Spis treści

[0 Abstrakt 1](#_Toc503023146)

[1 Wstęp 2](#_Toc503023147)

[1.1 Charakterystyka problematyki 2](#_Toc503023148)

[1.2 Uzasadnienie wyboru tematu pracy 2](#_Toc503023149)

[1.3 Cel i zakres pracy 2](#_Toc503023150)

[2 Przegląd stosowanych technologii 3](#_Toc503023151)

[2.1 OpenGL 3](#_Toc503023152)

[2.1.1 Czym jest OpenGL 3](#_Toc503023153)

[2.1.2 Wersje OpenGL 3](#_Toc503023154)

[2.1.3 Dodatkowe biblioteki 4](#_Toc503023155)

[2.1.4 Kolejność renderowania w OpenGL 4](#_Toc503023156)

[2.1.5 OpenGL jako maszyna stanów 5](#_Toc503023157)

[2.1.6 Transformacje 5](#_Toc503023158)

[2.1.7 Macierze transformacji 6](#_Toc503023159)

[2.1.8 Transformacja modelująca i transformacja obserwatora 7](#_Toc503023160)

[2.1.9 Transformacja projekcji - rzutowanie 8](#_Toc503023161)

[2.1.10 Oświetlenie 8](#_Toc503023162)

[2.1.11 Właściwości powierzchniowe. Materiały. 8](#_Toc503023163)

[2.1.12 Teksturowanie 8](#_Toc503023164)

[2.2 Wspierane formaty 8](#_Toc503023165)

[2.2.1 Omówienie wykorzystanych formatów plików 8](#_Toc503023166)

[2.2.2 Uzasadnienie wyboru 8](#_Toc503023167)

[2.2.3 Definicja geometrii sceny – format OBJ 8](#_Toc503023168)

[2.2.4 Definicja materiałów – format MTL 8](#_Toc503023169)

[2.2.5 Tekstury – popularne formaty rastrowe 8](#_Toc503023170)

[3 Architektura i implementacja rozwiązania 8](#_Toc503023171)

[3.1 Analiza wymagań 8](#_Toc503023172)

[3.2 Architektura rozwiązania 8](#_Toc503023173)

[3.3 Opis komponentów, uml 8](#_Toc503023174)

[3.4 Wykorzystanie technologie 8](#_Toc503023175)

[3.5 Problemy napotkane podczas implementacji 8](#_Toc503023176)

[4 Instrukcja użytkownika 9](#_Toc503023177)

[4.1 Instalacja 9](#_Toc503023178)

[4.2 Konfiguracja 9](#_Toc503023179)

[4.3 Instrukcja użytkowania 9](#_Toc503023180)

[5 Podsumowanie i wnioski 9](#_Toc503023181)

[5.1 Czego się nauczyłem 9](#_Toc503023182)

[5.2 Dalszy rozwój 9](#_Toc503023183)

[5.2.1 Wykorzystanie biblioteki Prism 9](#_Toc503023184)

[5.2.2 Rozbudowa interfejsu użytkownika 9](#_Toc503023185)

[5.2.3 Pokrycie kodu testami 9](#_Toc503023186)

[6 Bibliografia 9](#_Toc503023187)

# 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 zacząć przygodę z grafiką komputerową. Ma on prezentować podstawowe funkcjonalności bibliotek OpenGL. 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 bibliotek 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 Science. Its scope consists of design and implementation of an application allowing to easily start an adventure with computer graphics. 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 generates 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, często przypisywane Konfucjuszowi, głosi:

„Usłyszałem i zapomniałem. Zobaczyłem i zapamiętałem. Zrobiłem i zrozumiałem.”

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

* 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 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 bibliotek OpenGL takimi jak:

* modyfikacja oferowanych przez OpenGL zmiennych stanu
* transformacja projekcji
* transformacje modelująca i obserwatora
* teksturowanie
* cieniowanie / własności powierzchniowe

Dokumentacja powinna zawierać:

* instrukcję instalacji,
* instrukcję użytkowania,
* krótki opis wspieranych przez program funkcjonalności OpenGL.

# 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 renderowania grafiki komputerowej. Składa się on z około 250 funkcji pozwalających na tworzenie grafiki 2D oraz 3D. 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 wydawcy [B]. 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, co 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 [A] w porównaniu do nowszych wersji API. Taka wersja świetnie się jednak nadaje by przedstawić podstawowy potok renderowania (Rendering Pipeline) bez dodatkowego narzutu wprowadzanego przez shader’y. Program ten nie prezentuje 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 koncepty i założenia stosowane w nowszych wersjach OpenGL.

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

### Dodatkowe biblioteki

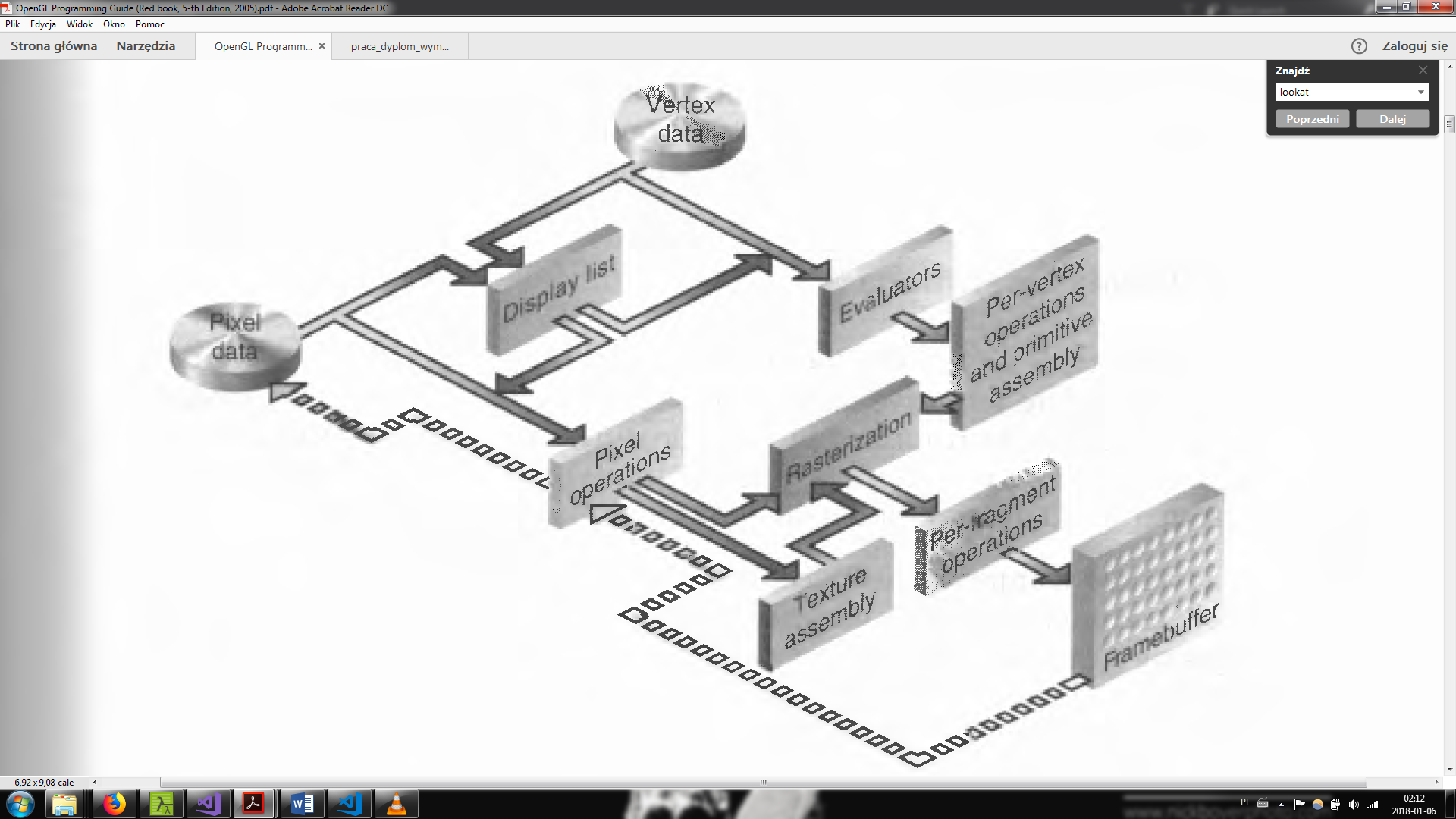
Ponieważ OpenGL zawiera jedynie zestaw podstawowych funkcji odpowiedzialnych za renderowanie grafiki komputerowej istnieje wiele bibliotek wspomagających pracę z OpenGL. Pierwszą biblioteką o której warto wspomnieć jest OpenGL Utility Library (GLU) [C]. Zawiera ona zestaw około 50 funkcji ułatwiających 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 [C]. 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 [D].

### Kolejność renderowania w OpenGL

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



Rys. 2.1 Kolejność renderowania w OpenGL [C]

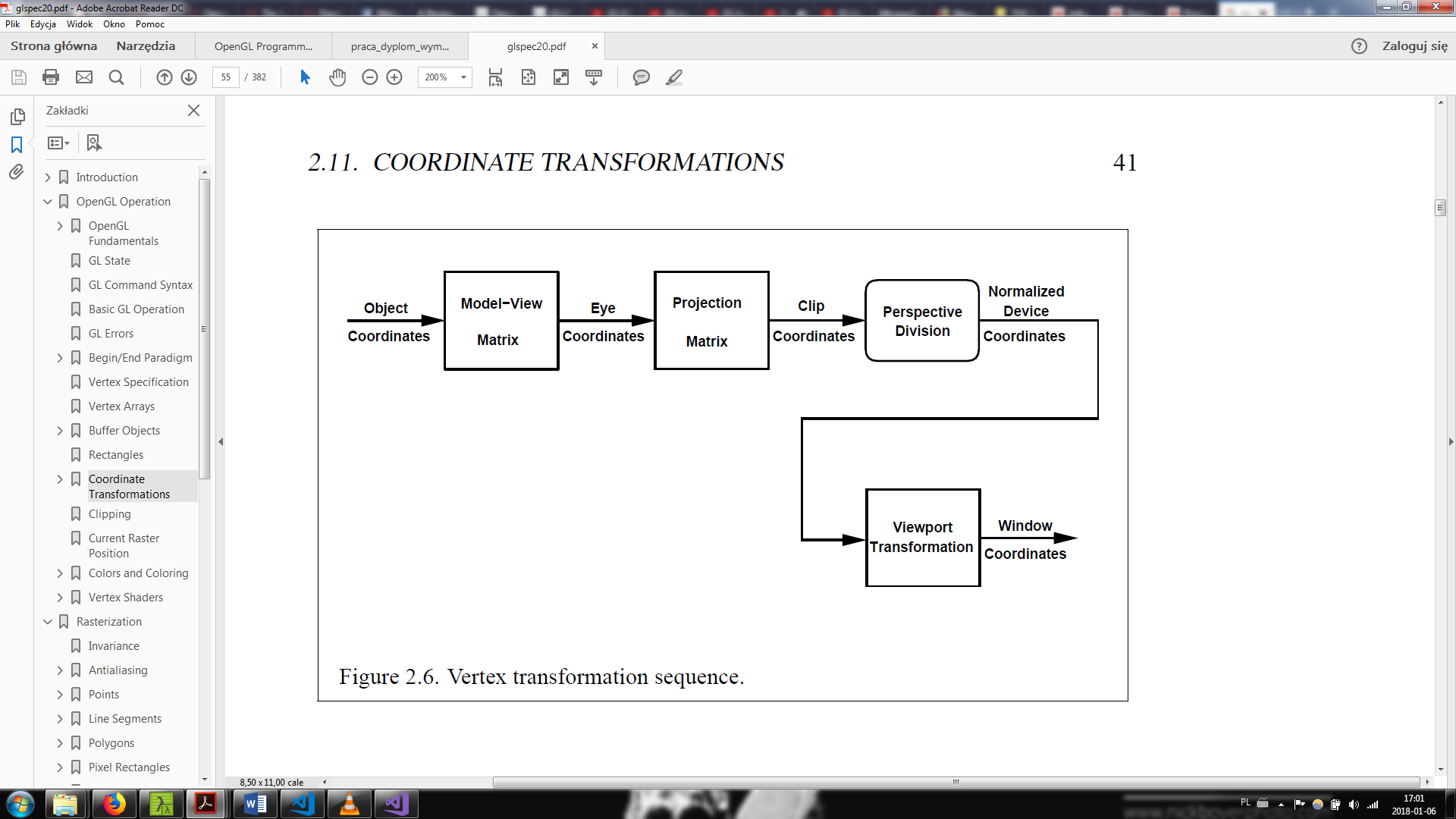
Analizując diagram możemy dostrzec dwa typy danych wejściowych, piksele i wierzchołki, każdy z własnym przepływem informacji. Pierwszym krokiem przetwarzania dla wierzchołków jest ich wczytanie i wyznaczenie ich pozycji w przestrzeni sceny. Na grupach wierzchołków rozpinane są proste figury geometryczne i ustalane są dla nich wektory normalne. Ustawione zostaje mapowanie tekstur oraz wykonywane są wyliczenia związane z oświetleniem. Jednocześnie wczytywane są tekstury z tablicy pikseli. W procesie rasteryzacji łączone są efekty obu operacji. Wyznaczane są tzw. fragmenty odpowiadające pikselom w końcowym obrazie. Dla każdego z fragmentów ustalany jest kolor i głębokość. 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 z wykorzystaniem komend **glEnable()** oraz **glDisable()**. Każda zmienna stanu posiada swoją domyślną wartość co pozwala uniknąć konieczności jej specyfikowania jeśli nie potrzebujemy z niej korzystać. 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 domyślnych wartościach znajduje się w dokumentacji API [E].

### Transformacje

Jednym z podstawowych konceptów, zrozumienie których jest kluczowe do pracy z OpenGL są transformacje. Pozwalają one na manipulację położenia wierzchołków w trój-wymiarowej przestrzeni sceny oraz pozycji i sposobu w jaki na nią patrzymy. W efekcie otrzymujemy zrzutowany na płaszczyznę obraz złożony z pikseli.



Rys. 2.2 Kolejność aplikowania transformacji [E]

Według kolejności przedstawionej na powyższym schemacie, 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 przekształcenie ich do wydzielonej 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.

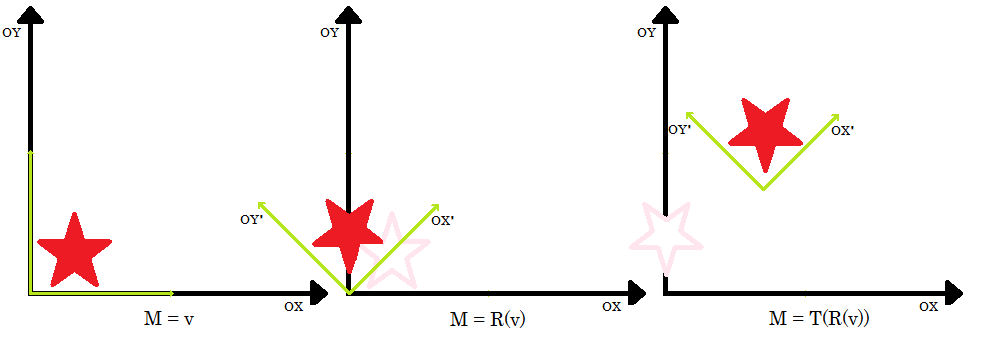
### Macierze transformacji

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.

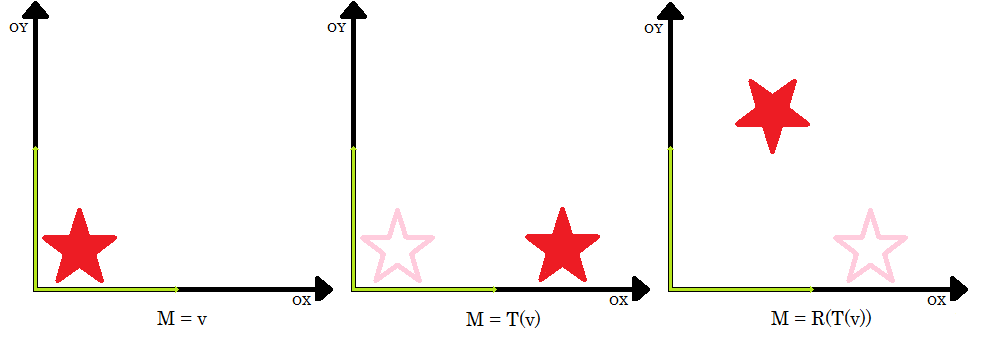
Pierwszym krokiem jest wczytywanie do bufora **B** macierzy tożsamości **I**. Każda kolejna zdefiniowana transformacja polega na przemnożeniu aktualnej macierzy **B** przez macierz transformacji **T**, co daje w efekcie nową macierz **BT**. Po zaaplikowaniu wszystkich transformacji, tj. wymnożeniu ich 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. Dla lepszego zobrazowania konceptu poniżej zamieszczony został przykład.

Przyjmując oznaczenia:

T – operacja translacji (przesunięcia),  
R – operacja rotacji,   
v – oryginalna pozycja wierzchołka



Rys. 2.3 Lokalny system odniesienia



Rys. 2.3 Globalny system odniesienia

Dążyć do osiągnięcia stanu przedstawionego na ostatniej pozycji na powyższych rysunkach można w dwójnasób:

Korzystając z lokalnego układu odniesienia należy w pierwszej kolejności obrócić obiekt, co spowoduje jednocześnie obrót układu odniesienia, a następnie przesunąć go wzdłuż nowej osi OX**´**. Kolejno aplikowane są transformacje **R**(**v**), **T**(**R**(**v**)), **T**(**R**(**v**)). Właściwą kolejnością definiowania macierzy dla tego scenariusza jest więc: **I 🡪 IT 🡪 ITR 🡪 ITRv**.

By skorzystać z globalnego układu odniesienia należy odwrócić kolejność transformacji. Pierwszą operacją jaką wykonamy będzie translacja względem osi OX, a następnie rotacja względem środka układu współrzędnych. Porządek aplikowania transformacji w tym wypadku wygląda następująco: **T**(**v**), **R**(**T**(**v**)), **R**(**T**(**v**)). Należy więc zdefiniować transformacje   
w następującej kolejności: **I 🡪 IR 🡪 IRT 🡪 IRTv**.

### Transformacja modelująca i transformacja obserwatora

Operacje transformacji modelującej i obserwatora wykonywane są na wspólnej macierzy. Transformacje modelujące służą 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 specyfikowania transformacji należy z pomocą komendy **glMatrixMode()** wybrać odpowiednią macierz (GL\_MODELVIEW). Następnie należy zresetować jej wartość poprzez załadowanie do niej macierzy tożsamości. Od tego momentu możemy nakładać na nią kolejne transformacje. Należy pamiętać, że aby uzyskać transformacje w globalnym układzie odniesienia trzeba odwrócić kolejność stosowania transformacji.

Transformacja obserwatora pozwala na ustawienie pozycji i orientacji z jakiej obserwujemy scenę. Jej ustawienie dobywa się na tej samej macierzy co transformacji modelującej za pomocą polecenia **glLookAt()**. Jest ona analogią pozycjonowania aparatu przed wykonaniem zdjęcia. Warto zaznaczyć, że efekt transformacji modelującej można uzyskać również za pomocą transformacji modelującej poprzez odpowiednie przesunięcie / obrót modelowanej sceny.

### Transformacja projekcji – rzutowanie

Transformacja projekcji odpowiada za ustalenie w jaki sposób obiekty sceny będą zachowywać się w zależności od głębokości ich położenia w scenie. Do jej ustawienia używamy polecenia **glMAtrixMode()**, tym razem jako parametr podając GL\_PROJECTION. Ponownie, pierwszą operacją jaką należy wykonać jest wczytanie macierzy tożsamości. Następnie należy wybrać pożądany sposób projekcji. OpenGL oferuje dwa tryby projekcji – perspektywiczną i ortograficzną.

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. Ustawiana jest z pomocą polecenia **glFrustum()** lub **glPerspective()**.

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. By ustawić projekcję ortograficzną należy skorzystać z polecenia **glOrtho()**.

### Oświetlenie

### Właściwości powierzchniowe. Materiały.

### Teksturowanie

## Wspierane formaty

### Omówienie wykorzystanych formatów plików

### Uzasadnienie wyboru

### Definicja geometrii sceny – format OBJ

### Definicja materiałów – format MTL

### Tekstury – popularne formaty rastrowe

# Architektura i implementacja rozwiązania

## Analiza wymagań

## Architektura rozwiązania

## Opis komponentów, uml

## Wykorzystanie technologie

## Problemy napotkane podczas implementacji

# Instrukcja użytkownika

## Instalacja

## Konfiguracja

## Instrukcja użytkowania

+ screeny

# Podsumowanie i wnioski

## Czego się nauczyłem

## Dalszy rozwój

### Wykorzystanie biblioteki Prism

### Rozbudowa interfejsu użytkownika

### Pokrycie kodu testami

# Bibliografia

[A] <https://www.khronos.org/opengl/wiki/Legacy_OpenGL>

[B] <https://www.khronos.org/opengl/wiki/History_of_OpenGL>

[C] OpenGL Programming Guide

[D] <https://www.opengl.org/resources/libraries/>

[E] The OpenGL® Graphics System: A Specification

1. Shirley P. et al. - Fundamentals of Computer Graphics, Second ed. AK Peters, Ltd.; 2 edition, 2002
2. Fernando R., Kilgard M.J. - Język Cg, Programowanie grafiki w czasie rzeczywistym, Helion, 2003
3. Wright B., Liptchak B. - OpenGL. Księga eksperta, Helion 2004
4. Ebert D.S., Musgrave F.K., Peachey D., Perlin K. - Texturing and Modeling, A Procedural Approach, 3rd edition, Morgan Kaufmann; 2002
5. Akenine-Moller T., Heines E. – Real Time Rendering – Third edition, A.K. Peters Ltd. 2008
6. Foley J.D., van Dam A. Feiner S., Hughes F.J. - Computer Graphics, Principles and Practice, Addison-Wesley, 1992 (ISBN 0-201-12110-7)
7. Zabrodzki J. [red] - Grafikia komputerowa, metody i narzędzia, WNT, W-wa, 1994

~15 pozycji