|  |  |  |  |
| --- | --- | --- | --- |
|  | | | |
|  | |  | **Wydział Informatyki i Zarządzania**  kierunek studiów: Informatyka  specjalność: brak  Praca dyplomowa - inżynierska  **Program do eksperymentowania z podstawowymi funkcjonalnościami biblioteki OpenGL**  Piotr Szymczyk  słowa kluczowe:  OpenGL, grafika komputerowa,  dydaktyka, interaktywna nauka  krótkie streszczenie:  Celem pracy jest stworzenie narzędzia dydaktycznego mającego wspomagać nauczanie grafiki komputerowej na Politechnice Wrocławskiej. Jej zakres obejmuje projekt i implementację programu prezentującego podstawowe funkcjonalności bibliotek OpenGL oraz dostarczenie dokumentacji składającej się z instrukcji użytkownika i opisu wspieranych funkcjonalności OpenGL.   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | opiekun pracy  dyplomowej | .................................................. | | ....................... | | ....................... | | | *Tytuł/stopień naukowy/imię i nazwisko* | | *ocena* | | *podpis* | | | Ostateczna ocena za pracę dyplomową | | | | | | | | | Przewodniczący Komisji egzaminu dyplomowego | | | ..................................................  *Tytuł/stopień naukowy/imię i nazwisko* | | ....................... | | ....................... | | *ocena* | | *podpis* |   *Do celów archiwalnych pracę dyplomową zakwalifikowano do:\**   1. *kategorii A (akta wieczyste)* 2. *kategorii BE 50 (po 50 latach podlegające ekspertyzie)*   *\* niepotrzebne skreślić*   |  | | --- | | pieczątka wydziałowa | |
|  |  |  | Wrocław 2017 |

Spis treści

[0 Abstrakt 1](#_Toc504010326)

[1 Wstęp 1](#_Toc504010327)

[1.1 Charakterystyka problematyki 1](#_Toc504010328)

[1.2 Uzasadnienie wyboru tematu pracy 2](#_Toc504010329)

[1.3 Cel i zakres pracy 2](#_Toc504010330)

[2 Przegląd stosowanych technologii 2](#_Toc504010331)

[2.1 OpenGL 2](#_Toc504010332)

[2.1.1 Czym jest OpenGL 2](#_Toc504010333)

[2.1.2 Wersje OpenGL 3](#_Toc504010334)

[2.1.3 Dodatkowe biblioteki 3](#_Toc504010335)

[2.1.4 Kolejność operacji w potoku wizualizacji OpenGL 4](#_Toc504010336)

[2.1.5 OpenGL jako maszyna stanów 4](#_Toc504010337)

[2.1.6 Składnia poleceń w OpenGL 5](#_Toc504010338)

[2.1.7 Transformacje 6](#_Toc504010339)

[2.1.8 Macierze transformacji 6](#_Toc504010340)

[2.1.9 Transformacja modelująca i transformacja obserwatora 7](#_Toc504010341)

[2.1.10 Transformacja projekcji – rzutowanie 8](#_Toc504010342)

[2.1.11 Oświetlenie – model Phonga 9](#_Toc504010343)

[2.1.12 Oświetlenie – materiały 10](#_Toc504010344)

[2.1.13 Oświetlenie – źródła światła 12](#_Toc504010345)

[2.1.14 Oświetlenie – tłumienie światła 14](#_Toc504010346)

[2.1.15 Teksturowanie 14](#_Toc504010347)

[2.2 Wspierane formaty 16](#_Toc504010348)

[2.2.1 Uzasadnienie wyboru wykorzystanych formatów plików 16](#_Toc504010349)

[2.2.2 Definicja geometrii sceny – format OBJ 16](#_Toc504010350)

[2.2.3 Definicja materiałów – format MTL 18](#_Toc504010351)

[2.2.4 Tekstury – popularne formaty rastrowe 19](#_Toc504010352)

[2.2.5 Pozostałe elementy sceny 20](#_Toc504010353)

[3 Projekt, architektura i implementacja rozwiązania 20](#_Toc504010354)

[3.1 Analiza wymagań 20](#_Toc504010355)

[3.2 Architektura rozwiązania 21](#_Toc504010356)

[3.2.1 Wzorce architektoniczne 21](#_Toc504010357)

[3.2.2 Opis implementacji 22](#_Toc504010358)

[3.3 Omówienie wykorzystanych technologii 23](#_Toc504010359)

[3.3.1 Metodyka agile 23](#_Toc504010360)

[3.3.2 Zarządzanie projektem 24](#_Toc504010361)

[3.3.3 System kontroli wersji 24](#_Toc504010362)

[3.3.4 Środowisko pracy 25](#_Toc504010363)

[3.3.5 Stos technologiczny 25](#_Toc504010364)

[4 Instrukcja użytkownika 25](#_Toc504010365)

[4.1 Instalacja 25](#_Toc504010366)

[4.1.1 Wymagania minimalne 25](#_Toc504010367)

[4.1.2 Proces instalacji 26](#_Toc504010368)

[4.2 Konfiguracja 26](#_Toc504010369)

[4.2.1 Uruchamianie programu w wybranej konfiguracji 26](#_Toc504010370)

[4.2.2 Struktura pliku i dozwolone wartości 26](#_Toc504010371)

[4.3 Instrukcja użytkowania 30](#_Toc504010372)

[4.3.1 Ustawianie binarnych zmiennych stanu 30](#_Toc504010373)

[4.3.2 Dodawanie i usuwanie transformacji modelujących 31](#_Toc504010374)

[4.3.3 Ustawianie transformacji obserwatora i transformacji projekcji 32](#_Toc504010375)

[4.3.4 Dodawanie i usuwanie źródeł światła 33](#_Toc504010376)

[4.3.5 Wczytywanie sceny 34](#_Toc504010377)

[4.3.6 Wczytywanie i ustawianie materiałów 35](#_Toc504010378)

[4.3.7 Wczytywanie i konfiguracja tekstur 36](#_Toc504010379)

[4.3.8 Generowanie kodu 37](#_Toc504010380)

[4.4 Przykłady 38](#_Toc504010381)

[5 Podsumowanie i wnioski 41](#_Toc504010382)

[5.1 Wnioski 41](#_Toc504010383)

[5.2 Spis artefaktów 41](#_Toc504010384)

[5.3 Dalszy rozwój 41](#_Toc504010385)

[5.3.1 Obsługa błędów 41](#_Toc504010386)

[5.3.2 Interfejsy 42](#_Toc504010387)

[5.3.3 Testy 42](#_Toc504010388)

[5.3.4 Wykorzystanie biblioteki Prism 42](#_Toc504010389)

[5.3.5 Rozbudowa interfejsu użytkownika 42](#_Toc504010390)

[5.3.6 Wsparcie dla Programowalnego Potoku Wizualizacji 42](#_Toc504010391)

[6 Bibliografia 42](#_Toc504010392)

# Abstrakt

|  |  |
| --- | --- |
| Celem pracy jest stworzenie narzędzia dydaktycznego mającego wspomagać nauczanie grafiki komputerowej na Politechnice Wrocławskiej. Jej zakres obejmuje projekt i implementację programu pozwalającego w prosty sposób opanować podstawowe zasady korzystania z biblioteki OpenGL. Ma on prezentować jej główne funkcjonalności. Powinien pozwalać na wczytanie sceny i eksperymentowanie z jej parametrami z poziomu graficznego interfejsu użytkownika. Program musi także oferować możliwość wygenerowania kodu który po skompilowaniu utworzy scenę odpowiadającą tej skonfigurowanej przez użytkownika. Do programu dołączona zostanie instrukcja oraz krótki opis funkcjonalności biblioteki OpenGL prezentowanych przez program. Zostaną one zawarte w tej pracy. | The aim of this thesis is creation of a didactic tool facilitating teaching process of computer graphics on Wroclaw University of Technology. Its scope consists of design and implementation of an application which allows to easily comprehend the basics of using an OpenGL library. It’s expected to present basic features of an OpenGL libraries. It should allow reading a scene definition and playing with its parameters by a graphic user interface. Moreover, program must allow generating code, compilation of which results in displaying a scene corresponding to the one configured by the user. A manual and a brief description of OpenGL libraries features presented by the program will be included in this thesis. |

# Wstęp

## Charakterystyka problematyki

Stare chińskie przysłowie głosi:

„Usłyszałem i zapomniałem. Zobaczyłem i zapamiętałem. Zrobiłem i zrozumiałem.” [12]

Według tej zasady, najlepszą metodą przyswajania wiedzy jest nauka interaktywna.   
Dotychczas dostępne metody nauki grafiki komputerowej obejmowały:

* prezentacje multimedialne dostarczane przez prowadzącego,
* literaturę tematyczną,
* wideo-poradniki dostępne w internecie,
* tutoriale dostępne w internecie,
* laboratoria powiązane z kursem.

Pierwsze trzy wymienione opcje skupiają się na biernym odbiorze i nie są przystosowane   
do efektywnej pracy z kodem oraz prezentacji szczegółów implementacyjnych omawianych tematów. Kolejne dwa podejścia wymagają posiadania odpowiednio skonfigurowanego środowiska programistycznego (IDE, kompilatory, biblioteki OpenGL).

Jak widać każda z wyżej wymienionych metod posiada braki.

## Uzasadnienie wyboru tematu pracy

Temat pracy dyplomowej wzbudził moje zainteresowanie ze względu na konieczność dogłębnego zapoznania się z biblioteką OpenGL podczas jego realizacji. Zaintrygowała mnie również możliwość zbudowania narzędzia które wspomagać będzie proces przyswajania wiedzy przez kolejne pokolenia.

## Cel i zakres pracy

Celem pracy jest stworzenie narzędzia wspomagającego proces nauczania grafiki komputerowej zgodnie z programem kursu ‘Grafika komputerowa’ oferowanego studentom na wydziale Informatyki i Zarządzania na Politechnice Wrocławskiej. Jego osiągnięcie ma zapewnić projekt, implementacja i dokumentacja programu komputerowego prezentującego   
w interaktywny sposób funkcjonalności oferowane przez biblioteki OpenGL.

Zadaniem programu będącego jednym z artefaktów tej pracy jest umożliwienie interaktywnej nauki bez konieczności skomplikowanej konfiguracji środowiska. Ma on pozwalać na eksperymentowanie z podstawowymi funkcjonalnościami biblioteki OpenGL takimi jak:

* oferowane przez OpenGL binarne zmienne stanu (flagi)
* transformacja projekcji
* transformacje modelująca i obserwatora
* teksturowanie
* cieniowanie / własności powierzchniowe

Opracowana w ramach pracy dokumentacja powinna zawierać:

* instrukcję instalacji
* instrukcję użytkowania
* opis wspieranych przez program funkcjonalności OpenGL
* opis architektury aplikacji

# Przegląd stosowanych technologii

## OpenGL

### Czym jest OpenGL

Przed przystąpieniem do jego omawiania warto wyjaśnić czym właściwie jest OpenGL.   
Open Graphics Library jest to interfejs programowania aplikacji (API) służący do generowania obrazów scen 3D i 2D. Składa się on z około 250 funkcji pozwalających na tworzenie grafiki. Pozwala on na uzyskanie przyspieszenia sprzętowego poprzez wykorzystanie procesora graficznego (GPU).

Ponieważ OpenGL jest interfejsem istnieje wiele jego implementacji dla różnych języków programowania. Mogą się one różnić implementacją oraz wydajnością jednak ich interfejs pozostaje taki sam.

### Wersje OpenGL

W momencie publikacji tej pracy najnowszą dostępną wersją OpenGL jest OpenGL 4.6.   
Opis funkcjonalności wprowadzanych wraz z kolejnymi wersjami biblioteki można znaleźć na stronie producenta [6]. Analizując historię wersji biblioteki można dostrzec zachowanie kompatybilności wstecznej dla większości wersji. Warto jednak zauważyć, że w wersji 3.0 nastąpiła zmiana paradygmatu związana z wprowadzeniem shader’ów. Shader’y są to krótkie programy komputerowe, uruchamiane na karcie graficznej, odpowiadające za definiowanie kolejnych etapów potoku wizualizacji odpowiedzialnych za oświetlenie. Ich wprowadzenie spowodowało porzucenie części dotychczasowej specyfikacji. Efektem jest brak kompatybilności wstecznej wersji 3.0 i kolejnych z wersjami wcześniejszymi niż wersja 3.0.

W celu czytelnego zaprezentowania podstaw grafiki komputerowej wykorzystana została stara wersja OpenGL ( Fixed Function Pipeline / Legacy OpenGL). Jej używanie jest niezalecane ze względu na brak kompatybilności z nowymi urządzeniami, ograniczone możliwości oraz niską wydajność [7] w porównaniu do nowszych wersji API. Taka wersja świetnie się jednak nadaje by przedstawić podstawowy potok wizualizacji (Rendering Pipeline) bez dodatkowego narzutu wprowadzanego przez shader’y.

Z tego powodu zdecydowano się oprzeć utworzony w ramach pracy program na tym właśnie paradygmacie. Program nie prezentuje więc, jak należy tworzyć nowoczesne aplikacje korzystając z OpenGL. Powinien być stosowany tylko w celu zrozumienia podstawowych mechanizmów występujących w grafice komputerowej które w wielu wypadkach zostały już zastąpione przez bardziej elastyczne i wydajne rozwiązania. Niemniej poznanie podstawowych idei omówionych w programie zapewnia solidne podstawy z zakresu grafiki komputerowej które pozwolą bez problemu pojąć idee i założenia stosowane w nowszych wersjach OpenGL.

W dalszej części pracy określenie OpenGL będzie odnosić się do Legacy OpenGL.

### Dodatkowe biblioteki

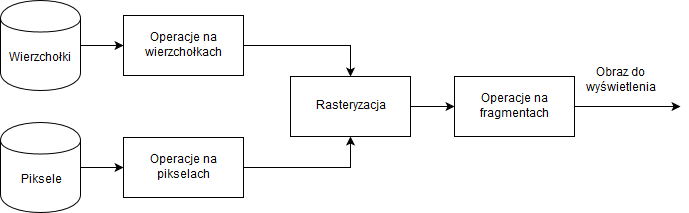
Ponieważ OpenGL zawiera jedynie zestaw podstawowych funkcji odpowiedzialnych za renderowanie grafiki komputerowej istnieje wiele bibliotek wspomagających pracę z OpenGL. Pierwszą biblioteką o której warto wspomnieć jest OpenGL Utility Library (GLU) [8]. Zawiera ona zestaw około 50 funkcji ułatwiających korzystanie z OpenGL i jest standardową częścią każdej jego implementacji.

W celu pozostania niezależnym od platformy, OpenGL nie jest powiązany z żadnym środowiskiem graficznym. Sprawia to, że kolejną użyteczną biblioteką jest OpenGL Utility Toolkit (GLUT) oferujący proste API odpowiadające za zarządzanie oknem aplikacji [8]. Jest ono również niezależne od platformy. Dla każdej platformy dostępne są więc kolejne biblioteki zapewniające wsparcie dla OpenGL w danym środowisku graficznym. Przykładowo dla systemów z rodziny Microsoft Windows jest to WGL a dla systemów UNIX’owych wspierających X Window System jest to GLX.

Ponadto istnieje wiele bibliotek abstrahujących niskopoziomowe funkcje na rzecz wysokopoziomowych dedykowanych do specyficznych zadań. Przykładami są np. Gizmo 3D, Open Inventor, Coin, OpenSceneGraph [5].

### Kolejność operacji w potoku wizualizacji OpenGL

OpenGL definiuje kolejność operacji wykonywanych w celu otrzymania obrazu wyświetlanego na ekranie. Została ona przedstawiona na Rys. 2.1.



Rys. 2.1 Potok wizualizacji OpenGL

Analizując diagram możemy dostrzec dwa typy danych wejściowych, piksele i wierzchołki, każdy z własnym przepływem informacji. Pierwszym krokiem przetwarzania dla wierzchołków jest ich wczytanie i wyznaczenie ich pozycji w przestrzeni sceny. Na grupach wierzchołków rozpinane są proste figury geometryczne (ściany). W celu wyznaczenia ich kolorów, wykonywane są wyliczenia związane z oświetleniem na podstawie pozycji wierzchołków, kierunków wektorów normalnych ściany, pozycji źródeł światła i własności powierzchniowych obiektu. Następnie figury te są rzutowane z wykorzystaniem transformacji projekcji na pozycję na płaszczyźnie ekranu. Jednocześnie z pamięci systemowej do pamięci tekstur wczytywane są tekstury w postaci tablicy pikseli. Następnie są one konwertowane z formatu w jakim są przechowywane na właściwą liczbę komponentów. W procesie rasteryzacji łączone są efekty obu operacji. Jest to krok polegający na konwersji geometrii sceny i pikseli tekstury na   
tzw. fragmenty. Są to kwadraty odpowiadające pikselom w końcowym obrazie zawierające informację o ich kolorze i odległości od obserwatora. Kolejnym krokiem są operacje na fragmentach takie jak teksturowanie, usuwanie ukrytych powierzchni, rozmywanie   
i maskowanie. Tak przetworzony fragment jest zapisywany jako piksel. Gotowy obraz może zostać wyświetlony na ekranie.

### OpenGL jako maszyna stanów

OpenGL działa na zasadzie maszyny stanów. Większość operacji, poczynając od ustawienia koloru rysowania, używanej tekstury, własności powierzchniowych czy ustawień oświetlenia na modyfikacji macierzy projekcji i transformacji skończywszy, polega na zmianie aktualnego stanu OpenGL. Zdefiniowanych jest wiele binarnych zmiennych stanu pozwalających na proste włączanie i wyłączanie danego trybu rysowania. Każda zmienna stanu posiada swoją domyślną wartość co pozwala uniknąć konieczności jej specyfikowania jeśli nie występuje potrzeba   
z niej korzystania. OpenGL umożliwia również sprawdzenie aktualnych wartości zmiennych stanu poprzez szereg zdefiniowanych metod w zależności od typu danej zmiennej. Pełna lista dostępnych zmiennych stanu wraz z informacją o ich działaniu i domyślnych wartościach znajduje się w dokumentacji API [11]. Oferowane konfigurowalne atrybuty obejmują:

* macierze transformacji
* parametry oświetlenia
* parametry dla efektu mgły
* parametry rysowania dla punktów, linii i wielokątów
* parametry teksturowania
* parametry bufora kolorów, głębokości, szablonowego
* parametry pikseli

Do ustawiania binarnych zmiennych stanu w OpenGL służą polecenia ***glEnable(…)***i ***glDisable(…)***. Przyjmują one jeden parametr którym jest ustawiana zmienna stanu. Przykładowe polecenie ma postać:   
glEnable(GL\_DEPTH\_TEST);

### Składnia poleceń w OpenGL

Wszystkie komendy API OpenGL rozpoczynają się od przedrostka **gl**, po którym następuje ich nazwa, w której każde słowo pisane jest z wielkiej litery (np. ***glLoadIdentity()***). Podobnie, zdefiniowane zostały stałe rozpoczynające się od przedrostka **GL\_**  po którym następuje ich nazwa, złożona jedynie z wielkich liter, gdzie kolejne słowa rozdzielone są znakiem podkreślenia ‘\_’.

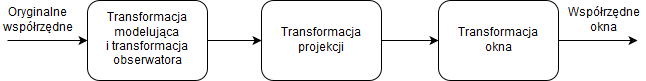
Ponadto, ponieważ OpenGL oferuje liczne przeciążenia dostępnych metod, na końcu części poleceń znajduje się postfiks. Składa się on z cyfry i jednej, dwóch lub trzech liter,   
np. ***glVertex3fv(…)***. Cyfra zawarta w postfiksie oznacza liczbę parametrów metody lub długość wektora będącego jej parametrem. Kolejna litera lub dwie oznaczają typ parametrów   
i mogą przyjmować jedną z następujących wartości:

* b – wartość typu bajt (8-bitowa liczba całkowita),
* s – wartość typu short (16-bitowa liczba całkowita),
* i – wartość typu int (32-bitowa liczba całkowita),
* f – wartość typu float (32-bitowa liczba rzeczywista),
* d – wartość typu double (64-bitowa liczba rzeczywista),
* ub – wartość typu bajt bez znaku (8-bitowa dodatnia liczba całkowita),
* us – wartość typu short bez znaku (16-bitowa dodatnia liczba całkowita),
* ui – wartość typu int bez znaku (32-bitowa dodatnia liczba całkowita)

Ostatnim znakiem który może znaleźć się na końcu polecenia jest litera ***v***. Jej wystąpienie informuje, że polecenie przyjmuje wektor / tablicę wartości jako jeden z parametrów. Wiele poleceń definiuje zarówno wersje wektorowe jak i wersje niewektorowe. W celu uproszczenia zapisu, polecenia przywoływane w tej pracy będą jedynie na podstawie ich nazwy. Dla odróżnienia poleceń zawierających postfiks, znak ‘\*’ będzie dopisywany na ich końcu, np. ***glColor\*()***.

### Transformacje

Jedną z podstawowych koncepcji, zrozumienie których jest kluczowe do pracy z OpenGL są transformacje. Pozwalają one na manipulację położenia wierzchołków w trójwymiarowej przestrzeni sceny i sposobu w jaki scena jest wyświetlana. W efekcie otrzymujemy zrzutowany na płaszczyznę obraz złożony z pikseli.

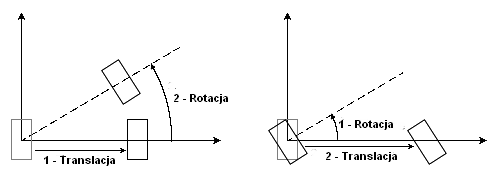


Rys. 2.2 Kolejność aplikowania transformacji

Według kolejności przedstawionej na powyższym schemacie (Rys.2.2), pierwszym etapem przetwarzania otrzymanych wierzchołków jest transformacja modelująca i transformacja obserwatora. Zostaną one szczegółowo omówione w rozdziale **2.1.8**. Jej wyniki są przekazywane do transformacji projekcji omówionej w rozdziale **2.1.9**. Ostatnią transformacją wykonywaną na wierzchołkach jest przekształcenie ich do przestrzeni okna (viewport). Na podstawie informacji o wysokości i szerokości dostępnego obszaru wyświetlania ustawiane są naturalne proporcje dla wyświetlanego obrazu. Tak przetworzony obraz złożony z pikseli może być bezpośrednio wyświetlony na ekranie.

### Macierze transformacji

Z matematycznego punktu widzenia transformacje są sekwencją wymnożonych ze sobą macierzy o wymiarach 4x4. Warto zrozumieć takie spojrzenie na transformacje ponieważ pozwala ono pojąć znaczenie kolejności ich nakładania – mnożenie macierzy nie jest przemienne. Dla zobrazowania tej koncepcji poniżej zamieszczony został przykład (Rys. 2.3).

 Rys. 2.3 Wpływ kolejności wykonywania transformacji na końcowy efekt

Pierwszym krokiem definiowania transformacji jest wczytywanie do bufora **B** macierzy tożsamości **I**. Każda kolejna zdefiniowana transformacja polega na przemnożeniu aktualnej macierzy **B** przez macierz transformacji **T**, co daje w efekcie nową macierz **BT**. Po zaaplikowaniu wszystkich transformacji, tj. wymnożeniu ich macierzy **IT1T2[...]TN** otrzymana macierz jest wykorzystywana do wyznaczania nowej pozycji wierzchołka w przestrzeni 3D. Dzieje się to poprzez wymnożenie macierzy **B** przez oryginalną pozycję wierzchołka **v**. Oznacza to, że transformacje dla wierzchołka aplikowane są w kolejności odwrotnej do ich specyfikacji **I**(**T1**(**T2**([…](**TN**(**v**))))) w tzw. lokalnym układzie odniesienia. By uzyskać globalny układ odniesienia należy odwrócić kolejność aplikowania transformacji przed ich wymnożeniem.

### Transformacja modelująca i transformacja obserwatora

Transformacje modelujące służą do przekształcenia współrzędnych z układu lokalnego do układu sceny czyli do rozmieszczenia poszczególnych obiektów w przestrzeni sceny. Istnieją trzy podstawowe rodzaje transformacji: translacja, rotacja i skalowanie. Specyfikując ujemne wartości dla transformacji skalowania można uzyskać odbicie względem wybranej osi. Przed rozpoczęciem definiowania transformacji należy wybrać odpowiednią macierz (***GL\_MODELVIEW***). Następnie należy zresetować jej wartość poprzez załadowanie do niej macierzy jednostkowej. Od tego momentu można nakładać na nią kolejne transformacje. Należy pamiętać, że aby uzyskać transformacje w globalnym układzie odniesienia trzeba odwrócić kolejność stosowania transformacji.

Transformacja obserwatora pozwala na wybranie pozycji i orientacji z jakiej obserwowana jest scena. Jej ustawienie odbywa się na tej samej macierzy co transformacji modelującej. Warto zaznaczyć, że efekt transformacji obserwatora można uzyskać również za pomocą transformacji modelującej poprzez odpowiednie przesunięcie / obrót modelowanej sceny.

Czytelną analogią dla obu rodzajów transformacji jest spojrzenie na zagadnienie transformacji   
w ujęciu fotograficznym. Pierwszym krokiem przed wykonaniem zdjęcia jest zestawienie sceny polegające na właściwym rozmieszczeniu w niej obiektów (transformacja modelująca). Kolejnym krokiem jest ustawienie aparatu we właściwej pozycji w stosunku do fotografowanego obiektu (transformacja w układzie obserwatora). Taki sam efekt możemy jednak uzyskać poprzez ustawienie obiektu we właściwy sposób względem kamery (transformacja modelująca w układzie świata).

W celu ustawienia transformacji modelującej i obserwatora wykorzystując API OpenGL należy w pierwszej kolejności wybrać odpowiednią macierz transformacji z wykorzystaniem polecenia ***glMatrixMode(GL\_MODELVIEW)***. Następnie należy wczytać do niej macierz jednostkową używając polecenia ***glLoadIdentity()***. Po wykonaniu powyższych czynności można przejść do definiowania kolejnych transformacji.

Do wygodnego ustawienia transformacji obserwatora służy polecenie ***gluLookAt(…)***, które jako parametry przyjmuje kolejno 3 współrzędne odpowiadające pozycji obserwatora,   
3 współrzędne wyznaczające kierunek patrzenia oraz 3 współrzędne odpowiadające kierunkowi wektora ‘do góry’. Przykładowe polecenie ma postać:   
gluLookAt(0,0,0, 0,0,-1, 0,1,0);

Do definiowania kolejnych przekształceń modelujących służą polecenia ***glTranslate\*(…)***, ***glRotate\*(…)*** oraz ***glScale\*(…)***.

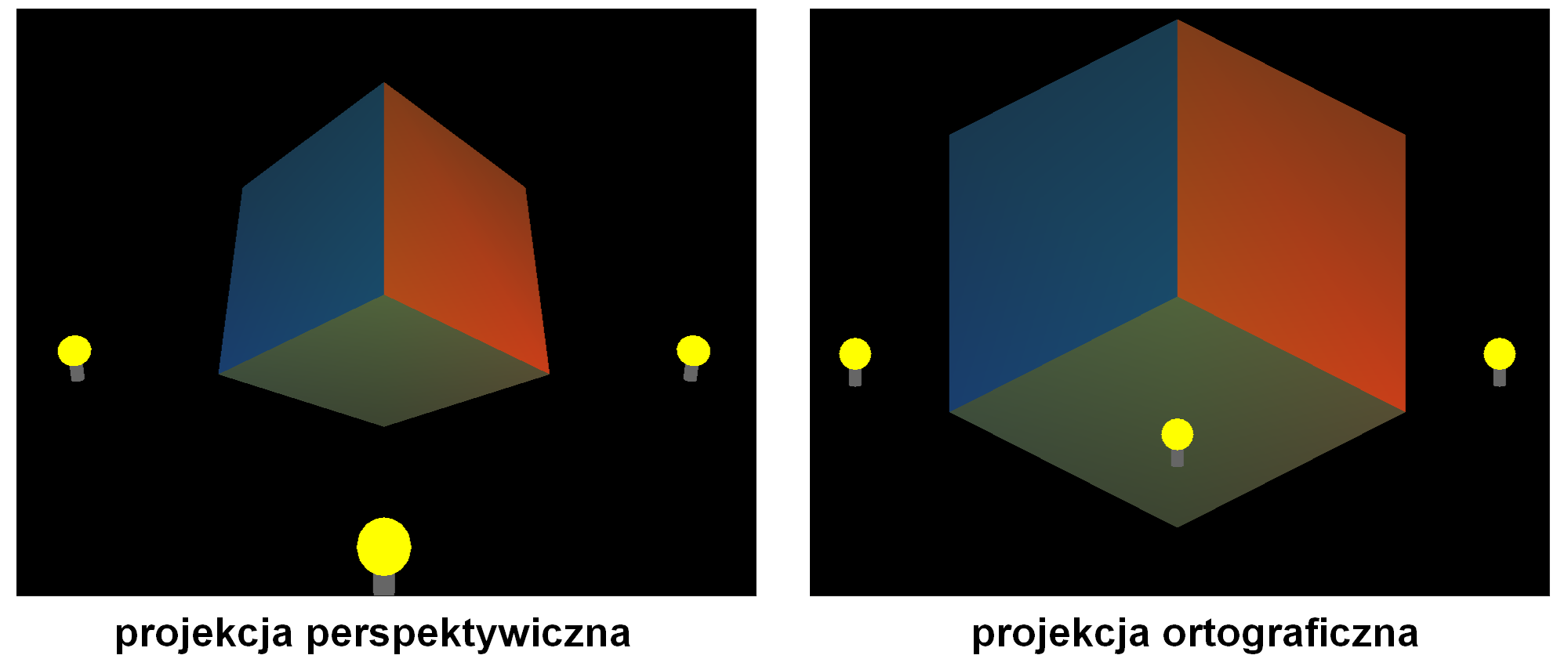
Polecenie ***glTranslate\*(…)*** przyjmuje 3 liczby rzeczywiste X, Y, Z odpowiadające przesunięciu względem odpowiednich osi. Przykładowe polecenie ma postać:   
glTranslatef(1.3, -2.1, 4.2);

Polecenie ***glScale\*(…)*** przyjmuje 3 liczby rzeczywiste X, Y, Z odpowiadające skalowaniu względem odpowiednich osi. Wartość ‘1’ oznacza oryginalną skalę, wartość -1 odbicie względem danej osi. Przykładowe polecenie odbijające obiekt względem osi X, rozciągające go dwukrotnie względem osi Y i redukujące dwukrotnie względem osi Z ma postać:   
glScalef(-1, 2, 0.5);

Polecenie ***glRotate\*(…)*** przyjmuje 4 parametry. Pierwszy z nich wyznacza kąt obrotu   
w kierunku przeciwnym do wskazówek zegara, wyrażony w stopniach. Kolejne trzy wyznaczają oś obrotu. Przykładowe polecenie obracające obiekt o 45º względem osi Y i 9º względem osi Z ma postać:   
glRotatef(45, 0, 1, 0.2);

### Transformacja projekcji – rzutowanie

Transformacja projekcji odpowiada za ustalenie w jaki sposób obiekty sceny będą zachowywać się w zależności od głębokości ich położenia w scenie. Przed rozpoczęciem pracy należy wybrać odpowiednią macierz, tym razem jako parametr podając GL\_PROJECTION. Ponownie, pierwszą operacją jaką należy wykonać jest wczytanie macierzy jednostkowej. Następnie należy wybrać pożądany sposób projekcji. OpenGL oferuje dwa tryby projekcji – perspektywiczną i ortograficzną (Rys. 2.4).



Rys. 2.4 Tryby transformacji projekcji w OpenGL

Naturalnym efektem obserwowalnym w naturze jest perspektywa sprawiająca, że obiekty oddalone od kamery wydają się mniejsze a dwie równoległe linie (np. tory) zbiegają się na horyzoncie. Transformacją zadaniem której jest symulowanie tego efektu jest transformacja perspektywiczna. Jest ona zwykle stosowana wtedy, gdy oczekuje się foto-realistycznego efektu mającego symulować rzeczywistość.

Inną dostępną metodą projekcji jest projekcja ortograficzna. W przeciwieństwie do projekcji perspektywicznej, odległość obiektów od obserwatora nie wpływa na ich wielkość. Ponadto, charakterystyczną cechą takiego podejścia jest to, że równoległe linie pozostają zawsze równoległe. Taki rodzaj projekcji sprawdza np. się w wypadku szkiców architektonicznych gdzie ważnym jest zachowanie właściwych kątów i rozmiarów obiektów.

Aby ustawić transformację projekcji wykorzystując API OpenGL należy w pierwszej kolejności wybrać odpowiednią macierz transformacji z wykorzystaniem polecenia ***glMatrixMode(GL\_PROJECTOIN)***. Następnie należy wczytać do niej wartość jednostkową używając polecenia ***glLoadIdentity()***. Po wykonaniu powyższych czynności można przejść do definiowania kolejnych transformacji. Należy wybrać jedno z poleceń odpowiadające pożądanemu sposobowi projekcji.

Do definicji transformacji ortograficznej służy polecenie ***glOrtho(…)***. Wymaga ono podania parametrów wyznaczających kolejno lewą, prawą, dolną i górną krawędź bliskiej płaszczyzny odcięcia, odległość bliskiej i dalekiej płaszczyzny odcięcia. Przykładowe polecenie ma postać:  
glOrtho(-1, 1, -1, 1, 0.1, 42);

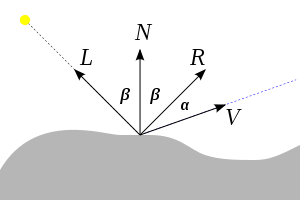
W celu uzyskania transformacji perspektywicznej można wykorzystać dwa polecenia. Pierwsze jest częścią biblioteki OpenGL, podczas gdy drugie jest częścią biblioteki GLU i oferuje wygodniejszy sposób na definiowanie tego rodzaju projekcji. Polecenie ***glFrustum(…)*** przyjmuje takie same parametry jak polecenie ***glOrtho(…)***. Polecenie ***gluPerspective(…)*** upraszcza proces definicji tego przekształcenia do podania zakresu kąta widzenia, stosunku szerokości do wysokości płaszczyzn odcięcia oraz odległości bliskiej i dalekiej płaszczyzny odcięcia. Przykładowe polecenie ma postać:   
gluPerspective(30, 1, 0.1, 42);

### Oświetlenie – model Phonga

Kolejnym ważnym aspektem OpenGL są możliwości oferowane przez API pozwalające na wygenerowanie realistycznego oświetlenia sceny. OpenGL symuluje oświetlenie wykorzystując model Phonga. Jest to empiryczny model opisujący sposób w jaki powierzchnie odbijają światło. Definiuje on odbicie rozproszone powierzchni matowych oraz odbicie lustrzane powierzchni błyszczących. Natężenie światła odbitego, docierającego ze źródła światła do obserwatora, zależne jest więc od natężenia światła odbitego lustrzanie / odblasku (Is), natężenia światła rozproszonego (Id), natężenia światła otoczenia (Ia) oraz odpowiadającym im własnościom odbicia dla powierzchni (ks, kd, ka), co prezentuje poniższy wzór:

(2.1)

Natężenie odbitego światła lustrzanego w danym punkcie płaszczyzny zależy od kąta pomiędzy wektorem skierowanym do obserwatora a wektorem wyznaczającym kierunek odbicia światła. Natężenie odbitego światła rozproszonego zależy od kąta padania światła na powierzchnię.



Rys. 2.5 Odbicie światła od nieregularnej powierzchni

Przyjmując oznaczenia jak na powyższym rysunku (Rys.2.5), gdzie N oznacza wektor normalny powierzchni, wektor L kierunek do źródła światłą, wektor R kierunek odbicia światła, a wektor V kierunek do obserwatora oraz wszystkie wektory są znormalizowane, natężenia odblasku (2.2) i odbitego światła rozproszonego (2.3) wynoszą odpowiednio:

(2.2)

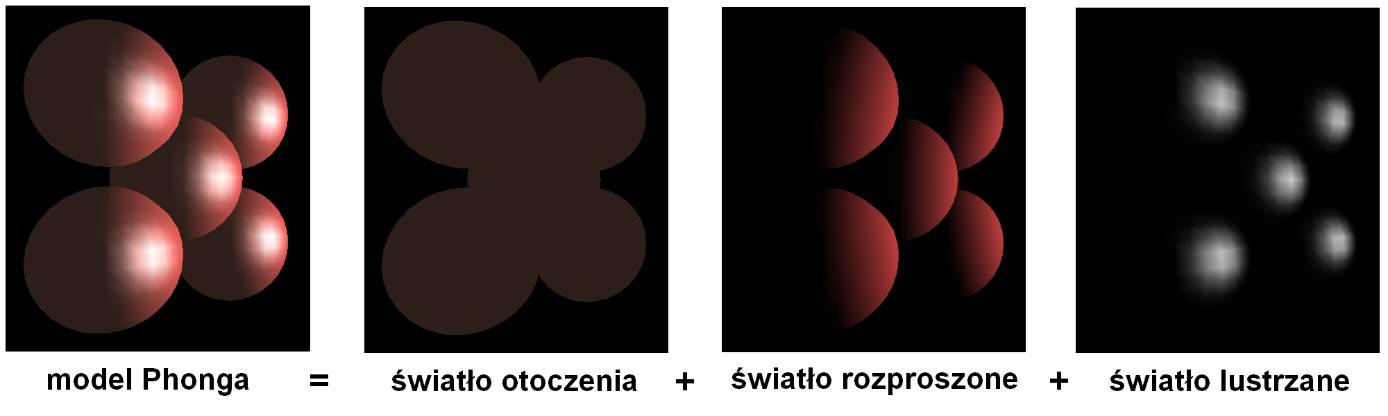
(2.3)

Wykładnik ‘n’ w powyższym wzorze oznacza zdefiniowany dla powierzchni współczynnik odbicia dla światła lustrzanego. Ostateczny wzór na natężenie światła docierającego do obserwatora przyjmuje postać:

(2.4)

### Oświetlenie – materiały

OpenGL oferuje możliwość definiowania jak powierzchnie na które pada światło wyglądają po ich oświetleniu. Proces ten polega na ustaleniu intensywności z jaką materiał odbija każdy   
z kanałów RGB dla wybranego typu światła. Pojęcie światła zostało rozbite na trzy komponenty, z których każdy posiada czerwony, zielony i niebieski kanał (RGB). Dostępne komponenty światła to światło otoczenia (ambient light), światło rozproszone (diffuse light) oraz światło lustrzane / odblask (specular light). Istnieje możliwość kontrolowania skupienia odblasku z pomocą wykładnika odblasku (specular exponent). Dodatkowo model oświetlenia w OpenGL definiuje kolor emitowany (emissive color) przez obiekt, co pozwala zdefiniować kolor obiektu, bez względu na istniejące w scenie źródła światła [8]. Ustalenie współczynnika odbicia poszczególnych kanałów wybranego komponentu światła dla powierzchni pozwala zdefiniować w jaki sposób będzie ona reagować na ten typ oświetlenia, co przekłada się na jej kolor. Dla każdego z wierzchołków wszystkie komponenty są wyliczane oddzielnie a następnie ich wynik jest sumowany aby uzyskać foto-realistyczne oświetlenie. Poniżej znajduje się ilustracja ukazująca poszczególne komponenty światła (Rys. 2.6) i ich omówienie.



Rys. 2.6 Model Phonga w rozbiciu na komponenty

Pierwszym dostępnym komponentem światła jest światło otoczenia. Modeluje ono światło wielokrotnie odbite, dochodzące ze wszystkich kierunków. Dodanie źródła światła posiadającego ten komponent zapewnia równomierne oświetlenie wszystkich elementów   
w scenie z każdej strony. Odbite od powierzchni jest rozpraszane równomiernie we wszystkich kierunkach. Pozwala więc ono w łatwy sposób regulować ogólny poziom jasności sceny.

Drugim z komponentów światła jest światło rozproszone. Jest to światło które pochodzi   
z określonego kierunku. Sprawia to, że jego jasność jest zależna od kąta pod jakim pada ono na oświetlaną powierzchnię. Po odbiciu od powierzchni, podobnie jak światło otoczenia, jest ono rozpraszane równomiernie we wszystkich kierunkach. Powoduje to, że oświetlona w ten sposób powierzchnia jest tak samo jasna bez względu na pozycje obserwatora.

Kolejnym rodzajem światła wspieranym przez OpenGL jest światło lustrzane, odblask. Odpowiada ono światłu pochodzącemu z określonego kierunku. W odróżnieniu od światła rozproszonego, światło odbija się od powierzchni w jednym kierunku, gdzie kąt odbicia zależny jest od kąta padania. Światło to może powodować obserwowalny często w świecie rzeczywistym odblask, tj. białe „plamy światła” w miejscach w których od metalicznej powierzchni odbija się silne światło, np. słoneczne.

Ostatnim z dostępnych komponentów światła jest emitowany kolor. Jest on definiowany jedynie dla powierzchni i ma symulować światło pochodzące z obiektu. W modelu OpenGL zwiększa on jasność obiektu bez względu na istniejące źródła światła. Ten rodzaj oświetlenia nie wprowadza dodatkowego źródła światła w scenie.

Do definiowania materiałów w API OpenGL wykorzystuje się polecenie ***glMaterial\*(…)***. Posiada ono trzy parametry. Pierwszy z nich wskazuje stronę ściany, do której jest aplikowany materiał. Przyjmuje jedną z wartości ***GL\_FRONT***, ***GL\_BACK***, ***GL\_FRONT\_AND\_BACK***. Drugi parametr wskazuje na definiowany komponent i może przyjmować wartości: ***GL\_AMBIENT***, ***GL\_DIFFUSE***, ***GL\_SPECULAR***, ***GL\_EMISSION***, ***GL\_SHININESS***. Ostatnim parametrem jest wektor definiujący wartości odbicia dla każdego z kanałów RGB lub w przypadku parametru ***GL\_SHININESS*** wartość wykładnika. Przykładowe wywołanie może mieć postać:   
GLfloat redMaterial[] = { 0.9, 0.3, 0.3 };   
glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, redMaterial);

### Oświetlenie – źródła światła

Aby zdefiniować źródło światła w OpenGL należy w pierwszej kolejności ustalić jakiego efektu oczekujemy. Możemy otrzymać światło pochodzące z określonego kierunku, z konkretnego punktu rozchodzące się we wszystkich kierunkach, lub emitowane stożkowo z danego punktu. Zostały one przedstawione na poniższym rysunku (Rys. 2.7).



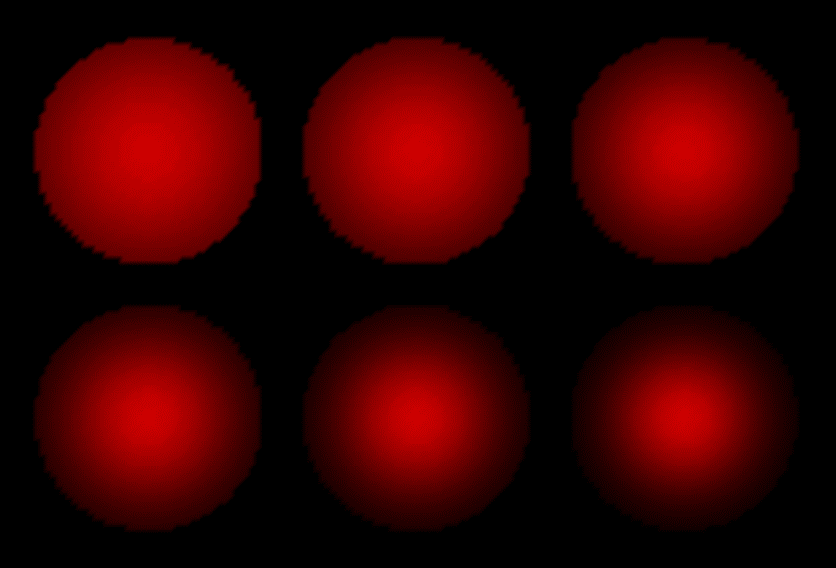
Rys. 2.7 Rodzaje źródeł światła w OpenGL

Najprostszym z możliwych źródeł światła jest źródło światła otoczenia (ambient light). Jedynym parametrem który należy dla niego zdefiniować jest jego nasilenie w każdym   
z kanałów RGB. Jako że jest to światło otoczenia nie posiadające źródła, jego pozycja nie ma znaczenia. Nie zostało ono przedstawione na powyższym rysunku (Rys.2.7), ponieważ jednolite oświetlenie wszystkich elementów sceny tym światłem uniemożliwia ich odróżnienie – w efekcie wszystkie obiekty w scenie przyjmują kolor światła.

Bardziej złożonym rodzajem światła jest światło kierunkowe (directional light). Modeluje ono nieskończenie odległe źródło światła, co sprawia, że jego promienie mogą być traktowane jako równoległe. W przypadku tego rodzaju oświetlenia wykorzystuje się zwykle światło rozproszone (diffuse) i / lub światło lustrzane (specular).

Kolejnym możliwym źródłem światła jest światło pozycyjne (positional light). Jest to światło umieszczone w wybranym punkcie w przestrzeni sceny. W przypadku tego rodzaju światła można uzyskać efekt jego tłumienia wraz ze wzrostem odległości od źródła światła. Jako, że domyślnie promienie rozchodzą się we wszystkich kierunkach z wybranego punktu, generowany zostaje efekt jasnego oświetlenia pobliskich powierzchni i słabszego oświetlenia powierzchni oddalonych. Ponownie, w przypadku tego rodzaju oświetlenia wykorzystuje się zwykle światło rozproszone (diffuse) i / lub światło lustrzane (specular).

Specjalnym typem światła pozycyjnego jest światło punktowe / reflektorowe (spotlight). Pozwala ono na ograniczenie do stożka kierunku w jakim emitowanie jest światło. Dodatkowo umożliwia kontrolowanie rozkładu intensywności emitowanego światła (najjaśniejsze   
w centrum stożka) za pomocą wykładnika – współczynnika skupienia. Im wyższa jego wartość tym światło jest bardziej skoncentrowane w środku stożka. Na poniższej ilustracji (Rys. 2.8) przedstawiono efekty dla współczynnika skupienia przyjmującego kolejno wartości: 25, 38, 49, 64, 81, 100.



Rys. 2.8 Światło reflektorowe dla różnych wartości współczynnika skupienia

Do ustawiania źródeł światła w API OpenGL służy polecenie ***glLight\*(…)***. Przyjmuje ono   
3 parametry. Pierwszy z nich wskazuje na światło dla którego definiowane są właściwości   
i przyjmuje wartości od ***GL\_LIGHT0*** do ***GL\_LIGHT7.*** Drugi parametr wyznacza definiowaną właściwość światła i może przyjmować wartości ***GL\_AMBIENT***, ***GL\_DIFFUSE***, ***GL\_SPECULAR***, ***GL\_POSITION***, ***GL\_SPOT\_DIRECTION***, ***GL\_SPOT\_EXPONENT***, ***GL\_SPOT\_CUTOFF***. Ostatni parametr jest wartością dla wybranej własności wybranego światła.

Własności ***GL\_AMBIENT***, ***GL\_DIFFUSE***, ***GL\_SPECULAR*** służą do definiowania koloru światła za pomocą wektora 4 wartości odpowiadających kanałom RGBA. Przykładowe polecenie ma postać:   
GLfloat redLight[] = { 0.9, 0.3, 0.3, 1 };   
glLightfv(GL\_LIGHT0, GL\_DIFFUSE, redLight });

Własność ***GL\_POSITION*** służy do definiowania pozycji światła za pomocą wektora 4 wartości odpowiadających pozycji X, Y, Z, W. Jeśli W przyjmuje jest równe 0 definiowane światło jest światłem kierunkowym, a wartości X, Y, Z odpowiadają wektorowi wyznaczającemu kierunek emisji światła. Przykładowe polecenie ma postać:   
GLfloat position[] = { 2, -1, 0.3, 1 };   
glLightfv(GL\_LIGHT0, GL\_POSITION, position);

Własność ***GL\_SPOT\_DIRECTION*** służy do definiowania kierunku dla światła reflektorowego za pomocą wektora 3 wartości odpowiadających kierunkowi wyznaczanemu przez wektor X, Y, Z. Przykładowe polecenie ma postać:   
GLfloat direction[] = { 0.8, 0.6, 0 };   
glLightfv(GL\_LIGHT0, GL\_SPOT\_DIRECTION, direction);

Własność ***GL\_SPOT\_EXPONENT*** służy do definiowania współczynnika skupienia dla światła reflektorowego. Przykładowe polecenie ma postać:   
glLightf(GL\_LIGHT0, GL\_SPOT\_EXPONENT, 49);

Własność ***GL\_SPOT\_CUTOFF*** służy do definiowania kąta odcięcia dla światła reflektorowego. Przykładowe polecenie ma postać:   
glLightf(GL\_LIGHT0, GL\_SPOT\_CUTOFF, 12);

### Oświetlenie – tłumienie światła

W celu modelowania spadku intensywności światła wraz ze wzrostem odległości od jego źródła API OpenGL umożliwia definiowanie współczynnika tłumienia. Jest on wyliczany na podstawie wzoru:

(2.5)

Oznaczenia przyjęte we wzorze to:

* kc – stała tłumienia
* kl – współczynnik liniowy tłumienia
* kq – współczynnik kwadratowy tłumienia
* d – odległość od źródła

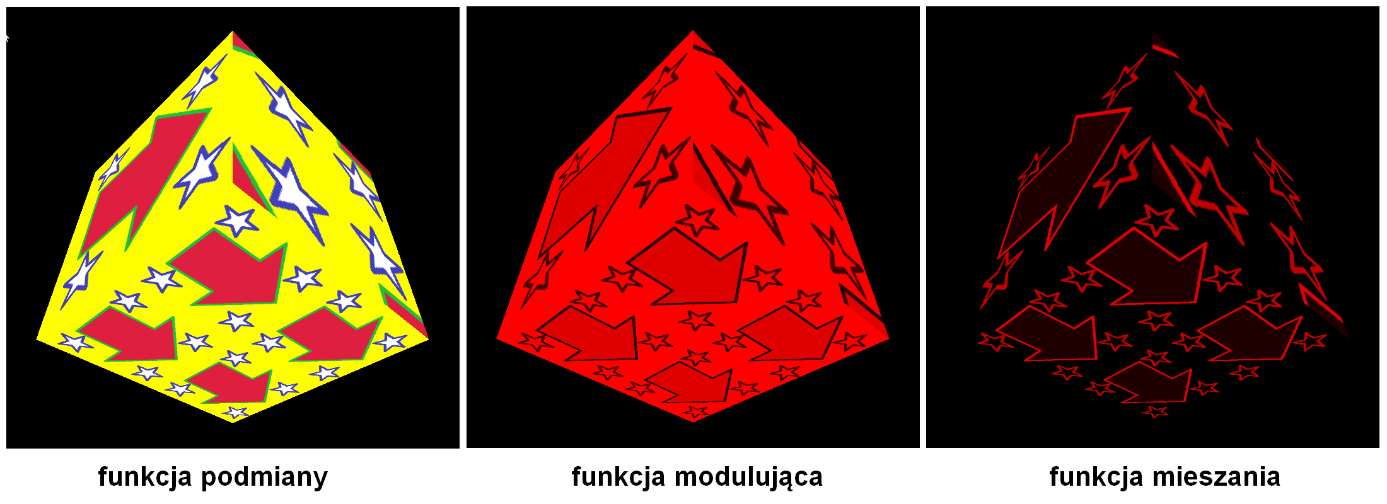
Do ustawiania tłumienia dla źródeł światła w API OpenGL służy polecenie ***glLight\*(…)*** którego składnia opisana została w poprzednim rozdziale (**2.1.13**).

Własności ***GL\_CONSTANT\_ATTENUATION***, ***GL\_LINEAR\_ATTENUATION***, ***GL\_QUADRATIC\_ATTENUATION*** służą do definiowania odpowiednio stałej tłumienia, liniowego współczynnika tłumienia i kwadratowego współczynnika tłumienia. Przykładowe polecenie ma postać:   
glLightf(GL\_LIGHT0, GL\_LINEAR\_ATTNUATION, 3);

### Teksturowanie

Proces teksturowania jest kolejną funkcją OpenGL mogącą nadać scenie realizmu. Pozwala on na naniesienie dwu-wymiarowego obrazu na powierzchnie obiektu, zapewniając łatwy sposób na przedstawienie obiektów podobnych do ich rzeczywistych odpowiedników. Sprawdza się to w przypadku dążenia do uzyskania powierzchni przypominających prawdziwe materiały takie jak drewno, kamień, roślinność lub tkaniny. Ponadto pozwala na zmniejszenie liczby obiektów potrzebnych do modelowania wielu przedmiotów – bez teksturowania pojedyncza powierzchnia posiadać może, w zależności od użytego trybu cieniowania, jeden kolor lub kilka interpolowanych na podstawie kolorów wierzchołków. Przykładowo, do modelowania powierzchni ceglanej ściany konieczne jest zdefiniowanie obiektu dla każdej cegły oraz przestrzeni między nimi. Stosując teksturowanie możemy osiągnąć zbliżony efekt definiując jedynie jeden obiekt, na powierzchnię którego naniesiony zostanie powielony obraz przedstawiający wycinek ceglanej ściany. API OpenGL umożliwia wykorzystanie tekstur jedno, dwu i trójwymiarowych. W programie zaimplementowano wsparcie dla tekstur dwuwymiarowych.

Przed rozpoczęciem procesu teksturowania należy przekonwertować otrzymany na wejściu obraz na tablicę bajtów. OpenGL wspiera tekstury jedno-, dwu-, trój- i czterokomponentowe, gdzie liczba komponentów oznacza liczbę bajtów przypadających na każdy teksel. Tekstury jednokomponentowe wykorzystywane mogą być wykorzystane do modulacji jaskrawości (***GL\_LUMINANCE***), intensywności (***GL\_INTENSITY***) lub przezroczystości (***GL\_APLHA***). Tekstury dwukomponentowe mogą być wykorzystane jako maska jaskrawości (***GL\_LUMINANCE\_ALPHA***). Tekstury trójkomponentowe służą do teksturowania   
z wykorzystaniem kanałów RGB tekstury (***GL\_RGB***) i dają efekt nałożenia obrazu na teksturowaną powierzchnię, który to efekt intuicyjnie kojarzy się z teksturowaniem. Tekstury czterokomponentowe, działające podobnie jak tekstury trójkomponentowe, wykorzystują dodatkowo kanał alpha tekseli (***GL\_RGBA***). W programie wykorzystano tekstury trójkomponentowe (***GL\_RGB***).



Rys. 2.9 Efekty różnych funkcji teksturujących

Powyższa ilustracja (Rys. 2.9) demonstruje, że tekstury mogą być aplikowane do powierzchni na kilka sposobów z wykorzystaniem dostępnych w OpenGL funkcji. Mogą być one nanoszone bezpośrednio na powierzchnie jako ostateczny kolor danej ściany, używane do modulowania koloru ściany pod oświetleniem lub mieszane z kolorem powierzchni. Ostateczny kolor tekstury ściany może być obliczany z pomocą pięciu funkcji opisanych poniżej. Zamieszczone zostają wzory poszczególnych funkcji dla tekstury trójkomponentowej ***GL\_RGB***. Wzory dla pozostałych trybów teksturowania znajdują się w specyfikacji OpenGL [11].

W poniższych wzorach przyjmuje się oznaczenia:

* Ct – kolor texela
* Co – oryginalny kolor teksturowanej powierzchni
* Cm – kolor dla funkcji mieszania

Funkcja podmiany (***GL\_REPLACE***) i funkcja kalki (***GL\_DECAL***) dla tekstur trójkomponentowych dają taki sam efekt. Polega on na całkowitym zastąpieniu oryginalnego koloru ściany kolorami RGB teksela. Wzór wyznaczający ostateczny kolor ma postać:

(2.6)

Funkcja modulująca (***GL\_MODULATE***) służy do modulacji koloru powierzchni na podstawie koloru teksela. Wzór na ostateczny kolor ma postać:

(2.7)

Funkcja sumująca (***GL\_ADD***) polega na prostym sumowaniu koloru teksela  
i oryginalnego koloru powierzchni. Nie zachowując ostrożności, łatwo można uzyskać efekt prześwietlenia. Wzór wyznaczający ostateczny kolor ma postać:

(2.8)

Funkcja mieszania (***GL\_BLEND***) polega na mieszaniu koloru powierzchni z ustalonym drugim kolorem zapisanym w zmiennej ***GL\_TEXTURE\_ENV\_COLOR***. Wartość koloru teksela służy jako współczynnik mieszania koloru powierzchni z kolorem zapisanym w zmiennej***.*** Ostateczny kolor wyznaczany jest na podstawie wzoru:

(2.9)

Do ustawiania tekstur służą w OpenGL polecenia ***glBindTexture(…), glTexParameter(..), glTexEnv(…)*** oraz ***glTexImage2D(…)***. Pierwsze z nich ustawia aktualnie używaną teksturę. Drugie z poleceń pozwala na ustawianie rozlicznych parametrów tekstury kontrolujących jak tekstury są aplikowane do ścian. Do wybrania jednej z opisanych powyżej funkcji służy polecenie ***glTexEnx(…).*** Ostatnia z przytoczonych funkcji służy do wczytania do pamięci tekstur tablicy reprezentującej teksturę. Przykładowe wywołania dla tych poleceń mają postać:

glBindTexture(GL\_TEXTURE\_2D, GL\_TEXTURE0);

glTexParameter(GL\_TEXTURE\_2D,GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);

glTexEnvf(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_REPLACE);

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGB, width, height, 0, GL\_RGB, GL\_UNSIGNED\_BYTE, texture);

## Wspierane formaty

### Uzasadnienie wyboru wykorzystanych formatów plików

W celu zapewnienia opcji prostej modyfikacji sceny umożliwione zostało jej wczytywanie  
z pliku. Wspierane formaty zostały wybrane ze względu na ich czytelność i łatwość modyfikacji. Geometrię sceny, wektory normalne, mapowanie tekstur oraz ściany są definiowane w pliku OBJ. Do przechowywania informacji o materiałach wybrany został format MTL. W celu umożliwienia prostego wczytywania tekstur obsługiwane są popularne rastrowe formaty obrazu.

### Definicja geometrii sceny – format OBJ

Do wczytywania geometrii sceny posłużył format OBJ. Został on wybrany ze względu na jego czytelność i łatwość modyfikacji. Program wspiera jedynie część oficjalnej specyfikacji dla tego formatu [10] pozwalającą na definiowanie ścian, zbudowanych z wierzchołków, ich wektorów normalnych oraz współrzędnych tekstury. Nie jest wspierane wczytywanie wielu materiałów dla pojedynczego obiektu. Poniżej znajduje się opis struktury pliku   
i obsługiwanych wartości.

W pliku OBJ każdy wpis zaczyna się od nowej linii. Dozwolone typy wpisów to:

* **v** – wierzchołki geometryczne
* **vt** – wierzchołki tekstur
* **vn** – wektory normalne dla wierzchołków
* **f** – ściany

Poniżej zamieszczone są zasady ich definiowania:

**v X Y Z**

Wierzchołki geometryczne **v** definiowane są poprzez zestaw 3 liczb rzeczywistych rozdzielonych spacjami odpowiadających pozycji X, Y i Z w przestrzeni.

**vt U V**

Wierzchołki tekstur **vt** są definiowane poprzez zestaw 2 liczb rzeczywistych dodatnich rozdzielonych spacjami odpowiadających pozycji odpowiadających wartościom UV. U jest położeniem w teksturze względem osi X. V odpowiada położeniu w teksturze względem osi OY. W lewym dolnym rogu tekstury znajduje się punkt (0,0), a w prawym górnym rogu punkt (1,1). Podanie wartości z zakresu [0, 1] pozwala na uzyskanie wycinka tekstury. Podanie wartości większej niż 1 skutkuje powtórzeniem tekstury N razy w celu mapowania jej na ścianę.

**vn I J K**

Wektory normalne **vn** są definiowane poprzez zestaw 3 liczb rzeczywistych rozdzielonych spacjami odpowiadających współrzędnym I, J, K. Wartości I, J i K są wartościami wyznaczającymi kierunek wektora odpowiednio dla osi X, Y i Z. Wektor nie musi być znormalizowany.

**f iv1[/ivt1/ivn1] iv2[/ivt2/ivn2] iv3[/ivt3/ivn3] […]**

Ściany **f** są definiowane za pomocą 3 lub więcej krotek rozdzielonych spacjami składających się z indeksów wierzchołków zdefiniowanych w pliku. Krotka zawiera indeksy wierzchołka geometrycznego, wierzchołka tekstur i wektora normalnego w podanej kolejności, rozdzielone jedynie przez ukośniki ‘/’. Wartość indeksu wierzchołka geometrycznego jest wymagana, pozostałe dwie wartości są opcjonalne. Indeksowanie wierzchołków rozpoczyna się od 1. Przykłady prawidłowych wpisów to:

* **f 1 2 3 –** minimalna poprawna definicja
* **f 1/1 2/2 3/3 -** definicja zawierająca referencje do wierzchołków tekstur
* **f 1//1 2//2 3//3 -** definicja zawierająca referencje do wektorów normalnych
* **f 1/1/1 2/2/2 3/3/3 -** definicja zawierająca referencje do wierzchołków tekstur   
  i wektorów normalnych
* **f 1 2 3 4 5 6 7 8 9 10 -** definicja ściany zbudowanej z wielu wierzchołków
* **f 1/1/1 2/2/2 3/3/3 4/4/4 5/5/5 -** definicja ściany zbudowanej z wielu wierzchołków zawierająca referencje do wierzchołków tekstur i wektorów normalnych

Przykładowy plik definiujący prosty sześcian z ustalonymi wektorami normalnymi   
i mapowaniem tekstur ma postać:

# Vertices

v 1 -1 -1

v 1 -1 1

v -1 -1 1

v -1 -1 -1

v 1 1 -1

v 1 1 1

v -1 1 1

v -1 1 -1

# Texture coordinates

vt 0 0

vt 1 0

vt 1 1

vt 0 1

vt 1 0

# Normals

vn 0 -1 0

vn 0 1 0

vn 1 0 0

vn -0 -0 1

vn -1 -0 -0

vn 0 0 -1

# Faces (vertex/texcoord/normal)

f 1/1/1 2/2/1 3/3/1 4/4/1

f 5/4/2 8/1/2 7/5/2 6/3/2

f 1/2/3 5/3/3 6/4/3 2/1/3

f 2/4/4 6/1/4 7/2/4 3/3/4

f 3/1/5 7/2/5 8/3/5 4/4/5

f 5/3/6 1/4/6 4/1/6 8/2/6

### Definicja materiałów – format MTL

Do przechowania i wczytywania materiałów dla powierzchni wybrany został format MTL. Oferuje on prosty i czytelny sposób definicji właściwości powierzchniowych. Ponownie, program zapewnia wsparcie tylko części oficjalnej specyfikacji tego formatu [10] istotnej dla aplikacji. Obsługiwane parametry obejmują współczynniki odbicia RGB dla światła otoczenia, światła rozproszonego, światła odbicia / odblasku, wykładnik dla światła odbicia oraz współczynnik RGB ilości światła emitowanego. Współczynniki te odpowiadają parametrom opisanym w rozdziale **2.1.12**. Poniżej znajduje się opis struktury pliku   
i obsługiwanych wartości.

Plik MTL może zawierać wiele definicji materiałów. Każda z nich, rozpoczyna się od linii zawierającej wpis **newmtl nazwa\_materiału**. Każda definicja może zawierać:

* Ka – współczynnik odbicia dla światła otoczenia (ambient color)
* Kd – współczynnik odbicia dla światła rozproszonego (diffuse color)
* Ke – współczynnik ilości światła emitowanego (emitted color)
* Ks – współczynnik odbicia dla światła lustrzanego / odblasku (specular color)
* Ns – wykładnik dla światła lustrzanego / odblasku (specular exponent)

Poniżej zamieszczone są zasady ich definiowania:

**newmtl nazwa\_materiału**

Wyznacza początek definicji materiału. Nazwa materiału nie może zawierać spacji, można zastąpić je znakiem ‘-‘ lub ‘\_’.

**Ka r g b**

Współczynnik odbicia dla światła otoczenia definiowany jest za pomocą rozdzielonych spacjami trzec

h liczb rzeczywistych z zakresu [0, 1] odpowiadających wartościom odbicia dla komponentów RGB światła otoczenia.

**Kd r g b**

Współczynnik odbicia dla światła rozproszonego definiowany jest za pomocą rozdzielonych spacjami trzech liczb rzeczywistych z zakresu [0, 1] odpowiadających wartościom odbicia dla komponentów RGB światła rozproszonego.

**Ks r g b**

Współczynnik odbicia dla odblasku definiowany jest za pomocą rozdzielonych spacjami trzech liczb rzeczywistych z zakresu [0, 1] odpowiadających wartościom odbicia dla komponentów RGB światła lustrzanego / odblasku.

**Ns e**

Wykładnik wartości odblasku jest definiowany za pomocą liczby rzeczywistej. Przyjmuje wartości z zakresu [0, 100]. Jego wysoka wartość skutkuje małym, skoncentrowanym odblaskiem.

**Ke r g b**

Współczynnik emisji definiowany jest za pomocą rozdzielonych spacjami trzech liczb rzeczywistych z zakresu [0, 1] odpowiadających ilości światła emitowanego dla   
komponentów RGB.

Przykładowy plik definiujący trzy materiały ma postać:

newmtl default

Ns 1

Ka 0.25 0.25 0.25

Kd 0.2 0.3 0.3

Ks 0.4 0.4 0.4

Ke 0.1 0.1 0.0

newmtl green\_diffuse\_only

Kd 0 0.8 0

newmtl specular\_only

Ks 0.8 0.8 0.8

Ns 60

### Tekstury – popularne formaty rastrowe

Choć istnieją gotowe formaty plików służące do przechowywania tekstur, aby zapewnić jak największą dowolność w dodawaniu własnych tekstur przez użytkownika, program wspiera wczytywanie popularnych rastrowych plików graficznych i tworzenie z nich tekstur. Jedynym kryterium jakie musi spełniać obraz wykorzystywany jako tekstura, są jego wymiary – wartości wysokości i szerokości muszą być równe i być potęgą liczby 2.

### Pozostałe elementy sceny

Pozostałe elementy definicji sceny (transformacja obserwatora, oświetlenie, wspierane flagi OpenGL, ścieżki do wyżej opisanych plików), nie będące częścią definicji żadnego z wyżej wymienionych formatów, znajdują się w pliku konfiguracyjnym. Jego struktura została opisana w rozdziale **4.2.2**.

# Projekt, architektura i implementacja rozwiązania

## Analiza wymagań

Przed przystąpieniem do pracy przeprowadzono proces analizy wymagań dla dostarczanego rozwiązania. Program będący jednym z artefaktów tej pracy ma umożliwiać wykładowcy prezentowanie na zajęciach podstawowych funkcjonalności OpenGL. Powinien on posiadać czytelny graficzny interfejs użytkownika pozwalający na ustawianie rozlicznych parametrów OpenGL. Ustawione wartości parametrów powinny być aplikowane do sceny wczytanej z pliku i wyświetlanej przez program. Dodatkowo, ma on umożliwiać łatwą konfigurację wspieranych funkcjonalności. Ponadto, program powinien pozwalać na wygenerowanie kodu źródłowego, który po skompilowaniu utworzy scenę identyczną do tej otrzymanej w programie. Dodatkowym istotnym aspektem aplikacji jest łatwość instalacji umożliwiająca udostępnienie jej studentom, by ci mogli własnoręcznie eksperymentować z OpenGL. Ostatnim wymaganym elementem jest dostarczenie podstawowych przykładów prezentujących wspierane przez aplikację możliwości OpenGL.

Zidentyfikowano następujące wymagania funkcjonalne:

* Program ma wyświetlać scenę w OpenGL
* Program ma pozwalać na konfigurację wyświetlanej sceny poprzez graficzny interfejs użytkownika
* Program ma umożliwiać łatwy sposób na przechowywanie i wczytywanie konfiguracji
* Program ma wspierać generowanie kodu odpowiadającego skonfigurowanej scenie
* Program ma zawierać kilka przykładowych konfiguracji prezentujących jego możliwości

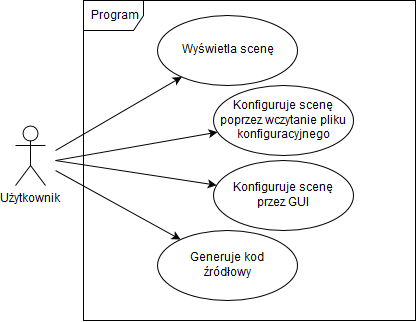
Oraz następujące wymaganie niefunkcjonalne:

* Program ma posiadać czytelny interfejs graficzny
* Program ma oferować łatwą instalację

Funkcjonalności OpenGL których wsparcie ma zapewniać program to:

* Przełączanie binarnych zmiennych stanu
* Ustawianie transformacji modelujących
* Ustawianie transformacji obserwatora
* Ustawianie transformacji projekcji
* Dodawanie i usuwanie źródeł światła
* Definiowanie materiału dla wszystkich obiektów w scenie
* Teksturowanie

Wymagania funkcjonalne przedstawione zostały na diagramie UML (Rys. 3.1).

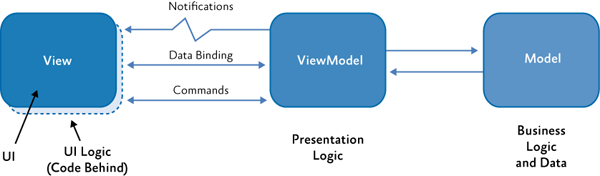


Rys. 3.1 Diagram przypadków użycia

## Architektura rozwiązania

### Wzorce architektoniczne

Program został zaprojektowany z wykorzystaniem architektury MVVM. Jej ogólny schemat został przedstawiony na poniższym diagramie.



Rys. 3.2 Schemat MVVM [4]

Jak widać na powyższym diagramie (Rys. 3.2) wzorzec MVVM wyróżnia trzy encje: widok (View), model widoku (ViewModel) oraz model. Jego zastosowanie pozwala rozdzielić logikę biznesową operującą na modelach od logiki interfejsu graficznego. Takie podejście oferuje solidną separacje odpowiedzialności. Mechanizm wiązania danych (data binding) zapewnia synchronizacje pomiędzy wyświetlanymi wartościami a wartościami przechowywanymi   
w modelach.

Ponadto, w rozwiązaniu zastosowany został paradygmat odwrócenia sterowania poprzez wstrzykiwanie zależności. Jest to wzorzec projektowy pozwalający obniżyć zależności występujące pomiędzy klasami oraz wspomagający stosowanie zasady pojedynczej odpowiedzialności. Osiągane to jest poprzez oddelegowanie tworzenia konkretnych serwisów od których zależna jest dana klasa do fabryki (kontenera IoC).

### Opis implementacji

W zgodzie z zasadą pojedynczej odpowiedzialności (single responsibility principle) zostały wydzielone serwisy dedykowane do konkretnych zadań. Poniżej zamieszczony został krótki opis poszczególnych serwisów zaimplementowanych na potrzeby programu oraz obrazujący wykres zależności między nimi (Rys. 3.3).

**CodeGenerationService** odpowiada za generowanie kodu na podstawie wczytanej sceny   
i aktualnego stanu ustawień OpenGL. W celu odczytania tych wartości korzysta   
z **OpenGLSettingsServiceModel** oraz **SceneServiceModel**.

**ConfigurationService** to serwis odpowiedzialny za odczytanie ustawień konfiguracji programu z pliku JSON.

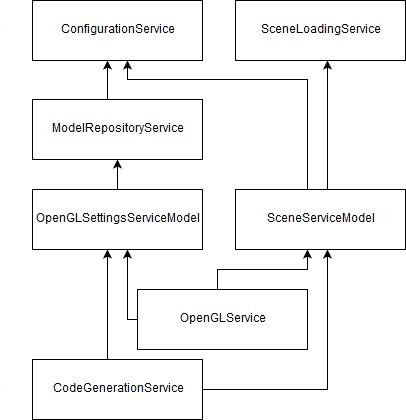
**ModelRepositoryService** jest odpowiedzialny za zarządzanie stanem modeli oraz zapewnia zunifikowany do nich dostęp. Używa **ConfigurationService** w celu odczytania domyślnych wartości dla modeli.

**OpenGLService** główny serwis odpowiedzialny za renderowanie sceny w OpenGL. Korzysta z **SceneServiceModel** i **OpenGLSettingsServiceModel** w celu odczytania aktualnego stanu OpenGL. Zawiera on całą logikę związaną z wykorzystaniem OpenGL API.

**OpenGLSettingsServiceModel** eksponuje aktualnie ustawione wartości modeli   
i zapewnia ich mapowanie do formatu z jakiego korzysta **OpenGLService**   
i **CodeGenerationService**. Implementuje wzorzec fasady dla **ModelRepositoryService.**

**SceneLoadingService** to serwis odpowiedzialny za wczytanie sceny z pliku. Pozwala na wczytanie pliku definicji sceny OBJ, pliku definicji materiałów MTL oraz tekstur w formie popularnych rastrowych plików graficznych.

**SceneServiceModel** jest odpowiedzialny za przechowywanie danych aktualnie wczytanej sceny. Korzysta z **ConfigurationService** w celu odczytania ścieżek plików OBJ, MTL i pliku tekstury. Wartości te są przekazywane do **SceneLoadingService** w celu wczytania wskazanych plików do pamięci programu.



Rys. 3.3 Zależności między serwisami

## Omówienie wykorzystanych technologii

### Metodyka agile

Ważnym aspektem realizacji pracy było zarządzanie procesem jego realizacji. Jest to jedna   
z kluczowych kwestii które należy uwzględnić przy realizacji dowolnego projektu informatycznego. Badania wykazują dużą skuteczność podejścia zwinnego (agile) w realizacji projektów informatycznych, także tych na małą skalę [3]. Z tego powodu podczas pracy nad programem wykorzystana została metodyka zwinna z elementami programowania ekstremalnego (Extreme Porgramming, XP). Manifest zwinnego programowania został zaimplementowany w następujących obszarach:

**Ludzie i interakcje ważniejsze od procesów i narzędzi**

Konsultacje z opiekunem pracy dyplomowej pozwoliły na ustalenie wymagań i zakresu pracy. Po przedstawieniu opiekunowi działającej wersji programu, następowała jego weryfikacja   
i dostosowywane wymagań jakie ma on spełniać.

**Działające oprogramowanie ważniejsze od szczegółowej dokumentacji**

Podczas wytwarzania oprogramowania nie była na bieżąco definiowana jego dokumentacja. Uwaga została skupiona na dostarczeniu narzędzia dydaktycznego oferującego wszystkie funkcjonalności wymagane przez opiekuna pracy.

**Współpraca z klientem ważniejsza od negocjacji umów**

Poza ogólnym wyznaczeniem dziedziny poprzez temat pracy dyplomowej, konkretne wymagania były definiowane we współpracy z opiekunem pracy. Przykładem jest wybrana wersja OpenGL. Początkowo planowano wykonać pracę z wykorzystaniem najnowszej wersji biblioteki, jednak po konsultacji z prowadzącym okazało się, że oczekuje on biblioteki w wersji legacy OpenGL w celu prostszej demonstracji podstawowych koncepcji grafiki komputerowej

**Reagowanie na zmiany ważniejsze od realizacji założonego planu**

Wszelkie zmiany sugerowane podczas konsultacji przez prowadzącego były wprowadzane   
w kolejnej iteracji wytwarzania oprogramowania.

Wykorzystane podczas pracy elementy Extreme programming obejmują priorytetyzację zadań oraz pracę w zamkniętych cyklach z których każdy polegał na dodaniu wsparcia dla kolejnej funkcjonalności OpenGL.

### Zarządzanie projektem

Istotną kwestią w realizacji dowolnego projektu jest odpowiednie nim zarządzanie. Ważnym jest zdefiniowanie pożądanych funkcjonalności, nadanie im priorytetów i kontrola liczby zrealizowanych zadań w stosunku do czasu jaki upłynął oraz liczby zadań oczekujących na realizację w stosunku do czasu jaki pozostał do planowanego terminu zakończenia pracy. Terminy rozpoczęcia i zakończenia pracy zostały wyznaczone przez początek i koniec semestru.

Do zarządzania listą zadań do wykonania wykorzystany został portal Trello. Pozwala on   
w wygodny sposób śledzić status zadań. Zdefiniowane zostały cztery statusy zadań: oczekiwanie na realizację, implementacja w trakcie, testowanie oraz zakończone. Każdemu   
z nich odpowiadała lista na której wyświetlane były zadania o danym statusie. Proces realizacji zadania rozpoczynał się zmianą jego statusu z początkowego ‘oczekuje na realizację’ na status ‘implementacja w trakcie’. Po ukończeniu implementacji funkcjonalności definiowanej przez zadanie jego status zmieniany został na ‘testowanie’. W tej fazie wykonywane były testy manualne nowo dodanej funkcjonalności weryfikujące czy spełnia ona wymagania zdefiniowane w zadaniu. W przypadku wykrycia błędów lub nie spełnienia zdefiniowanych wymagań status zadania był zmieniany ponownie na ‘implementacja w trakcie’. W przypadku nie znalezienia żadnych błędów oraz spełnienia wszystkich wymagań zadanie otrzymywało status ‘zakończone’ i rozpoczynana była praca nad kolejnym zadaniem o najwyższym priorytecie.

Portal Trello pozwala na wygodną integrację z wieloma platformami w tym z systemem kontroli wersji git. Ta możliwość została wykorzystana w celu umożliwienia śledzenia postępów implementacji bezpośrednio z poziomu tablicy projektu.

### System kontroli wersji

Aby umożliwić bezpieczne i łatwe dodawanie nowych funkcjonalności zastosowano system kontroli wersji git. Został on wybrany ze względu na jego popularność i wcześniejsze doświadczenie w pracy z jego wykorzystywaniem. Podstawową jednostką w pracy z git’em jest zatwierdzenie zmian (commit). Dobrą praktyką stosowania tego narzędzia jest częste zatwierdzanie zmian. Pozwala ono na łatwe ich wycofanie jeśli okaże się, że wprowadziły one błędy. System zapewnia także system gałęzi (branch’owania) pozwalający na równoczesną pracę nad osobnymi zadaniami w oddzielnych kontekstach. Po zakończeniu pracy nad daną funkcjonalnością zmiany dokonane w gałęzi (branch’u) są aplikowane do głównego kontekstu (merge’owane do głównego branch’a). W celu uniknięcia utraty dokonanej pracy w wypadku utraty lub awarii komputera zmiany przechowywane były również w zdalnym repozytorium na github’ie.

### Środowisko pracy

Podczas pracy korzystano ze zintegrowanego środowiska programowania Microsoft Visual Studio Community 2017. Dodatkowo, w celu wygodniejszej pracy podczas procesu wytwarzania oprogramowania wykorzystano dodatek Resharper Ultimate 2017. Do edytowania plików definicji sceny i materiałów wykorzystywane były zamiennie Notepad++ oraz Microsoft Visual Studio Code. Do przygotowania modeli 3D użyto programu blender [2]. System kontroli wersji git był obsługiwany z poziomu Visual Studio Team Explorer oraz z poziomu konsoli z wykorzystaniem nakładki na konsolę Cmder.

### Stos technologiczny

Do implementacji programu wykorzystana została platforma .NET ze względu na wcześniejsze doświadczenie w pisaniu aplikacji w tej technologii. W celu zastosowania wzorca MVVM użyty został Windows Presentation Foundation (WPF). Aby uniknąć konieczności ponownej implementacji popularnych funkcjonalności wykorzystano biblioteki dostępne w systemie zarządzania pakietami Nuget. Poniżej znajduje się lista wykorzystanych bibliotek, wraz pełnioną przez nie funkcją:

* SharpGL – biblioteka będąca implementacją API OpenGL; wybrana ze względu na dobrą integrację z WPF
* Newtonsoft.Json – biblioteka służąca do deserializacji konfiguracji przechowywanej   
  w pliku JSON
* JeremyAnsel.Media.WavefrontObj – biblioteka służąca do wczytywania plików OBJ oraz MTL
* Unity – biblioteka zapewniająca mechanizm odwrócenia sterowania (Inversion of Control) poprzez wstrzykiwanie zależności (dependency injection)

# Instrukcja użytkownika

## Instalacja

### Wymagania minimalne

Ponieważ program napisany został w technologii .NET Framework 4.6.1 do jego prawidłowego działania wymagany jest system z rodziny Windows, poczynając od wersji Windows 7.

### Proces instalacji

Program będzie można pobrać ze źródeł udostępnionych przez prowadzącego zajęcia. Instalacja programu polega na pozyskaniu wyżej wymienionego archiwum zawierającego skompilowany kod programu oraz jego rozpakowaniu z użyciem wybranego menedżera archiwów, np. dostępnego za darmo programu 7-zip. Po przeprowadzeniu tych operacji program jest gotowy do uruchomienia. Włączenie programu polega na wybraniu pliku ‘LegacyOpenGL.exe’ znajdującego się w głównym katalogu rozpakowanego archiwum i jego uruchomieniu poprzez dwukrotne nań kliknięcie.

## Konfiguracja

### Uruchamianie programu w wybranej konfiguracji

W celu umożliwienia przechowywania i łatwego przełączania pomiędzy konfiguracjami dodana została opcja uruchomienia programu z linii komend z możliwością specyfikacji konfiguracji innej niż domyślna. Aby to osiągnąć należy:

1. korzystając z powershell’a nawigować do głównego folderu programu i wywołać polecenie:

**Start-Process LegacyOpenGlApp.exe ścieżka\_do\_pliku\_z\_konfiguracją**

1. korzystając z wiersza polecenia nawigować do głównego folderu programu i wywołać polecenie:

**start LegacyOpenGlApp.exe ścieżka\_do\_pliku\_z\_konfiguracją**

Wartość **ścieżka\_do\_pliku\_z\_konfiguracją** należy zastąpić prawidłową ścieżką do pliku JSON. Wspierane są zarówno ścieżki bezpośrednie (absolute path) jak i pośrednie (relative path). Należy pamiętać, że jeśli ścieżka zawiera spację konieczne jest zawarcie jej   
w cudzysłowie. Jeśli plik znajduje się w tym samym katalogu co program, wystarczy podać jego nazwę (wraz z rozszerzeniem).

### Struktura pliku i dozwolone wartości

Wczytywanie predefiniowanych ustawień programu i sceny zostało zrealizowanie za pomocą przechowywania konfiguracji w pliku w formacie JSON. Oferuje on dostosowywanie następujących wartości:

#### Ustawianie binarnych zmiennych stanu

Sekcja ta zawiera listę deskryptorów binarnych zmiennych stanu OpenGL. Każdy deskryptor odpowiada jednej zmiennej eksponowanej dla użytkownika przez GUI. Ustawianie tych zmiennych w OpenGL opisane zostało w rozdziale **2.1.5**, a pełna lista obsługiwanych zmiennych dostępna jest w dokumentacji [11]. Wymaganym parametrem jest ‘StateVariable’, odpowiadający nazwie danej zmiennej OpenGL.

Opcjonalnymi wartościami są:

* DisplayName – nazwa zmiennej wyświetlana w programie
* IsActive – informacja o początkowym stanie danej zmiennej; w przypadku jej braku przyjmowana jest wartość domyślna ‘false’
* Description – opis zmiennej stanu wyświetlany jako podpowiedź po najechaniu na nią myszą

Struktura definicji:

"OpenGlFlags":   
[  
 {  
 "StateVariable": string,   
 "DisplayName": string,  
 "IsActive": boolean,  
 "Description": string  
 },  
 ...  
]

#### Ustawianie źródeł światła w scenie

Sekcja ta służy do definiowania źródeł światła w scenie. Ustawiane są one za pomocą listy zawierającej ich definicje. Dla każdego z jej elementów, wspierane są wartości odpowiadające parametrom światła opisanym w rozdziałach **2.1.13** i **2.1.14**. Dostępnymi parametrami są:

* Ambient – lista wartości dla kanałów R, G, B, A światła otoczenia
* Diffuse – lista wartości dla kanałów R, G, B, A światła rozproszonego
* Specular – lista wartości dla kanałów R, G, B, A światła odbitego / odblasku
* Position – lista wartości X, Y, Z, W odpowiadających pozycji światła w scenie
* SpotlightDirection – lista wartości X, Y, Z odpowiadających kierunkowi światła reflektorowego w scenie
* SpotlightExponent – wykładnik służący do kontrolowania rozkładu intensywności emitowanego światła reflektorowego
* SpotlightCutoff – parametr definiujący kąt odcięcia dla światła reflektorowego
* ConstantAttenuation – parametr definiujący stałą tłumienia światła; domyślną jego wartością jest 1
* LinearAttenuation – parametr definiujący współczynnik liniowy tłumienia światła
* QuadraticAttenuation – parametr definiujący współczynnik kwadratowy tłumienia światła

Domyślnymi wartościami dla parametrów (jeśli nie zaznaczono inaczej) są:

* dla list – listy o określonej długości, zawierające domyślne wartości dla   
  typu float: 0.0
* dla wartości typu float: 0.0

Zmienne X, Y, Z, W mogą przyjmować dowolne wartości rzeczywiste. Zmienne R, G, B, A mogą przyjmować wartości rzeczywiste z przedziału [0, 1]. Zmienne ConstantAttenuation, LinearAttenuation oraz QuadraticAttenuation mogą przyjmować wartości z zakresu liczb rzeczywistych nieujemnych. Współczynnik SpotlightCutoff przyjmuje wartości z przedziału [0, 90].

Struktura definicji:

"Lights":   
[  
 {  
 "Ambient": [ float, float, float, float ],  
 "Diffuse": [ float, float, float, float ],  
 "Specular": [ float, float, float, float ],  
 "Position": [ float, float, float, float ],  
 "SpotlightDirection": [ float, float, float ],  
 "SpotlightExponent": float,   
 "SpotlightCutoff": float,   
 "ConstantAttenuation": float,   
 "LinearAttenuation": float,   
 "QuadraticAttenuation": float  
 },  
 ...  
]

#### Ustawianie transformacji modelującej

Sekcja ta poświęcona jest definicji listy przekształceń modelujących opisanych w rozdziale **2.1.9**. Zawiera deskryptory elementarnych transformacji (przesunięcie, skalowanie, obrót), których złożenie daje pełną transformację modelującą, czyli przekształcenie współrzędnych  
z układu lokalnego do układu sceny. Dla każdego z jej elementów wspieranymi   
wartościami są:

* Transform – rodzaj transformacji; może przyjmować jedną z trzech wartości: ‘Translate’, ‘Rotate’, ‘Scale’; domyślną wartością jest pierwsza z wymienionych
* X – wartość transformacji względem osi X
* Y – wartość transformacji względem osi Y
* Z – wartość transformacji względem osi Z
* Angle – kąt obrotu (tylko dla transformacji ‘Rotate’)

Wartość transformacji względem danej osi w zależności od wybranej transformacji oznacza:

* Translate – liczbę jednostek o jaka obiekt zostanie przesunięty względem wybranej osi
* Scale – wartość o jaką zostanie przeskalowany obiekt względem danej osi, gdzie  
  1 odpowiada oryginalnej skali; ponieważ domyślną wartością jest 0, należy podać wartość 1 dla każdej z osi względem której nie jest dokonywane skalowanie
* Rotate – parametr ‘Angle’ definiuje w stopniach kąt obrotu a wartości X, Y, Z oś transformacji.

Struktura definicji:

"Transformations":

[

{

"Transform": string,

"X": float,

"Y": float,

"Z": float,

"Angle": float

},

...

]

#### Ustawianie transformacji obserwatora

Sekcja pozwala na zdefiniowanie parametrów dla transformacji obserwatora, opisanej   
w rozdziale **2.1.9.** Są to parametry dla polecenia ***gluLookAt(…)***, tj. początkowa pozycja, orientacja i kierunek patrzenia dla transformacji obserwatora. Określane tutaj deskryptory to:

* PositionX, PositionY, PositionZ – pozycja obserwatora odpowiednio względem   
  osi X, Y i Z
* AimX, AimY, AimZ – punkt X, Y, Z w przestrzeni sceny wyznaczający kierunek patrzenia
* UpX, UpY, UpZ – kierunek „do góry” ustalający obrót transformacji obserwatora względem osi wyznaczającej kierunek patrzenia.

Struktura definicji:

"Camera": {

"PositionX": float,

"PositionY": float,

"PositionZ": float,

"AimX": float,

"AimY": float,

"AimZ": float,

"UpX": float,

"UpY": float,

"UpZ": float

}

#### Ustawianie domyślnych ścieżek

Następujące parametry określają lokalizację plików definiujących pozostałe istotne elementy sceny. Wspierane są zarówno ścieżki bezpośrednie, jaki i pośrednie. W przypadku ścieżek pośrednich jako katalog bazowy służy katalog w którym znajduje się aplikacja.

Domyślana ścieżka pliku definicji sceny OBJ, który zostanie wczytany przy uruchomieniu programu jest podawana w następujący sposób:

"DefaultObjFilePath": string

Domyślana ścieżka pliku definicji materiałów MTL, który zostanie wczytany przy uruchomieniu programu jest podawana w następujący sposób:

"DefaultMtlFilePath": string

Domyślana ścieżka pliku graficznego wykorzystywanego do teksturowania, który zostanie wczytany przy uruchomieniu programu jest podawana w następujący sposób:

"DefaultTexturePath": string

## Instrukcja użytkowania

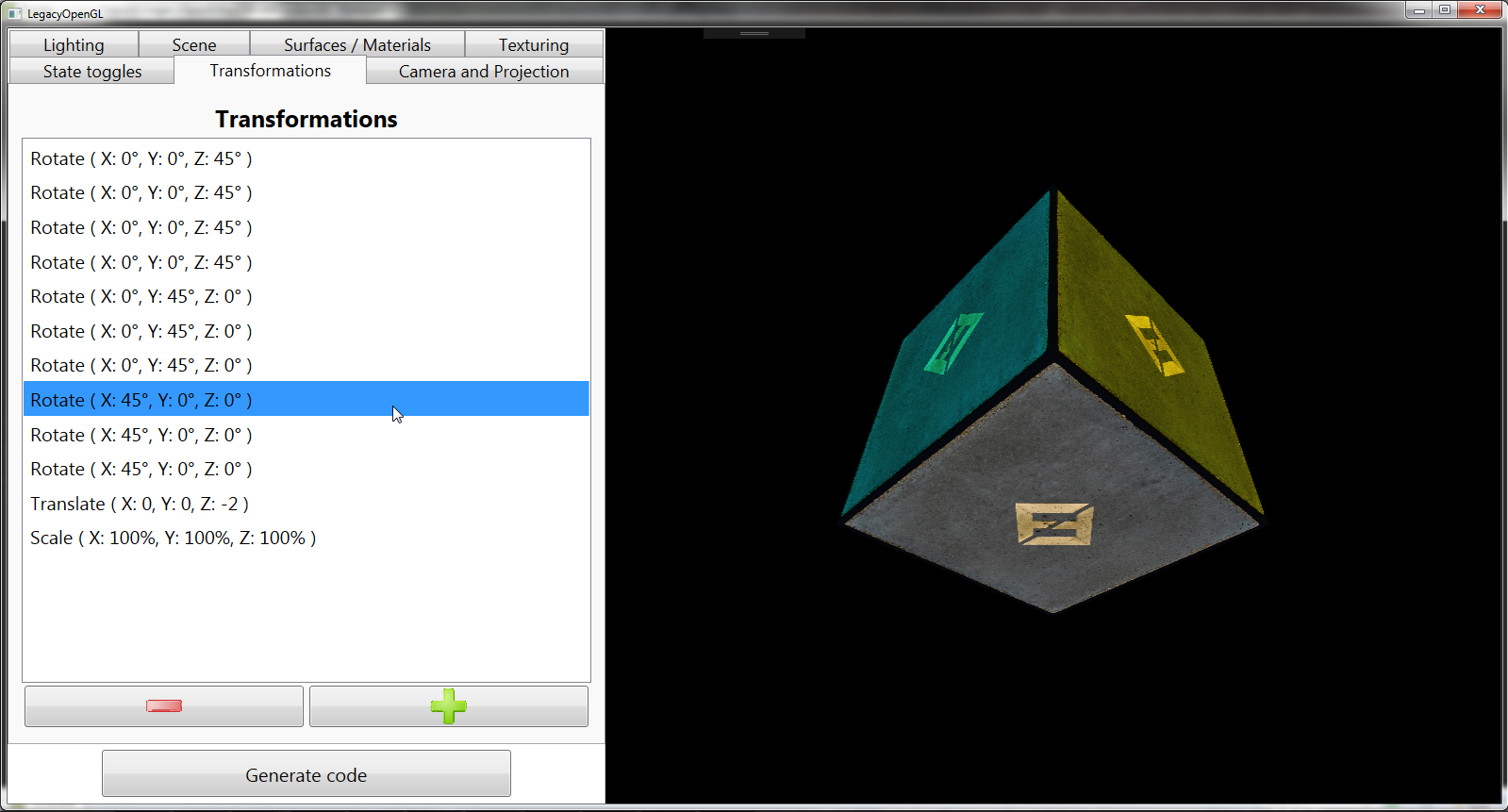
### Ustawianie binarnych zmiennych stanu



Rys. 4.1 Panel edytowania binarnych zmiennych stanu

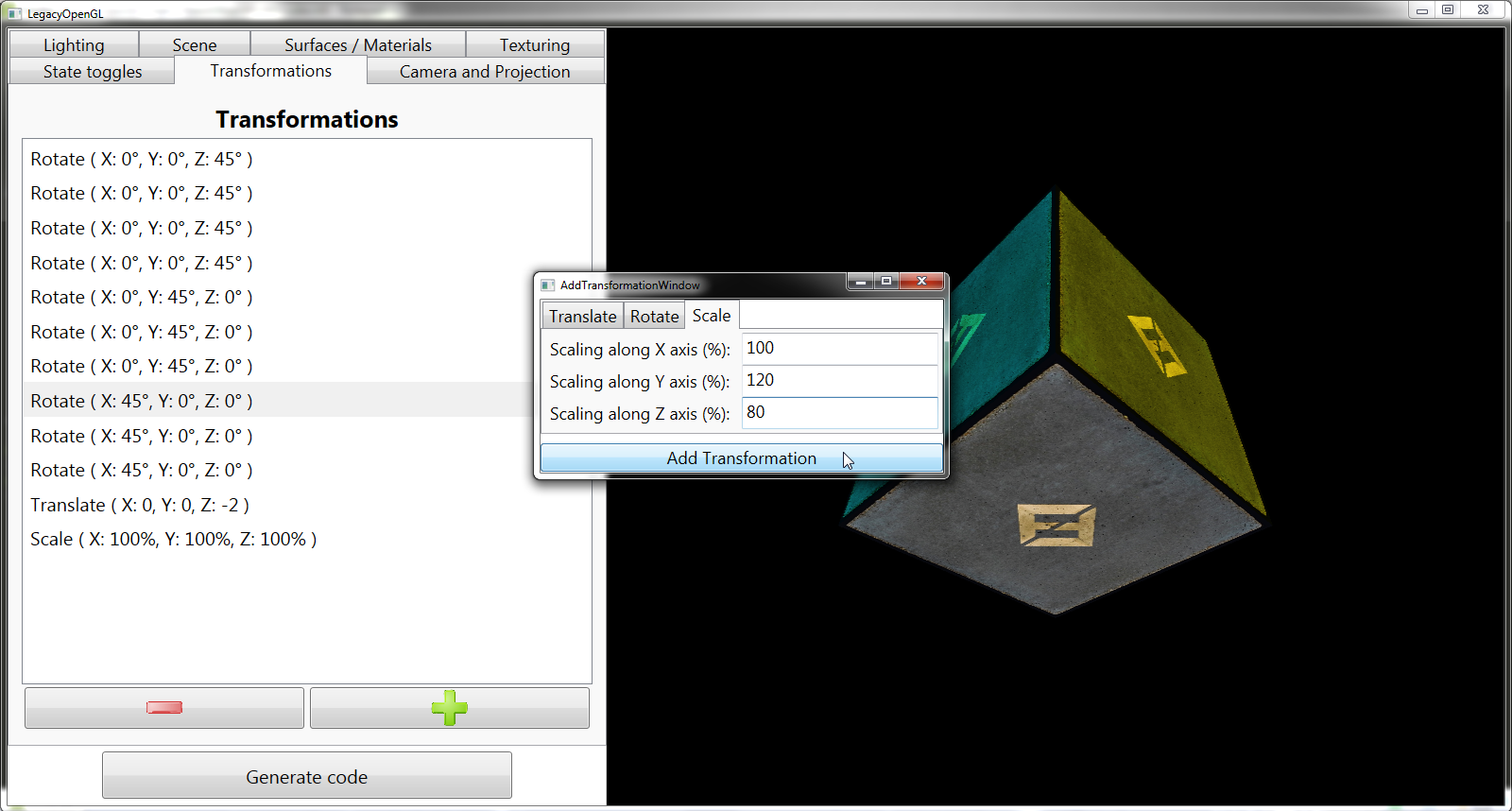
Na zaprezentowanym powyżej ekranie (Rys. 4.1) znajduje się lista dostępnych zmiennych stanu. W celu ułatwienia procesu nauki OpenGL, po najechaniu kursorem na zmienną, wyświetla się podpowiedź informująca o jej znaczeniu i działaniu. Istnieje możliwość przełączania pomiędzy stanem zmiennej wykorzystując pole wyboru znajdujące się przy lewej krawędzi listy.

### Dodawanie i usuwanie transformacji modelujących



Rys. 4.2 Panel dodawania i usuwania transformacji modelujących

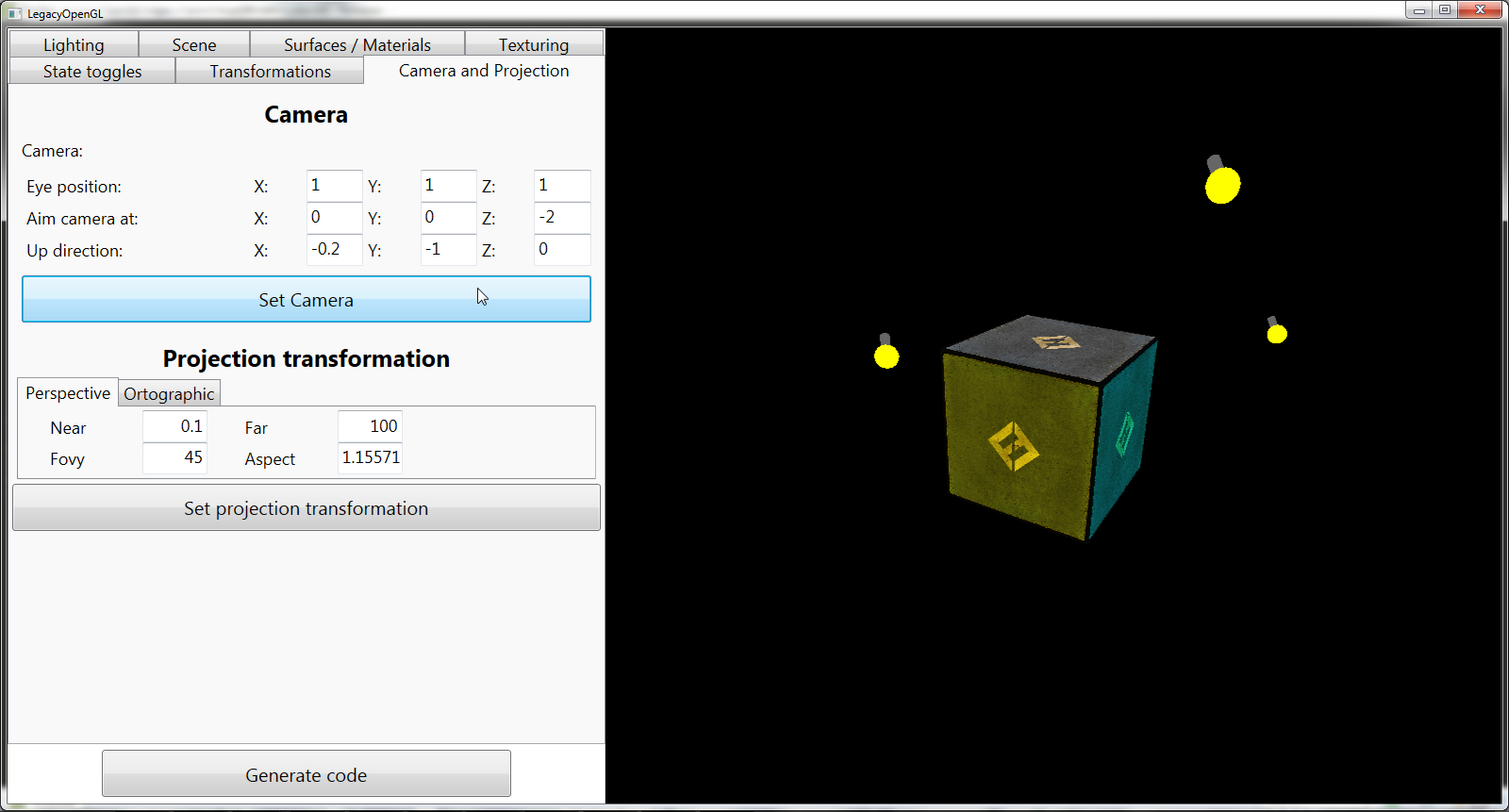
Na ekranie zawartym na powyższym rysunku (Rys. 4.2) przedstawiony został panel dodawania i usuwania transformacji modelujących. Po wybraniu dowolnej pozycji z listy istnieje możliwość jej usunięcia wykorzystując przycisk zawierający znak minus. Aby dodać nową transformację należy wybrać przycisk plus. Otwiera on poniższe okno.



Rys. 4.3 Okno dodawania transformacji

Okno dodawania transformacji (Rys. 4..3) pozwala na definiowanie nowych transformacji modelujących. Należy w tym celu wybrać pożądaną zakładkę z paska u góry okna odpowiadającą typowi dodawanego przekształcenia, a następnie zdefiniować wartości transformacji względem każdej z osi. Po ustaleniu tych wartości należy nacisnąć przycisk   
‘Add Transformation’ aby zaaplikować transformację do sceny. Okno nie zamyka się samoczynnie, dając możliwość dodawania kolejnych transformacji. Po zakończeniu procesu dodawania transformacji należy zamknąć okno.

### Ustawianie transformacji obserwatora i transformacji projekcji

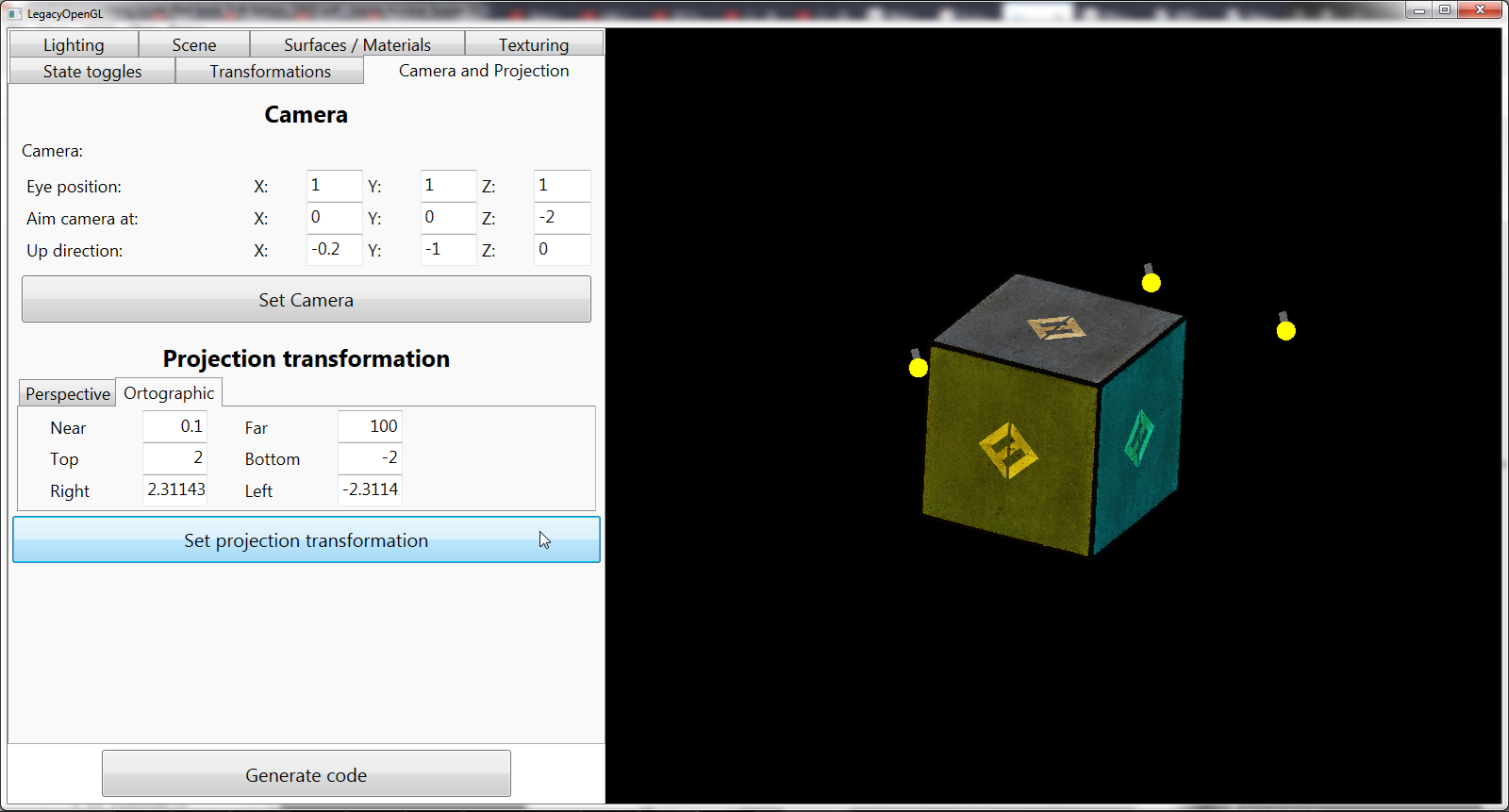


Rys. 4.4 Panel ustawiania transformacji obserwatora i transformacji projekcji (perspektywiczna)

Panel zaprezentowany powyżej (Rys. 4.4) agreguje dwie funkcjonalności OpenGL mające wpływ na sposób patrzenia na scenę.

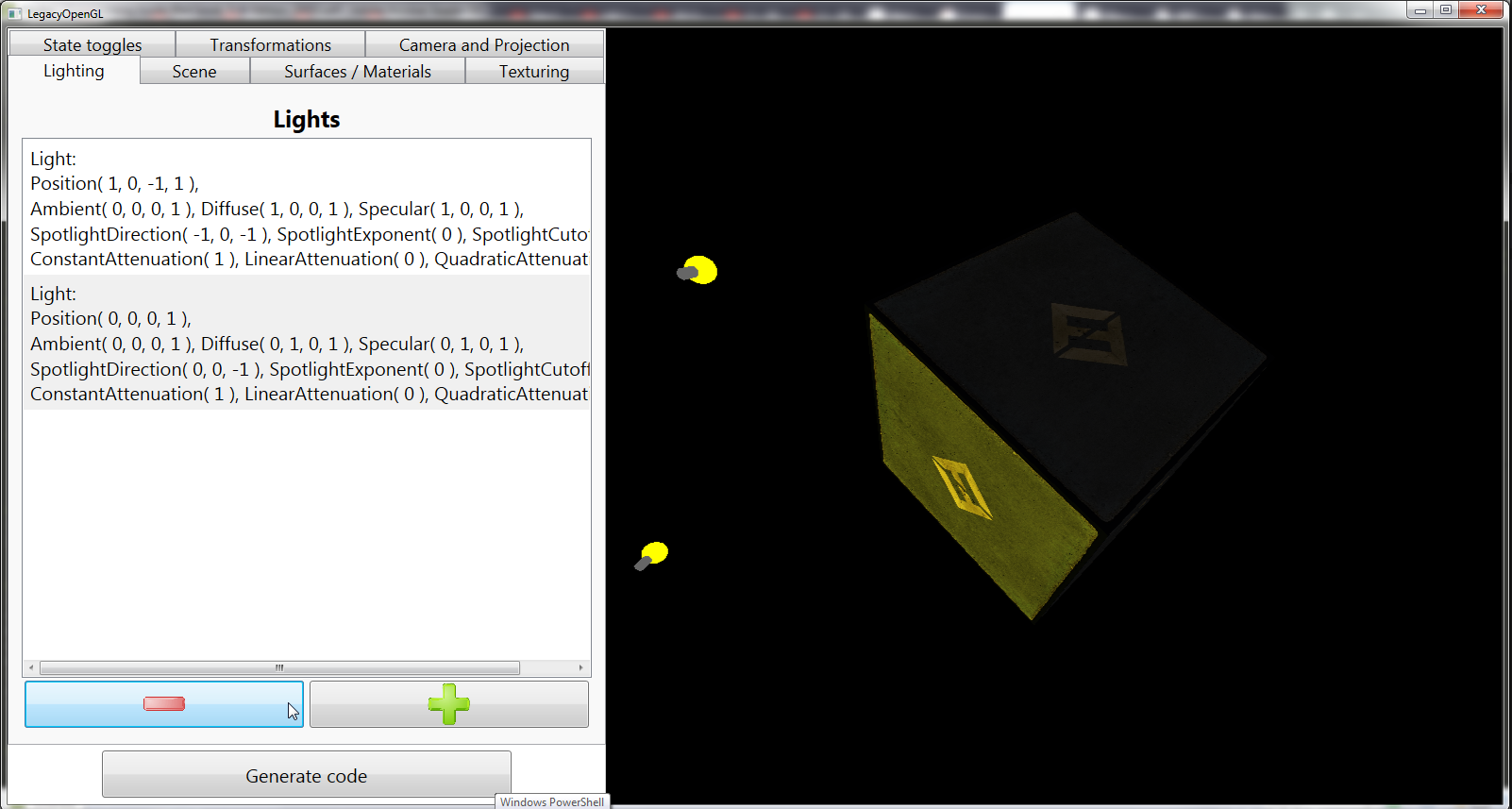
Pierwszą z nich jest ustawienie pozycji i kierunku patrzenia obserwatora. Aby je ustawić należy wprowadzić wartości dla pozycji obserwatora, kierunku obserwacji oraz kierunku ‘do góry’ wyznaczającego orientację obserwatora w przestrzeni.

Kolejną funkcjonalnością panelu jest ustawianie transformacji projekcji. Należy wybrać jedną z dwóch dostępnych zakładek – ‘Perspective’ lub ‘Orthographic’. W przypadku wybrania transformacji perspektywicznej możliwe jest ustawienie odległości płaszczyzn odcięcia wykorzystując pola ‘Near’ i ‘Far’. Pole ‘Fovy’ to kąt wyznaczający szerokość pola widzenia,   
a pole ‘Aspect’ stosunek szerokości do wysokości pola w którym wyświetlana jest scena. Jeśli wybrana zostanie transformacja ortograficzna (Rys. 4.5) należy zdefiniować wartości ‘Near’  
 i ‘Far’ oraz ‘Right’, ‘Left’, ‘Top’ i ‘Bottom’ wyznaczające krawędzie odcięcia. Dla porównania obu typów projekcji poniżej zamieszczony zostaje przykład tej samej sceny co powyżej w ujęciu ortograficznym (Rys. 4.5).



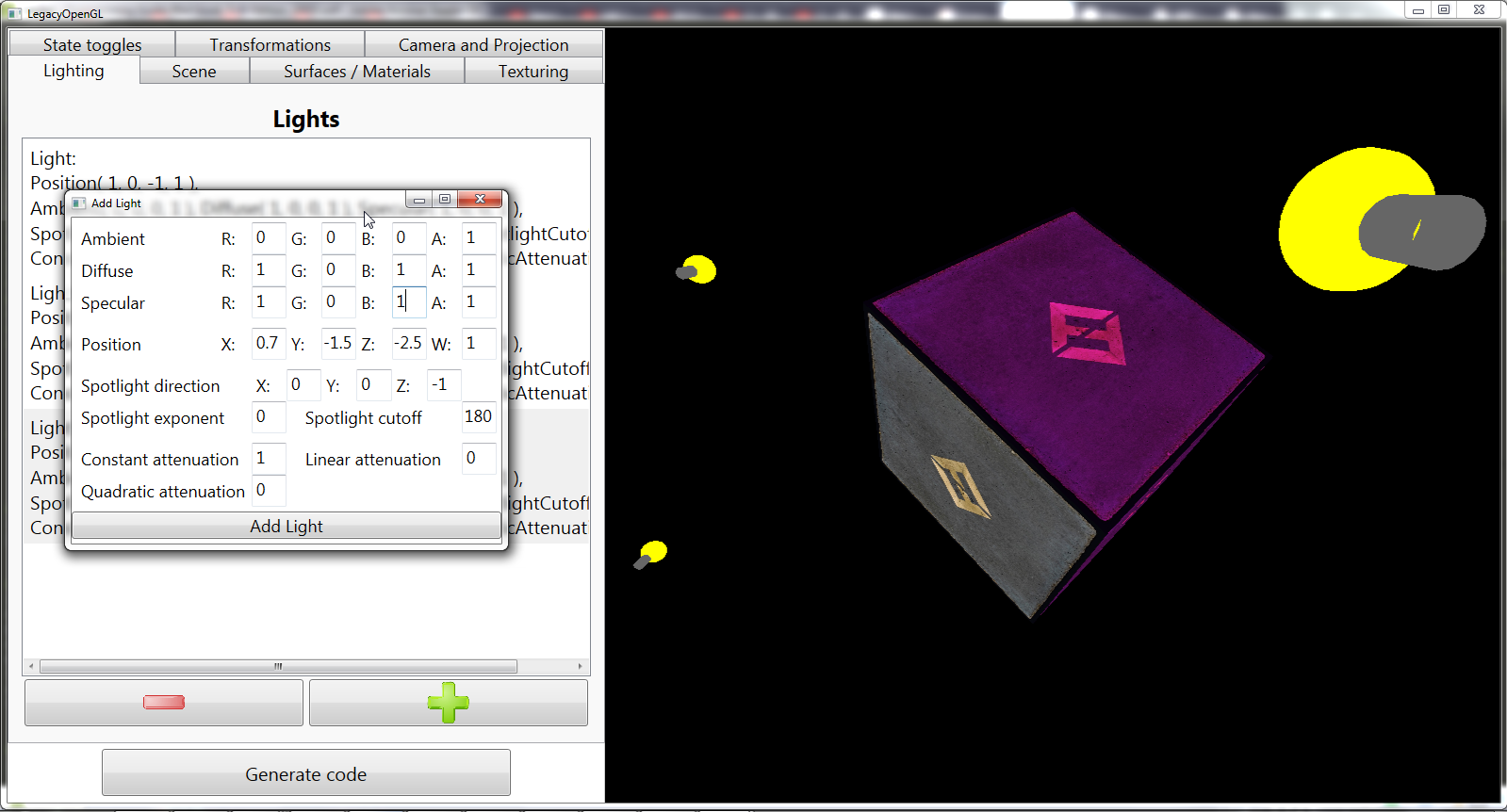
Rys. 4.5 Panel ustawiania transformacji obserwatora i transformacji projekcji (ortograficzna)

### Dodawanie i usuwanie źródeł światła



Rys. 4.6 Panel dodawania i usuwania świateł

Na ekranie znajdującym się na powyższym rysunku (Rys. 4.6) przedstawiony został panel dodawania i usuwania źródeł światła. Po wybraniu dowolnej pozycji z listy istnieje możliwość jej usunięcia wykorzystując przycisk zawierający znak minus. Aby dodać nowe źródło światła należy wybrać przycisk plus. Otwiera on poniższe okno.



Rys. 4.7 Okno dodawania nowego światła

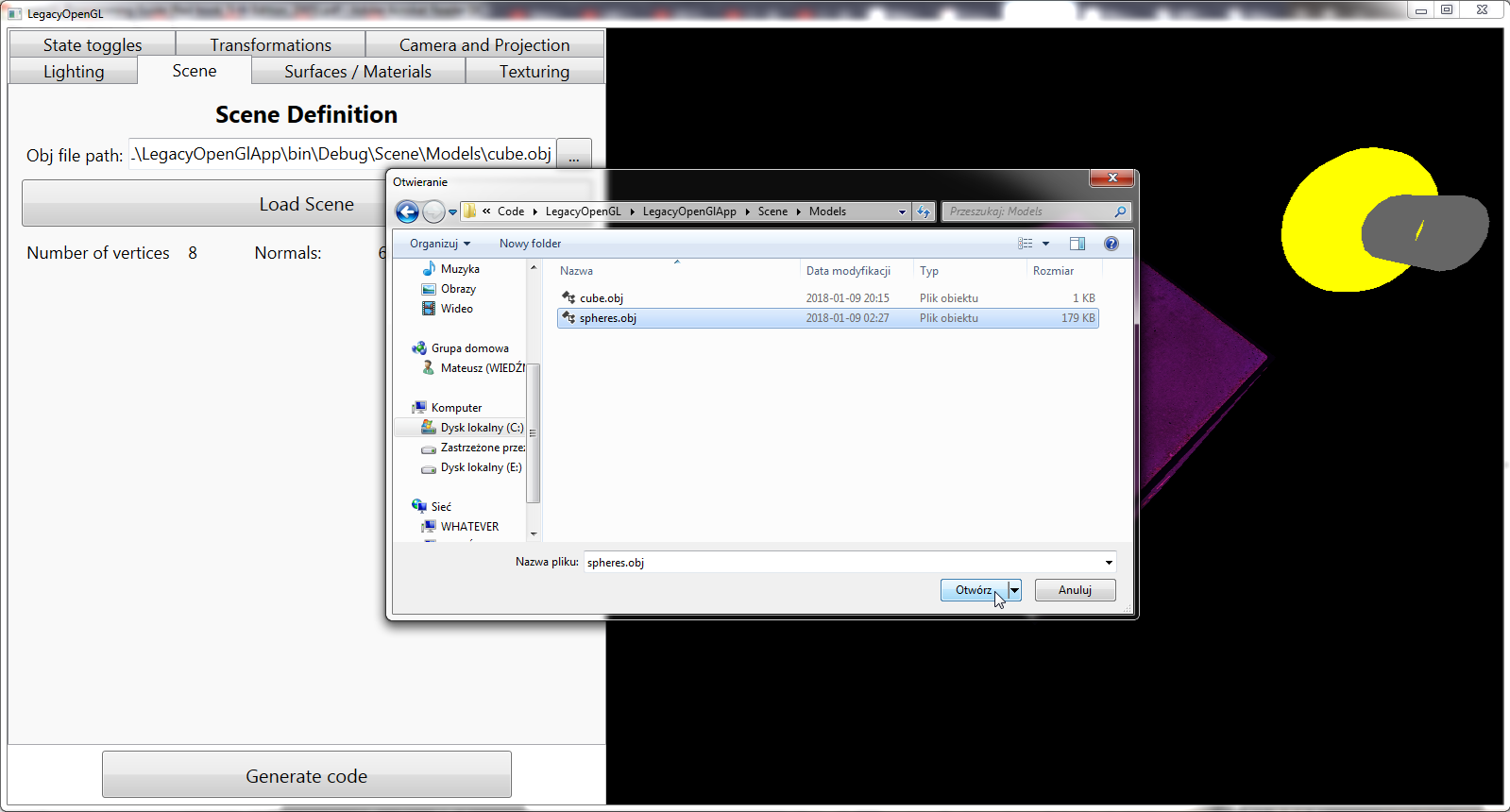
Okno dodawania źródła światła (Rys. 4.7) pozwala na definiowanie dodatkowych źródeł światła. Umożliwia ono zdefiniowanie wszystkich parametrów światła w OpenGL. Warto pamiętać, że parametry światła reflektorowego (spotlight) są aplikowane tylko w wypadku światła pozycyjnego (gdy wartość pozycji światła ‘W’ ma wartość różną od zera) .

### Wczytywanie sceny



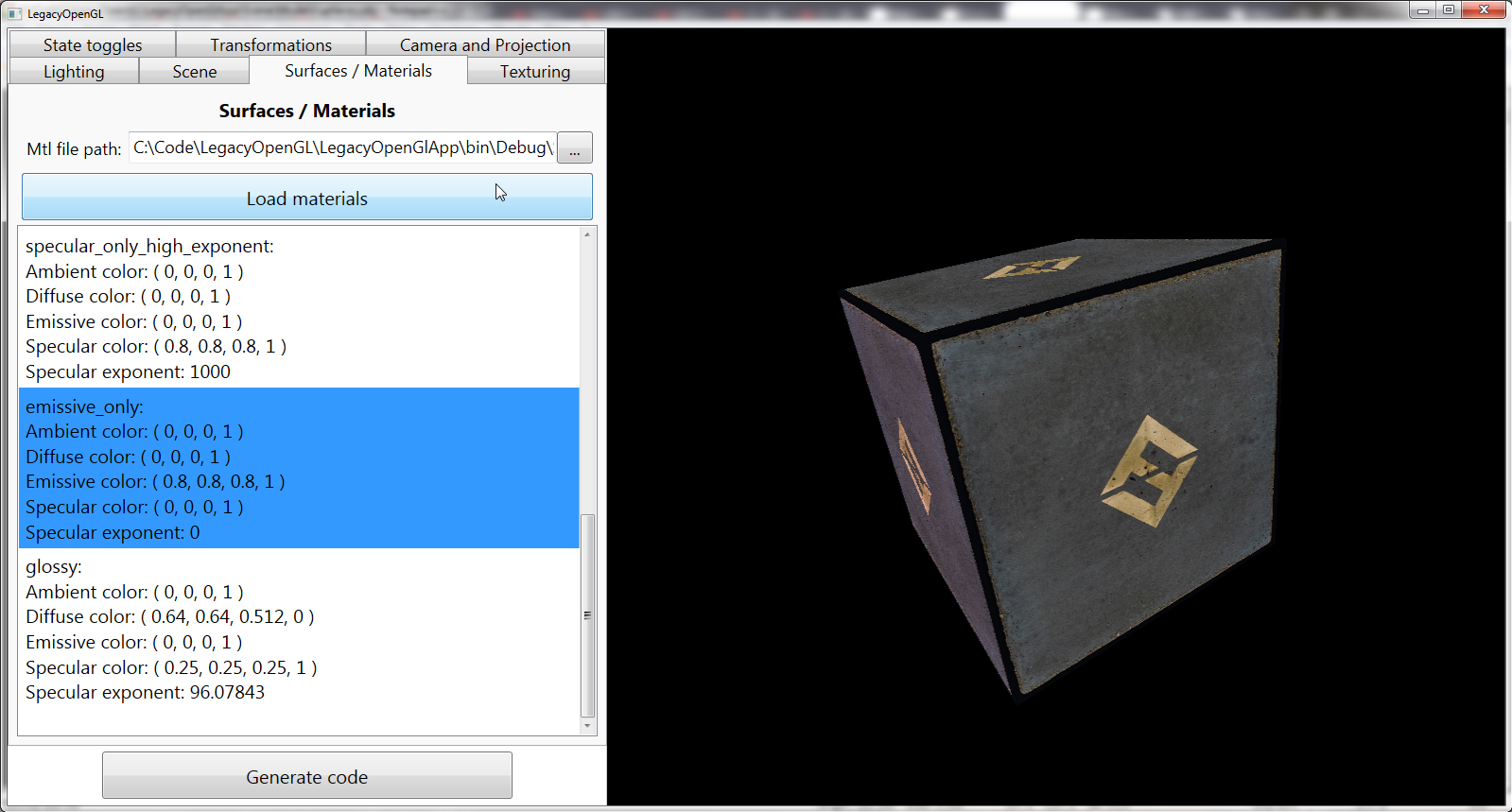
Rys. 4.8 Panel wczytywania sceny

Przedstawiony powyżej panel (Rys. 4.8) umożliwia wczytywanie definicji sceny z pliku OBJ. Pozwala on na ręczne wprowadzenie ścieżki do pliku lub wybranie jej za pomocą dedykowanego okna przedstawionego poniżej (Rys. 4.9). Po ustawieniu poprawnej ścieżki należy nacisnąć przycisk ‘Load scene’ w celu załadowania pliku. Dla wczytanego pliku wyświetlana jest liczba jego wierzchołków, wektorów normalnych i ścian.



Rys. 4.9 Okno wyboru pliku

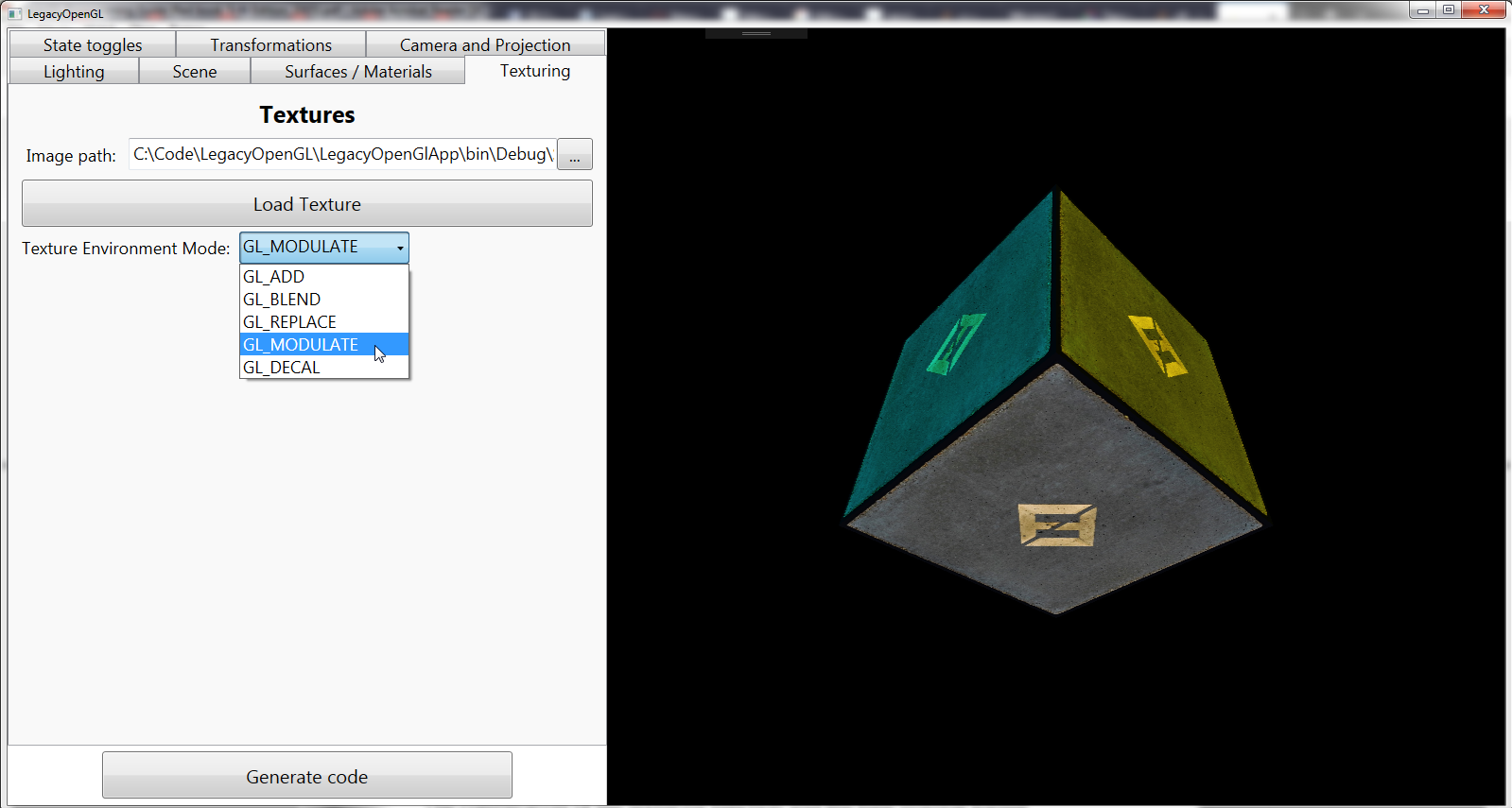
### Wczytywanie i ustawianie materiałów



Rys. 4.10 Panel wczytywania i ustawiania materiałów

Panel wczytywania i ustawiania materiałów (Rys. 4.10) pozwala na wczytanie definicji materiałów z pliku MTL. Podobnie jak panel wczytywania sceny, pozwala on na ręczne wprowadzenie ścieżki do pliku lub wybranie jej za pomocą dedykowanego okna. Po ustawieniu poprawnej ścieżki należy nacisnąć przycisk ‘Load materials’ w celu załadowania pliku. Wczytane materiały wyświetlane są w formie listy. Wybranie elementu z listy powoduje zaaplikowanie odpowiadającego mu materiału do wszystkich obiektów w scenie.

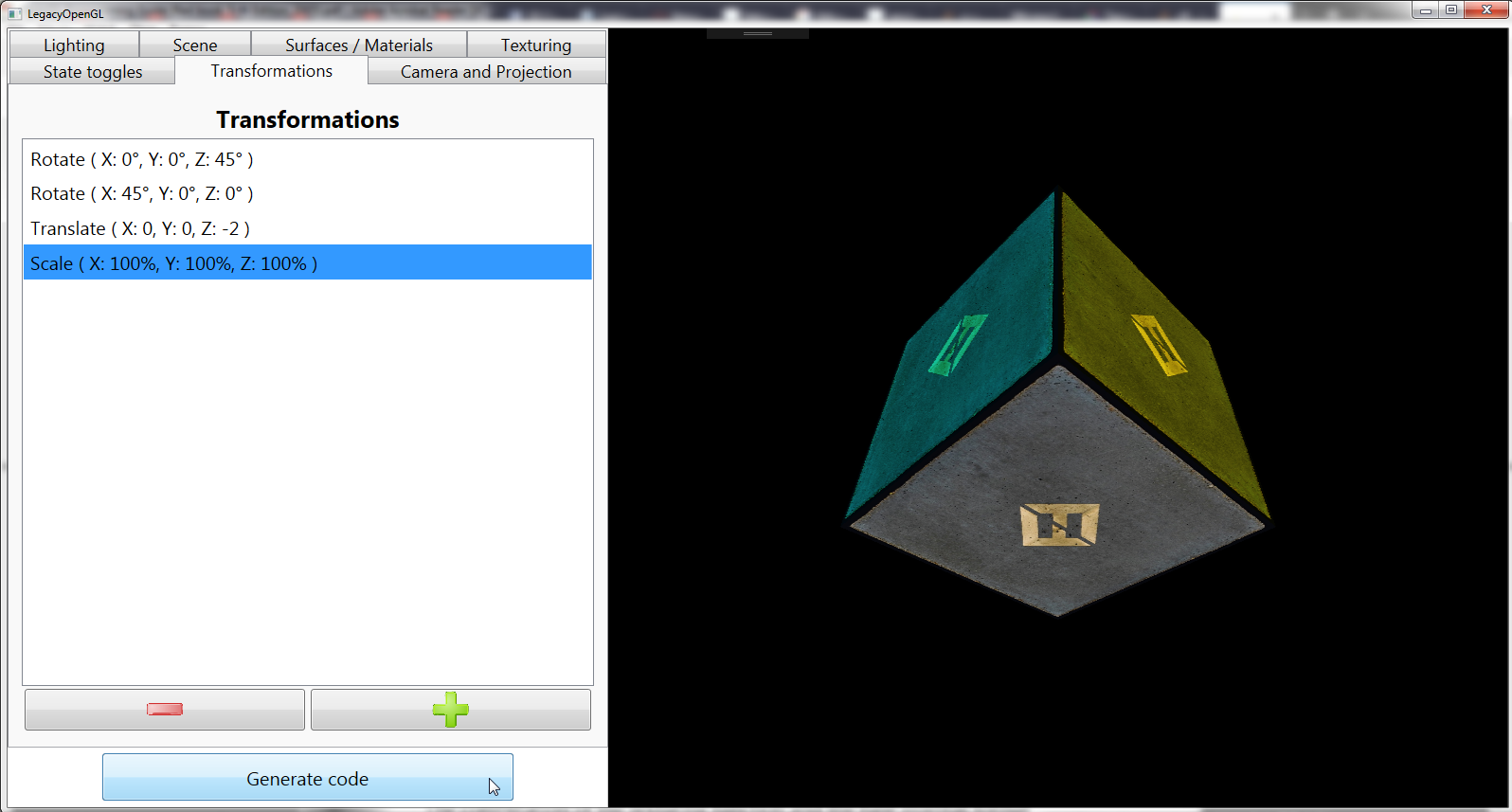
### Wczytywanie i konfiguracja tekstur



Rys. 4.11 Panel wczytywania i ustawiania tekstur

Panel wczytywania i ustawiania tekstur pozwala na wczytanie tekstury z pliku graficznego. Podobnie jak panele wczytywania sceny i wczytywania materiałów, pozwala on na ręczne wprowadzenie ścieżki do pliku lub wybranie jej za pomocą dedykowanego okna. Po ustawieniu poprawnej ścieżki należy nacisnąć przycisk ‘Load Texture w celu załadowania pliku. Po załadowaniu tekstury istnieje możliwość wybrania funkcji używanej do nałożenia tekstury na ściany.

### Generowanie kodu



Rys. 4.12 Generowanie kodu

Aby wygenerować kod odpowiadający wyświetlanej scenie należy kliknąć przycisk znajdujący się w lewym dolnym obszarze okna, pod opisanymi w poprzednich punktach panelami   
(Rys. 4.12). Jego naciśnięcie powoduje wygenerowanie kodu, zapisanie go do pliku (generatedCode.cpp w głównym folderze programu) oraz jego otwarcie z wykorzystaniem domyślnego edytora dla tego typu plików.

Kod wygenerowany przez program dla prostego przykładu ma postać:

#include <GL/glut.h>

int main(int argc, char\*\* argv)

{

    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

    glutInitWindowSize(800, 600);

    glutInitWindowPosition(100, 100);

    glutCreateWindow(argv[0]);

    init();

    glutDisplayFunc(display);

    glutMainLoop();

    return 0;

}

void init(void)

{

    glViewport(0, 0, 800, 600);

    glMatrixMode(GL\_PROJECTION);

    glLoadIdentity();

    gluPerspective(60.00, 1.16, 0.10, 100.00);

    glClearColor(0.0, 0.0, 0.0, 0.0)

    glShadeModel(GL\_SMOOTH);

    glEnable(GL\_DEPTH\_TEST);

}

void draw(void)

{

    glMatrixMode(GL\_MODELVIEW);

    // Clear the color and depth buffers.

    glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

    // Reset the modelview matrix.

    glLoadIdentity();

    glLookAt(

        3.00, 0.00, 3.00,

        0.00, 0.00, -1.00,

        0.00, 1.00, 0.00);

    glBegin(GL\_QUADS);

    glVertex3f(1.00, 1.00, 0.00);

    glVertex3f(-1.00, 1.00, 0.00);

    glVertex3f(-1.00, -1.00, 0.00);

    glVertex3f(1.00, -1.00, 0.00);

    glEnd();

    glFlush();

}

## Przykłady

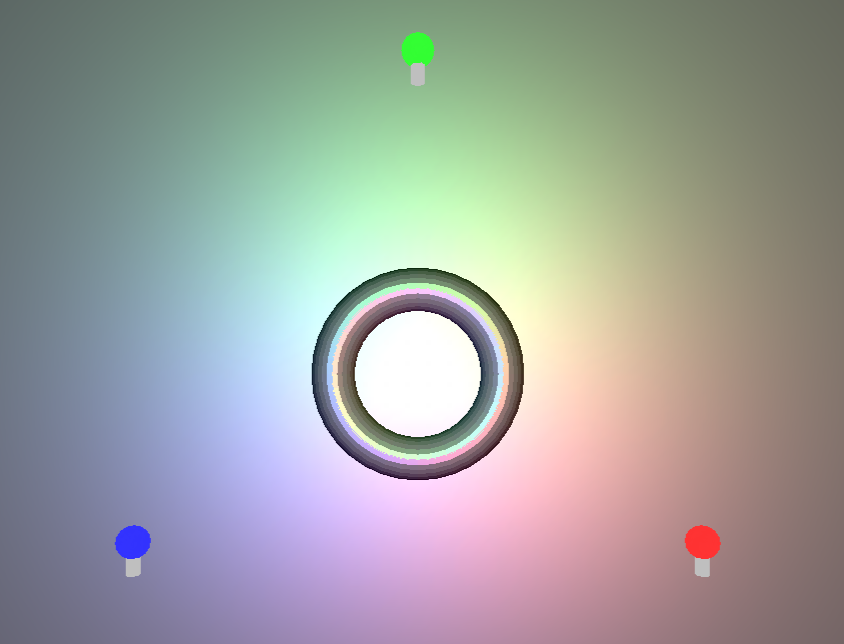
W ramach realizacji pracy przygotowanych zostało kilka przykładowych zestawów danych:

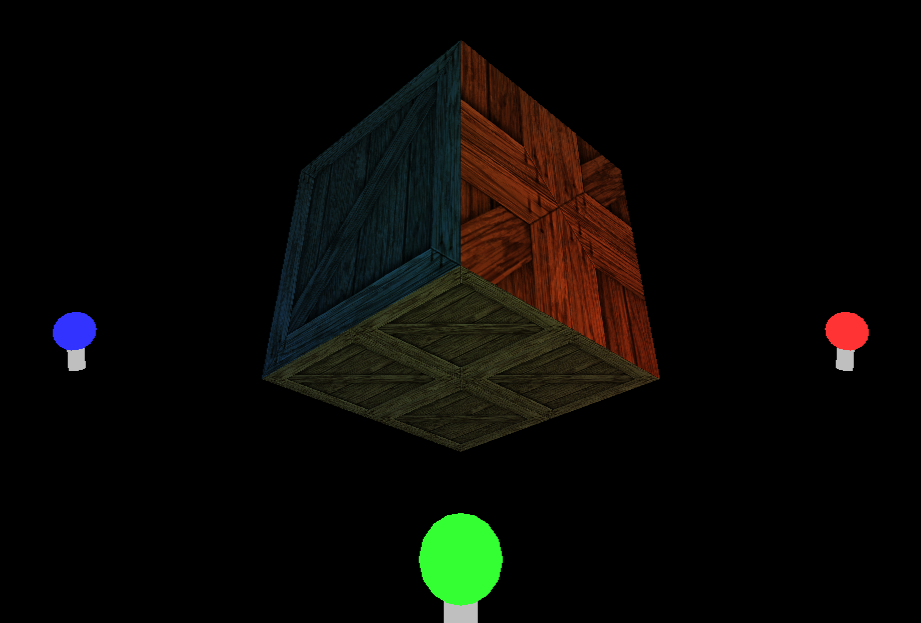
Wygenerowano następujące modele 3D:

* Płaszczyzna
* Prosty sześcian
* Pięć połączonych ze sobą sfer
* Torus na płaszczyźnie
* Pokemon (na podstawie modelu udostępnionego przez Mike’a Fong’a na licencji   
  CC-Attribution-NonCommercial [1] <https://sketchfab.com/models/64815cda802746b8b1be2e2246db4b35#>)

Z wykorzystaniem powyższych modeli zdefiniowano następujące konfiguracje startowe :

* config.json – przykład zawierający prosty teksturowany sześcian, obrócony   
  i oświetlony
* config\_specular.json – przykład zawierający model złożony z pięciu połączonych ze sobą sfer, obrócony i przesunięty w celu czytelnego zaprezentowania lustrzanego komponentu światła (odblasku)
* config\_spotlight\_exponents.json – przykład ukazujący wpływ współczynnika skupienia na efekt generowany przez światło reflektorowe; przedstawia 6 świateł reflektorowych o różnych współczynnikach skupienia padających na płaszczyznę
* config\_spotlights.json – przykład prezentujący oświetlenie torusa znajdującego się na płaszczyźnie z wykorzystaniem świateł reflektorowych o różnych pozycjach i kolorach
* config\_positional\_lights.json – przykład prezentujący oświetlenie torusa znajdującego się na płaszczyźnie z wykorzystaniem świateł pozycyjnych o różnych pozycjach   
  i kolorach; został on zaprezentowany na Rys. 4.13
* config\_directional\_lights.json – przykład prezentujący oświetlenie torusa znajdującego się na płaszczyźnie z wykorzystaniem świateł kierunkowych o różnych kierunkach   
  i kolorach
* config\_teture\_multi.json – przykład prezentujący prosty teksturowany sześcian   
  o różnych mapowaniach tekstur dla każdej ze ścian, obrócony i oświetlony; został on zaprezentowany na Rys. 4.14
* config\_pokemon.json – złożony model pokemona, ze zdefiniowanym mapowaniem tekstur, oświetlony z pomocą dwóch świateł; został on zaprezentowany na Rys. 4.15

  
Rys. 4.13 Scena wygenerowana na podstawie pliku konfiguracyjnego config\_positional\_lights.json



Rys. 4.14 Scena wygenerowana na podstawie pliku konfiguracyjnego config\_texture\_multi.json



Rys. 4.15 Scena wygenerowana na podstawie pliku konfiguracyjnego config\_pokemon.json

# Podsumowanie i wnioski

## Wnioski

Samodzielna praca nad wytworzeniem programu pozwoliła na zrozumienie jak istotną rolę odgrywa prawidłowe zarządzanie projektem i organizacja pracy. Pokazała ona, że realizacja projektu informatycznego obejmuje znacznie więcej niż samo programowanie. Poprzez przejście pełnego cyklu wytwarzania oprogramowania, wyeksponowane zostało jak wiele kroków należy podjąć zanim możliwa będzie praca nad kodem programu. Pozwala to dostrzec i docenić pracę jaką wykonują na co dzień analitycy biznesowi, architekci oprogramowania oraz menadżerowie projektów, która umożliwia programistom ich pracę.

W początkowej fazie realizacji projektu, przez zastosowaniem zwinnych technik zarządzania miał miejsce bardzo powolny postęp prac. Dopiero wprowadzenie organizacji projektu   
w postaci spriorytetyzowanej listy zadań oraz zbliżający się ostateczny termin oddawania pracy wzmogły mobilizację do działania.

## Spis artefaktów

W procesie realizacji pracy inżynierskiej wygenerowane zostały następujące artefakty:

* Kod programu
* Pliki wykonywalne programu
* Przykładowe modele i konfiguracje

Oraz zawarte w niniejszej pracy:

* Wprowadzenie do OpenGL
* Opis architektury projektu
* Opis procesu implementacji projektu
* Instrukcja instalacji i obsługi programu
* Przykłady możliwości programu

## Dalszy rozwój

Program implementuje wszystkie początkowo ustalone funkcjonalności. Jednakże, podczas jego implementacji i dokumentacji dostrzeżonych zostało kilka obszarów które posiadają potencjał do dalszego rozwoju.

### Obsługa błędów

Jako, że z programu korzystać będą studenci, mogący nie posiadać jeszcze wiedzy z zakresu OpenGL, należy spodziewać się, że mogą oni próbować wprowadzić niepoprawne wartości dla eksponowanych parametrów OpenGL. Aby uniknąć niespodziewanego zachowania programu w takiej sytuacji warto zaimplementować w nim mechanizm obsługi błędów.

### Interfejsy

Aby zapewnić łatwo testowalny kod zgodny z zasadami SOLID należy wydzielić interfejsy dla serwisów, a następnie zastąpić wstrzykiwanie konkretnych klas poprzez wstrzykiwanie interfejsów. Pozwoli to uniezależnić kod od konkretnych implementacji, pozostawiając jedynie zależność od definiowanych przez interfejs funkcjonalności.

### Testy

W celu ułatwienia utrzymania kodu oraz zapewnienia bezpieczeństwa późniejszego dodawania nowych funkcjonalności i wprowadzania zmian, należy zwiększyć pokrycie kodu testami jednostkowymi i integracyjnymi. Pozwoli to wyłapać wszelkie zmiany które mogą naruszyć prawidłowe działanie programu.

### Wykorzystanie biblioteki Prism

By w pełni wykorzystać potencjał oferowany przez WPF oraz mechanizm wstrzykiwania zależności i poprawić jakość i czytelność kodu odpowiedzialnego za interfejs użytkownika warto skorzystać z biblioteki Prism. Wspiera ona re-używalność kodu, poprawne rozdzielenie odpowiedzialności oraz łatwe zarządzanie zależnościami i interakcjami pomiędzy komponentami [9].

### Rozbudowa interfejsu użytkownika

Program zapewnia prosty interfejs użytkownika pozwalający na specyfikowanie parametrów dla sceny OpenGL. Istnieje duży potencjał do rozbudowy jego możliwości w wielu aspektach,   
między innymi o:

* wsparcie pozycjonowania obserwatora z wykorzystaniem klawiatury i myszy
* wsparcie nawigowania po interfejsie graficznym z wykorzystaniem klawiatury   
  i skrótów klawiaturowych
* walidacja typu i zakresu wprowadzanych przez użytkownika wartości
* dodanie tekstów opisujących znaczenie poszczególnych parametrów
* uczynienie interfejsu bardziej intuicyjnym i responsywnym

### Wsparcie dla Programowalnego Potoku Wizualizacji

Aby zaprezentować jak należy pisać nowoczesne aplikacje korzystając z API OpenGL rozważa się dodanie wsparcia dla programowalnego potoku wizualizacji występującego w nowszych wersjach OpenGL.

# Bibliografia

[1] Attribution-NonCommercial 4.0 International Licence [online], Creative Commons, dostęp: 12.01.2018, dostępny w Internecie: < <https://creativecommons.org/licenses/by-nc/4.0/legalcode> >

[2] Blender Reference Manual [online], Blender Foundation, dostęp: 12.01.2018, dostępny w Internecie: < <https://docs.blender.org/manual/en/dev/index.html> >

[3] CHAOS Report 2015, Standish Group, 2015

[4] Developer's Guide to Microsoft Prism Library 5.0 for WPF [online], Microsoft Corporation, dostęp: 12.01.2018, dostępny w Internecie:   
< <https://msdn.microsoft.com/en-us/library/gg406140.aspx> >

[5] GLUT and OpenGL Utility Libraries [online], Khronos Group, dostęp: 12.01.2018, dostępny w Internecie: < <https://www.opengl.org/resources/libraries/> >

[6] History of OpenGL [online], Khronos Group, dostęp: 12.01.2018, dostępny   
w Internecie: < <https://www.khronos.org/opengl/wiki/History_of_OpenGL> >

[7] Legacy OpenGL [online], Khronos Group, dostęp: 12.01.2018, dostępny w Internecie: < <https://www.khronos.org/opengl/wiki/Legacy_OpenGL> >

[8] OpenGL Programming Guide; the official guide to learning OpenGL, wyd. 2, Silicon Graphics, Inc., ISBN 0-321-33573-2

[9] Prism Documentation [online], dostęp: 12.01.2018, dostępny w Internecie:   
< <http://prismlibrary.readthedocs.io/en/latest/WPF/01-Introduction/> >

[10] Ramey D., File Formats: Version 4.2, Alias Wavefront, 1995

[11] Segal M., Akeley K., The OpenGL® Graphics System: A Specification, wyd. 2, Silicon Graphics, Inc., 22-10-2004

[12] Yargs S., Origin of “I hear and I forget. I see and I remember. I do and I understand.”, StackExchange, dostęp: 12.01.2018, dostępny w Internecie:   
< <https://english.stackexchange.com/questions/226886/origin-of-i-hear-and-i-forget-i-see-and-i-remember-i-do-and-i-understand> >