Prosty edytor grafiki 3D

Zespół
Imię i nazwisko
Maciej Brocki
Kacper Plutowski
Wojciech Lupa

Spis treści

Opis projektu

Celem projektu jest stworzenie prostego edytora grafiki 3D. Okno aplikacji jest podzielone na cztery segmenty, w których wyświetlane są różna widoki kreowanych obiektów. Tworzenie obiektów będzie się odbywało za pomocą komend wprowadzanych w linii poleceń w aplikacji. Ponadto w oknie aplikacji wyświetlana będzie lista utworzonych obiektów wraz z podstawowymi informacjami o nich.

Założenia wstępne przyjęte w realizacji projektu

Wstępnymi założeniami projektu było realizacja wymagań podstawowych, czyli:

- Stworzenie konsoli do wprowadzania komend
- Tworzenie obiektów takich jak: linia, prostopadłościan, sfera, stożek, walec
- Zmiana koloru linii dla rysowanych obiektów
- Rysowanie obiektów na ekranie w różnych widokach:
    - Od przodu ((0,0,1)->(0,0,0))
    - Z góry ((0,1,0)->(0,0,0))
    - Z boku ((1,0,0)->(0,0,0))

    - Z perspektywy ((1,1,1)->(0,0,0))
- Zmiana koloru linii, którą rysowane będą obiekty
- Wyświetlanie listy utworzonych obiektów
- Modyfikowanie obiektów, operacje przesuwania obiektu o wektor oraz obracanie obiektu
- Usuwanie obiektów
- Zapisanie utworzonych obiektów do pliku, a następnie umożliwienie jego odczytu

Powyższe wymagania zostały uzupełnione o następujące funkcjonalności:

- Obsługa błędów podczas wprowadzania komend
- Zapisu widoku z okna “perspektywa” do bitmapy o zadanych wymiarach
- Przesuwania oka kamery dla zadanego widoku na podaną odległość
- Przesuwanie punktu skupienia kamery w widoku “perspektywa” do zadanego punktu
- Zmiana ustawienia pozycji kamery w widoku “perspektywa” do zadanego punktu
- Zmiana pola widzenia w widoku perspektywa o zadany kąt
- Podświetlenie wskazanego obiektu
- Tworzenie grup obiektów
- Przeprowadzanie operacji na grupach: przesuwanie i obracanie

Analiza Projektu

Dane wejściowe

Użytkownik wprowadza do aplikacji predefiniowane komendy o ściśle określonej postaci, które to następnie są interpretowane na odpowiadające im operacje. Są one typu WxString

Dane wyjściowe

Wyjściem aplikacji są wyrysowane na ekranie stworzone obiekty. Na podstawie obiektów są tworzone odpowiednie bitmapy zawierające różne widoki, a następnie wyświetlane są one w odpowiednim oknie.

Struktury danych

Klasa Drawable::Camera

Opis:

Klasa `Camera` jest odpowiedzialna za przekształcenia i projekcje 3D do 2D, zarządzanie parametrami kamery i jej ustawieniami.

Pola:

Wszystkie pola są przechowywane w statycznych doubla’h albo w struct Postition.

static double frontDistance; Odległość do przedniej płaszczyzny obcinania.

static double topDistance; Odległość do górnej płaszczyzny obcinania.

static double rightDistance; Odległość do prawej płaszczyzny obcinania.

static Position cameraPosition; Pozycja kamery.

static Position lookAtPosition; Punkt, na który skierowana jest kamera.

static double fieldOfView; Pole widzenia kamery w stopniach.

static Position cameraDirection; Kierunek kamery.

static Position rightVector; Wektor skierowany w prawo.

static Position upVector; Wektor skierowany w górę.

static constexpr double nearPlane = 0.01; Odległość do bliskiej płaszczyzny obcinania.

static constexpr double farPlane = 1000.0; Odległość do dalekiej płaszczyzny obcinania.

static double fovInRadians; Pole widzenia w radianach.

static double tanFov; Tangens pola widzenia.

static double aspectRatio; Stosunek szerokości do wysokości obrazu.

static double panelHeight; Wysokość panelu (ekranu).

static double panelWidth; Szerokość panelu (ekranu).

Klasa Drawable

Opis:

Klasa `Drawable` zarządza zbiorami obiektów rysowalnych (`DrawableObject`) i oferuje funkcjonalności takie jak dodawanie, usuwanie, przesuwanie, obracanie, podświetlanie i rysowanie obiektów.

Pola:

static std::vector<DrawableObject*> figures; Wektor figur

static wxColour penColor; Kolor do rysowania

static bool fill_style; Czy wypełnienie (false jeśli nie, true jeśli tak)

static wxColour fill_color; Kolor wypełnienia

static view view_style; Rodzaj widoku (wire, lines lub solid)

static int penWidth; Szerokość pióra

static int highlight_duration_ms; Czas trwania podświetlenia w milisekundach

static double highlight_factor; Współczynnik podświetlenia

Klasa DrawableObject

Opis:

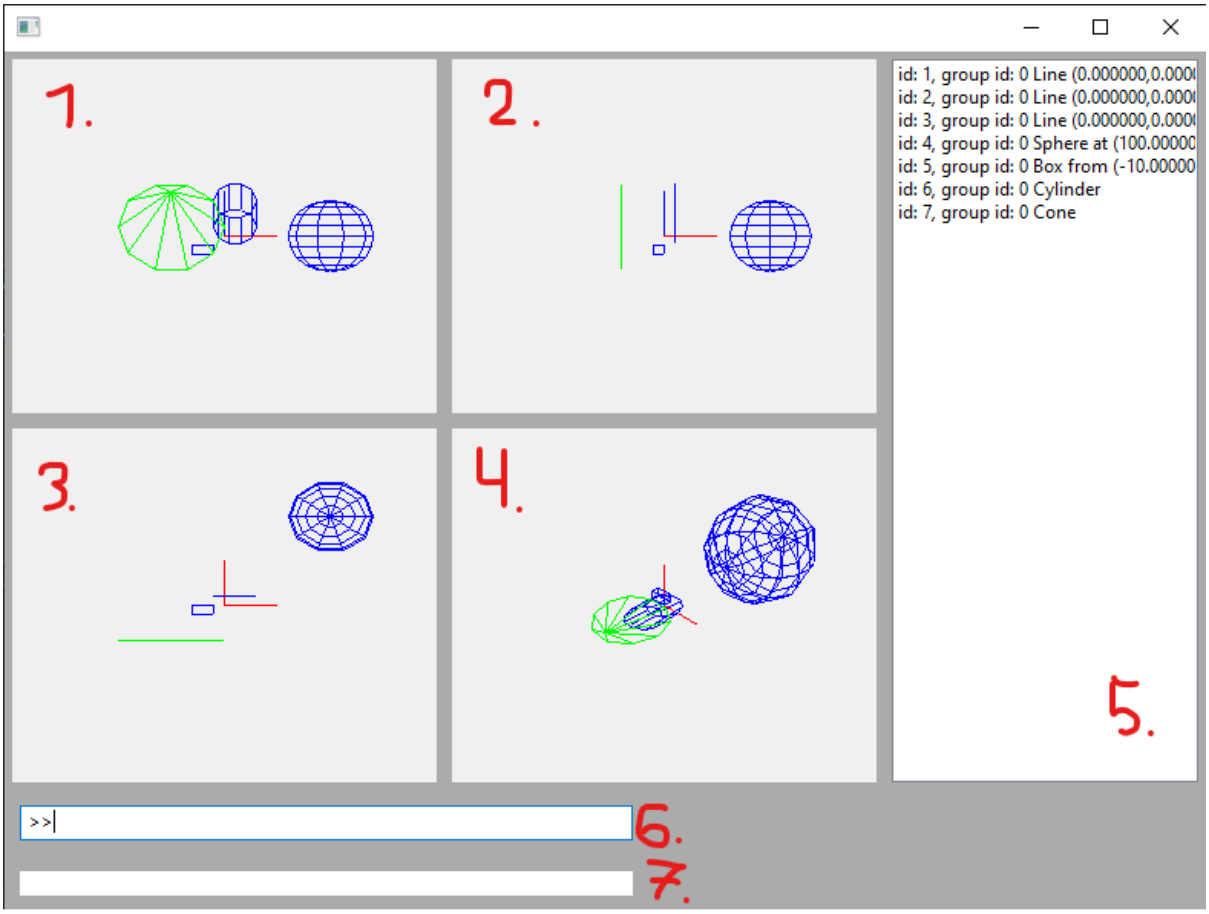
Klasa `DrawableObject` dziedziczy po klasie `Drawable` i jest podstawową klasą dla wszystkich rysowalnych obiektów w aplikacji. Poniżej znajduje się szczegółowy opis jej pól, metod i struktur danych.

Pola:

- **_group_id** (int):
  - Przechowuje identyfikator grupy, do której należy obiekt.
  - Domyślnie ustawiony na 0.
- **_type** (std::string):
  - Typ obiektu rysowalnego.
  - Domyślnie ustawiony na "DrawableObject".
- **_color** (wxColour):
  - Kolor obiektu.
  - Inicjalizowany domyślnym kolorem pióra `Drawable::penColor`.
- **_lineWidth** (int):
  - Szerokość linii używanej do rysowania obiektu.
  - Inicjalizowany domyślną szerokością pióra `Drawable::penWidth`.
- **_vertices** (std::vector<Position>):
  - Wektor przechowujący wierzchołki obiektu w przestrzeni 3D.
- **_highlightTimer** (wxTimer*):
  - Timer używany do resetowania podświetlenia obiektu po określonym czasie.

Interfejs użytkownika

Interfejs użytkownika składa się z następujących elementów.



Rysunek 1. Poglądowy wygląd interfejsu użytkownika aplikacji

Objaśnienie elementów

1. Panel do wyświetlania rzutu pionowego
    - Obiekty są rysowane z perspektywy obserwatora znajdującego się w punkcie (0,0,1) układu patrzącego na punkt (0,0,0)
2. Panel do wyświetlania rzutu bocznego
    - Obiekty są rysowane z perspektywy obserwatora znajdującego się w punkcie (1,0,0) układu patrzącego na punkt (0,0,0)
3. Panel do wyświetlania rzutu poziomego
    - Obiekty są rysowane z perspektywy obserwatora znajdującego się w punkcie (0,1,0) układu patrzącego na punkt (0,0,0)
4. Panel do wyświetlaniu rzucie perspektywicznym
    - Obiekty są rysowane z perspektywy obserwatora znajdującego się w punkcie (1,1,1) układu patrzącego na punkt (0,0,0)
5. Lista obiektów
    - Element z wxBoxList z biblioteki wxWidgets, który w tym wypadku został wykorzystany do wyświetlania wszystkich obiektów jakie są stworzone
    - Każdy rekord posiada swój indywidualny numer (id), który to jest wykorzystywany w niektórych komendach np. W komendzie delete.

- Kolejne pole to id grupy, do której należy dany obiekt. Przy tworzeniu domyślnie jest on przypisany do grupy 0. Za pomocą komend można zmienić przypisanie do grupy
- Kolejne pole to typ danego obiektu, np. line, box
- Kolejne pola to informacje dodatkowe o wykreowanych obiektach takich jak ich wartość promienia, informacje o pozycji punktu lub podstawy

6. Konsola do komend
    - Główne narzędzie do komunikowania się użytkownika z aplikacją. Całe sterowanie i wszystkie akcje są przeprowadzane za pomocą aplikacyjnej konsoli.
    - W poniższej tabeli będą zdefiniowane wszystkie dostępne komendy z objaśnieniem ich działania
    - Poszczególne składowe komendy oddziela spacja
    - Po wpisaniu komendy by ją zatwierdzić należy kliknąć przycisk ENTER

Komendy		
Sygnatura	parametry	opis
set_line_color (R,G,B)	• R,G,B - wartości kolorów przyjmujące wartości od 0 do 255	Ustawienie koloru linii rysowania
line (x1,y1,z1) (x2,y2,z2)	• (x1,y1,z1) - punkt początkowy • (x2,y2,z2) - punkt końcowy	Rysownie linii o zadanych współrzędnych
box (x1,y2,z1) (x2,y2,z2)	• (x1,y1,z1) - punkt początkowy • (x2,y2,z2) - punkt przeciwległy	Rysowanie prostopadłościanu o zadanych parametrach
sphere (x,y,z) r (n,m)	• (x,y,z) - punkt środka sfery • r- promień sfery • (n,m)- liczba południków i równoleżników	Rysowanie sfery o środku w zadanym punkcie, promieniu r, podzielonej na zdefiniowaną liczbę południków i równoleżników
cone (x1,y1,z1) r1 (x2,y2,z2) r2 n	• (x1,y1,z1) - współrzędne pierwszej podstawy • r1- promień pierwszej podstawy	Rysowanie stożka o zadanych podstawach, podzielonego na n czworokątów

	• (x2,y2,z2) - współrzędne drugiej podstawy • r2- promień drugiej podstawy • n- liczba czworokątów	
cylinder (x1,y1,z1)(x2,y2,z2) r n	• (x1,y1,z1) - współrzędne pierwszej podstawy • (x2,y2,z2) - współrzędne drugiej podstawy • r- promień podstaw • n- liczba czworokątów	Rysowanie walca o zadanych podstawach, o promieniu r, podzielonego na n czworokątów
delete id	• id – id obiektu, który chcemy usunąć	Usuwanie obiektu o podanym id
clear_all	• NONE	Usuwa wszystkie obiekty
move id (x,y,z)	• id- id obiektu • (x,y,z)- wektor, o który chcemy przesunąć figurę	Przesunięcie obiektu z podanym id o zadany wektor
rotate id (x,y,z) (a,b,g)	• id – id obiektu • (x,y,z)- punkt, względem którego checmy obracać obiekt • (a,b,g) - wartość obrotu wokół osi x,y,z	Obrót obiektu z podanym id względem punktu (x,y,z) o podane kąty
add_to_group id_group id	• id_group- id grupy • id- id obiektu	Dodanie obiektu id do grupy o podanym id_group, id_group musi być wartością większą od 0
move_group id_group (x,y,z)	• id_group- id grupy • (x,y,z)- wektor, o który chcemy przesunąć grupę figur	Przesunięcie grupy obiektów z podanym id_group o zadany wektor

rotate_group id_group (x,y,z) (a,b,g)	• id_group- id grupy • x,y,z)- punkt, względem którego checmy obracać obiekty • (a,b,g) - wartość obrotu wokół osi x,y,z	Obrót grupy obiektów z podanym id_group względem punktu (x,y,z) o podane kąty
save name	• name – nazwa pliku, do którego zapisujemy utworzone obiekty	Zapis danych do pliku o nazwie name.txt
load name	• name – nazwa pliku, z którego będziemy czytać dane	Odczytanie danych z pliku. Podając nazwę pliku nie podajemy jego rozszerzenia
render_to_file w h name	• w- szerokość bitmapy • h- wysokość bitmapy • name- nazwa pliku	Zapis bitmapy o szerokości w i wysokości h tego co aktualnie znajduje się w oknie “perspektywa”
set_view_range right | front | top r	• right | front | top - wybór rzutu, który chcemy modyfikować (należy wybrać jeden) • r- nowa odległość kamery	Przesuwa oko kamery dla zadanego widoku z domyślnej odległości 1 na r
camera_look_at (x,y,z)	• (x,y,z)- punkt, na który patrzy kamera	Przesunięcie punktu, na który patrzy kamera z widoku “prespektywa” na punkt (x,y,z)
camera_fov alfa	• alfa-nowy kąt widzenia	Zmiana kąta widzenia kamery w widoku "perspektywa”
touch id	• id- id obiektu	Podświetlenie wskazanego obiektu

7. Informacja o błędach
    - Gdy podana komenda jest niewłaściwa, wtedy użytkownik może skorzystać z komunikatu o błędzie
    - Są cztery typy błędów:

- “Error: No command given” - podana komenda jest pusta
- “Error: Not enough arguments” - podana komenda nie zawiera wszystkich argumentów
- “Error: Unrecognized command” - podana komenda nie została rozpoznana
- "Error: Invalid command arguments" - argument podany w komendzie jest nieprawidłowy

Wyodrębnienie i zdefiniowanie zadań

Projekt został podzielony na poszczególne etapy

I. Wstępne planowanie
II. Tworzenie Interfejsu
III. Moduł związany z obsługą komend
    a. Interpretacja komend
    b. Obsługa błędów związanych z komendami
IV. Tworzenie szkieletu projektu
    a. Tworzenie klas związanych z zarządzaniem obiektami Drawable
    b. Tworzenie klas reprezentujących poszczególne obiekty: DrawableObject oraz klasy do niej pochodne
    c. Definiowanie operacji na obiektach
V. Moduł rysowanie obiektów
    a. Przekształcenia perspektywy
    b. Deklaracje metod rysujących
VI. Dodawanie dodatkowych funkcjonalności - grupy
VII. Testy
VIII. Tworzenie dokumentacji

Narzędzia programistyczne użyte w projekcie

Przy doborze narzędzi programistycznych wykorzystywanych przy tworzeniu projektu kierowano się ich znajomością przez autorów oraz prostotą implementacji.

Program napisano w języku C++ przy użyciu środowiska programistycznego Visual Studio 2022 z uwagi na doświadczenie w płynnym posługiwaniu się obu narzędziami.

Projekt zrealizowany został przy wykorzystaniu biblioteki wxWidgets. Bibliotekę tę wybrano z uwagi na jej znajomość oraz pewne cechy upraszczające tworzenie kodu:

- tworzenie interfejsu graficznego
- możliwość wprowadzania komend oraz rysowania obiektów
- implementacje struktur danych wzbogacających interfejs programisty
- obszerna dokumentacja oraz narzędzia debugowe

Celem synchronizacji poszczególnych wersji kodu pomiędzy autorami projektu użyto narzędzia Git. Repozytorium projektu dla szerokiej dostępności przechowywane jest na platformie GitHub.

Podział pracy i analiza czasowa

Lp.	zadania	Maciej	Kacper	Wojciech
1	Tworzenie repozytorium	30 min		
2	Planowanie działań	1h	2h	2h
3	Projekt i utworzenie interfejsu użytkownika	1h 30 min		
4	Obsługa i interpretacja komend, klasa CommandParser, funkcje klasy MyFrame1	2h		1h 30min
5	Obsługa błędów komend	2h		30 min
6	Tworzenie szkieletu struktury danych aplikacji: deklaracja klas Drawable, DrawableObject, Position			3h
7	Utworzenie klas reperzentujących obiekty do rysowania: Line, Box, Sphere, Cone,Cylinder		5h	30 min
8	Zdefiniowanie metod move i		1h	

	rotate dla obiektów			
9	Rysowanie obiektów		2h	
10	Definiowanie metod związanych z perspektywą: klasa Camera		4h	
11	Zapis i odczyt do pliku		3h	
12	Zapis do pliku jako bitmapę		30min	
13	Wypisywanie listy obiektów		30min	
14	Implementacja komendy touch (wyróżnienie obiektu)		30min	
15	Implementacja grup obiektów	1h 30min		
16	testowanie	2h	2h	5h
17	Tworzenie dokumentacji	10 h	2h	8h

Opracowanie i opis niezbędnych algorytmów

1. Algorytm Przesunięcia (shift)

Algorytm przesunięcia dodaje wartości przesunięcia x, y i z do aktualnych współrzędnych punktu. Jest to prosta operacja arytmetyczna, która aktualizuje współrzędne punktu.

2. Algorytm Obracania (rotate)

Algorytm obracania wykorzystuje macierze obrotu w trójwymiarowej przestrzeni. Każdy obrót jest przekształceniem punktu przy użyciu odpowiednich macierzy obrotu wokół osi x, y i z. Procedura jest następująca:

- **Przesunięcie do układu współrzędnych z początkiem w `center`**:
  - Przesuwamy punkt tak, aby `center` stał się początkiem układu współrzędnych.

- **Obrót:**
  - Obracamy punkt najpierw wokół osi z (gamma), następnie wokół osi y (beta) i na końcu wokół osi x (alpha).
  - Używamy macierzy obrotu dla każdej osi.
- **Przesunięcie z powrotem do oryginalnego układu współrzędnych**:
  - Po obrocie punkt jest przesuwany z powrotem na swoje miejsce w oryginalnym układzie współrzędnych.

3. Metody Projekcji

- projectPerspective()
  - Projekcja perspektywiczna przekształca punkt z przestrzeni 3D na 2D. Algorytm najpierw przesuwa punkt względem pozycji kamery, następnie obraca go zgodnie z orientacją kamery i wykonuje projekcję perspektywiczną.
- projectFront()
  - Projekcja z przodu ignoruje współrzędną z i przekształca tylko x i y.
- projectTop()
  - Projekcja z góry ignoruje współrzędną y i przekształca x i z.
- projectSide()
  - Projekcja z boku ignoruje współrzędną x i przekształca y i z.

Kodowanie

class MyFrame1 : public wxFrame{...}

Klasa definiujące główne okno programu. Zawiera ona deklaracje wszystkich elementów interfejsu użytkownika. Klasa ta zawiera:

Element klasy	opis
wxPanel* vertical_side_panel;	Panel, na którym wyświetlany jest rzut pionowy obiektów
wxPanel* side_panel;	Panel, na którym wyświetlany jest rzut boczny obiektu
wxPanel* horizontal_side_panel;	Panel, na którym wyświetlany jest rzut poziomy
wxPanel* perspective_panel;	Panel, na którym wyświetlany jest rzut perspektywiczny obiektów
wxListBox* Elements_ListBox;	Służy do wyświetlenia listy utworzonych obiektów oraz ich podstawowych danych
wxTextCtrl* Command_panel;	Element odpowiedzialny za wprowadzanie komend przez użytkownika
wxStaticText* m_error_message_box;	Daje informację zwrotną do użytkownika o niepowodzeniu związanym z

	wprowadzeniem nieprawidłowej komendy
virtual void Update(wxCommandEvent& event);	Główna funkcja realizująca komendy wprowadzone przez użytkownika oraz aktualizująca panele widoków
virtual void OnSize(wxSizeEvent& event);	Funkcja zapewniająca aktualizację paneli przy zmianie rozmiaru okna
virtual void Redraw();	Funkcja rysująca odpowiednie widoki na ekran

struct Position {...};

Struktura przechowująca położenie punktu o współrzędnych (x,y,z)

Elementy klasy	opis
Position(double a = 0, double b = 0, double c = 0);	Konstruktor klasy Position. Parametry a,b,c to wartości współrzędnych x, y, z. Mają one domyślną wartość równą 0
double x, y, z;	Współrzędne punktu
std::string toString() const;	Metoda konwertująca pozycje na typ string
Position operator-(const Position& other) const;	Operator odejmowania wektorowego dwóch pozycji
Position operator+(const Position& other) const;	Operator dodawania wektorowego dwóch pozycji
static Position fromString(const std::string& str);	Metoda statyczna, konwertująca zmienną typu string na obiekt typu Position
void shift(double x_shift, double y_shift, double z_shift);	Metoda realizująca przesunięcie aktualnej pozycji o zadany wektor
void rotate(const Position& center, double alpha, double beta, double gamma);	Metoda realizująca obrócenie pozycji względem podanego punktu, gdzie • Center- punkt względem, którego dokonujemy obrotu • Alpha - obrót wokół osi x • Beta - obrót wokół osi y • Gamma - obrót wokół osi z

class CommandParser {...}

Klasa, w której są zdefiniowane wszystkie niezbędne funkcje do obsługi wprowadzanych komend. Przekształcają tekst komendy na odpowiednie zmienne, które są następnie używane do inicjalizacji obiektów.

Elementy klasy	opis

static std::vector<std::string> parse_to_vector(wxString command);	Dzieli tekst komendy na poszczególne składowe zwracane w postaci wektora
static Position get_a_point(std::string command);	Zamienia wyrażenie (...,...,...) na obiekt typu Position
static wxColour get_a_color(std::string command);	Zamienia wyrażenie podane wartości RGB podane jako string na zmienną typu wxColour
static std::vector<int> get_a_sphere_lines(std::string command);	Metoda statyczna wykorzystywana w komnedzie sphere, która przekształca wyrażenie (m,n) na liczbę południków i równoleżników, które dzielą sferę
static bool command_length_check(std::vector<std::string> command, int length);	Metoda statyczna sprawdzająca czy komenda ma odpowiednią długość zgadzającą się z postacią komendy

class Drawable {...}

Klasa Drawable służy do obsługi obiektów do rysowania. Przechowuję ona vector stworzonych obiektów oraz metody służące do ich modyfikacji. Zawiera ona również klasę Camera, która definiuje wszystkie niezbędne parametry i funkcje niezbędne do prawidłowego wyświetlania obiektów.

Element klasy	opis
Class Camera	Klasa zawiera dane dotyczące perspektywy i jej ustawień (będzie zdefiniowana dokładniej poniżej)
static void addObj(DrawableObject* fig);	Metoda dodająca nowy obiekt figury do listy istniejących obiektów
static void deleteObj(int index);	Metoda usuwająca obiekt o zadanym id
static void clearAll();	Metoda usuwająca wszystkie obiekty z listy
static void moveObj(int index, double x_shift, double y_shift, double z_shift);	Metoda realizująca przesunięcie zadanego obiektu o podany wektor
static void moveGroup(int group_id, double x_shift, double y_shift, double z_shift);	Metoda przesuwająca grupę obiektów o zadany wektor
static void rotateObj(int index, double x_cord, double y_cord, double z_cord, double alpha, double beta, double gamma);	Metoda obracająca obiekt o zadanym indeksie względem punktu (x,y,z) o podane kąty wokół poszczególnych osi

static void rotateGroup(int grouo_id, double x_cord, double y_cord, double z_cord, double alpha, double beta, double gamma);	Metoda obracająca grupę obiektów o zadanym indeksie grupy względem punktu (x,y,z) o podane kąty wokół poszczególnych osi
static void touchObj(int index);	Metoda realizująca podświetlenie figury o zadanym indeksie
static void DrawAll(wxDC& dcFront, wxDC& dcTop, wxDC& dcSide, wxDC& dcPerspective);	Metoda rysująca wszystkie figury. Parametry: • *dcFront* - panel z widokiem z przodu • *dcTop* - panel z widokiem z góry • *dcSide* - panel z widokiem z boku • *dcPerspective* - panel z widokiem z perspektywa
static DrawableObject* getObj(int index);	Metoda zwracająca obiekt o podanym indeksie
static std::vector<DrawableObject*> getAllObjs();	Metoda zwracająca wszystkie obiekty jakie zostały utworzone
static void SetLineColor(const wxColour& newColour);	Metoda ustawiająca nowy kolor linii
static void saveToFile(const std::string& fileName);	Metoda realizująca zapisująca aktualną listę obiektów do pliku o nazwie fileName
static void loadFromFile(const std::string& fileName);	Metoda realizująca odczyt z pliku o nazwie podanej jako argument
static std::vector<std::string> getFiguresInfo();	Metoda zwracająca informacje jakiego rodzaju elementy znajdują się w liście. Informacja zwraca jest w postaci vectora stringów
static void render_panel_to_bitmap(const std::string& filename, int width, int height, wxPanel* panel);	Metoda zapisująca do pliku obraz z widoku “perspektywa” jako bitmapę o zadanych wymiarach
static void add_to_group(int group_id, int element_id);	Dodanie elementu o wskazanym indeksie do konkretnej grupy

Najważniejsze parametry w klasie Drawable:

- static std::vector<DrawableObject*> figures; - wektor wszystkich utworzonych obiektów do narysowania
- static wxColour penColor; - kolor linii jaką będą rysowane obiekty
- static double highlight_factor; /// Highlight factor

class Camera{...}

Klasa, która jest zawarta w klasie Drawable, odpowiedzialna jest za zachowanie odpowiedniej perspektywy podczas wyświetlania obiektów na ekranie.

Elementy klasy	opis
static wxPoint projectPerspective(const Position& point);	Metoda przeliczająca punkty w 3D na punkt znajdujący się w przestrzeni 2D dla widoku “perspektywa”
static wxPoint projectFront(const Position& pos);	Metoda przeliczająca punkty w 3D na punkt znajdujący się w przestrzeni 2D dla widoku od przodu
static wxPoint projectTop(const Position& pos);	Metoda przeliczająca punkty w 3D na punkt znajdujący się w przestrzeni 2D dla widoku z góry
static wxPoint projectSide(const Position& pos);	Metoda przeliczająca punkty w 3D na punkt znajdujący się w przestrzeni 2D dla widoku z boku
static void update();	Metoda uaktualniająca położenie kamery dla aktualnych ustawień
static void setPosition(const double x, const double y, const double z);	Metoda ustawiająca współrzędne dla nowego położenia kamery
static Position getPosition();	Metoda zwracająca obecną pozycję kamery
static void setLookAt(const double x, const double y, const double z);	Metoda ustawiająca nowy punkt skupienia kamery (punkt, na który patrzy kamera)
static Position getLookAt();	Metoda zwracająca współrzędne aktualnego punktu skupienia kamery
static void setFov(const double newFov);	Metoda ustawiająca nowy kąt widzenia kamery
static double getFov();	Metoda pobierająca aktualne położenie kamery
static double getFrontDistance();	Metoda zwracająca aktualną odległość kamery dla widoku od przodu
static double getTopDistance();	Metoda zwracająca aktualną odległość kamery dla widoku z góry
static double getRightDistance();	Metoda zwracająca aktualną odległość kamery dla widoku z boku
static void setFrontDistance(const double front);	Metoda ustawiająca nową odległość kamery dla widoku od przodu
static void setTopDistance(const double top);	Metoda ustawiająca nową odległość kamery dla widoku z góry

static void setRightDistance(const double right);	Metoda ustawiająca nową odległość kamery dla widoku z boku
static void SetViewSize(const double x, const double y);	Metoda aktualizująca położenie kamery w zależności od rozmiaru okna

Najważniejsze parametry w klasie Camera:

- static double frontDistance;.
  static double topDistance; -odległości kamery
  static double rightDistance;
- static Position cameraPosition; - pozycja kamery
- static Position lookAtPosition; - punkt skupienia kamery
- static double fieldOfView; - obszar widzenia kamery
- static Position cameraDirection;
- static Position rightVector;
- static Position upVector;
- static constexpr double nearPlane = 0.01; ///< Distance to the near clipping plane.
- static constexpr double farPlane = 1000.0; ///< Distance to the far clipping plane.
- static double panelHeight; - wysokość okna
- static double panelWidth; - szerokość okna

class DrawableObject : public Drawable{...}

Klas ta jest podstawową klasą, po której dziedziczą pozostałe klasy definiujące konkretne figury. Dziedziczy ona po klasie Drawable.

Element klasy	opis
DrawableObject(wxColour color = Drawable::penColor, const std::string& type = "DrawableObject");	Konstruktor klasy
void setColor(const wxColour& newColor);	Metoda ustawiająca kolor linii
wxColour getColor() const;	Metoda zwracająca kolor linii
void setLineWidth(int newLineWidth);	Metoda ustawiająca szerokość linii
int getLineWidth() const	Metoda zwracająca szerokość linii
virtual void draw(wxDC& dcFront, wxDC& dcTop, wxDC& dcSide, wxDC& dcPerspective) const;	Metoda rysująca dany obiekt. Parametrami metody są referencje na obszary, gdzie będą rysowane poszczególne widoki
virtual void move(double xShift, double yShift, double zShift);	Metoda przesuwająca obiekt o zadany wektor

virtual void rotate(double xPivot, double yPivot, double zPivot, double alpha, double beta, double gamma);	Metoda obracająca obiekt względem podanego punktu względem trzech osi
virtual void render(wxDC& dc, wxPoint(*projectionFunc)(const Position&)) const = 0;	Metoda rysująca obiekt 3D na 2-wymiarowej powierzchni wykorzystując podaną funkcję perspektywy projectionFunc do obliczenia konkretnych współrzędnych obiektu
virtual std::string getInfo() const;	Metoda podająca informacje o obiekcie, a dokładniej zwraca zapisane w stringu jakiego jest typu
virtual std::string save() const;	Metoda zapisująca dane dotyczące obiektu do pliku
int getGroupId() const;	Metoda zwracająca id grupy, której należy obiekt
void setGroupId(int newGroupId);	Metoda dodająca obiekt do nowej grupy
void highlightObject();	Metoda realizująca podświetlenie wskazanego obiektu
virtual void computeVertices() = 0;	Metoda czysto wirtualna, która oblicza wierzchołki danej bryły
const wxColour generateHighlight() const;	Metoda obliczająca kolor na jaki ma się podświetlić wskazana bryła

Najważniejsze parametry w klasie DrawableObject:

- int _group_id = 0; - id grupy, do której należy obiekt. Domyślną wartością jest 0 co oznacza, że obiekt nie został przypisany do żadnej z grup
- std::string _type; - typ jakiego jest obiekt
- std::vector<Position> _vertices; - wektor przechowując wierzchołki bryły
- wxColour _color; - kolor jakiego jest obiekt
- int _lineWidth = Drawable::penWidth; - grubość linii, jaką jest rysowany obiekt

class Line : public DrawableObject {...}

Klasa reprezentująca Linię

Element klasy	opis
Line(double x1, double y1, double z1, double x2, double y2, double z2, wxColour color = DrawableObject::penColor, const std::vector<Position>& vertices = {});	Konstruktor klasy Line. Parametry: ○ Double x1, y1, z1 - współrzędne pierwszego punktu ○ Double x2, y2, z2 - współrzędne drugiego punktu ○ wxColour color – kolor linii jak będzie miała rysowana linia ○ const std::vector<Position>& vertices - współrzędne linii

Line(Position start, Position end, wxColour color = DrawableObject::penColor, const std::vector<Position>& vertices = {});	Konstruktor klasy Line. Parametry: ○ Position start - współrzędne pierwszego punktu ○ Position end- współrzędne drugiego punktu ○ wxColour color – kolor linii jak będzie miała rysowana linia ○ const std::vector<Position>& vertices - współrzędne linii
std::string getInfo() const override;	Metoda odziedziczona z klasy DrawableObject, zwracająca typ obiektu
std::string save() const override;	Metoda odziedziczona z klasy DrawableObject, przygotowująca dane funkcji do zapisu do pliku
void render(wxDC& dc, wxPoint(*projectionFunc)(const Position&)) const override;	Metoda odziedziczona z klasy DrawableObject, rysująca dany obiekt w podanej perspektywie
void computeVertices() override;	Metoda odziedziczona z klasy DrawableObject, obliczająca wierzchołki figury

Najważniejsze parametry klasy Line:

- Position _start – pozycja początkowa linii
- Position _end – pozycja końcowa linii

class Box : public DrawableObject {...}

Klasa reprezentująca prostopadłościan

Element klasy	opis
Box(double x1, double y1, double z1, double x2, double y2, double z2, wxColour color = DrawableObject::penColor, const std::vector<Position>& vertices = {});	Konstruktor klasy Box. Parametry: ○ Double x1, y1, z1 - współrzędne początkowego punktu ○ Double x2, y2, z2 - współrzędne końcowego punktu(na przeciwległym boku) ○ wxColour color – kolor linii jak będzie miała rysowana linia ○ const std::vector<Position>& vertices - współrzędne linii
Box(Position start, Position end, wxColour color = DrawableObject::penColor, const std::vector<Position>& vertices = {});	Konstruktor klasy Box. Parametry: ○ Position start - współrzędne początkowego punktu ○ Position end- współrzędne końcowego punktu (na przeciwległym boku)

	○ wxColour color – kolor linii jak będzie miała rysowana linia ○ const std::vector<Position>& vertices - współrzędne linii
std::string getInfo() const override;	Metoda odziedziczona z klasy DrawableObject, zwracająca typ obiektu
std::string save() const override;	Metoda odziedziczona z klasy DrawableObject, przygotowująca dane funkcji do zapisu do pliku
void render(wxDC& dc, wxPoint(*projectionFunc)(const Position&)) const override;	Metoda odziedziczona z klasy DrawableObject, rysująca dany obiekt w podanej perspektywie
void computeVertices() override;	Metoda odziedziczona z klasy DrawableObject, obliczająca wierzchołki figury

Najważniejsze parametry klasy Box:

- Position _start – pozycja początkowego punktu
- Position _end – pozycja końcowa przeciwległego punktu

class Sphere : public DrawableObject {...}

Klasa reprezentująca sferę

Element klasy	opis
Sphere(double x, double y, double z, double radius, int numMeridians, int numParallels, const wxColour& color = DrawableObject::penColor, const std::vector<Position>& vertices = {});	Konstruktor klasy Sphere. Parametry: ○ Double x, y, z - współrzędne środka sfery ○ Double radious - promień sfery ○ Int numMeridians, numParallels - ilość równoleżników i południków na jakie jest podzielona sfera ○ wxColour color – kolor linii jak będzie miała rysowana linia ○ const std::vector<Position>& vertices - współrzędne wierzchołków
Sphere(const Position& center, double radius, int numMeridians, int numParallels, const wxColour& color = DrawableObject::penColor, const std::vector<Position>& vertices = {});	Konstruktor klasy Sphere. Parametry: ○ Position centre - współrzędne środka sfery ○ Double radious - promień sfery ○ Int numMeridians, numParallels - ilość równoleżników i południków na jakie jest podzielona sfera ○ wxColour color – kolor linii jak będzie miała rysowana linia

	○ const std::vector<Position>& vertices - współrzędne wierzchołków
std::string getInfo() const override;	Metoda odziedziczona z klasy DrawableObject, zwracająca typ obiektu
std::string save() const override;	Metoda odziedziczona z klasy DrawableObject, przygotowująca dane funkcji do zapisu do pliku
void render(wxDC& dc, wxPoint(*projectionFunc)(const Position&)) const override;	Metoda odziedziczona z klasy DrawableObject, rysująca dany obiekt w podanej perspektywie
void computeVertices() override;	Metoda odziedziczona z klasy DrawableObject, obliczająca wierzchołki figury
void rotate(double xPivot, double yPivot, double zPivot, double alpha, double beta, double gamma) override;	Metoda odziedziczona z klasy DrawableObject, obracająca sferę względem punktu o współrzędnych (xPivot, yPivot, zPivot) wokół osi układu spółrzędnych o zadane kąty

Najważniejsze parametry klasy Sphere:

- double _radius; -promień sfery
- int _numMeridians; - liczba na ile południków ma być podzielona sfera
- int _numParallels; - liczba na ile równoleżników ma być podzielona sfera
- Position _center; - współrzędne środka sfery

class Cone :public DrawableObject{...}

Klasa reprezentująca stożek

Element klasy	opis
Cone(double x1, double y1, double z1, double radius1, double x2, double y2, double z2, double radius2, int sides, wxColour color = DrawableObject::penColor, const std::vector<Position>& vertices = {});	Konstruktor klasy Cone. Parametry: ○ Double x1, y1, z1 - współrzędne środka pierwszej postawy ○ Double radious1 - promień pierwszej podstawy ○ Double x2, y2, z2 - współrzędne środka drugiej postawy ○ Double radious2 - promień drugiej podstawy ○ Int sides - ilość czworokątów na jakie będzie podzielony stożek ○ wxColour color – kolor linii jak będzie miała rysowana linia ○ const std::vector<Position>& vertices - współrzędne wierzchołków

Cone(const Position& base1, double radius1, const Position& base2, double radius2, int sides, const wxColour& color = DrawableObject::penColor, const std::vector<Position>& vertices = {});	○ Konstruktor klasy Cone. Parametry: ○ const Position& base1 - współrzędne środka pierwszej postawy ○ Double radious1 - promień pierwszej podstawy ○ const Position& base2 - współrzędne środka drugiej postawy ○ Double radious2 - promień drugiej podstawy ○ Int sides - ilość czworokątów na jakie będzie podzielony stożek ○ wxColour color – kolor linii jak będzie miała rysowana linia ○ const std::vector<Position>& vertices - współrzędne wierzchołków
std::string getInfo() const override;	Metoda odziedziczona z klasy DrawableObject, zwracająca typ obiektu
std::string save() const override;	Metoda odziedziczona z klasy DrawableObject, przygotowująca dane funkcji do zapisu do pliku
void render(wxDC& dc, wxPoint(*projectionFunc)(const Position&)) const override;	Metoda odziedziczona z klasy DrawableObject, rysująca dany obiekt w podanej perspektywie
void computeVertices() override;	Metoda odziedziczona z klasy DrawableObject, obliczająca wierzchołki figury
void rotate(double xPivot, double yPivot, double zPivot, double alpha, double beta, double gamma) override;	Metoda odziedziczona z klasy DrawableObject, obracająca stożek względem punktu o współrzędnych (xPivot, yPivot, zPivot) wokół osi układu spółrzędnych o zadane kąty

Najważniejsze parametry klasy Cone:

- Position _base1; -współrzędne pierwszej podstawy
- double _baseRadius1; - promień pierwszej podstawy
- Position _base2; -współrzędne pierwszej podstawy
- double _baseRadius2; - promień pierwszej podstawy
- int _sides; - ilość czworokątów na jakie będzie podzielony stożek

class Cylinder : public Cone{...}

Klasa reprezentuje walec. W przeciwieństwie do innych klas figur dziedziczy ona po klasie Cone, ponieważ są one do siebie bardzo podobne i metody do renderowania oraz obracania działają dokładnie w ten sam sposób. Klasa ta zawiera następujące

Elementy klasy	opis
Cylinder(const Position& base1, double radius1, const Position& base2, double radius2, int sides, const wxColour& color = DrawableObject::penColor, const std::vector<Position>& vertices = {});	○ Konstruktor klasy Cylinder. Parametry: ○ const Position& base1 - współrzędne środka pierwszej postawy ○ Double radious1 - promień pierwszej podstawy ○ const Position& base2 - współrzędne środka drugiej postawy ○ Double radious2 - promień drugiej podstawy ○ Int sides - ilość czworokątów na jakie będzie podzielony walec ○ wxColour color – kolor linii jak będzie miała rysowana linia ○ const std::vector<Position>& vertices - współrzędne wierzchołków
Cylinder(double x1, double y1, double z1, double radius1, double x2, double y2, double z2, double radius2, int sides, wxColour color = DrawableObject::penColor, const std::vector<Position>& vertices = {});	Konstruktor klasy Cone. Parametry: ○ Double x1, y1, z1 - współrzędne środka pierwszej postawy ○ Double radious1 - promień pierwszej podstawy ○ Double x2, y2, z2 - współrzędne środka drugiej postawy ○ Double radious2 - promień drugiej podstawy ○ Int sides - ilość czworokątów na jakie będzie podzielony walec ○ wxColour color – kolor linii jak będzie miała rysowana linia ○ const std::vector<Position>& vertices - współrzędne wierzchołków
std::string getInfo() const override;	Metoda dziedziczona z klasy bazowej, która zwraca nazwę typu obiektu

std::string save() const override;	Metoda odziedziczona z klasy bazowej przygotowująca dane obiektu do zapisu do pliku

Testowanie

1. Testy niezależnych bloków

Testy obejmowały zachowanie obiektów poszczególnych klas i funkcji globalnych tuż po ich implementacji, w tym między innymi:

- formularza wxWidgets
- statycznych członków klasy bazowej Drawable
- obiektów klas Position, Camera oraz poszczególnych figur
- metod klasy CommandParser

Dla łatwiejszego zaobserwowania zmian po napisaniu nowych funkcjonalności oraz ścisłą relację pewnych klas i funkcji, większość testów jednostkowych przeprowadzono w ramach testów powiązań różnych bloków.

2. Testy powiązań bloków

Testy integracyjne bloków stanowiły główną część procesu testowania aplikacji przed jej zgłoszeniem. W czasie ich trwania sprawdzano relacje pomiędzy wywołaniem pewnych funkcji a zmianą wartości zmiennych statycznych oraz atrybutów innej klasy, mających na celu:

- tłumaczenie komend na metody klas Drawable i Camera
- dodawanie figur do listy i rysowanie ich na panelach wxWidgets
- dodawanie figur do grup
- modyfikacje parametrów figur oraz grup
- zapis i odczyt listy figur z pliku
- zapis obrazu panelu do pliku bitmapy

Testy przeprowadzano stopniowo, za każdym razem po wprowadzeniu nowej funkcjonalności celem upewnienia się, iż działają one poprawnie przed wdrożeniem kolejnej.

Wybrany sposób testowania pozwolił na przyspieszenie procesu eliminacji błędów po ich zaraportowaniu dzięki ograniczeniu ilości kodu do kontroli - miało to szczególne

znaczenie biorąc pod uwagę, iż działanie niektórych narzędzi mocno zależało od poprawnego działania poprzednio zaimplementowanych funkcji.

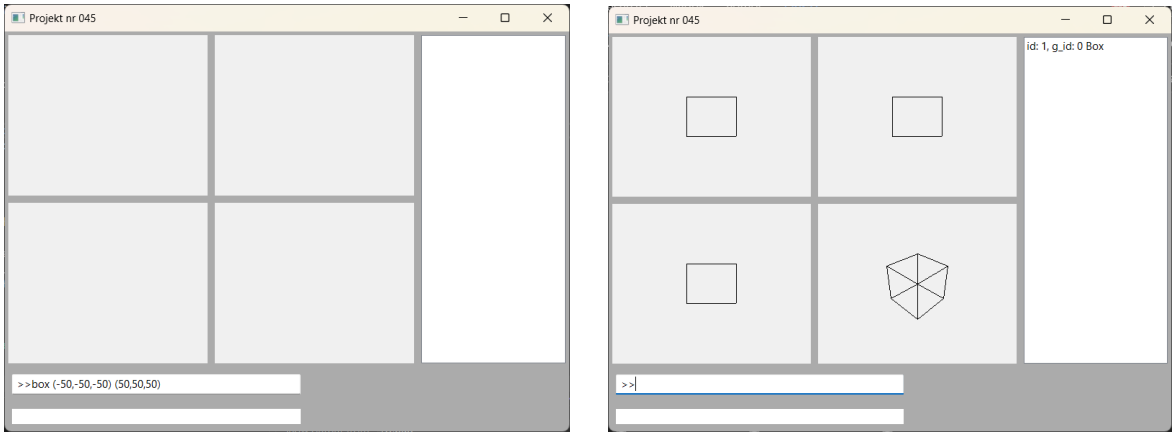
3. Testy całościowe

Testy całościowe przeprowadzane były po implementacji wszystkich założeń projektu. Testy te miały na celu upewnienie się, iż w ostatecznej wersji programu nie występują poważne błędy oraz zachodzi oszczędne zarządzanie pamięcią.

W ramach ostatecznych testów ponownie sprawdzono działanie poszczególnych metod oraz wszystkich relacji międzyklasowych na wypadek, gdyby poprzednio pominięto jakiekolwiek nieścisłości.

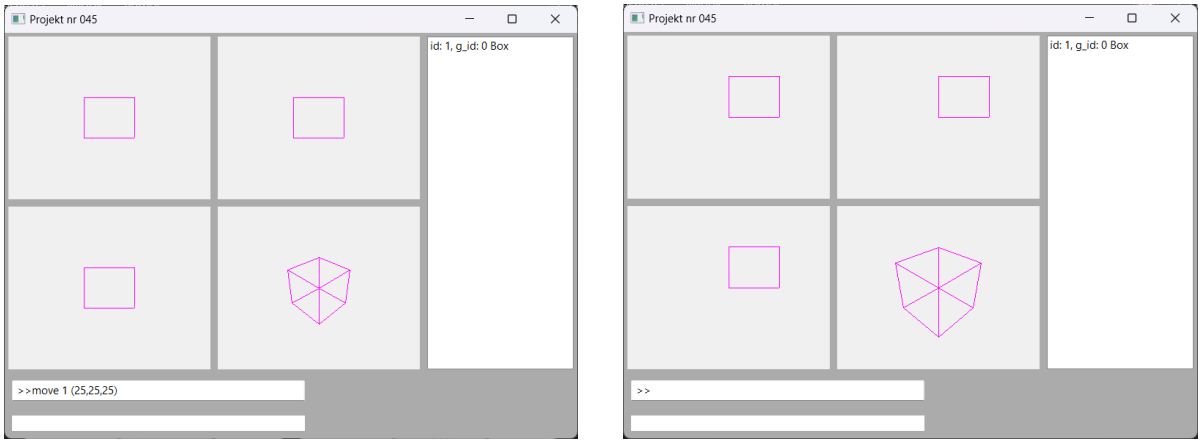
Testy wykonywano na danych wejściowych spełniających zakładany format. Nie sprawdzono wielu przypadków, w których dane niezgodne są z szablonem z powodu niskiego doświadczenia w przeprowadzaniu testów oraz ograniczonego czasu na ukończenie projektu – w programie mogą pojawić się zatem nieprzewidziane zachowania.

Testy częściowo oparto o pliki zawierające dane dla każdego rodzaju figur stereometrycznych w celu przyspieszenia procesu.

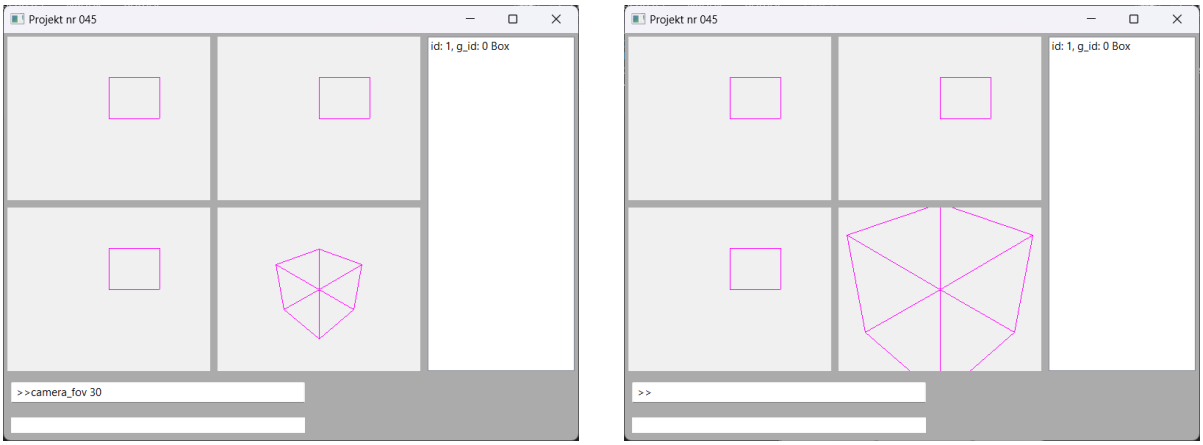


Rysunek 2. Utworzenie obiektu Box

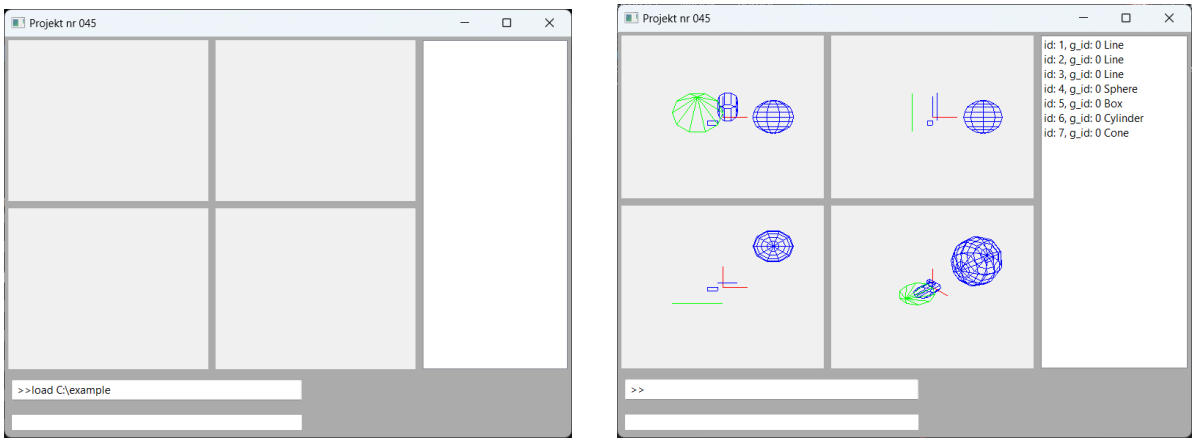
Rysunek 3. Zmiana koloru linii i utworzenie obiektu



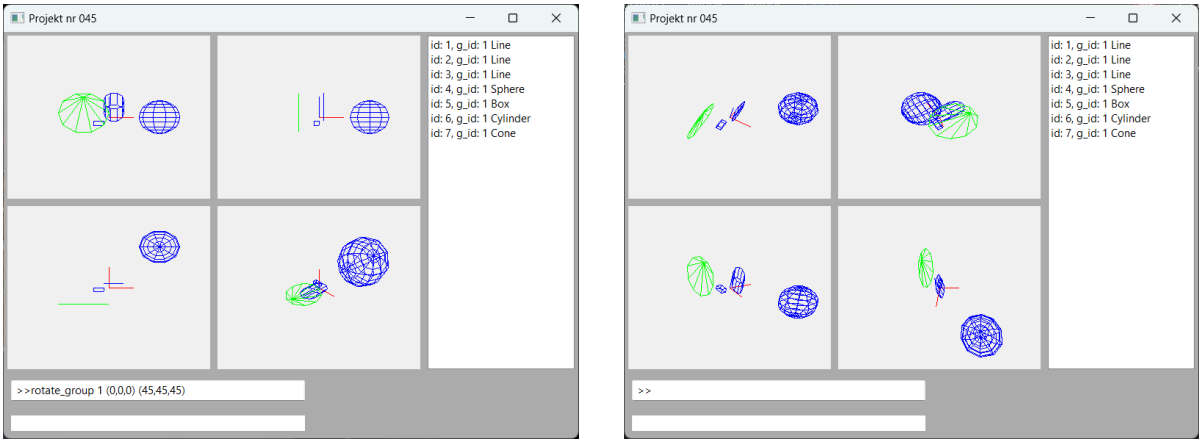
Rysunek 4. Przesunięcie figury



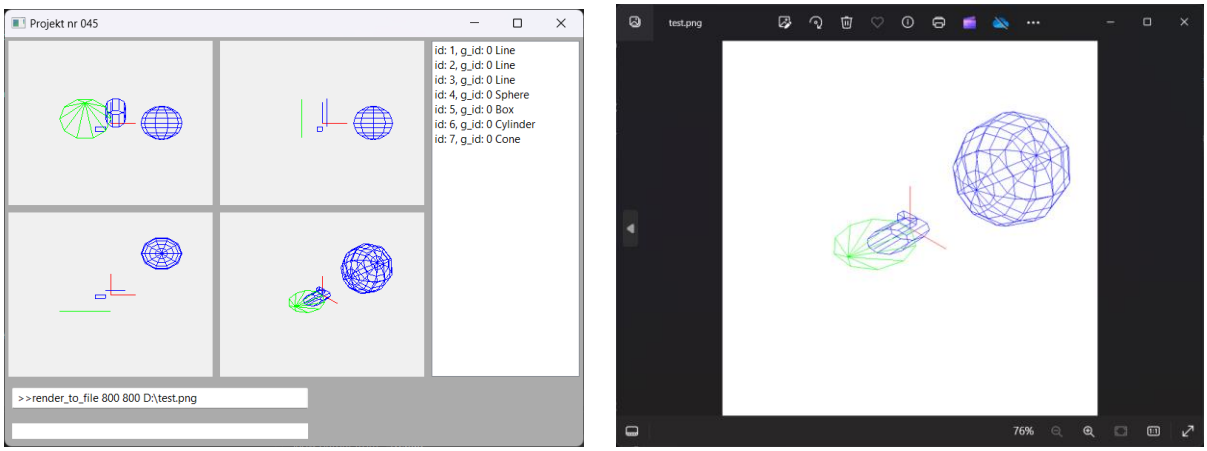
Rysunek 5. Modyfikacja perspektywy



Rysunek 6. Wczytanie pliku



Rysunek 7. Obrót grupy figur



Rysunek 8. Zapis obrazu kamery do pliku

Wdrożenie, raport i wnioski

Podsumowując całkowity czas trwania projektu jego realizacja zajęła nam ok 2.5 tygodnia. W tym czasie udało nam się zrealizować wymagania podstawowe projektu oraz rozszerzyć je o dodatkowe funkcjonalności rozszerzone.

Pracując nad projektem utrwaliliśmy sposoby wykorzystania biblioteki wxWidgets do tworzenie aplikacji, jak również sposoby transformacji perspektyw oraz transformacji obiektów takich jak obracanie czy przesuwanie z wykorzystaniem odpowiednich algorytmów.

Tworząc projekt zapoznaliśmy się z platformą GitHub oraz jej możliwościami. Tym samym zdobyliśmy cenne umiejętności, które będą procentować w przyszłych projektach. Rozwinęliśmy również nasze kompetencje miękkie związane z pracą w grupie, organizacją i planowaniem zadań.

Podczas realizacji projektu była implementacja wyświetlania różnych rzutów perspektywicznych. Początkowo napotkaliśmy trudności w zachowaniu oczekiwanej perspektywy podczas skalowania rozmiaru okna. Ostatecznie jednak po poprawieniu wzoru przeliczającego współrzędnych udało się uzyskać pożądany rezultat.

Największą problem stanowiła realizacja funkcjonalności rozszerzonej dotyczącej niewyświetlania linii znajdującej się za jakimś obiektem. Po analizie problemu stwierdziliśmy, że jej implementacja wymagała dużej reorganizacji istniejącego projektu. Postanowiliśmy zatem, że zastąpi ją funkcjonalność tworzenia grup obiektów oraz działania na ich (np. Obracanie grupy obiektów).

Podsumowując uważamy, że udało nam się pomyślnie zaimplementować wymagane funkcjonalności, oraz dostarczyć stabilnie działającą aplikację do prostego generowania grafiki 3D.