Projekt nr 3 - Anaglify

Bartłomiej Leśnicki

Paweł Bielecki

Mateusz Niepokój

1. Cel projektu

Celem projektu było przygotowanie programu, który na podstawie odpowiednio przygotowanych danych wejściowych, wyświetli na ekranie trójwymiarowy obraz i pozwoli nim obracać wokół trzech prostopadłych osi oraz na inne funkcjonalności.

2. Opis projektu

Widzenie przestrzenne jest wynikiem interpretacji jakiej dokonuje mózg porównując dwa, różniące się nieco obrazy pochodzące z każdego oka niezależnie. Tworzenie obrazów przestrzennych polega na przygotowaniu dwóch obrazów, oddzielnie dla lewego i prawego oka, a następnie wyświetlanie ich w taki sposób, aby odpowiedni obraz trafił do właściwego oka. Istnieje wiele metod realizacji tego zadania. Jedna z bardziej znanych polega na wyświetleniu każdego z obrazów w innych kolorach na tym samym monitorze, a następnie oglądaniu ich przez specjalne, dwukolorowe okulary. Obrazy takie nazywamy anaglifami. Celem naszego projektu było stworzenie programu wyświetlające anaglify wraz z dodatkowymi funkcjonalnościami. Program został napisany w języku C++ przy użyciu bibliotek wxWidgets oraz SFML.

3. Opis działania

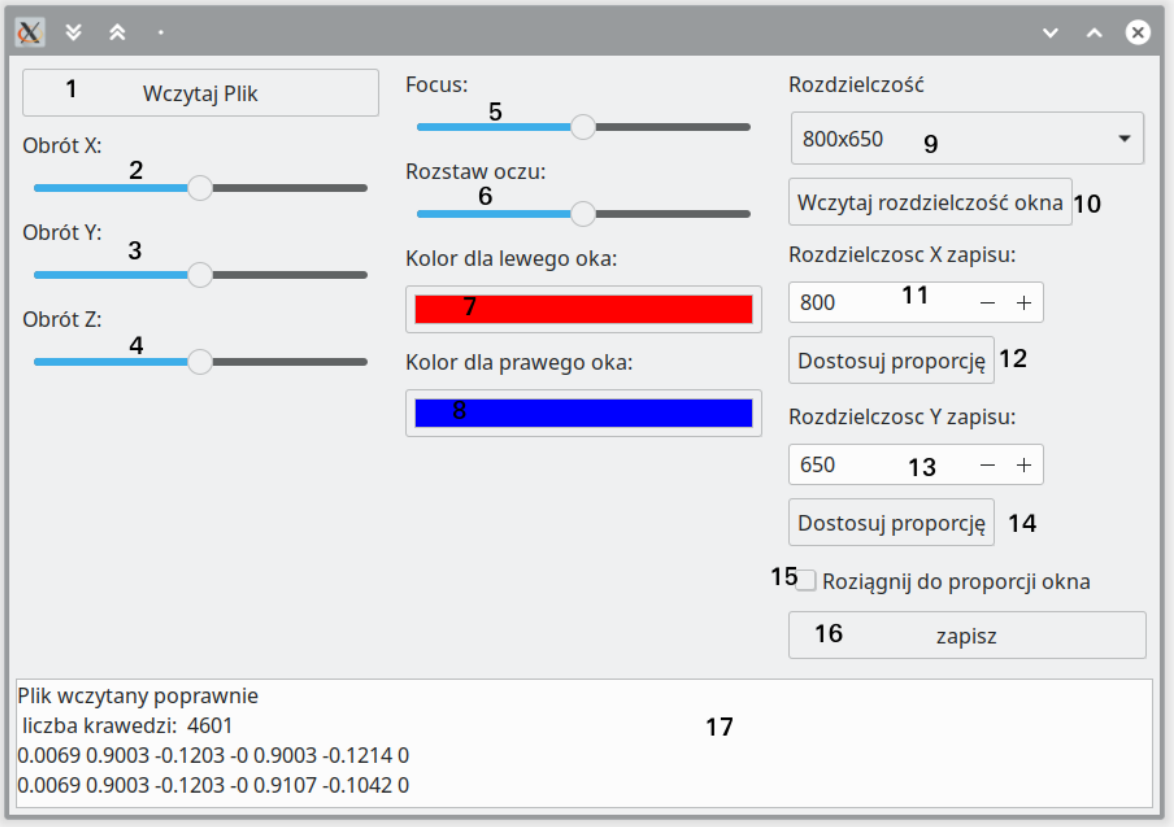
3.1 Wczytywanie pliku do programu:

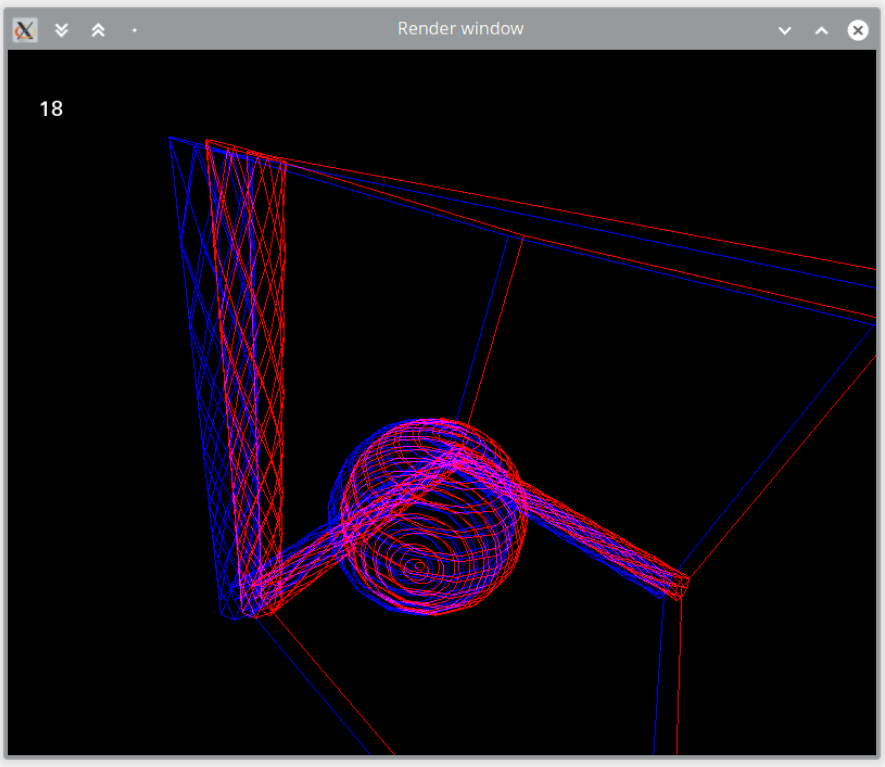
Program wczytuje odpowiednio przygotowany plik tekstowy:

- Jeśli wczytujemy bryłę szkieletową pierwszym znakiem w każdej linii jest cyfra 1 następnie dwie trójki liczb odpowiadają za krawędzie początku i końca odcinka. Ósma liczba mówi o grubości krawędzi. Wszystkie znaki później w linii traktowane są jako komentarz (nie wnoszą nic do działania programu).
- Jeśli wczytujemy kulę pierwszy znak linii to 2, następnie podawane są trzy liczby odpowiadające za położenie środka kuli, następnie jedna liczba określająca promień. Wszystkie znaki później w linii traktowane są jako komentarz (nie wnoszą nic do działania programu).

3.2 Interfejs graficzny:

Elementy interfejsu





1. Przycisk do wczytywania pliku z zapisaną geometrią
2. Suwak do obracania figury wokół osi X

   Każda jego zmiana powoduje zresetowanie zmian transformacji naniesionych przez okno renderowania - transformacja zostanie dostosowana do ustawień z menu.

3. Suwak do obracania figury wokół osi Y

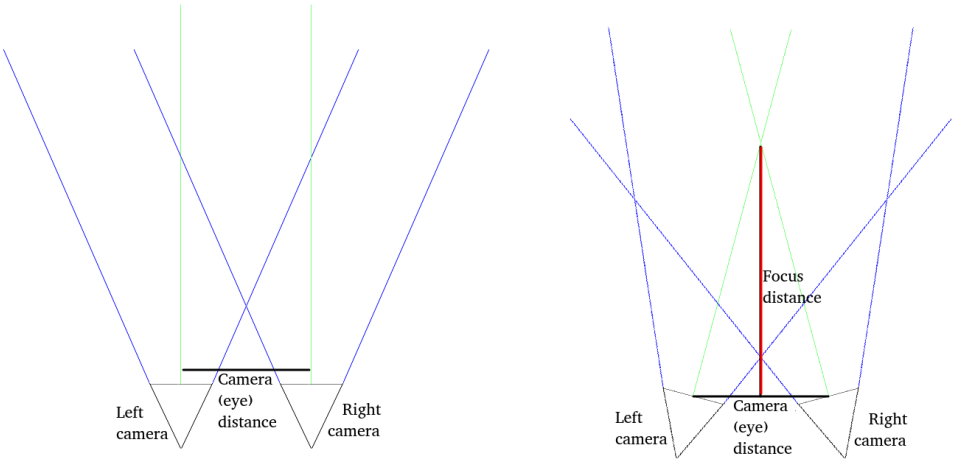
   Każda jego zmiana powoduje zresetowanie zmian transformacji naniesionych przez okno renderowania - transformacja zostanie dostosowana do ustawień z menu.

4. Suwak do obracania figury wokół osi Z

   Każda jego zmiana powoduje zresetowanie zmian transformacji naniesionych przez okno renderowania - transformacja zostanie dostosowana do ustawień z menu.

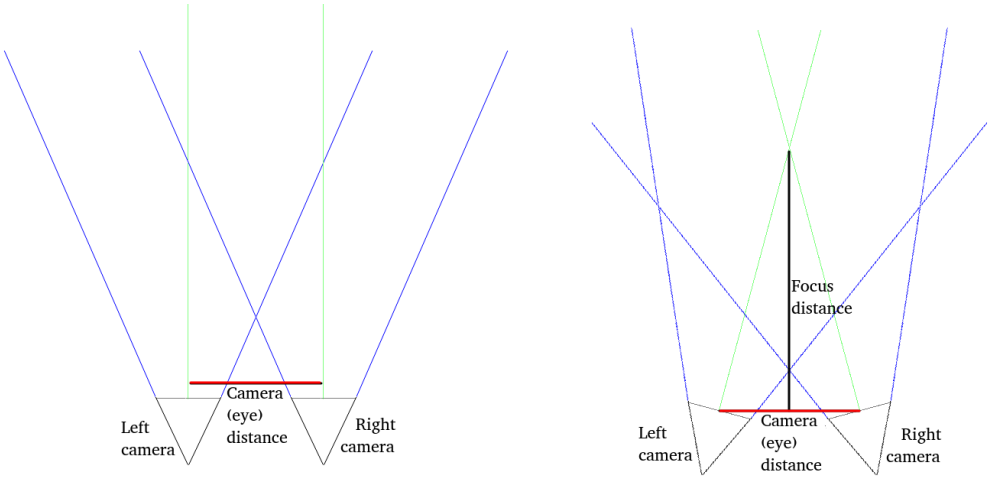
5. Suwak do ustawiania punktu skupienia osi optycznych kamer (focus)

   Ustawia punkt przecięcia się osi optycznych wirtualnych kamer.



6. Suwak do regulacji odległości między kamerami (oczyma)

   Ustawia szerokość między wirtualnymi kamerami.



7. Ustawianie koloru figury dla lewego oka
8. Ustawianie koloru figury dla prawego oka
9. Ustawianie predefiniowanych rozdzielczości

   Ustawia elementy 11 i 13 na zależną od wyboru jednej z kilku predefiniowanych rozdzielczości zapisu.

10. Przycisk ustawiający wielkość zapisywanego obrazu na rozdzielczość okna

    Dosłownie kopiuje wymiary okna do pól wyboru rozdzielczości zapisywanego obrazu.

11. Ustawianie szerokości zapisywanego obrazu
12. Przycisk ustawiający szerokość tak, aby proporcje obrazu były zgodne z proporcjami okna renderowania

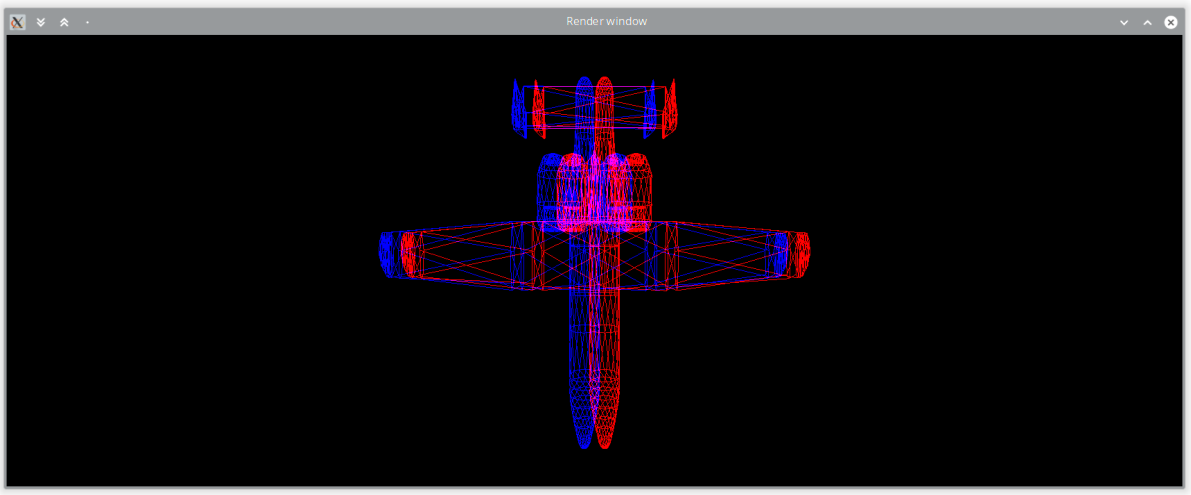
    Często się zdarza, że wybrane rozmiary zapisywanego rozmiaru mogą nie pokrywać się z proporcjami okna renderowania. Ten przycisk umożliwia dostosowanie szerokości tak, by stosunek szerokości do wysokości zapisywanego obrazu był równy stosunkowi tych samych wielkości renderowanego okna.

13. Ustawianie wysokości zapisywanego obrazu
14. Przycisk ustawiający wysokość tak, aby proporcje obrazu były zgodne z proporcjami okna renderowania

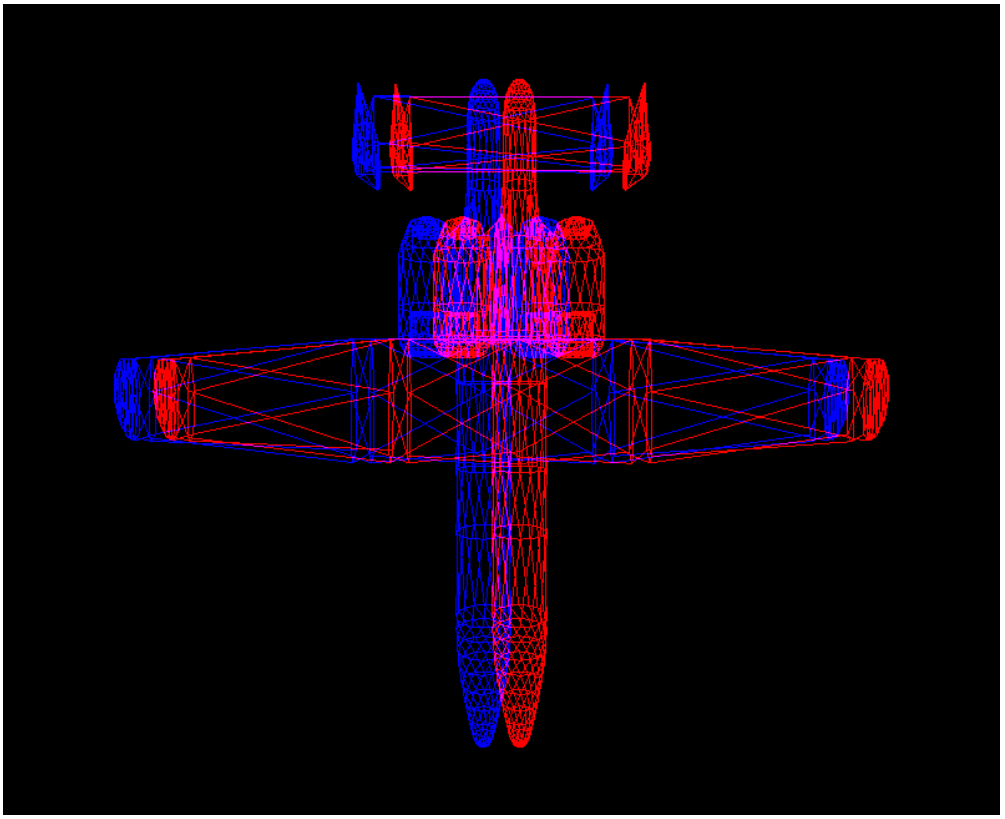
    Często się zdarza, że wybrane rozmiary zapisywanego rozmiaru mogą nie pokrywać się z proporcjami okna renderowania. Ten przycisk umożliwia dostosowanie wysokości tak, by stosunek szerokości do wysokości zapisywanego obrazu był równy stosunkowi tych samych wielkości renderowanego okna.

15. Zaznaczenie powoduje, że obraz z okna renderowania jest rozciągany do proporcji obrazu, jego odznaczenie zapobiega deformacji obrazu

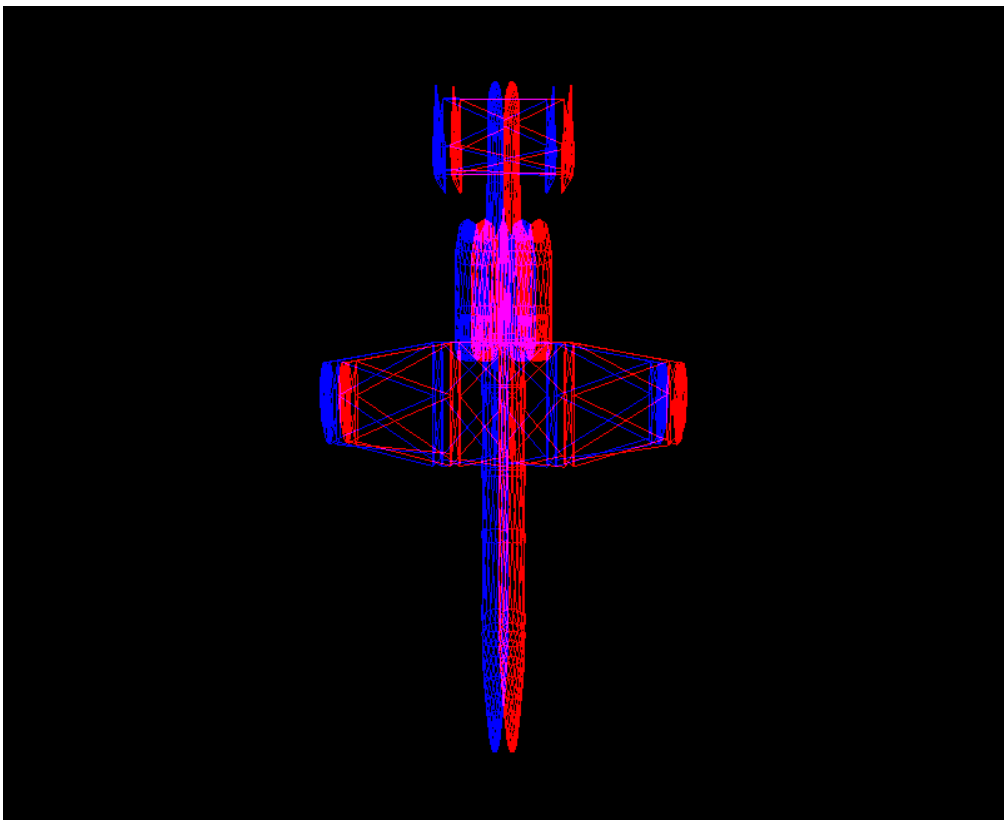
    Przykładowo dla nienaturalnie rozciągniętego okna:



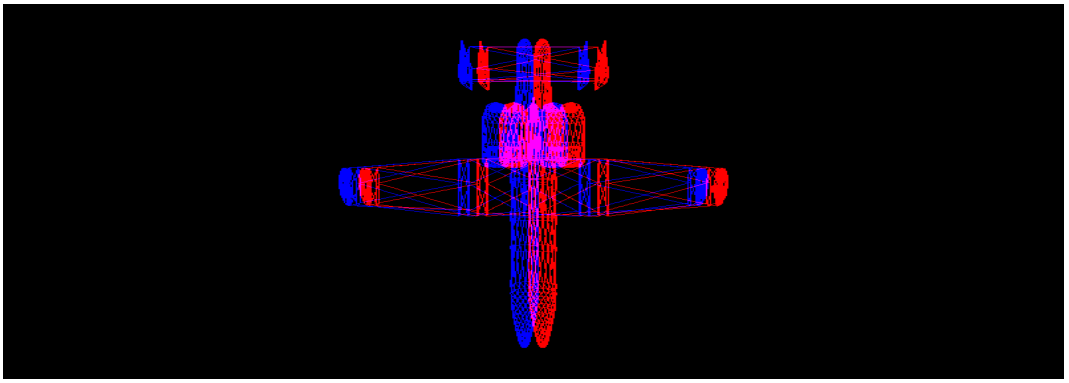
Mogą zostać zapisane obrazy o rozmiarze 800x650, które wyglądają następująco:
Bez zaznaczonej wskazanej opcji:



Z zaznaczoną wskazaną opcją:



Dzieje się tak dlatego, że drugi obraz po rozciągnięciu (przeskalowaniu) do proporcji renderowanego okna będzie wyglądał następująco:



Czyli będzie miał zachowane naturalne proporcje.

16. Przycisk do zapisywania obrazu

    Możliwe jest zapisanie wyrenderowanego obiektu do pliku. Obsługiwane rozszerzenia to: *png, bmp, jpg, jpeg*.

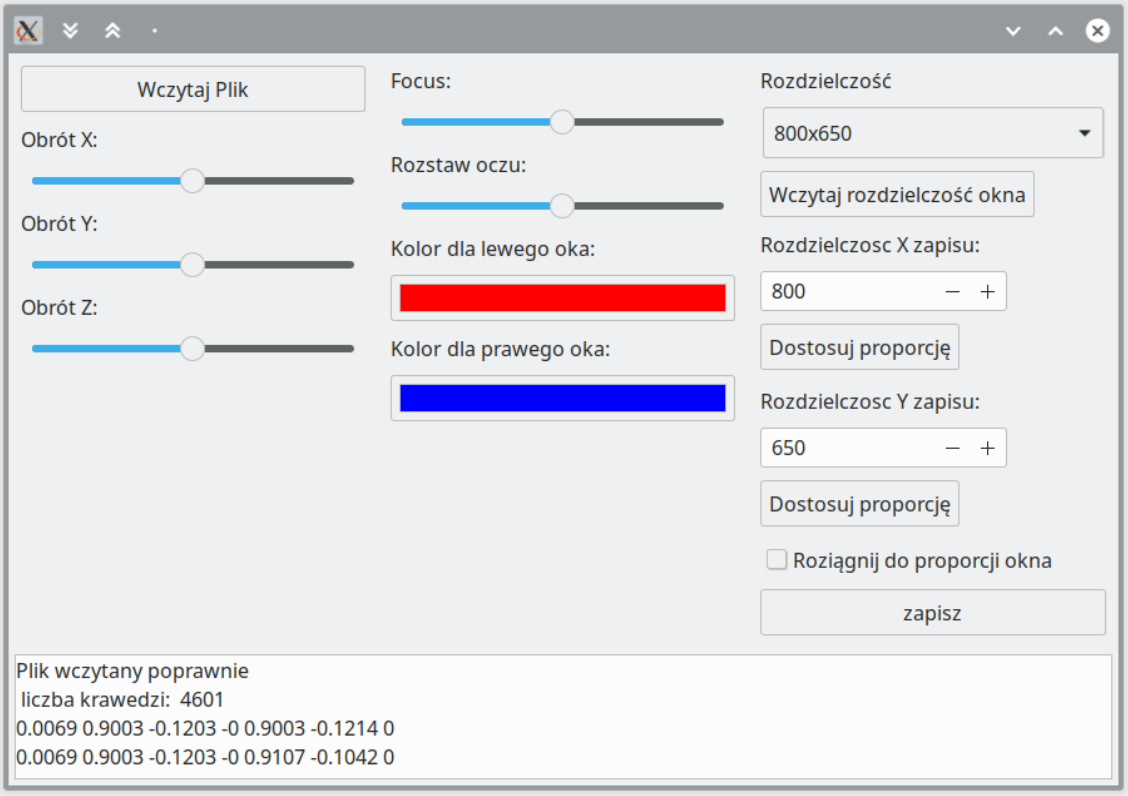
17. Konsola z informacjami pokazująca szczegóły wczytanego pliku i położenie wierzchołków.

    Praktycznie dla zwykłego użytkownika nie służy do niczego - pokazuje jedynie informacje, które można uznać za ciekawostki.

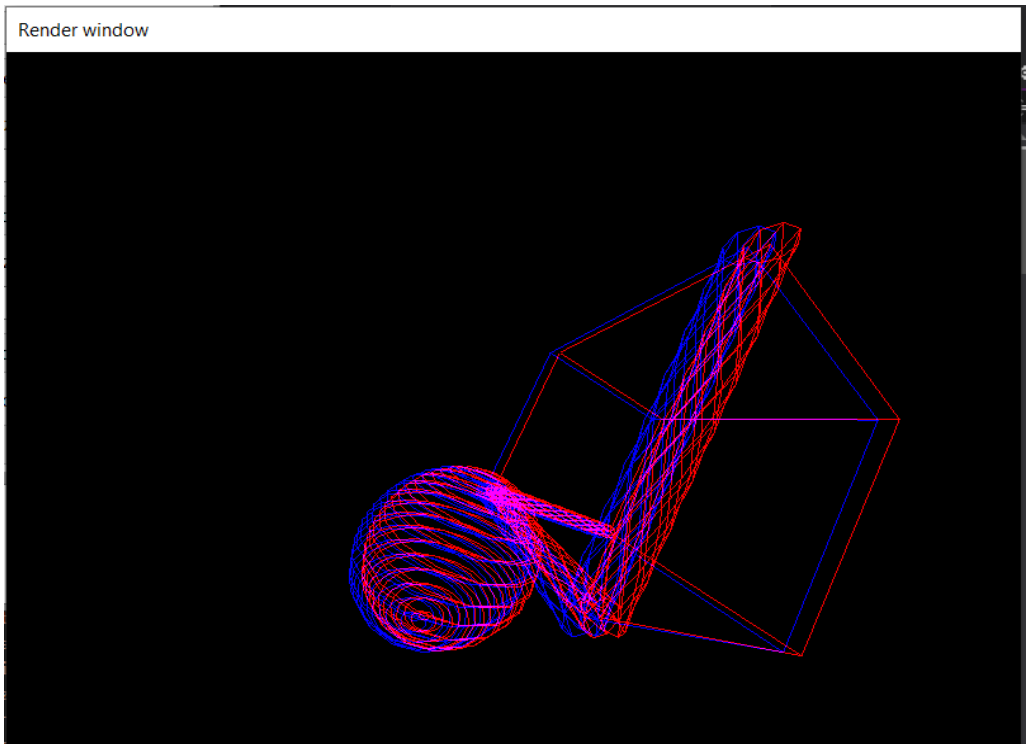
18. Okno renderowania

    Okno w którym pokazywany jest otwarty trójwymiarowy obiekt. Posiada możliwości manipulowania obrazem. Przeciąganie myszką z wciśniętym klawiszem powoduje obracanie obiektu. Puszczenie klawisza w trakcie ruchu powoduje animację dalszego obrotu, który z czasem ustaje. Można też wymusić zatrzymanie animacji poprzez kliknięcie lewym klawiszem myszy na okno. Używając scrolla można powiększać i pomniejszać obraz. Przy pomocy strzałek na klawiaturze można przesuwać obraz w górę, dół, oraz w prawo i w lewo w przypadku jakby podczas transformacji (obrotów) obiekt wyszedł poza widoczną część renderowania. Nie ma możliwości zamknięcia tego okna - zamyka się razem z programem, który zamyka się przez zamknięcie okna menu.

Zrzuty ekranu



Wygląd okna menu dla Linux Kubuntu 20.04 - w innych systemach / środowiskach graficznych program może wyglądać inaczej.



Wygląd okna renderowania dla Windows 10 - w innych systemach / środowiskach graficznych program może wyglądać inaczej.

3.3 Funkcjonalności programu:

Program po uruchomieniu włącza dwa okna. Pierwsze okno reprezentuje menu do sterowania programem. Drugie okno służy do renderowania brył wczytanych z pliku. Okna można dowolnie rozciągać. Okno do renderowania zostało zaimplementowane z przy użyciu biblioteki SFML natomiast okno z menu za pomocą wxWidgets.

Program umożliwia następujące funkcjonalności:

- Obrót wokół trzech prostopadłych osi: X,Y, Z za pomocą trzech suwaków. Bryłę można również obracać za pomocą myszki. Obrót za pomocą myszki jest animowany w zależności od szybkości ruchu myszką.
- Ustawianie odległości punktu przecięcia się osi optycznych kamer (suwaka “Focus”).
- Ustawienie odległości między wirtualnymi kamerami w celu dokładniejszego ustawienia lepszego widzenia trójwymiarowego (suwak “Rozstaw oczu”).
- Wybranie koloru brył do dostosowania do okularów 3D.
- Zapis do pliku.
- Ręczny wybór rozdzielczości zapisywanego obrazu.
- Dostosowanie proporcji zapisywanego obrazu.
- Rozciągnięcie zapisywanego obrazu do proporcji okna
- Po wczytaniu danych, dane są wypisywane w oknie menu.

4. Instrukcja kompilacji

4.1 Linux

Aby skompilować projekt w systemie Linux należy wpisać:
make all
w katalogu z plikiem *Makefile* (znajduje się on w głównym katalogu projektu). Do tego wymagany jest pakiet *make*, który może już być zainstalowany na komputerze, jednak w przypadku jego braku można go także zainstalować np.: przy użyciu komendy:
sudo apt update && sudo apt install make
do wykonania polecenia niezbędne są uprawnienia administratora.
Jeśli po wpisaniu *make all* zostają wypisane jakiekolwiek błędy (i jednocześnie w katalogu *bin/Linux* nie powstaje gotowy program), oznacza to, że nie są zainstalowane w systemie wymagane pakiety / biblioteki. Aby to zrobić należy wpisać komendę:
make install_packets
albo można zamiast tego wpisać następujący ciąg (odpowiadających) komend:
sudo apt update
sudo apt install make gcc g++
sudo apt install libwxgtk3.0-dev || sudo apt install libwxgtk3.0-gtk3-dev
sudo apt install libsfml-dev libsfml-window2.5 libsfml-system2.5 libsfml-graphics2.5

Może być wymagane wprowadzenie hasła administratora, oraz w trakcie procesu użytkownik może być pytany, czy zezwala na instalację poszczególnych pakietów

(i zajęcie przez nie dodatkowego miejsca na dysku).
Po ich zainstalowaniu można ponownie wpisać *make all*, co powinno skutkować powstaniem pliku wykonywalnego *main* w katalogu *bin/Linux*.

W przypadku problemów z uruchomieniem wersji programu skompilowanej na innym komputerze, należy także wpisać komendę *make install_packets* lub odpowiadający jej ciąg komend podany powyżej.

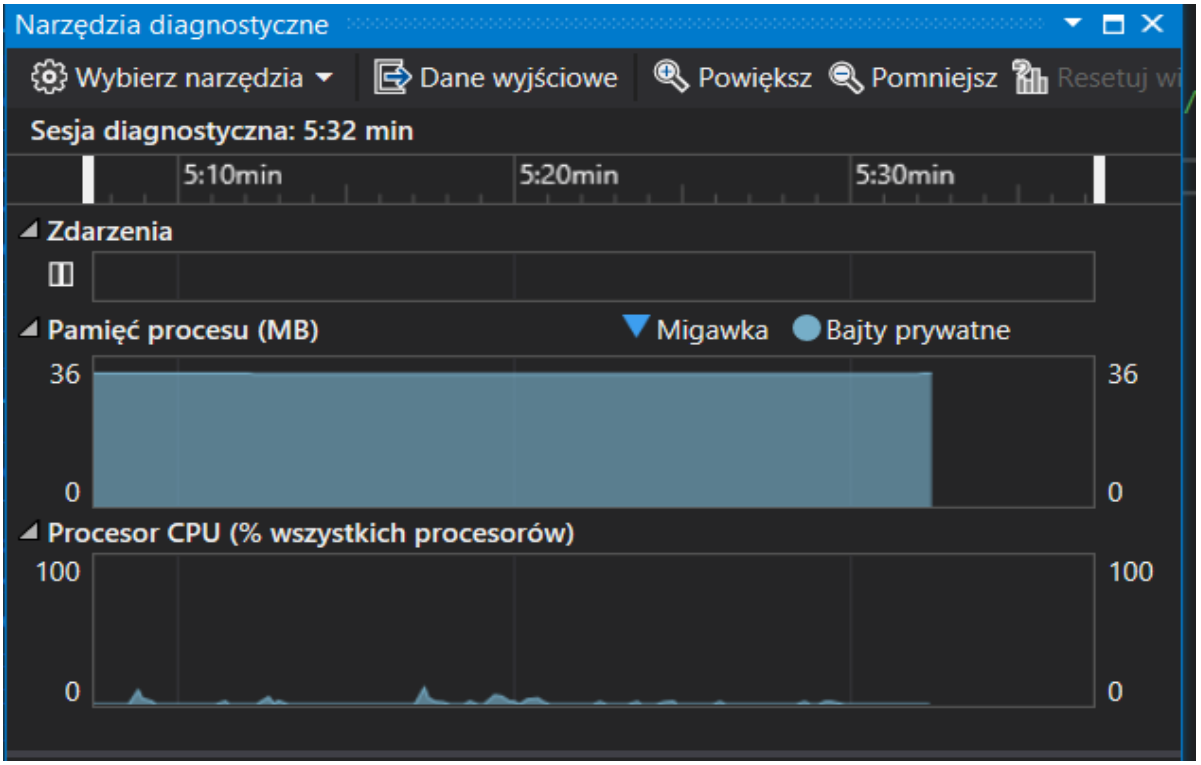
W przypadku problemów, szczególnie z komendami wymagającymi uprawnień nie posiadanych przez danego użytkownika, należy skontaktować się z administratorem danego komputera, ponieważ można nie być uprawnionym do nanoszenia zmian
w postaci instalowania dodatkowych bibliotek w systemie.

4.2 Windows

Aby skompilować program w systemie Windows należy w wybranym IDE obsługującym język c++ stworzyć projekt i dodać pliki .h i .cpp do projektu, następnie dodać potrzebne biblioteki (SFML i wxWidgets), oraz umieścić pliki dll w katalogu z plikiem wykonywalnym (.exe).
Na przykładzie Visual Studio tworzymy nowy projekt za pomocą: “Kreator aplikacji klasycznej systemu Windows”. Wpisujemy nazwę i wybieramy lokalizację projektu.
W dalszej części instrukcji jako nazwę projektu przyjmę “Projekt”. Klikamy “utwórz“ wybieramy “aplikacja klasyczna” i zaznaczamy dodatkowo “pusty projekt”. Następnie dodajemy pliki. To lokalizacji projektu. Można to zrobić następująco: klikamy prawym przyciskiem myszy na projekt w sekcji eksplorator rozwiązań (domyślnie znajduje się po lewej stronie) i wybieramy “otwórz folder w eksploratorze plików” następnie tam wypakowujemy wszystkie pobrane pliki .h i .cpp. Następnie w Visual studio ponownie klikamy prawym przyciskiem mysz na projekt w sekcji eksplorator rozwiązań i wybieramy “dodaj” i tam “istniejący element” przechodzimy do lokalizacji gdzie wypakowaliśmy pliki zaznaczamy wszystkie pliki .cpp i .h i klikamy “dodaj”. następnie klikamy na środku u góry i wybieramy opcje kompilacji debug x64. Następnie dodajemy biblioteki. Klikamy prawym przyciskiem mysz na projekt w sekcji eksplorator rozwiązań i wybieramy “właściwości”. We właściwościach konfiguracji w dziale Katalogi VC++ dodajemy:
ścieżki do wxWidgets\include\mscv i wxWidgets\include w dziale “katalogi plików nagłówkowych: oraz ścieżkę do \wxWidgets\lib\vc_x64_lib w dziale “katalogi bibliotek”
w dziale “C\C++” “ogólne” “dodatkowe katalogi plików nagłówkowych” \SFML-2.5.1\include
w dziale “C\C++ “generowanie kodu” “biblioteka środowiska uruchomieniowego” wybieramy “debugowanie wielowątkowe (/Mtd)”
W dziale “konsolidator” “ogólne” “Dodatkowe katalogi bibliotek” dodajemy ścieżkę do SFML-2.5.\lib oraz wxWidgets\lib\vc_x64_lib.
W dziale “konsolidator” “dane wejściowe” “dodatkowe zależności” dodajemy:
sfml-graphics-d.lib
sfml-window-d.lib
sfml-system-d.lib
Następnie wystarczy tylko dodać pliki .dll do miejsca lokalizacji pliku wykonywalnego (...\projekt\x64\Debug).

5. Testy Wydajności Programu

Wszystkie przedstawione funkcjonalności działają poprawnie, wszystkie cele projektu zostały zrealizowane. Do projektu zostały dołączone trzy pliki z danymi wejściowymi. Pierwszy z nich o nazwie plik.txt pokazuje możliwości rysowania programu. Drugi z nich o nazwie samolot.txt pokazuje natomiast jakość optymalizacji projektu. Trzeci plik jest identyczny z pierwszym, jednak zawiera kilka dodatkowych komentarzy na końcu, które zostały dodane w celu sprawdzenia, czy program ignoruje zbędne dane. Działanie programu jest bardzo wydajne (wszelkie wyświetlanie, szczególnie animacje są całkowicie płynne). Program zużywa mało pamięci, użycie procesora jest na bardzo niskim poziomie mimo zastosowania do obliczeń ośmiu wątków, co teoretycznie umożliwia użycie znacznej mocy obliczeniowej (w przypadku procesora ośmiowątkowego nawet teoretycznie do 100%).



6. Podział pracy i analiza czasowa

Zadanie realizowaliśmy przez 4 tygodni. Projekt wykonywaliśmy przy pomocy platformy GitHub, przez którą zsynchronizowaliśmy kolejne wersje projektu.
Tydzień 1.

- Zaprojektowanie schematu projektu.
- Podział zadań do wykonania.
- Napisanie podstawowego podstawowego GUI.
- Implementacja funkcji obliczających przekształcenia i rysujących model.
- Implementacja funkcji do wczytywania danych.
- Napisanie pliku z przykładowymi danymi wejściowymi.

Tydzień 2.

- Implementacja elementów GUI.
- Implementacja animacji ruchu.
- Implementacja kuli.
- Implementacji ustawień grubości linii.
- Dodanie możliwości wyboru rozdzielczości zapisu pliku.

Tydzień 3.

- Wytworzenie drugiego pliku z przykładowymi danymi.
- Naprawienie znalezionych błędów.
- Ponowne rozmieszczenie elementów interfejsu użytkownika.

Tydzień 4.

- Końcowe poprawki do projektu.
- Napisanie sprawozdania z projektu.

Rozkład zadań prezentował się następująco:

- Bartłomiej Leśnicki: stworzenie GUI, podstaw projektu oraz repozytorium Git. Implementacja wczytywania danych, przygotowanie przykładowej bryły szkieletowej do wczytania.

- Paweł Bielecki: ogólny projekt działania aplikacji (podział funkcjonalności bibliotek i zaprojektowanie obsługi komunikatów dla dwóch okien jednocześnie), stworzenie klasy odpowiedzialnej za renderowanie (wyświetlanie, zapisywanie), stworzenie animacji, proceduralne projektowanie kul i grubszych odcinków (walców), zaprojektowanie systemu realizacji zadań na wielu wątkach jednocześnie, pomoc przy tworzeniu GUI menu, dostosowanie projektu dla Linux, optymalizacja (wyrzucenie niepotrzebnych zmiennych, zamknięcie zmiennych globalnych w klasach).

- Mateusz Niepokój: obsługa kontrolek GUI, pomoc przy tworzeniu GUI, przygotowanie przykładowego pliku wejściowego, wykorzystującego znaczną liczbę krawędzi

7. Kodowanie

Klasy:

- **MyFrame1** – jest to klasa wygenerowana automatycznie przez wxFormBuilder, zawierająca wszystkie elementy interfejsu

- **GUIMyFrame1** – klasa dziedzicząca po MyFrame1, zawierająca implementację elementów interfejsu
    - Rotacja – pomocnicza funkcja zwracające macierz obrotu
    - ViewWindow* viewWindow – przechowuje klasę rysującą anaglif
    - Std::vector<section> data – przechowuje wczytane krawędzie
    - Unsigned liczba_k – zmienna przechowująca liczbę wczytanych krawędzi
    - wczytajOnButtonClick – metoda wczytująca dane
    - zapiszOnButtonClick – metoda zapisująca obraz w formacie png w zadanej rozdzielczości
    - sfmlTimerOnTimer – metoda wewnątrz której obsługiwane jest okno rysujące
    - FitProportionXBtnOnButtonClick – metoda, która dopasowuje rozmiar zapisywanego pliku, według proporcji okna rysującego, względem współrzędnej X
    - FitProportionYBtnOnButtonClick - metoda, która dopasowuje rozmiar zapisywanego pliku, według proporcji okna rysującego, względem współrzędnej Y
    - closeRenderWindow – metoda, do zamykania okna rysujące
    - openRenderWindow – metoda, do otwierania okno rysywania
    - wxTimer sfmlTimer - komponent odpowiedzialny za wykonywanie metody odpowiedzialnej za przetwarzanie komunikatów w oknie biblioteki SFML (w tym także renderowanie obiektu w oknie).
- **Point** - to struktura reprezentująca punkt w trójwymiarowej przestrzeni

- **Section** – struktura przechowująca odcinek w postaci dwóch punktów: początku i końca odcinka

- **Ray** – metoda zwracająca odległość punktu od początku układu współrzędnych

- **ViewWindow** – klasa rysująca anaglif
  Zawiera składniki:
    - double Zoom – zmienna odpowiadająca za przybliżenie lub oddalenie anaglifu
    - double eyeDistance - zmienna odpowiadająca za odległość między kamerami(oczami)
    - double eyeTarget- zmienna odpowiadająca za punkt skupienia optyczn kamer
    - double rotationDensity – zmienna odpowiadająca za czułość (zależność prędkości obrotowej od prędkości myszki w pikselach) obrotu kamery
    - double rotationResistance – zmienna odpowiadająca za stopniowe wytracanie prędkości obrotu
    - sf::RenderWindow window – okno wewnątrz którego rysowany będzie anaglif

- sf::VertexArray rightVertexArray – tablica zawierająca przekształcone punkty do rysowania dla prawego oka
- sf::VertexArray leftVertexArray – tablica zawierająca przekształcone punkty do rysowania dla lewego oka
- bool mouseButtonIsDown – flaga sprawdzająca czy mysz przycisk myszy jest naciśnięty
- int mousePositionX, mousePositionY – aktualna pozycja myszy
- ParallerMultiplier multipliers – obiekt klasy obliczającej wielowątkowo przekształcenia punktów
- Point center – centralny punkt obiektu, wokół którego następują przekształcenia (obroty)
- Matrix4 translationMatrix – macierz przesunięcia, potrzebna do przekształceń
- Matrix4 rotationMatrix macierz obrotu, potrzebna do przekształceń
- Matrix4 mainMatrix – główna macierz przekształcenia
- double rotationSpeedX, rotationSpeedY – zmienna służące do płynnego obracania anaglifu myszką (i podtrzymywania ewentualnej późniejszej animacji samoczynnego obrotu)
- double offsetX, offsetY – zmienne służące do przemieszczania renderowanego obrazu po oknie (jeśli by podczas transformacji "uciekł" z widoku)

Zawiera metody:

- processMessages – metoda wewnątrz której obsługiwane są zdarzenia, obliczane są nowe współrzędne odcinków, które następnie są rysowane
- heartBeat – funkcja wykonywana z każdym przetwarzaniem komunikatów, wewnątrz niej obliczana jest macierz przekształceń oraz zmniejszana jest prędkość obrotu,
- setData – metoda przekazująca tablicę odcinków z klasy GuiMyFrame do obiektu multipliers
- Update – metoda służąca do ustawiania przekształceń macierzowych (z zewnątrz)
- SaveToFile – metoda zapisuje aktualną zawartość okna do pliku
- UpdateEyeMatrixes – metoda służąco do obliczenia macierzy dla lewego i prawego oka
- Render – oblicza nowe współrzędne odcinków na podstawie macierzy przekształceń
- getSize – zwraca rozmiar okna w pikselach
- RenderTo – rysuje przekształcone odcinki do wskazanego celu
- HandleEvents – obsługuje zdarzenia okna window
- Paint – metoda, która rysuje nową klatkę i wyświetla ją w oknie

- **Vector4** - struktura przechowująca 4-wymiarowy wektor

- **Matrix4** – struktura reprezentująca macierz o wymiarach 4x4

- **ParallerMultiplier** - klasa służy do renderowania modeli trójwymiarowych na obiekty typu *VertexArray*, które są obsługiwane przez SFML.
  Zawiera składniki publiczne:

- leftColor, rightColor - przechowują informacje o kolorach, na jakie mają być pomalowane odcinki dla każdego oka.
- leftMatrix, rightMatrix - przechowują macierze przekształceń dla każdego oka z osobna.
- vertexArrayLeft, vertexArrayRight - są to obiekty zawierające przetworzone dane, gotowe do wyświetlenia przez SFML.
- sections - wskaźnik do wektora zawierającego listę odcinków obiektu.
- cameraDepth - głębokość kamery - jako że wirtualna kamera jest graniastosłupem, ten parametr określa jego wysokość. W praktyce oznacza to regulację kąta widzenia.

Zawiera metody:

- ParallerMultiplier - konstruktor, przyjmuje jako parametr liczbę wątków, domyślnie osiem.
- MultiplePoint - rzutuje zadany odcinek zadany w *sections* na *vertexArrayLeft* i *vertexArrayRight* mnożąc je przez odpowiednie macierze. Może działać asynchronicznie i rownolegle.
- Done - zwraca prawdę, jeśli zakończono obliczenia, w przeciwnym wypadku zwraca fałsz.
- getCalcID - zwraca identyfikator obliczeń - zwiększa się wraz z każdym zleceniem ponownego wykonania obliczeń. Dzięki obserwacji tej metody, poszczególne wątki mogą zauważyć, że jest coś nowego do obliczenia.
- Terminating - na ogół zwraca fałsz, jednak od momentu rozpoczęcia wykonywania destruktora, do końca, zwraca prawdę. Jest to sygnał dla wątków, że należy wyjść z pętli i zakończyć działanie.
- getPointCount - zwraca liczbę odcinków.
- getThreadCount - zwraca liczbę utworzonych wątków.
- wait - czeka na zakończenie obliczeń.
- DoneStatus - sprawdza czy wątek o danym indeksie zakończył już swoje obliczenia.
- setColors - ustawia kolory dla każdego oka.

- **SingleMultiplier**
  Zawiera metody:
  - SingleMultiplier - konstruktor, przyjmuje informację, którym jest indeksem, oraz przyjmuje referencję do *ParallerMultiplier*.
  - operator() - wykonuje się w nim główna pętla wątku.

Moduły z funkcjami:

- StructureMakers
    - makeSphere
      Dla podanej lokalizacji środka kuli, jej promienia, oraz zadanej gęstości, tworzy przybliżenie kuli złożonej z odcinków (klasa *Section*). Odcinki są zapisywane
      w wektorze przekazanym w ostatnim parametrze.
    - makeSection
      Dla podanych współrzędnych początku i końca odcinka, jego grubości, oraz zadanej gęstości, tworzy odcinek, którym jest pojedynczy odcinek klasy *Section* dla grubości równej zero, lub przybliżenie walca o promieniu równym zadanej grubości, złożone z wielu odcinków klasy *Section*. Odcinki są zapisywane w wektorze przekazanym w ostatnim parametrze.

- wec - za wyjątkiem klas zawiera także funkcje do ich przygotowywania i obsługi:
    - CreateRotationMatrix – na podstawie argumentów tworzy odpowiednią macierz rotacji o zadany kąt wokół wybranej osi
    - CreateMoveMatrix – na podstawie argumentów tworzy odpowiednią macierz translacji
    - CreateScaleMatrix – na podstawie argumentów tworzy odpowiednią macierz skalowania
    - CreateRotationMatrixFromVector – na podstawie wektora tworzy macierz obrotu wokół tego wektora o kąt zadany przez jego długość
    - IdentityMatrix - zwraca macierz jednostkową

8. Wdrożenie, raport i wnioski

- W programie udało nam się wykonać wszystkie zadania podstawowe. Możliwe jest rysowanie linii wczytanych z pliku, obracanie szkieletu wokół trzech prostopadłych osi oraz kalibracja obrazu do posiadanych okularów. Warto tutaj zwrócić uwagę na dobrą optymalizację programu. Program działa płynnie nawet dla dużej liczby krawędzi i jednocześnie nie zużywa znacznej mocy obliczeniowej.

- Z wymagań rozszerzonych wprowadziliśmy możliwość zmiany grubości linii, wyświetlanie kul, możliwość zbliżania oraz oddalanie obiektu, płynnego poruszania się po układzie oraz zmiany rozdzielczości zapisywanych obrazów.