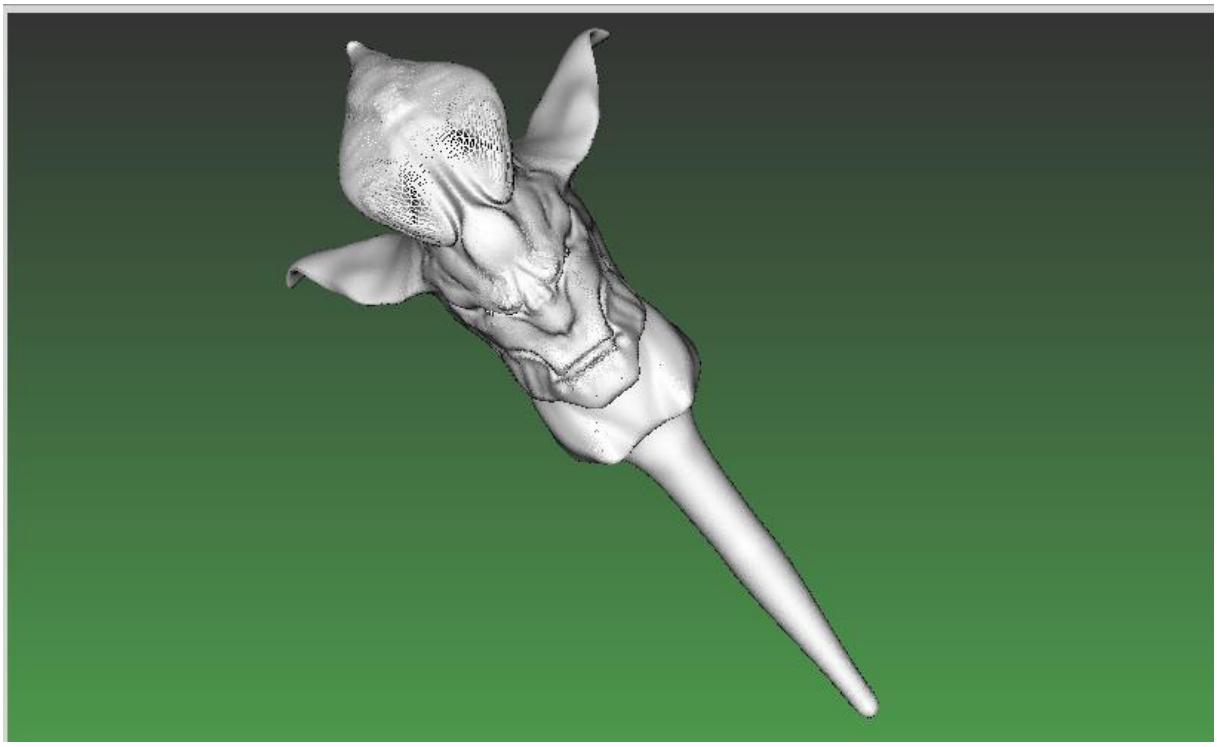
Zrzut 1: Wyglądu interfejsu programu "GUT" przed wczytaniem jakiegokolwiek obiektu.



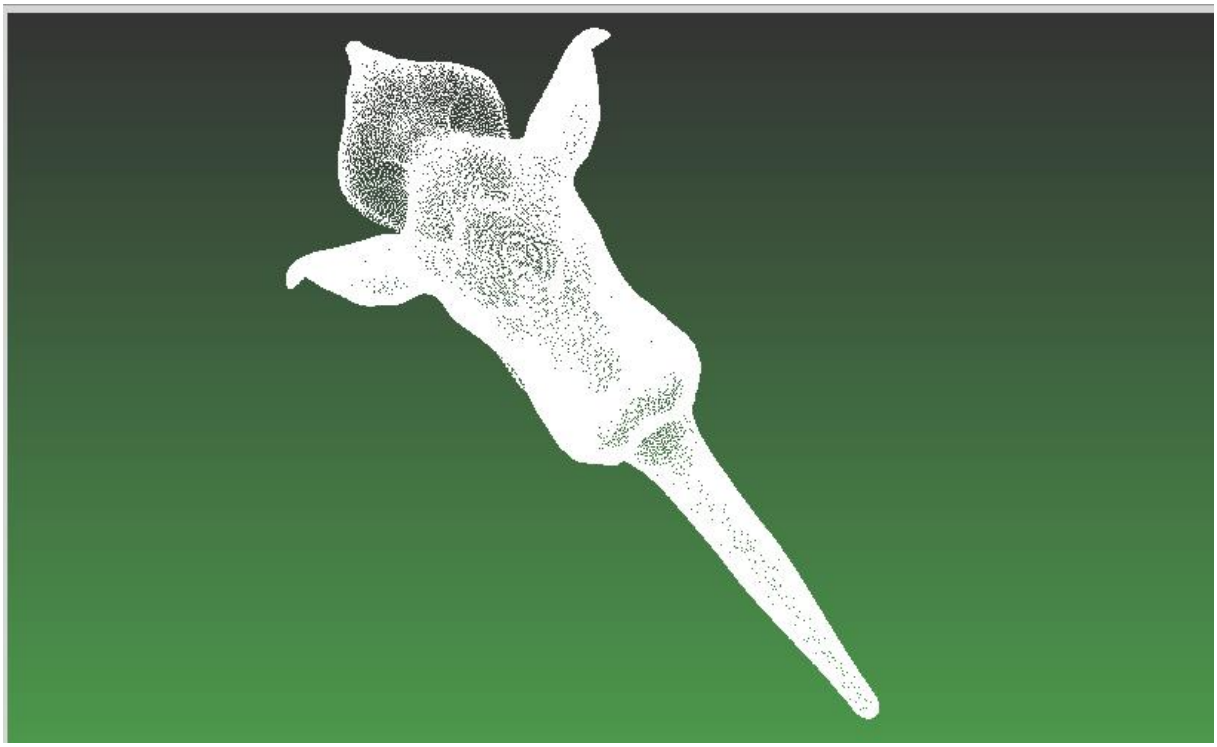
Zrzut 2: Wyglądu interfejsu programu "GUT" po wczytaniem przykładowego obiektu.



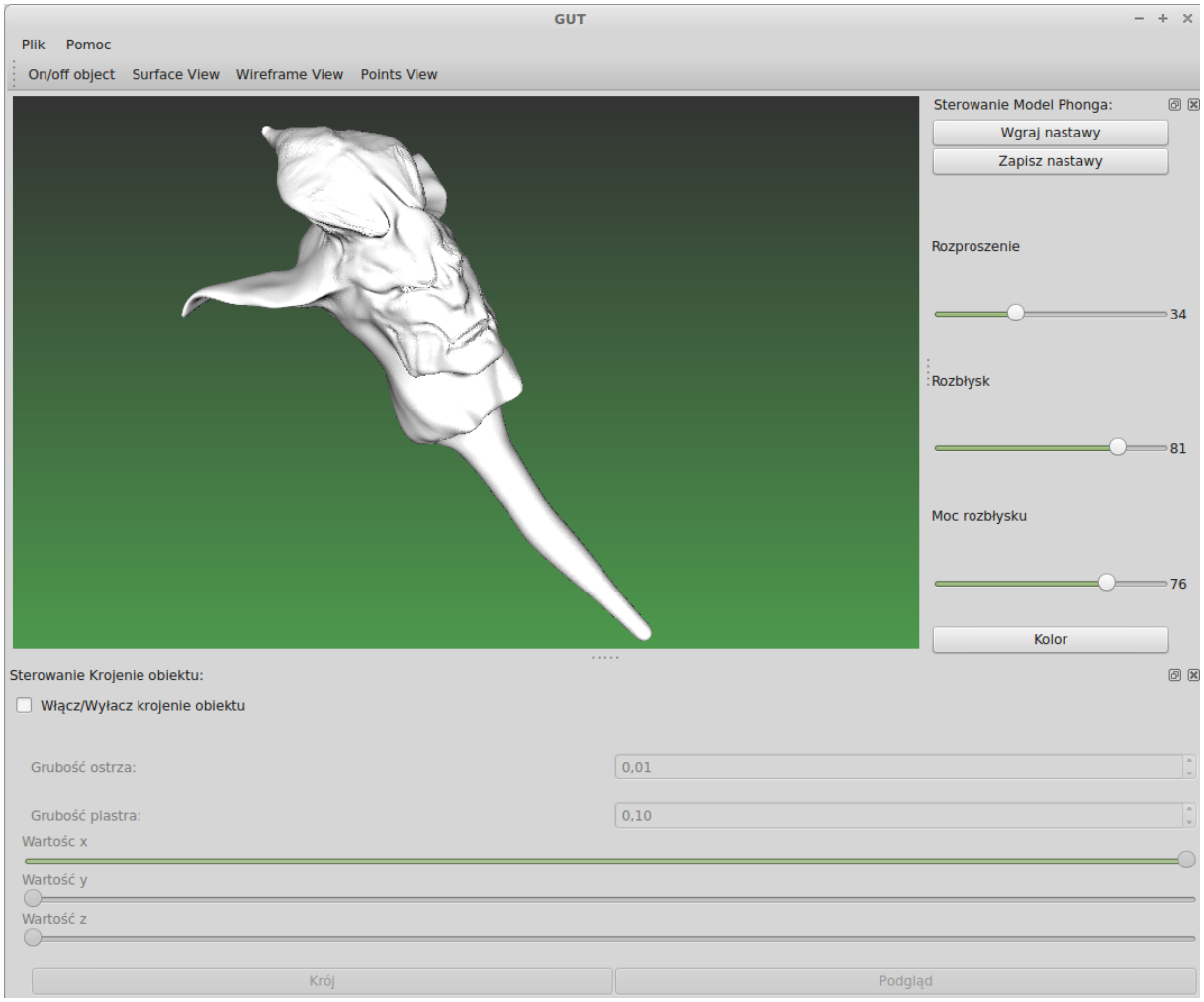
Zrzut 3: Program oferuje 3 sposoby wyświetlania obiektów typu PolyData. Pierwszy tryb to wizualizacja powierzchniowa.



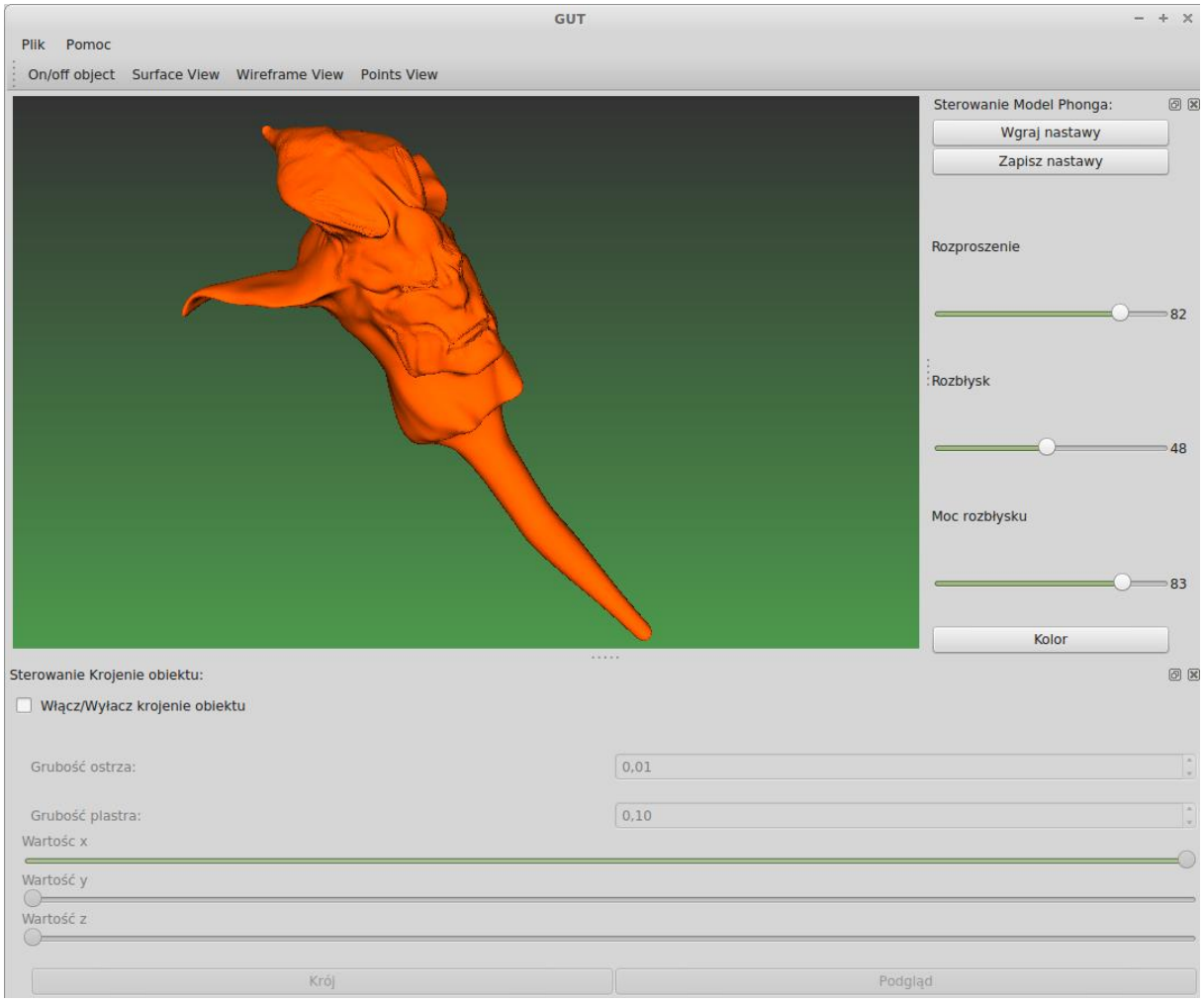
Zrzut 4: Program oferuje 3 sposoby wyświetlania obiektów typu PolyData. Drugi tryb to wizualizacja w widoku szkieletowym.



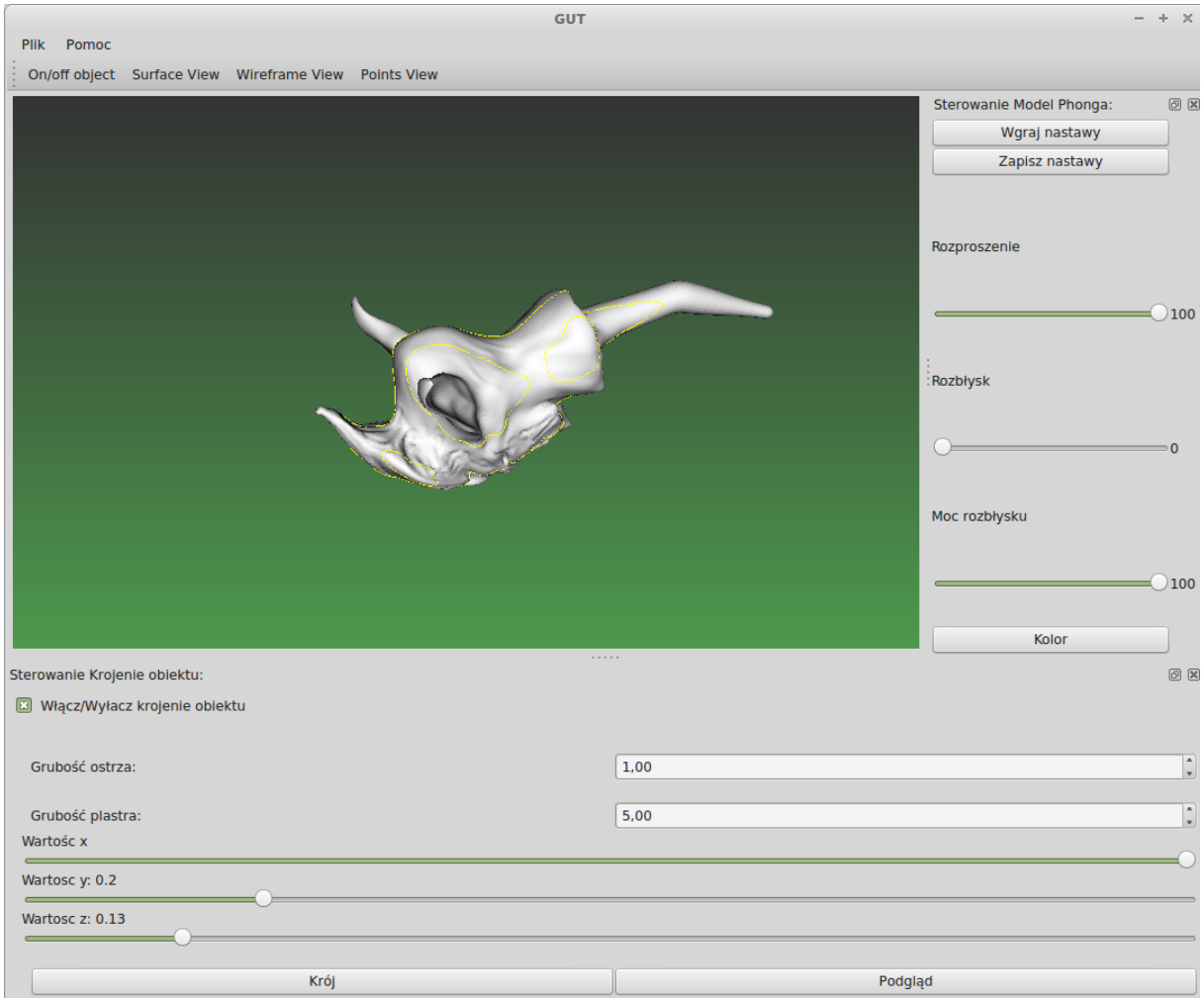
Zrzut 5: Program oferuje 3 sposoby wyświetlania obiektów typu PolyData. Trzeci tryb to wizualizacja punktowa.



Zrzut 6: Aplikacja umożliwia dowolną zmianę parametrów modelu Phonga. Do tego celu zostało stworzone dokowane menu z prawej strony okna głównego aplikacji. Możliwa jest zmiana rozproszenia, rozbłysku i mocy rozbłysku, a także zmiana sposobu kolorowania obiektu



Zrzut 7: Wygląd przykładowego obiektu po zmianie kolorowania powierzchni.



Zrzut 8: Program umożliwia wizualizację krojenia obiektów, jak i automatyczne krojenie całego obiektu o zadanej grubości plastra i noża. Na powyższym zrzucie widać przykładowy pogląd krojenia o zadanych przez użytkownika parametrach.

Poniżej przedstawione zostanie mała część zrzutów obrazujących krojenie automatyczne obiektu o zadanej grubości ostrza (0,01), grubości plastra (0,1) oraz w płaszczyźnie XZ (1,0,0)

Zrzut 9: Plaster wycięty z obiektu.

Zrzut 10: Plaster wycięty z obiektu.

Zrzut 11: Plaster wycięty z obiektu.

Zrzut 12: Plaster wycięty z obiektu.

Zrzut 13: Plaster wycięty z obiektu.

Zrzut 14: Plaster wycięty z obiektu.

Zrzut 15: Plaster wycięty z obiektu.

Zrzut 16: Plaster wycięty z obiektu.

Zrzut 17: Plaster wycięty z obiektu.

Zrzut 18: Plaster wycięty z obiektu.

Zrzut 19: Plaster wycięty z obiektu.