Podstawy Grafiki Komputerowej

Dokumentacja do projektu 45

Prosty edytor grafiki 3D

Piotr Deda, Jakub Kraśniak, Łukasz Bartoszek

15 czerwca 2021

Spis treści

1 Opis Projektu

Celem projektu jest napisanie prostego edytora grafiki 3D.
Okno programu dzieli się na 3 główne części:

1. Konsole do wpisywania komend
2. Listę rysowanych obiektów wraz z podstawowymi informacjami
3. Główne panel składający się z 4 mniejszych okienek, w których jest podgląd na rysowane obiekty z różnych perspektyw

W celu narysowania danego obiektu w oknie należy podać odpowiednią komendę w konsoli. Podstawowe informacje dotyczące obiektu pojawiają się na liście wraz z przypisanym id, które ułatwia późniejszą jego edycję. Argumenty są bezpośrednio uzależnione od wywoływanej funkcji rysującej.

2 Założenia wstępne przyjęte w realizacji projektu

Program daje podgląd na rysowane obiekty z 3 różnych perspektyw, wzdłuż osi X, wzdłuż osi Y, oraz wzdłuż osi Z, które wyświetlane są w trzech okienkach. Czwarte z nich wypełnia rysowanie z uwzględnieniem perspektywy z punktu (1, 1, 1) (ten podpunkt był jednym z wymagań rozszerzonych i nie został w pełni zrealizowany). Program oferuje wybór koloru linii obiektów oraz rysowanie następujących figur:

1. odcinek
2. prostopadłościan
3. sfera
4. stożek (ścięty)
5. cylinder

Każdy z obiektów można usuwać, obracać, oraz przesuwać. Sam program oferuje zapisywanie projektu do pliku, oraz jego ponowny odczyt.

3 Analiza projektu

3.1 Specyfikacja danych wejściowych

3.1.1 Konsola

Główny sposób komunikacji polega na pisaniu komend w konsoli. Poszczególne parametry komend są rozdzielone pojedynczą spacją. Jeśli parametr jest postaci (x,y,z), to pomiędzy przecinkami nie może być spacji. Zaimplementowane komendy to:

- Komendy podstawowe
  - set_line_color (r,g,b)
  - line (x1,y1,z1) (x2,y2,z2)
  - box (x1,y1,z1) (x2,y2,z2)
  - sphere (x,y,z) r (n,m)
  - cone (x1,y1,z1) r1 (x2,y2,z2) r2 n
  - cylinder (x1,y1,z1) (x2,y2,z2) r n
  - delete id

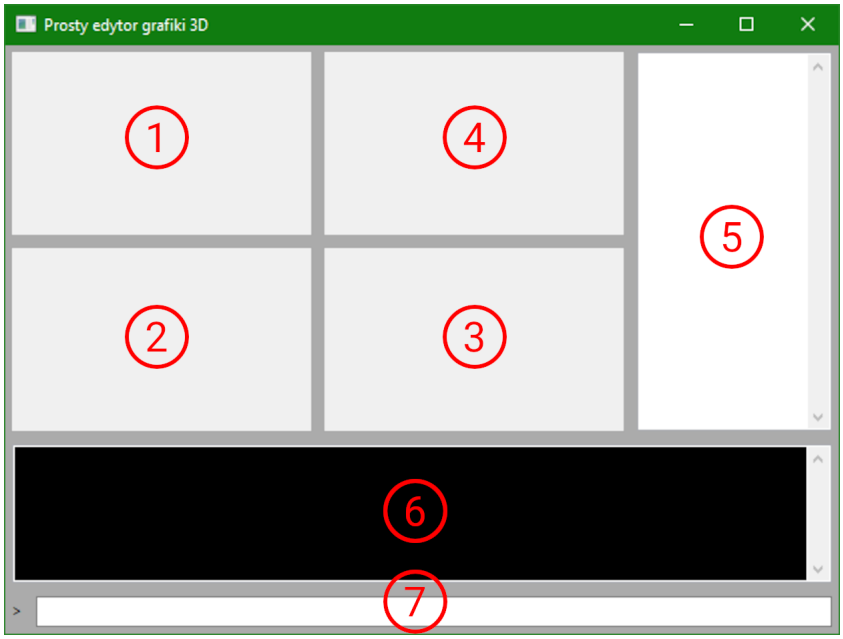
  - clear_all
  - move id (x,y,z)
  - rotate id (x,y,z) (α,β,γ)
  - save name
  - load name
- Komendy rozszerzone
  - set_view_range right — front — top r
  - camera_at (x,y,z)
  - camera_fov alfa
- Komendy debugowania (nieuwzględnione w opisie projektu, używane do testowania programu)
  - debug_scale
  - debug_culling

Konsola zaprojektowana jest w sposób pozwalający łatwo wykorzystać ją w dowolnym projekcie *wxWidgets* – wykorzystanie polega na utworzeniu obiektu konsoli, jako parametr przesyłając wxTextCtrl, który będzie używany do wypisywania tekstu. Klasa udostępnia metody pozwalające na zarejestrowanie komendy, przesłanie komendy (pobieranie komend program realizuje samodzielnie) i wypisanie tekstu w dowolnym kolorze (typu wxColour).
Konsola posiada wbudowane sprawdzanie poprawności wysyłanego tekstu – w przypadku wpisania niepoprawnej nazwy komendy lub błędnych parametrów poinformuje użytkownika o błędzie czerwonym tekstem.

3.1.2 Odczyt z pliku

Aplikacja pozwala na zapis i odczyt danych z pliku. Wczytywane dane to wszystkie utworzone obiekty, używany kolor linii i dane kamer (odległość w przypadku płaskich widoków, pozycja i pole widzenia w przypadku kamery perspektywicznej), zastępują one wszystkie poprzednie obiekty i wartości. Dane są zapisane czystym tekstem, przez co można je modyfikować, jednak jeśli interpreter natknie się na linijkę, której nie potrafi odczytać, plik nie zostanie wczytany, o czym użytkownik zostanie poinformowany w konsoli.

3.2 Specyfikacja interfejsu użytkownika



Rysunek 1: Interfejs użytkownika

1. Okno z widokiem z góry
2. Okno z widokiem z przodu
3. Okno z widokiem z prawej
4. Okno z widokiem perspektywicznym
5. Lista brył
6. Wyjście konsoli
7. Pole do wprowadzania komend

3.3 Wyodrębnienie i zdefiniowanie zadań

3.3.1 Interfejs

Interfejs został wygenerowany przy pomocy *wxFormBuilder*, zawiera 3 wcześniej opisane części.

3.3.2 Konsola

Konsola do wpisywania komend, która automatycznie dzieli je na funkcje, oraz przekazywane im argumenty. Umożliwia to szybką komunikację użytkownika z programem.

3.3.3 Rysowanie

Realizacja rysowania obiektów w 4 okienkach, oparta o osobną metodę dla każdego z nich.

3.3.4 Obiekty

Utworzenie klasy wirtualnej będącej bazą dla każdej z figur, oraz kontenera przechowującego wszystkie figury wraz z ich zmianami.

3.3.5 Transformacje

Utworzenie metod umożliwiających edytowanie obiektów poprzez usunięcie, przesunięcie, oraz obrót.

3.3.6 Zapisywanie i wczytywanie pliku

Zapisywanie aktualnej zawartości programu w pliku zrealizowane przy pomocy konsoli, oraz możliwość późniejszego go wczytania.

3.4 Narzędzia programistyczne

Użyte przez nas środowisko programistyczne to *Microsoft Visual Studio 2019* wraz z wbudowanym kompilatorem. Zdecydowaliśmy się na nie ze względu na wygodę, oraz wsparcie dla repozytorium *Git* i jego hostingu *GitHub*, które pozwoliło nam na automatyzację wspólnego tworzenia projektu. Do realizacji naszego wykorzystaliśmy bibliotekę *wxWidgets*, oraz przeznaczony dla niej *wxFormBuilder* do projektowania i generacji interfejsu graficznego.
Cały projekt napisany został w języku *C++* w standardzie *C++17*.

4 Podział prac i analiza czasowa

Pracę nad projektem rozpoczęliśmy od wspólnego stworzenia i omówienia interfejsu w programie *wxFormBuilder*.
Następnie zaprogramowaliśmy konsolę, która zwracała ostrzeżenia w przypadku wywołania niepoprawnych komend, oraz była w stanie wywołać podstawowe funkcje i metody potrzebne do rysownia. Jej dalszy rozwój było prowadzony równolegle do dalszych prac nad programem
Po utworzeniu konsoli i jej podstawowych funkcji zaczęliśmy rozwijać pozostałe elementy programu, zaczynając od wirtualnej klasy kształtu i dziedziczących po niej klas jak linia, czy prostopadłościan. Jednocześnie pracowaliśmy nad rozwojem rysowania obiektów w przypadku którego niezbędny okazał się kontener na obiekty. Prace zakończyliśmy na bardziej skomplikowanych figurach, ich transformacjach, oraz zapisie i odczycie z pliku.
Napisanie projektu zajęło około 2 tygodni, Piotr napisał główną strukturę programu, Jakub tworzenie obiektów, a Łukasz dokumentację.

5 Opracowanie i opis niezbędnych algorytmów

5.1 Perspektywa

Perspektywa oparta jest na prostych wzorach, służących uwzględnieniu współrzędnej z, mimo prezentowania na układzie x-y:

$$\begin{aligned} x' &= x \cdot \frac{\text{fov}}{-z_{cam} + z} - x_{cam}, \\ y' &= y \cdot \frac{\text{fov}}{-z_{cam} + z} - y_{cam}. \end{aligned}$$

5.2 Algorytmy konstruowania figur

5.2.1 Odcinek i Prostopadłościan

Odcinek konstruujemy trywialnie jako jeden segment łączący dwa punkty podane jako argumenty.
Konstrukcja prostopadłościanu polega na wyrażeniu jego wszystkich narożników za pomocą koordynatów narożników przeciwległych podanych jako argumenty oraz połączenie odpowiednich narożników segmentami.

5.2.2 Stożek i walec

Budowanie podstaw stożka i cylindra oparte jest na algorytmie znanym jako *midpoint circle algorithm*. W ogólności służy do rasteryzacji okręgów o zadanym promieniu – u nas tym promieniem jest parametr n, a uzyskana jest figura jest skalowana do parametrów reprezentujących promień później.

5.2.3 Sfera

Konstrukcja sfery rozpoczyna się od wyznaczenia zbioru punktów na tej sferze, pomiędzy którymi będą przebiegały segmenty. Współrzędne tych punktów wyrażają się wzorami:

$$\begin{aligned} x &= r \cdot sin(\alpha) \cdot cos(2 \cdot \beta) \\ y &= r \cdot cos(\alpha) \\ z &= r \cdot sin(\alpha) \cdot sin(2 \cdot \beta) \end{aligned}$$

gdzie α to kąt 180 stopni podzielony na ilość równoleżników, a β to kąt 360 stopni podzielony na ilość południków. Następnie idąc od górnego czubka sfery, po równoleżnikach, prowadzimy segmenty z punktów na aktualnym równoleżniku do odpowiadających im punktów na kolejnym. Poza tym, na każdym takim poziomie łączymy w segmenty kolejne pary punktów leżących na równoleżniku.

5.3 Algorytmy transformacji figur

5.3.1 Przesunięcie

Przesunięcie figury o dany wektor polega na przesunięciu każdego z jej segmentów o ten wektor. Odbywa się to poprzez dodanie do punktów będących końcami segmentu, współrzędnych wektora podanego jako argument.

5.3.2 Obrót

Obrót figury wokół danego punktu o współrzędnych (x,y,z) odbywa się poprzez przesunięcie figury o wektor (-x,-y,-z), następnie obrócenie jej o dane kąty względem każdej z osi, przykładowo dla obrotu wokół osi Z:

$$\begin{aligned} x' &= x \cdot cos(\gamma) - y \cdot sin(\gamma) \\ y' &= x \cdot sin(\gamma) + y \cdot cos(\gamma) \end{aligned}$$

Na koniec ponowne przesunięcie, tym razem o wektor (x,y,z).

6 Kodowanie

Dokumentacja wygenerowana z pomocą *Doxygen* znajduje się w pliku Doxy.pdf.

7 Testowanie

Testowanie naszego programu rozpoczęliśmy od sprawdzenia błędnie wpisanych komend.

Rysunek 2: Przykładowe komunikaty dla błędnych danych wejściowych

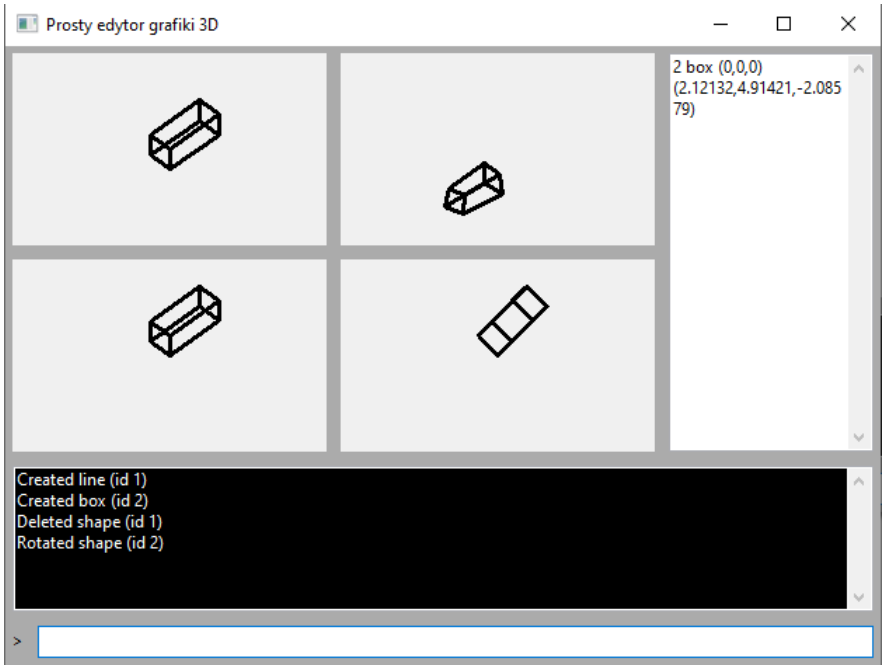
Jak można zauważyć, konsola informuje nas o błędach.

Następnie sprawdziliśmy generację brył, zaczynając od prostej.

Rysunek 3: Przykładowa linia

Prosta została wygenerowana poprawnie (Rys. 3).

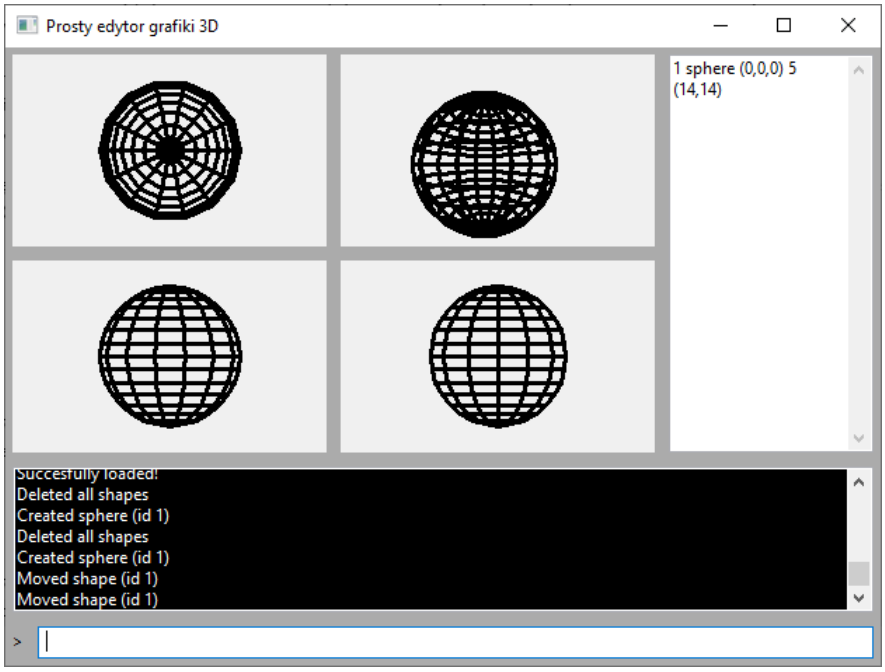
Generacja prostopadłościanu. Przy okazji sprawdziliśmy obrót.



Rysunek 4: Przykład prostopadłościanu i rotacji

Prostopadłościan został wygenerowany poprawnie (Rys. 4).

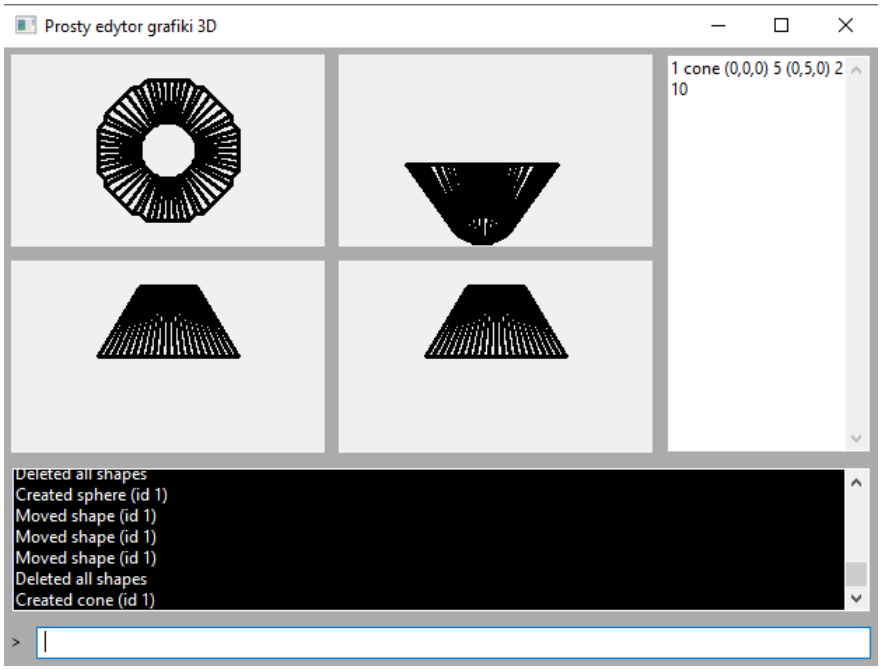
Generacja sfery.



Rysunek 5: Przykład sfery

Sfera została wygenerowana poprawnie (Rys. 5).

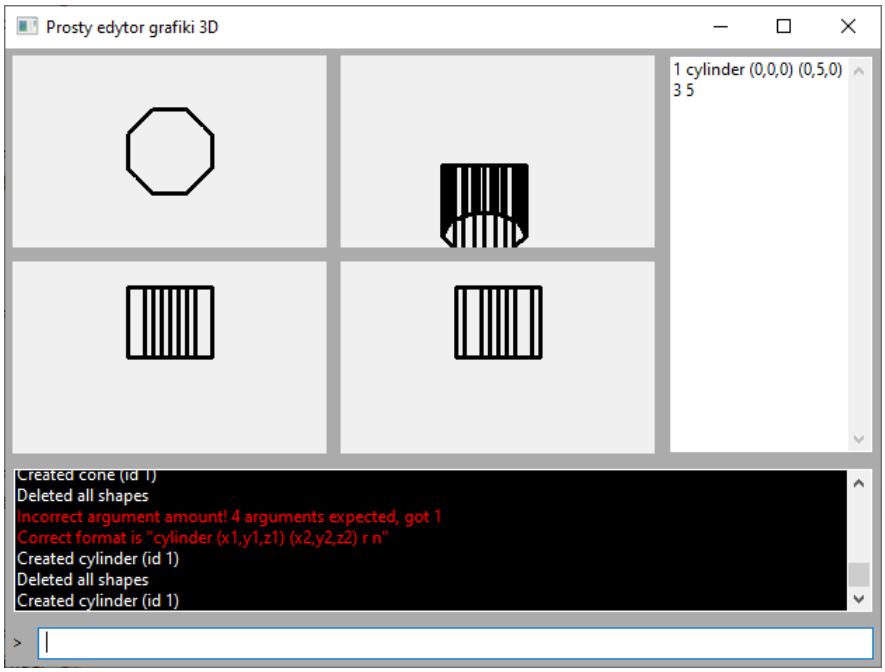
Generacja stożka.



Rysunek 6: Przykładowy stożek

Stożek został wygenerowany poprawnie (Rys. 6).

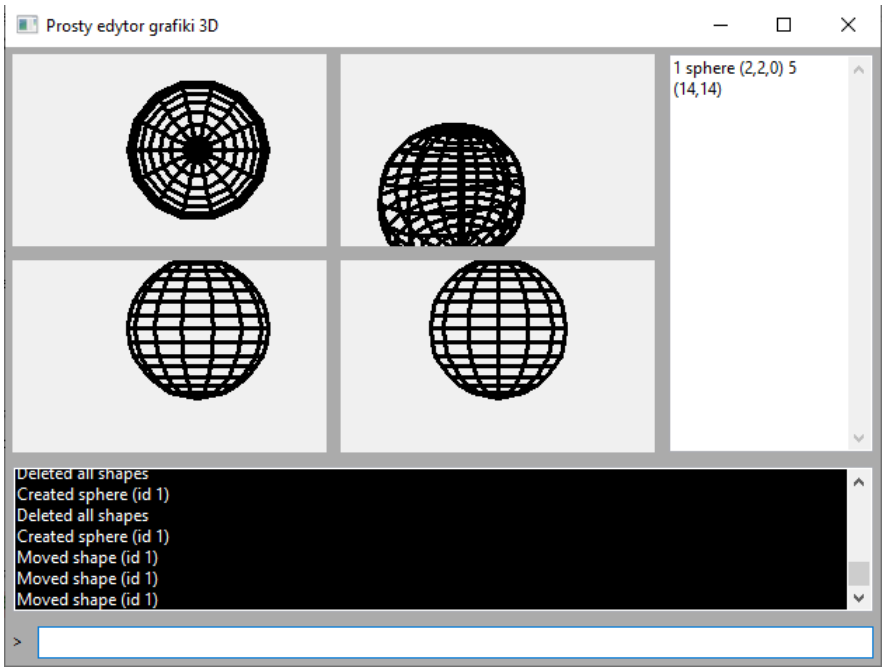
Generacja cylindra.



Rysunek 7: Przykładowy cylinder

Cylinder został wygenerowany poprawnie (Rys. 7).

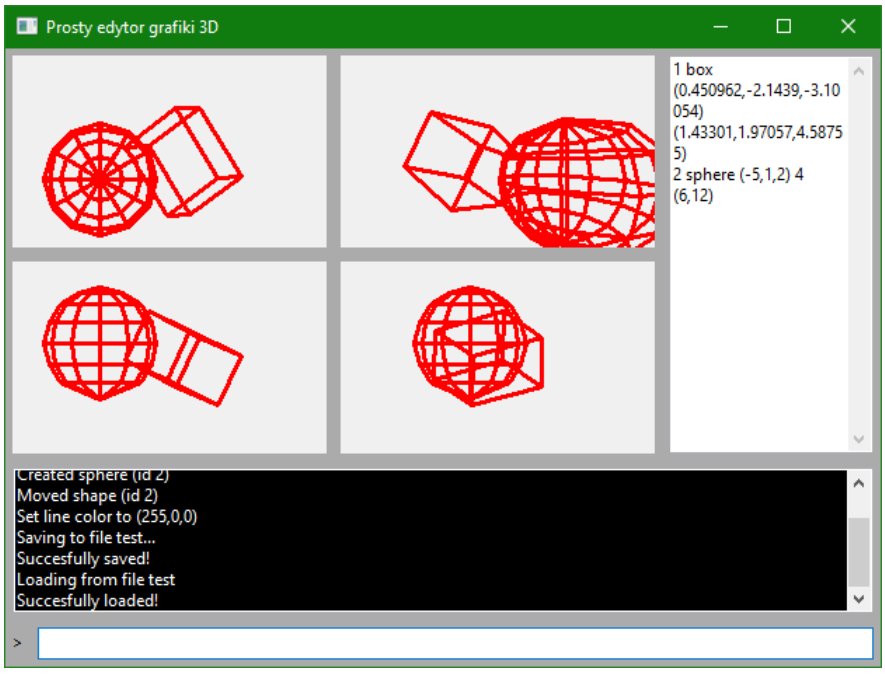
Działanie funkcji move.



Rysunek 8: Przesunięcie na przykładzie sfery

W każdym z okien sfera została przesunięta poprawnie (Rys. 8).

Ostatnim testem było tworzenie wielu figur naraz, ustawianie koloru linii oraz zapisywanie i wczytywanie z pliku.



Rysunek 9: Przykładowe dodatkowe operacje

Wszystkie operacje zostały wykonane poprawnie (Rys. 9).

8 Wdrożenie, raport i wnioski

Projekt zrealizował wszystkie wymagania podstawowe, pozwala na tworzenie prostych grafik 3D poprzez dodawanie podstawowych brył i ich edycję. Poprawnie działa zapis i odczyt pliku. Czasami można zaobserwować błąd nieznanego pochodzenia powodujący niepoprawne wyświetlanie się kształtów po obrocie. Gdy zapiszemy i wczytamy dane z pliku powinien on zniknąć.
Z wymagań rozszerzonych zrealizowana została komenda ustawiająca odległość kamer widoku płaskiego od środka układu. Poza tym, częściowo zrealizowany został widok perspektywiczny – można ustawić pozycję kamery i pole widzenia, jednak kamer nie może zmieniać kąta patrzenia.
Zastosowanie konsoli jak głównego kanału komunikacji z użytkownikiem w przypadku tego typu programu nie jest wygodnym rozwiązaniem. Proces dodawania i edycji obiektów jest powolny i nieintuicyjny, ponadto brak możliwości szybkiego oglądania brył ze wszystkich stron.
Praca nad aplikacją pozwoliła nam zapoznać się z podstawami grafiki 3D – fakt, że środowisko w którym pracowaliśmy było przystosowane do pracy w 2D spowodował, że musieliśmy przemyśleć sprawy dotyczące takich rzeczy jak perspektywy czy rzutowanie.
Jednak co prawdopodobnie ważniejsze, zapoznaliśmy się w praktyce z pracą w środowisku *wxWidgets* oraz z projektowaniem i implementacją bardziej skomplikowanych programów niż te, które dotąd tworzyliśmy na laboratoriach.
Przyzwyczajaliśmy się również do pracy z programami wspomagającymi pracę programisty, zarówno w pisaniu programów jak i dokumentacji – *Visual Studio*, *Github*, *Doxygen*, *Overleaf*.