WOJSKOWA AKADEMIA TECHNICZNA

**im. Jarosława Dąbrowskiego**

WYDZIAŁ CYBERNETYKI



**PRACA DYPLOMOWA**

Temat pracy: **Modelowanie oświetlenia z wykorzystaniem światła pozycyjnego w OpenGL**

**INFORMATYKA**

…………………………………………………………………………

(kierunek studiów)

**SIECI KOMPUTEROWE**

…………………………………………………………………………

(specjalność)

|  |  |
| --- | --- |
| Dyplomant: | Promotor pracy: |
| **Mateusz KOŁACZYK** | **dr inż. Marek SALAMON** |

**Warszawa 2021**

**OŚWIADCZENIE**

*„Wyrażam zgodę na udostępnianie mojej pracy w czytelni  
Archiwum WAT”.*

Dnia ........................ .................................

(podpis)

*Pracę przyjąłem*

*promotor pracy*

*dr inż. Marek SALAMON*

**Zadania**

Właściwości i sposób definiowania światła pozycyjnego w OpenGL.

Specyfikacja funkcjonalna aplikacji.

Opracowanie interfejsu graficznego aplikacji.

Implementacja aplikacji w wybranym środowisku programowym.

Wnioski z prac projektowych i implementacyjnych.

[WSTĘP 6](#_Toc73289339)

[ROZDZIAŁ I WŁAŚCIWOŚCI I SPOSÓB DEFINIOWANIA ŚWIATŁA POZYCYJNEGO W OPENGL 14](#_Toc73289340)

[1.1. Położenie źródła światła 14](#_Toc73289341)

[1.2. Kierunek świecenia światła 15](#_Toc73289342)

[1.3. Współczynnik światła otoczenia (Ambient) 16](#_Toc73289343)

[1.4. Współczynnik światła odbijanego w sposób rozproszony (Diffuse) 18](#_Toc73289344)

[1.5. Współczynnik światła odbijanego od obiektów kierunkowo (Specular) 22](#_Toc73289345)

[1.6. Współczynnik tłumienia atmosferycznego 27](#_Toc73289346)

[1.7. Materiały 28](#_Toc73289347)

[ROZDZIAŁ II SPECYFIKACJA FUNKCJONALNA APLIKACJI 30](#_Toc73289348)

[2.1. Scena 30](#_Toc73289349)

[2.2. Źródło światła 30](#_Toc73289350)

[2.3. Kamera 31](#_Toc73289351)

[2.4. Transformacje 31](#_Toc73289352)

[ROZDZIAŁ III OPRACOWANIE INTERFEJSU GRAFICZNEGO APLIKACJI 32](#_Toc73289353)

[3.1. Jak korzystać z programu? 32](#_Toc73289354)

[3.2. Realizacja graficznego interfejsu użytkownika 36](#_Toc73289355)

[ROZDZIAŁ IV IMPLEMENTACJA APLIKACJI W WYBRANYM ŚRODOWISKU PROGRAMOWYM 55](#_Toc73289356)

[4.1. Scena 56](#_Toc73289357)

[4.2. Źródło światła 62](#_Toc73289358)

[4.3. Kamera 69](#_Toc73289359)

[4.4. Transformacje 71](#_Toc73289360)

[ZAKOŃCZENIE 73](#_Toc73289361)

[Bibliografia 74](#_Toc73289362)

# WSTĘP

**Przedstawienie technologii OpenGL**

OpenGL jest to interfejs programistyczny aplikacji (API), który wiedzie prym wśród technologii przetwarzania grafiki komputerowej. Niezliczona ilość programów, od tych naukowych po zaawansowane gry komputerowe, korzysta z bibliotek opracowanych na podstawie tej specyfikacji. Przykładem takich są między innymi:

* Autodesk AutoCAD
* Scilab
* Google Earth
* SAP2000
* Adobe After Effects
* Adobe Photoshop
* I wiele innych…

Nie da się zaprzeczyć, że jest to kierunek informatyki, w którym warto się rozwijać. Projekty opracowywane są wciąż do dnia dzisiejszego w OpenGL i nie widać, aby ta specyfikacja miała odejść w niepamięć.

**Dlaczego OpenGL nazywamy specyfikacją?**

Jest tak dlatego, że sam OpenGL, jako specyfikacja, to zbiór wymagań postawionych producentom sterowników kart graficznych w celu utrzymania jednolitego standardu na rynku[[1]](#footnote-1).

Można to porównać do jazdy samochodem. Gdyby każdy producent samochodów osobowych wymyślił swój kompletnie inny od konkurencji sposób budowy i prowadzenia pojazdu, każdy potencjalny kierowca musiałby oddzielnie zapoznawać się ze sposobem działania podzespołów aut każdej z marek zanim zacznie używać pojazdu, aby robić to poprawnie i niczego nie uszkodzić. Dlatego utrzymywanie jednolitych standardów na rynku w danej branży jest tak ważne, aby końcowy użytkownik miał jak największą wygodę korzystania z nich.

Co więcej, OpenGL stawia również wymagania w postaci funkcji, które mają zostać zawarte w bibliotece. To jest druga rzecz, która jest dużym ukłonem w stronę końcowego użytkownika.

Wracając do wcześniejszego porównania, tym razem można rzec, że takie podejście, które prezentuje organizacja Khronos Group, która utrzymuje ten standard API, jest niczym zbiór uchwytów, przycisków oraz wajch niezbędnych do poprawnego użytkowania samochodu. Dzięki obecności wajchy skrzyni biegów nie musimy znać szczegółowo mechanizmu przekładni, aby nią zarządzać. Ten sam przykład reprezentuje kierownica: nie musimy znać, jak działa układ kierowniczy aby być w stanie skręcić koła samochodu w wybranym przez nas kierunku. Jedno wspólne podejście, jakie przyjęli producenci samochodów sprawia, że użytkowanie tak bardzo skomplikowanej maszyny staje się proste.

To samo podejście prezentuje OpenGL jako specyfikacja. Jedna wspólna dokumentacja, te same funkcje, prowadzą do wyniku, którym jest obecny OpenGL. Programista, który używa bibliotek OpenGL nie musi dzięki temu znać dokładnych procesów zachodzących w procesorze karty graficznej, a jedynie funkcje wykonujące całe ich zestawy, co pozwala na szybsze tworzenie grafiki i większą dostępność samej technologii. Ta spójność doprowadziła między innymi do tak dużego spopularyzowania się tego standardu na całym świecie.

**Czym OpenGL wyróżnia się na tle konkurencji?**

OpenGL wyróżnia się także wśród innych specyfikacji swoją uniwersalnością w kontekście platformy na której pracuje. To oznacza, że napisanie aplikacji wykorzystującej OpenGL jest możliwe na ogromnej liczbie platform, od Windows, Mac OS X, czy Linux i jego różne dystrybucje, aż po konsole, takie jak Xbox 360, czy Nintendo DS. Można to porównać do popularnego DirectX, który jest dostępny jedynie na Windows oraz konsole Xbox, czy inne produkty firmy Microsoft[[2]](#footnote-2).

Dzieje się tak, gdyż DirectX nie może być nazywany „wolną” specyfikacją, ponieważ został stworzony przez Mirosoft, gdzie OpenGL jest niezależny od jakiejkolwiek konkretnej platformy. Każdy producent sprzętu elektronicznego może stworzyć swoją wersję biblioteki OpenGL, aby była jak najbardziej dopasowana do jego potrzeb i specyfikacji sprzętowej. Dlatego nazwa OpenGL zawiera w sobie słowo „Open”. Jest to otwarta biblioteka.

**Kto jest autorem i pomysłodawcą OpenGL?**

Pierwowzór obecnej specyfikacji powstał 1992 roku. Była to alternatywa dla API o nazwie Iris GL, które zaprojektowane zostało specjalnie na potrzeby stacji roboczych Silicon Graphics o nazwie OpenGL 1.0. Potrzeba zaprojektowania takiej alternatywy wynikała z braku formalnej specyfikacji dla Iris GL, co uniemożliwiało rozpowszechnienie tej technologii dla większej grupy odbiorców.

Autorami pierwszej wersji OpenGL byli Mark Segal i Kurt Akeley, którzy podjęli się sformalizowania specyfikacji Iris GL, aby można było jej używać na wielu platformach sprzętowych w tamtym czasie.

Jednak do dziś wielu deweloperów grafiki komputerowej, za pierwszą w pełni poprawną wersję uważa się OpenGL 1.1, który wprowadził na przykład tablice wierzchołków, które znacznie przyspieszyły ich odczyt, w sposób, który wymaga wykonania mniejszej ilości zapytań do pamięci w celu dostarczenia informacji o atrybutach każdego wierzchołka[[3]](#footnote-3).

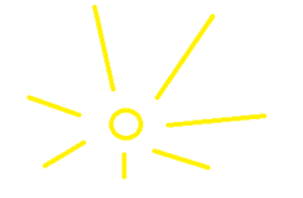
**W jakich celach OpenGL jest obecnie używany?**

Głównym zastosowaniem OpenGL jest grafika dwu- i trójwymiarowa. To oznacza że efekt jego działania możemy zobaczyć w przeróżnych miejscach: od programów projektujących części mechaniczne, przez aplikacje architektoniczne, po gry i filmy.

**Oświetlenie modelowanego obiektu**

Można wyróżnić kilka rodzajów generowanego oświetlenia w bibliotekach bazujących na OpenGL[[4]](#footnote-4), są to:

* Oświetlenie punktowe (pozycyjne):



Rysunek 1 Zasada działania światła punktowego

Źródło 1 Opracowanie własne

Źródło tego rodzaju oświetlenia traktuje się jako punkt. Dlatego, wielkość źródła jest pomijana przy jego projektowaniu. Oświetlenie takie symuluje klasyczne źródła światła, takie jak lampy pokojowe, żarówki, czy ogień. Ten rodzaj oświetlenia jest tematem pracy.

Na poniższym zdjęciu można zobaczyć przykład wykorzystania takiego rodzaju światła w praktyce:



Rysunek 2 Przykład wykorzystania światła pozycyjnego.

Źródło 2 Derek Key - https://www.flickr.com/photos/32211216@N06

* Oświetlenie kierunkowe



Rysunek 3 Zasada działania oświetlenia kierunkowego

Źródło 3 Opracowanie własne

Ten rodzaj światła przypomina światło słoneczne, które możemy obserwować dookoła nas w rzeczywistości. Oznacza to, że powierzchnia, którą modelujemy jest równomiernie oświetlona promieniami padającymi z góry, lub określonego konkretnie kierunku. Światło to nie ma jednego, konkretnie określonego źródła. Równoległe „promienie” padają na scenę, oświetlając ją równomiernie pod ustalonym kątem.

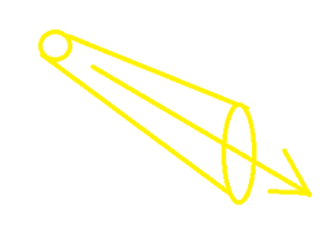
Na poniższym zdjęciu można zobaczyć przykład wykorzystania takiego rodzaju światła w praktyce:



Rysunek 4 Wykorzystanie światła kierunkowego w praktyce.

Źródło 4 Robert Pittman - https://www.flickr.com/photos/50144889@N08

* Oświetlenie typu SPOT



Rysunek 5 Zasada działania oświetlenia typu SPOT

Źródło 5 Opracowanie własne

W świetle tego rodzaju, ściśle określony jest punkt, z którego światło pada, czyli jego źródło. Jego kierunek musi być ustalony, podobnie jak przy wyżej opisanej pracy kamery w OpenGL. Światło rozchodzi się w kształcie stożka z zdefiniowanym kątem odcięcia i tłumiennością kątową rozchodzenia światła w stożku, która specyfikuje poziom rozmywania się światła na krawędziach stożka.

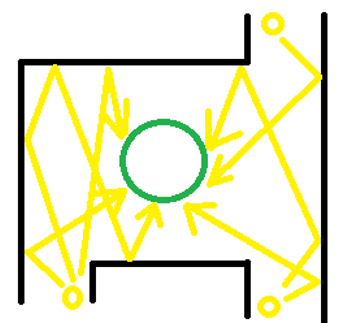
Na poniższym zdjęciu można zobaczyć przykład wykorzystania takiego rodzaju światła w praktyce:



Rysunek 6 Wykorzystanie światła typu SPOT w praktyce

Źródło 6 kjeik - https://www.flickr.com/photos/37622685@N02

* Oświetlenie otaczające (Ambient)

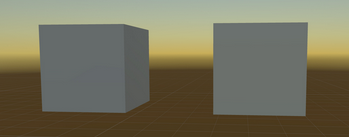


Rysunek 7 Zasada działania oświetlenia otaczającego

Źródło 7 Opracowanie własne

Światło otoczenia to światło, które nie pochodzi z określonego kierunku. Posiada swoje źródło, jednak jest ono dla nas niewidoczne, ale dochodzące od niego „promienie” mogą odbijać się na przykład od ścian pomieszczenia, w którym obserwowany model się znajduje i oświetlać go równomiernie z każdej strony.

Na poniższym zdjęciu można zobaczyć przykład wykorzystania takiego rodzaju światła w praktyce:



Rysunek 8 Przykład użycia światła ambient w praktyce

Źródło 8 Opracowanie własne

**Opis aplikacji**

Aplikacja, którą zdecydowałem się zaprojektować, jest program, który pozwoli nam poeksperymentować z ustawieniami oświetlenia.

Każdy programista grafiki komputerowej spotkał się zapewne kiedyś z problemem dopasowania odpowiedniego oświetlenia do swojej mapy w generowanym świecie, lub w programie służącym do modelowania.

Za oświetlenie to odpowiadają parametry, które znane są w świecie grafiki komputerowej od wielu lat, takie jak na przykład współczynnik intensywności połysku światła odbitego (ang. Specular shininess factor).

Moja aplikacja wychodzi naprzeciw problemowi doboru tych współczynników. Po jej uruchomieniu mamy dostęp do wyboru kilku rodzajów brył, na których przetestujemy nasze oświetlenie. Po dokonaniu wyboru mamy możliwość w czasie rzeczywistym zmieniać parametry zarówno samej bryły jak i jej oświetlenia. Mamy do wyboru dwa rodzaje oświetlenia: kierunkowe, aby zobaczyć, jak bryła prezentuje się w pełnej okazałości, oraz pozycyjne, które jest przedmiotem mojej pracy i pozwala zasymulować oświetlenie ulokowane w konkretnej pozycji w układzie współrzędnych i świeci we wszystkich kierunkach, niczym żarówka.

Wewnątrz aplikacji zawrę również implementację pracy kamery na dwa sposoby: obrót obserwowanego modelu za pomocą kursora oraz latająca kamera, dzięki której zobaczymy naszą bryłę z różnych odległości i pod różnymi kątami.

**Powód wybrania tematu**

Sam temat wybrałem grafiki komputerowej nie jest mi obcy już od długiego czasu. Od momentu, w którym dostałem mój pierwszy komputer oczywistością było, że chce na nim grać w przeróżne gry. Z początku było to zainteresowanie związane wyłącznie z ich fabułą i rozgrywką Jednak z czasem przerodziło się w zainteresowanie tym, jak są one stworzone. Ta praca jest częścią drogi, którą idę aby w przyszłości być autorem gier dla innych. Dlatego wybrałem ten temat pracy.

**Ocena użyteczności aplikacji**

Program, który jest tu przedstawiony może zostać wykorzystany do dalszej rozbudowy jako podstawa silnika graficznego. Moja implementacja mechanizmu oświetlenia jak i obsługi buforów, obiektów modelowanych i pracy kamery jest podstawową funkcją takiego silnika i otwiera możliwości do generowania takich obiektów, jak całe wirtualne światy, przy wyskalowaniu jej do większej ilości modeli.

Co więcej, jak już wspomniałem wcześniej, pozwala ona na przetestowanie parametrów oświetlenia kierunkowego i pozycyjnego oraz dopasowania parametrów materiału, z którego stworzona jest wyświetlana bryła.

W Internecie można spotkać całe zestawy gotowych danych współczynników oświetlenia dla uzyskania określonych efektów, co pokazuje skalę problemu z jakim spotykają się projektanci modeli do programów graficznych. Mój program wychodzi temu naprzeciw i pozwala samemu eksperymentować w poszukiwaniu tych pożądanych współczynników.

**Zawartość pracy**

Rozdział I zawiera opis dostępnych sposobów definiowania oświetlenia, jaki oferuje OpenGL. Zawarte są tam składowe oświetlenia Phonga i aspekty związane z ustawieniem kamery, a także obsługa tłumienia mocy światła wraz ze wzrastającą odlegóścią od źródła.

Rozdział II to specyfikacja funkcjonalna, czyli określa możliwości funkcjonalne programu. Opisuje czynności, jakie użytkownik może wykonać obsługując aplikacje.

Rozdział III traktuje o interfejsie użytkownika. Zostały opisane okna dostępne w aplikacji i sposób ich obsługi.

Rozdział IV objaśnia wszelkie zagadnienia związane z implementacją programu. Zostało opisane środowisko sprzętowo-programowe oraz wykorzystane biblioteki. Następnie zaprezentowane zostały uzyskane wyniki i przeprowadzone zostały eksperymenty na generowanym oświetleniu z użyciem omawianej aplikacji.

# WŁAŚCIWOŚCI I SPOSÓB DEFINIOWANIA ŚWIATŁA POZYCYJNEGO W OPENGL

Oświetlenie w moim programie wykonane jest z użyciem metody Phonga. Metoda ta sprawia, że powierzchnia, która jest oświetlana sprawia wrażenie bliskie rzeczywistemu. Składa się z trzech składowych, z których każda ma osobne przeznaczenie.

W OpenGL wyróżniamy kilka parametrów podstawowych współczynników definiujących światło używane na modelowanej scenie[[5]](#footnote-5), są to:

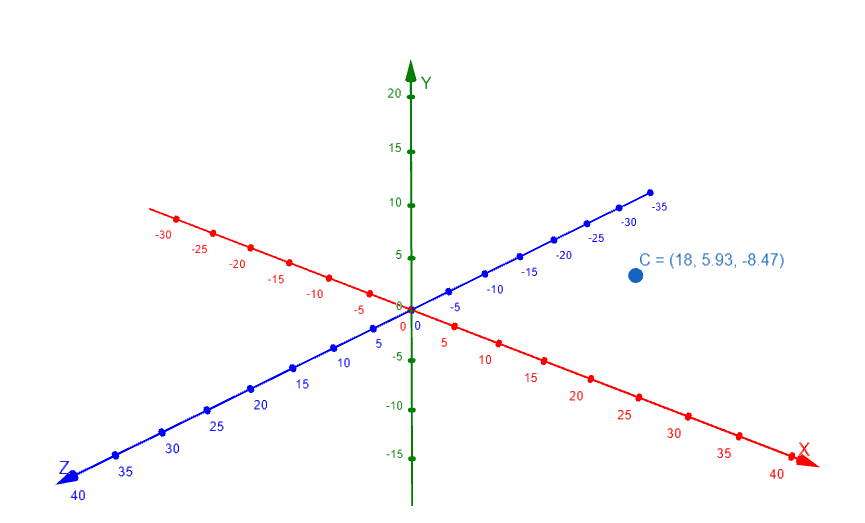
* Położenie źródła światła
* Kierunek świecenia (ta wielkość jest ważna w przypadku światła kierunkowego)
* Współczynnik odbicia światła otoczenia (ang. Ambient)
* Współczynnik światła odbijanego od obiektów kierunkowo (ang. Specular)
* Współczynnik światła odbijanego w sposób rozproszony (ang. Diffuse)
* Współczynnik tłumienia atmosferycznego

Rodzajem światła, które zostanie wykorzystane w mojej aplikacji, będzie światło pozycyjne. Można je porównać do świecącej żarówki lub pochodni. To oznacza, że wielkość źródła światła będzie wartością pomijalną, ponieważ traktuje się je jako punkt, a nie jako świecący obszar. Światło to będzie się rozprzestrzeniało w każdą stronę, tracąc swoje natężenie wraz z odległością od źródła.

Dodanie tego rodzaju światła do oświetlanej sceny polega na pomnożeniu wartości koloru na danym modelu przez współczynnik oświetlenia otoczenia, który dobierany jest przez nas według potrzeb. Następnie, używamy wynikowej wartości, jako kolor fragmentu w opracowywanym shaderze.

## Położenie źródła światła

Omawiana aplikacja obsługuje modelowanie świata w przestrzeni trójwymiarowej. Oznacza to, że każdy punkt w tym świecie musi zostać zdefiniowany z użyciem trzech współrzędnych: wysokości, szerokości i głębi[[6]](#footnote-6).



Rysunek 9 Przestrzeń współrzędnych wykorzystywana w OpenGL. Wi-doczny przykładowy punkt C ze współrzędnymi

Źródło 9 Opracowanie własne

Definicja położenia punktu – źródła światła:

Gdzie:

położenie punktu źródła światła na osi X

położenie punktu źródła światła na osi Y

położenie punktu źródła światła na osi Z

położenie punktu źródła światła

## Kierunek świecenia światła

Kierunek świecenia możemy wyznaczyć w prosty sposób. Posiadając wektor pozycji źródła światła oraz wektor pozycji fragmentu, który ma zostać oświetlony, odejmujemy od siebie te dwa wektory, dzięki czemu powstaje wektor kierunku świecenia.

Gdzie:

wektor kierunku świecenia światła

wektor współrzędnych oświetlanego fragmentu obiektu

wektor współrzędnych źródła światła

W OpenGL, pozycję tą definiuję jako zmienne w kodzie aplikacji, których wartość wyliczam w zależności od potrzeb. Następnie przekazuję ją do funkcji lookAt(), z biblioteki GLM, która generować będzie macierz widoku przydatną do obliczenia pozycji wierzchołków w świecie względem pozycji kamery.

## Współczynnik światła otoczenia (Ambient)

Światło otoczenia to światło, które nie pochodzi z określonego kierunku. Ma swoje źródło, jednak może ono nie być widoczne, ale światło z niego może docierać do interesującego nas modelu na przykład w wyniku odbić od ścian pomieszczenia, w którym model się znajduje.

Obiekty, które są oświetlone takim światłem są równomiernie oświetlone na powierzchniach skierowanych we wszystkich kierunkach**.**

Omawiany rodzaj jest składową oświetlenia Phonga. Jest to najprostszy rodzaj światła, jednak bardzo potrzebny. Prawdziwa czerń w naturze występuje niezwykle rzadko, ponieważ oznacza kompletny brak światła, co w ziemskich warunkach jest trudne do zaobserwowania na co dzień. Zwykle nawet w kompletnej ciemności potrafimy odróżnić kontury otaczających nas przedmiotów. W grafice komputerowej za tym efektem stoi światło otaczające. Potrafi nadać barwę obiektowi nawet wtedy, gdy nie można dostrzec źródła światła.

Do zaimplementowania światła otaczającego wystarczy jeden atrybut. Jest to jego intensywność, przy założeniu, że świecimy światłem białym. Kolor można zmieniać, lecz jest to jedynie zmiana wartości składowych RGBA.

Implementacja tego rodzaju oświetlenia w OpenGL nie należy do najtrudniejszych. Polega na przemnożeniu współczynnika intensywności przez kolor materiału ustalony dla tej składowej. Na koniec wartość koloru danego fragmentu zostaje dodana do pozostałych składowych.

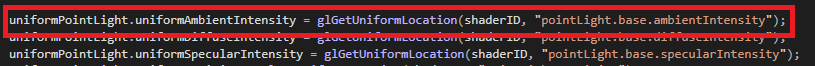
Algorytm, który pozwla utworzyć takie oświetlenie nie jest długi:

1. **Utworzenie i nadanie wartości współczynnikowi intensywności światła otaczającego**

Parametr ten przechowywany jest w aplikacji w postaci zmiennej, która zapisywana jest w buforze uniform OpenGL.

1. **Przekazanie współczynnika do shadera fragmentów**

Bufor zawierający współczynnik dowiązywany jest do zmiennej w shaderze fragmentów.

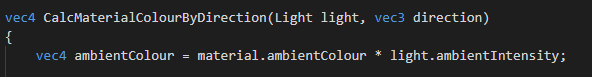


Rysunek 10 Powiązanie bufora uniform dla współczynnika intensywności światła otoczenia ze zmienną w shaderze fragmentów

Źródło 10 Opracowanie własne

1. **Przemnożenie współczynnika przez wektor koloru światła otaczającego dla wybranego materiału**

Przekazany współczynnik widoczny jest w shaderze, w którym wykonane są operacje obliczenia finalnego koloru generowanego fragmentu.



Rysunek 11 Fragment funkcji w shaderze wierzchołków odpowiedzialnej za obliczenie koloru fragmentu

Źródło 11 Opracowanie własne

1. **Dodanie obliczonej wartości do finalnego koloru fragmentu**

Kolor fragmentu reprezentowny jest przez 4 składowe: czerwień, zieleń, błękit i współczynnik przezroczystości. Każda ze składowych może mieć wartość od zera do jednego. Wartość jeden oznacza maksymalną intensywność danego koloru, a w przypadku współczynnika przezroczystości, po prostu nie przezroczystość. W przeciwnym wypadku, zero oznacza brak danego koloru, lub przezroczystość.

Każdy rodzaj światła składa się, dodając wartości swoich składowych do jednej puli, na finalny kolor fragmentu.

Efekt uzyskany na podstawie samego oświetlenia otaczającego nie należy do zadowalających, ponieważ nie tworzy wrażenia głębi.

****

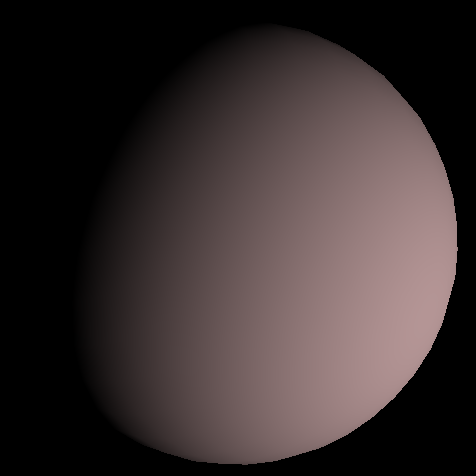
Rysunek 12 Przykład obiektów oświetlonych światłem otaczającym

Źródło 12 Opracowanie własne

Dodanie światła typu Ambient do naszej sceny polega na pomnożeniu wartości naszego koloru przez współczynnik oświetlenia otoczenia, który sami dobieramy. Następnie używamy wynikowej wartości jako kolor fragmentu w shaderze.

## Współczynnik światła odbijanego w sposób rozproszony (Diffuse)

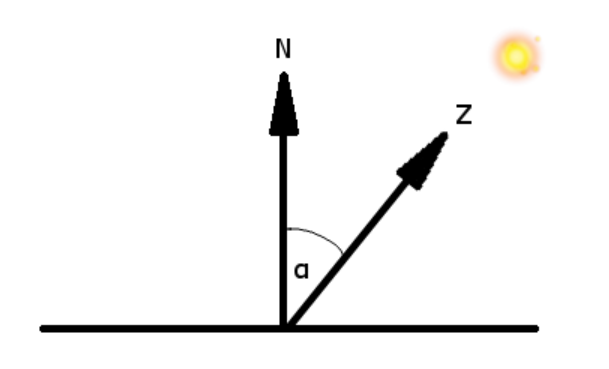
Kolejnym rodzajem światła w oświetleniu Phonga jest światło rozproszone. Można dzięki niemu określić, z której strony położone jest źródło światła w stosunku do obiektu.



Rysunek 13 Kula oświetlona jedynie światłem rozproszonym

Źródło 13 Opracowanie własne

Światło to uzyskiwane jest na podstawie analizy kąta między wektorem normalnym oświetlanej powierzchni, a wektorem prowadzącym od oświetlanego frgmentu do oka kamery. Im większy jest ten kąt, tym mniejsza intensywność oświetlenia rozproszonego.



Rysunek 14 Prezentacja wyliczania intensywności światła rozproszonego

Źródło 14 Opracowanie własne

Na powyższej grafice można dostrzec dwa wektory:

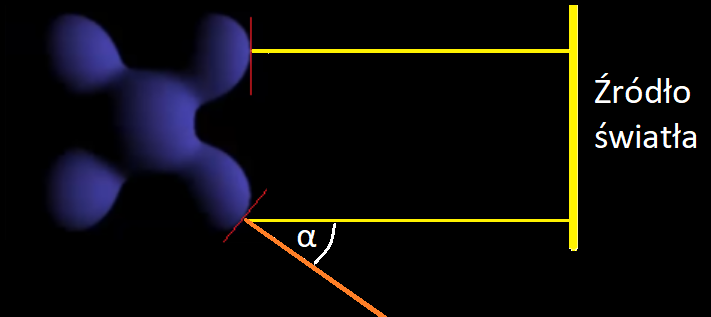
* N – normalna powierzchni
* Z – wektor wskazujący źródło światła

Jak już zostało wspomniane, większa wartość kąta α powoduje mniejszą intensywność oświetlenia rozproszonego. Podobną przeciwproporcjonalnością charakteryzuje się funkcja cosinus, której wartość rośnie wraz z malejącym kątem podanym jako jej argument dla pewnych przedniałów wartości kąta (na przykład od 0 do 90 stopni).

Cosinus kąta pomiędzy dwoma wektorami możemy uzyskać z pomocą iloczynu skalarnego:

Po znormalizowaniu wektorów , można skrócić to równanie do poniższej postaci:

To oznacza, że wyliczając iloczyn skalarny na tych obu wektorach, można dostać w wyniku współczynnik odpowiadający za intensywność światła rozproszonego! Tak też zostało to zrealizowane w omawianej aplikacji.



Rysunek 15 Przykład wyznaczenia współczynnika światła rozproszonego. Linie żółte reprezentują wektor biegnący od fragmentu do źródła światła. Linia pomarańczowa reprezentuje wektor normalny powierzchni

Źródło 15 Opracowanie własne

Na powyższym obrazie można zobaczyć dwie sytuacje: pierwsza, górna, obrazuje to, jak fragment leżący naprzeciw źródła światła rozjaśniany jest w stopniu większym, niż fragment pokazany na przykładzie dolnym. Dolny przykład pokazuje, że kąt pomiędzy wektorem normalnym (pomarańczowym), a wektorem zwróconym w stronę źródła światła znacząco wpływa na jasność fragmentu.

Wartości wektorów wyliczane są w następujący sposób:

Gdzie:

Lewy wierzchołek trójkątnego prymitywu

Środkowy wierzchołek trójkątnego prymitywu (pozycja oświetlanego fragmentu)

Prawy wierzchołek trójkątnego prymitywu

Pozycja źródła światła

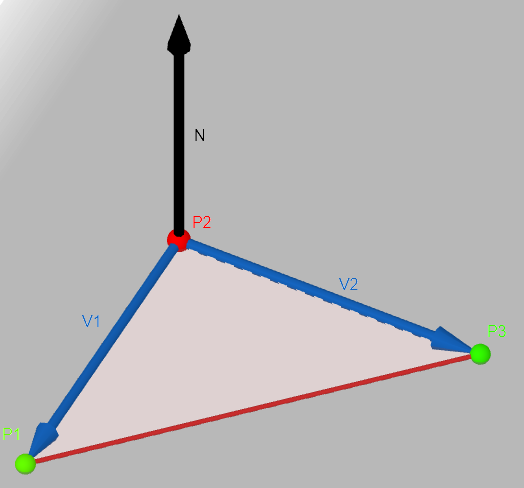
Pierwszy wektor pomocniczy

Drugi wektor pomocniczy

Współczynnik światła rozproszonego

kąt pomiędzy wektorem normalnym i skierowanym w stronę źródła światła

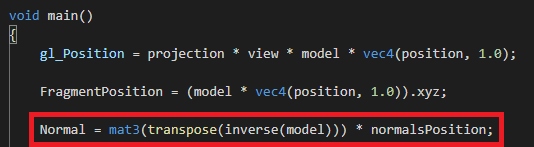
Ostatni wzór na współczynnik światła rozproszonego rozdzielony jest na dwie opcje, ponieważ dla światła rozproszonego zakłada się, że dla danego źródła światła nie dociera ono ze stron innych niż było to zamierzone.



Rysunek 16 Wyznaczanie wektora normalnego na podstawie prymitywu trójkąta. P2 jest punktem, dla którego obliczana jest jasność i kolor

Źródło 16 Opracowanie własne

Światło rozproszone wymaga jasno zdefiniowanych normalnych. Co się stanie jednak jeśli obiekt, którego normalne zostały już policzone, zostanie przekształcony w jakiś sposób? Może to być obrót modelu, bądź jego powiększenie w którymś z wymiarów. Wektory normalne też muszą wtedy zostać przekształcone w ten sam sposób. Dlatego w omawianej aplikacji są przemnożone przez macierz przekształceń w shaderze wierzchołków.



Rysunek 17 Przekształcenie pozycji normalnych za pomocą macierzy przekształceń (tutaj pod nazwą model)

Źródło 17 Opracowanie własne

Następnie, normalne wraz z pozycją oświetlanego fragmentu przekazywane są do shadera fragmentów, gdzie użyte zostaną w celu uzyskania współczynnika intensywności światła rozproszonego.

Finalnie, wyliczany jest kolor światła rozproszonego:

Gdzie:

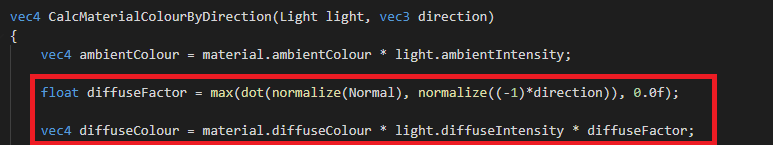
finalny kolor światła rozproszonego

wektor współczynników RGBA dla światła rozproszonego dla użytego materiału

intensywność światła rozproszonego w skali od 0 do 1

współczynnik światła rozproszonego

Współczynnik światła rozproszonego różni się od intensywności światła rozproszonego tym, że ten pierwszy wyliczany jest na bazie kąta padania śwaitła, a ten drugi jest informacją, w jakim stopniu ten rodzaj światła jest emitowany ze źródła.



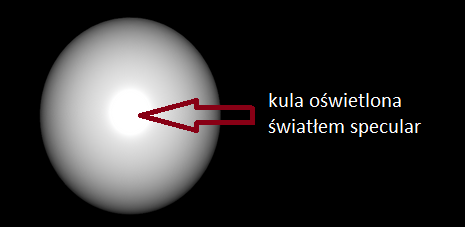
Rysunek 18 Fragment funkcji obliczającej współczynnik intensywności światła rozproszonego

Źródło 18 Opracowanie własne

## Współczynnik światła odbijanego od obiektów kierunkowo (Specular)

Światło odbłysku (ang. Specular), posiada określony kierunek świecenia, jednak jest odbijane tylko w jedną stronę, za to z dużą intensywnością.

W tym rodzaju światła musimy też uwzględnić z której strony umieścimy kamerę, czyli z którego punktu patrzymy na to światło.



Rysunek 19 Przykład oświetlenia odbitego

Źródło 19 Opracowanie własne

Ostatnią składową oświetlenia Phonga jest światło odbite. Występuje ono tylko wtedy, gdy aktywne jest światło rozproszone. Można odnieść wrażenie, że znajduje się na jego powierzchni.

Większość przedmiotów w rzeczywistym świecie charakteryzuje się choćby w małym stopniu odblaskowością. Nawet najbardziej matowe materiały potrafią rozjaśnić się po zaświeceniu nań latarką. Można powiedzieć, że jest to swego rodzaju lustrzana właściwość materiałów, ponieważ tak naprawdę owe rozjaśnienie spowodowane jest odbiciem się postaci źródła światła w obserwowanym obiekcie. Nie jest ono tak dokładne jak w przeciętnym lustrze ze względu na nierówności powierzchni i jej własciwości, jednak zasada jest podobna.



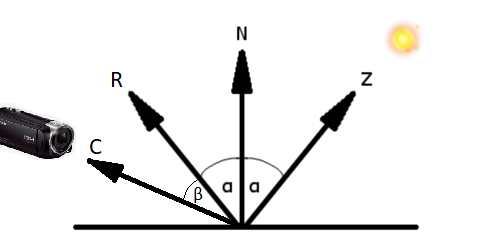
Rysunek 20 Przedstawienie oświetlenia odbitego w rzeczywistym świecie (wskazane strzałką)

Źródło 20 Opracowanie własne

Phong spróbował zasymulować ten efekt za pomocą światła odbitego i udało mu się to naprawdę dobrze. Omawiana aplikacja również implementuje ten rodzaj oświetlenia. Będzie do tego potrzeba więcej danych, niż przy poprzednich składowych oświetlenia, ponieważ obliczenia oparte są nie tylko na pozycji fragmentu i źródła światła, lecz także na pozycji oka kamery.

Potrzebne są cztery wektory[[7]](#footnote-7):

* Wektor normalny -
* Wektor prowadzący od fragmentu do źródła światła -
* Wektor odbicia światła od fragmentu –
* Wektor od fragmentu do oka kamery -

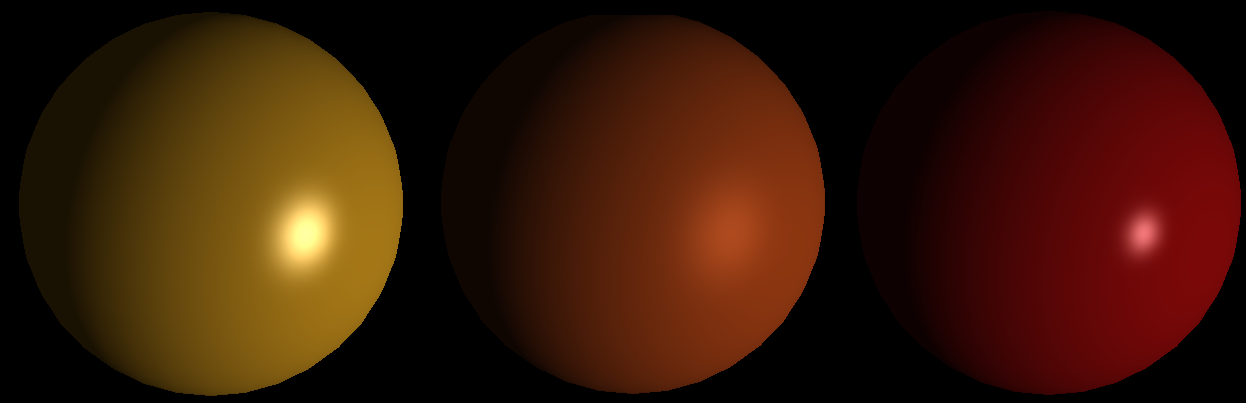


Rysunek 21 Wektory potrzebne do wyliczenia współczynnika światła odbitego

Źródło 21 Opracowanie własne

Głównym parametrem, który decyduje o tym, czy zobaczymy światło odbite, jest kąt β. Poruszając kamerą, należy trafić w promień odbity, czyli wektor powinien zrównać się z wektorem , aby zobaczyć charakterystyczną „plamkę” światła odbitego. Oczywiście nie jest to zaimplementowane zero jedynkowo. Istnieje płynne przejście, jak widać to na grafice prezentującej światło odbite.

O przejściu tym decyduje współczynnik połysku dla użytego materiału.



Rysunek 22 Prezentacja różnych współczynników odbłysku, od lewej: mosiądz, miedź i rubin

Źródło 22 Opracowanie własne

Przedstawione powyżej różne rodzaje materiałów mają znacząco różniące się od siebie współczynniki odbłysku. Zostały one podane w poniższej tabeli:

Tabela 1 Parametry materiałów

Źródło: Opracowanie własne

|  |  |
| --- | --- |
| Materiał | Współczynnik odbłysku |
| Mosiądz | 27,89 |
| Miedź | 12,8 |
| Rubin | 76,8 |

Jak można spostrzec, wyższa wartość wspólczynnika odbłysku charakteryzuje się większym skupieniem światła w jednym punkcie i lekko ostrzejszymi krawędziami „plamki” odbitej. Natomiast mniejsza wartość tego współczynnika pozwala wygenerować materiał sprawiający wrażenie matowego.

Podobnie jak światło rozproszone, odbite również wymaga obliczenia wspóczynnika tego światła, który później zostanie pomnożony przez kolor materiału dla tego rodzaju światła.

Algorytm obliczania współczynnika światła odbitego wygląda następująco:

1. Domyślnie, kolor wynikający z oświetlenia odbitego ma wartość zerową, która zmieni się dopiero wtedy, gdy oświetlany fragment znajdzie się w zasięgu światła rozproszonego.
2. Po sprawdzeniu, że analizowany fragment znajduje się w zasięgu światła rozproszonego, wyliczany jest wektor od fragmentu do oka kamery (), wektor od fragmentu do źródła światła () oraz wektor odbity (). Można przyjąć założenie, że wektor normalny został już obliczony wraz z obliczaniem współczynnika światła rozproszonego.

Gdzie:

Pozycja oka kamery

Pozycja oświetlanego fragmentu

Pozycja źródła światła

Wektor normalny do powierzchni oświetlanej

1. Obliczenie wartości współczynnika światła odbitego

Gdzie:

Współczynnik światła odbitego

Współczynnik odbłysku

1. Obliczenie koloru światła odbitego (zwiększenie jego wartości z początkowo zerowej)

Gdzie:

Barwa oświetlenia odbitego dla wybranego materiału w formacie RGBA, w skali od 0 do 1

Intensywność światła odbitego

Współczynnik światła odbitego

## Współczynnik tłumienia atmosferycznego

Tłumienie światła, jest to zjawisko zmniejszenia natężenia światła wraz z pokonywaną przez nie odległością.

Wiąże się to także z faktem, że promienie wraz z coraz większą odległością od źródła światła zaczynają zanikać, czyli stają się coraz słabsze. Dlatego ważnym jest aby prawidłowo określić położenie światła w przestrzeni, aby oświetlić pożądane modele.

Rysunek 23 Przykład obecności tłumeinia światła w świecie rzeczywitym

Źródło: StockyPics - https://search.creativecommons.org/photos/b599a375-5198-429e-b50b-a7f578b49804

Należy wyliczyć parametr, który definiuje tłumienie atmosferyczne światła:

Gdzie:

d - parametr oznaczający odległość od źródła światła.

, , - współczynniki tłumienia światła:

(constant) – współczynnik stały

(linear) – współczynnik liniowy

(quadratic) – współczynnik kwadratowy

W przypadku braku tłumienia światła:

= 1, = 0, = 0

W innych przypadkach można manipulować wartością tych współczynników, aby uzyskać pożądany efekt tłumienia światła.

To, jakie wartości dobierzemy dla światła, powinno być motywowane rodzajem „ośrodka”, w którym rozchodzi się światło. Na przykład, w czystym powietrzu światło dociera daleko. Natomiast w pomieszczeniu zadymionym lub zamglonym światło jest przyćmione, stłumione.

Po obliczeniu każdego rodzaju światła, wystarczy je na siebie nałożyć. Biorąc pod uwagę, że wartości wynikowe barwy każdej ze składowych oświetlenia reprezentowana jest jako czteroelementowy wektor RGBA w przestrzeni wartości od 0 do 1, wartości te wystarczy jedynie dodać do siebie.

Gdzie:

Barwa oświetlenia otaczającego

Barwa oświetlenia rozproszonego

Barwa oświetlenia odbitego

Jeśli którakolwiek ze składowych finalnej wartości K jest większa niż 1, pozostaje jedynką.

## Materiały

Opisywana aplikacja oferuje szereg dostępnych materiałów.

Materiał w grafice komputerowej jest definiowany jako opis sposobu zachowania się powierzchni obiektów na poszczególne rodzaje światła[[8]](#footnote-8). Oznacza to, że możemy zdefiniować parametry materiału w taki sposób, aby jak najwierniej oddawały istniejącą w rzeczywistym świecie substancję.

Parametrami określającymi rodzaj materiału są:

* Współczynnik odbłysku światła – wyznacza on stopień występowania rozbłysku światła odbitego w kierunku kamery
* Składowe RGBA dla oświetlenia otaczającego
* Składowe RGBA dla oświetlenia rozproszonego
* Składowe RGBA dla oświetlenia odbitego
* Składowe RGBA dla światła emitowanego przez obiekt (te są pominięte w omawianej aplikacji)

Fakt istnienia składowych RGBA dla poszczególnych rodzajów oświetlenia Phonga oznacza, że możliwe jest zaimplementowanie materiału, który będzie mienił się różnymi kolorami. Umożliwia to symulację szerokiego wachlarza substancji, z których może być wykonany generowany obiekt.



Rysunek 24 Przykład różnych materiałów w świecie rzeczywistym, od lewej: chrom i miedź

Źródło 23 Opracowanie własne

# SPECYFIKACJA FUNKCJONALNA APLIKACJI

Jak zostało już wspomniane w poprzednich rozdziałach, omawiana aplikacja oferuje szereg funkcji, które zostały opisane poniżej.

## Scena

Scena aplikacji to czarna, trójwymiarowa przestrzeń, po środku której umiejscowiony jest wygenerowany model. Oferowane są 4 modele do wyboru:

* Sfera – użytkownik może wybrać ilość wierzchołków, z których zbudowana jest sfera w pionie i w poziomie
* Prostopadłościan
* Ostrosłup
* Walec

Użytkownik może wygenerować dowolny z nich w dowolnym momencie działania aplikacji. Model dostępny jest do wglądu z każdej strony.

Dostępne są również dwa dodatkowe tryby podglądu bryły:

* Z nałożoną teksturą
* Jako siatka wielokątów

Tryb siatki wielokątów pomaga zrozumieć użytkownikowi, jak zbudowana jest bryła z prymitywów.

Dodatkowo, dostępne są do wyboru dla użytkownika dwa sposoby generowania wektorów normalnych dla powierzchni oświetlanej bryły:

* Uśrednione – tryb ten zaokrągla ostre krawędzie prymitywów, dzięki czemu użytkownik może odnieść wrażenie obłości bryły
* Równoległe – bryła posiada ostre krawędzie pomiędzy prymitywami, co widać też w jej oświetleniu

Dostępna jest też funkcja umożliwiająca wybór materiału, który będzie symulowany.

## Źródło światła

Aplikacja oferuje dwa rodzaje oświetlenia. Głównym jest oświetlenie pozycyjne, które można modyfikować na wymienione poniżej sposoby:

* Zmiana barwy emitowanego oświetlenia pozycyjnego
* Zmiana intensywności oświetlenia – poprzez zmianę intensywności rozumiana jest jednoczesna zmiana intensywności trzech składowych oświetlenia Phonga (oświetlenia otaczającego, rozproszonego i odbitego), z użyciem dla nich tych samych wartości w tym samym czasie
* Niezależna zmiana wartości współczynników intensywności oświetlenia otaczającego, rozproszonego i odbitego
* Zmiana wartości współczynników tłumienia oświetlenia pozycyjnego (współczynnika stałego, liniowego i kwadratowego)

Zmiana pozycji źródła światła pozycyjnego realizowana jest poprzez wbudowany algorytm, powodujący nieustanne poruszanie się go wokół generowanej bryły. Użytkownik może zmienić wartość prędkości poruszania się tego źródła światła, co oznacza że może je też zatrzymać.

## Kamera

Aplikacja oferuje dwa tryby poruszania się kamery w celu dokładnego obejrzenia efektów działania oświetlenia pozycyjnego:

* Tryb obracania – polega na obserwacji bryły z takiej samej odległości z każdej strony. Użytkownik może zmieniać pozycję kamery poprzez kliknięcie lewego klawisza myszy i przeciągnięcie kursora w kierunku, w którym chce obrócić kamerę względem środka układu współrzędnych
* Tryb latania – umożliwia obserwację bryły z dowolnej perspektywy. Użytkownik może zmieniać pozycję i zwrot kamery poprzez naciskanie klawiszy WSAD oraz poruszanie kursorem. Tryb ten pozwala na przykład na zbliżenie się do bryły bardzo blisko i zwrócenie kamery w stronę jej brzegu, co może stworzyć iluzję „stania” na jej powierzchni

## Transformacje

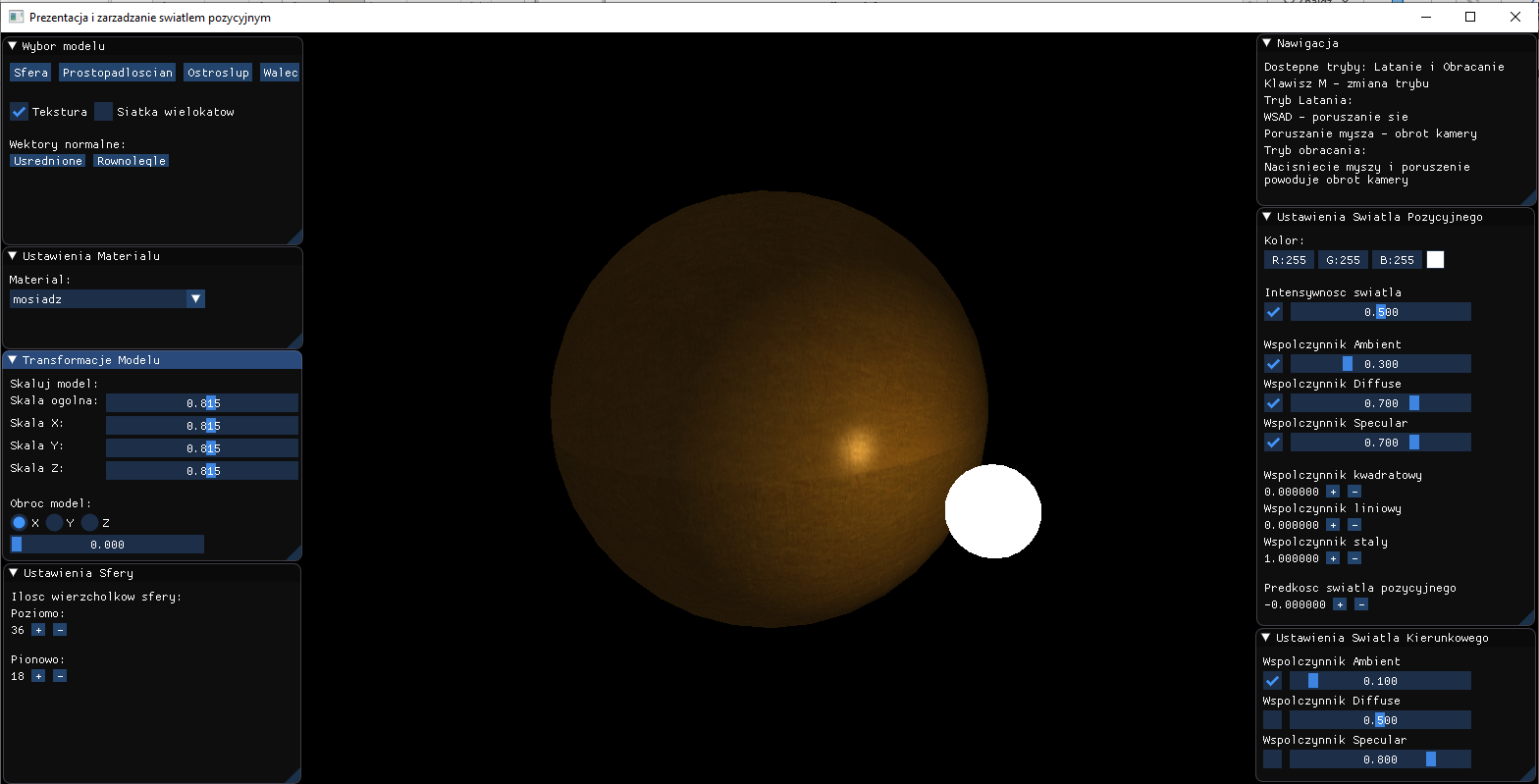
Aplikacja oferuje opcję transformowania oglądanej bryły. Można ją wyskalować w 3 kierunkach, wzdłuż osi X, Y oraz Z. Dostępna jest też opcja obrócenia modelu, aby dojrzeć jak prezentuje się on oświetlony w innym miejscu.

# OPRACOWANIE INTERFEJSU GRAFICZNEGO APLIKACJI

## Jak korzystać z programu?

Obsługa aplikacji nie jest trudna. Po jej uruchomieniu, oczom użytkownika ukazuje się wygenerowana scena z umiejscowioną na środku kulą, którą oświetla lekko światło otaczające wraz z okrążającym ją powoli światłem pozycyjnym.

Dla ułatwienia zrozumienia położenia tego światła, przedstawione jest ono za pomocą małej, białej kuli, która zmienia swój kolor wraz z barwą światła.



Rysunek 25 Domyślnie generowana kula, czyli model główny, wraz z okrążającą ją mniejszą kulą – światłem pozycyjnym.

Źródło 25 Opracowanie własne

Oprócz oczywistego modelu, program wita nas również zestawem nieruchomych okienek, które zawierają w sobie dużą ilość przycisków i suwaków.

Służą one do dostosowania szczegółów generowanego modelu oświetlenia oraz wyświetlanego obiektu w celu umożliwienia użytkownikowi wykonania kilku eksperymentów związanych z oświetleniem. Sprawiają one, że wykonanie ich jest bardzo proste.

Użytkownik nie ma narzuconej konkretnej kolejności ustawiania wszystkich szczegółów. Wszystkie okna dostępne są od samego uruchomienia i działają niezależnie od pozostałych. Jedynym wyjątkiem jest okno „Ustawienia sfery”, które wyświetla się tylko wtedy, gdy na scenie głównej wyświetlona jest sfera, zamiast innego obiektu.

Pierwszym omawianym oknem jest okno „Wybór modelu”. Użytkownik korzystając z niego, jest w stanie skorzystać z opcji wygenerowania innych modeli niż tylko kula. Dostępne są: Sfera, Prostopadłościan, Ostrosłup oraz Walec.

Po naciśnięciu któregokolwiek z nich, model zostaje momentalnie wygenerowany.

Poniżej dostępne są również dwa tryby generowania wektorów normalnych, a co za tym idzie, gładkości krawędzi powierzchni bryły. Tryb „Uśrednione” powoduje, że program zaczyna wykorzystywać mechanizm interpolacji, który oferuje OpenGL i wygładza krawędzie.

Tryb „Równoległe” nie korzysta z tego mechanizmu i renderuje bryłę, po prostu łącząc sąsiadujące ze sobą wierzchołki prostymi liniami, co powoduje efekt „kanciastości” sfery.

Po wybraniu i naciśnięciu trybu, efekt jest nanoszony na scenę momentalnie.

Kolejną opcją tego okna jest opcja nałożenia tekstury. Każdy z dostępnych materiałów ma domyślnie ustawioną swoją teksturę, którą możemy aktywować tym przyciskiem. Zostaną one zaprezentowane w sekcji omówienia materiałów.

Użytkownik może też w tym oknie wybrać tryb rysowania. Zaznaczając opcję „Siatka wielokątów”, uruchamia się tryb rysowania za pomocą linii tworzonych na granicach prymitywów. Wyłączając tą opcję, zostaje użyty tryb rysowania z wypełnieniem, czyli taki jaki jest widoczny na powyższych grafikach.

W oknie o nazwie „Nawigacja”, znajduje się instrukcja objaśniająca sterowanie kamerą.

Dostępne są dwa tryby:

* Obracania, w którym obracamy kamerą wokół bryły za pomocą myszy
* Latania, w którym klawiszami WSAD przemieszczamy kamerę, a ruchami myszy obracamy kamerę.

Z pomocą okna „Transformacje Modelu”, użytkownik może wykonywać operacje skalowania wyświetlanej bryły w każdym kierunku, oraz obracania jej w 360 stopniach.

Aby zmienić wartość któregokolwiek z wyżej wymienionych parametrów, wystarczy „chwycić” kursorem z którykolwiek z suwaków i przesunąć go w prawo, aby zwiększyć wartość, bądź w lewo, aby ją zmniejszyć.

W przypadku obracania bryły, wybór osi wzdłuż której będziemy obracać, odbywa się za pomocą zaznaczenia odpowiedniej opcji spośród: X, Y i Z.

W oknie „Ustawienia światła pozycyjnego” odbywa się kontrola nad głównym tematem tej pracy. Użytkownik jest w stanie kontrolować wszystkimi współczynnikami decydującymi o generowanym świetle pozycyjnym. Dzięki temu panelowi, może dostosować je tak, aby uzyskać pożądany efekt, zbliżony do światła, jakie możemy zaobserwować w rzeczywistym świecie.

Pierwszym parametrem, którym można zarządzać jest barwa oświetlenia. Po naciśnięciu białego kwadracika, zostanie wyświetlona paleta barw, w której użytkownik może wybrać interesującą go barwę i jej intensywność.

Można także określić wartość współczynników światła otoczenia (ang. Ambient), rozproszenia światła pozycyjnego (ang. Diffuse) oraz odbicia (ang. Specular). Zmiana dokonywana jest za pomocą „chwytania” za suwak i przesuwania go. Światło wygenerowane z nowymi parametrami widziane jest od razu na podglądzie na środku ekranu. Dostępne są również przyciski, które umożliwiają kompletne wyłączenie danej składowej oświetlenia Phonga, aby po ponownym ich przyciśnięciu wrócić do poprzednio ustawionej wartości. Poruszając suwakiem podpisanym „Intensywność światła” można jednocześnie zmienić wartość intensywności wszystkich składowych oświetlenia Phonga.

W tym oknie użytkownik może też dokonać zmiany współczynników zanikania światła. Są to współczynniki: kwadratowy, liniowy i stały. Projekt interfejsu zakłada, że użytkownik posiada wiedzę na temat grafiki komputerowej, dlatego współczynniki zostały zostawione do dowolnego modyfikowania użytkownikowi. Zmiany można dokonać w sposób znany z poprzednich okien, czyli przyciskami „+” oraz „-”.

Ostatnią cechą generowanego światła pozycyjnego, jest fakt, że od uruchomienia aplikacji jest on w nieustannym ruchu wokół bryły. Uzytkownik jednak może potrzebować, aby źródło stało w miejscu, bądź poruszało się szybciej. Dlatego jest widoczna opcja zmiany prędkości poruszania się źródła światła. Klikając „+”, możemy przyspieszyć okrążanie bryły, analogicznie jest z przyciskiem oznaczonym „-”. Po kliknięciu ich, źródło światła będzie zmieniać swoją prędkość zgodnie z ustawieniem.

Istnieje jeszcze okno „Ustawienia światła kierunkowego”. Oświetlenie kierunkowe nie jest przedmiotem tej pracy, dlatego opcje dostępne w tym oknie ograniczają się tylko do omówionych wcześniej składowych oświetlenia Phonga.

Domyślnie, od uruchomienia programu dostępne jest jedynie światło otaczające, które umożliwia dokładne spostrzeżenie granic bryły tam, gdzie światło pozycyjne nie dosięga.

Okno „Ustawienia Sfery” pojawia się jedynie wtedy, gdy na scenie wygenerowana jest bryła. Możemy w nim ustawić, w jaki sposób ma zostać utworzona. To znaczy, że możemy wybrać, z ilu wierzchołków w kierunku pionowym i poziomym bryła będzie się składać. Mniejsza ilość wierzchołków spowoduje efekt „kanciastości” bryły. Większa nastomiast sprawi, że będzie ona wyglądac bardziej gładko. Dokonywanie tej zmiany realizowane jest przez naciskanie przycisków „+” i „-”.

Program oferuje również możliwość wyboru materiału w oknie „Ustawienia Materiału”, z którego wykonana jest bryła. Dostępnych jest aż dziewiętnaście materiałów, z których każdy ma przypisane odpowiednie wartości dla generowania koloru bryły w zależności od padającego światła.

Lista dostępnych materiałów:

* Mosiądz
* Brąz
* Polerowany brąz
* Chrom
* Miedź
* Polerowana miedź
* Złoto
* Polerowane złoto
* Grafit
* Srebro
* Polerowane srebro
* Szmaragd
* Jadeit
* Obsydian
* Perła
* Rubin
* Turkus
* Czarny plastik
* Czarna guma

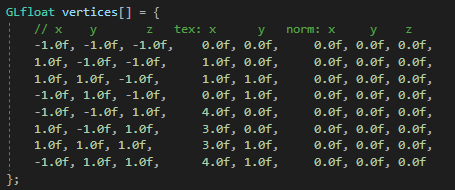
Użytkownik może wybrać interesujący go materiał, korzystając z rozwijanej listy.

## Realizacja graficznego interfejsu użytkownika

Podstawową informacją, od której zależy to, co zobaczymy w finalnie generowanej ramce, są wierzchołki. Każda wygenerowana figura, którą zawarłem w swoim programie, bazuje na wierzchołkach, które wprowadziłem do programu.

W opisywanym programie, każdy wierzchołkek zawiera 3 rodzaje informacji, jakie dla niego wprowadzam. Są to:

* Pozycja wierzchołka (współrzędne X, Y i Z) – miejsce, w którym osadzony jest wierzchołek we współrzędnych generowanego świata
* Pozycja tekstury (współrzędne X i Y) – powierzchnia modelu pokryta jest teksturą. Dlatego każdy wierzchołek ma zdefiniowaną pozycję w granicach tekstury, która zostanie „przypięta” do niego w miejscu o podanych współrzędnych
* Zwrot wektoru normalnego w stosunku do generowanej powierzchni (współrzędne X, Y i Z) – W celu poprawnego generowania oświetlenia, każdy wierzchołek musi mieć zdefiniowany swój wektor normlany, który wskaże, w którą stronę zwrócona jest powierzchnia, na którą składa się wierzchołek



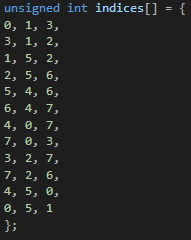
Rysunek 26 Przykład tablicy z wierzchołkami sześcianu. Każda linia to dane jednego wierzchołka.

Źródło 26 Opracowanie własne

Na powyższym zdjęciu widoczne są zera w miejscach składowych wektorów normalnych. Zostaną one dopiero wypełnione w późniejszym etapie z pomocą odpowiednich metod w programie.

Same wierzchołki jednak nie sprawią, że OpenGL będzie w stanie wygenerować model. Kolejnym krokiem, który trzeba podjąć, aby przygotować te dane, jest połączenie ich według wzorca, którym jest wykorzystany prymityw.

W omawianej aplikacji, prymitywem tym jest trójkąt. To oznacza, że każdy model, który zostanie utworzony w niniejszym programie, złożony jest z wielu trójkątów. Sposobem, w jaki „powiemy” OpenGL’owi o sposobie połączenia wymienionych wierzchołków, są tak zwane indeksy. Jest to kolejna tablica z liczbami, które oznaczają jedynie numery wierzchołków z poprzedniej tablicy, według kolejności występowania rzędów.



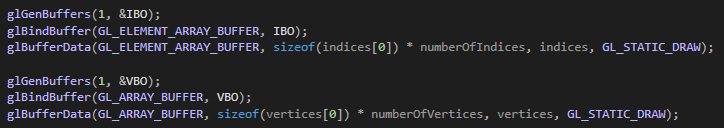
Rysunek 27 Tablica indeksów dla sześcianu

Źródło 27 Opracowanie własne

Każdy rząd tej tablicy odpowiada pojedynczemu trójkątowi. Dzięki takiemu rozwiązaniu, tworząc sześcian, nie jesteśmy zmuszeni do definiowania osobnych wierzchołków dla każdego boku generowanej figury, co oszczędza użytkowaną przez program pamięć.

Następnym etapem jest utworzenie buforów, które będą przekazane na wejście potoku OpenGL:

* Bufor wierzchołków (VBO – Vertex Buffer Object) – to w nim przechowywane są dane z tablicy wierzchołków
* Bufor indeksów (IBO – Index Buffer Object) - w nim przechowywane są dane z tablicy indeksów

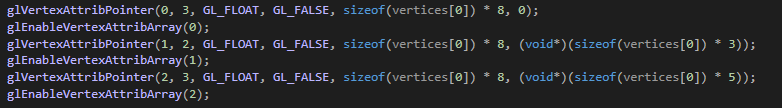


Rysunek 28 Utworzenie buforu wierzchołków i indeksów

Źródło 28 Opracowanie własne

Analizując powyższe zdjęcie, można zauważyć, że określiłem użycie powyższych buforów jako GL\_STATIC\_DRAW. Oznacza to, że dane, które są tam składowane, są tam wprowadzone raz, i nie będą modyfikowane w czasie działania programu. Jest to prawda, ponieważ po wprowadzeniu ich w tym miejscu, model jest modyfikowany za pomocą macierzy przekształceń, bez wpływu na dane źródłowe, które przez cały czas działania programu zostają takie same. To podejście jest optymalne z perspektywy działania programu, ze względu na wykorzystanie mocy obliczeniowej procesora w zamian za rzadszą potrzebę uruchomienia operacji zapisu w pamięci.

Kolejnym krokiem jest zdefiniowanie wskaźników na poszczególne atrybuty w buforach. Oznacza to, że istnieje potrzeba określenia ilości miejsca, które zajmuje każdy wierzchołek w bajtach, a także pozycji poszczególnych parametrów w obrębie pojedynczego wierzchołka.



Rysunek 29 Definicja wskaźników na poszczególne parametry wierzchołka w buforach

Źródło 29 Opracowanie własne

W przypadku pokazanym na powyższym zdjęciu, wskazane są trzy parametry (o indeksach 0, 1 i 2), oznaczające kolejno:

* 3 miejsca na dane typu GLfloat, poczynając od miejsca zerowego (dla współrzędnych x, y, z wierzchołka),
* 2 miejsca na dane typu GLfloat, poczynając od miejsca trzeciego (dla współrzędnych x, y przypięcia tekstury),
* 3 miejsca na dane typu GLfloat, poczynając od miejsca piątego (dla współrzędnych x, y, z wektora normalnego danego wierzchołka)

Razem, miejsce w buforze dla każdego wierzchołka to osiem pozycji dla zmiennych typu zmiennoprzecinkowego, czyli zgodnie z przedstawioną wcześniej tablicą wierzchołków.

Kolejnym etapem jest połączenie utworzonych buforów wierzchołków z tablicą wierzchołków (VAO – Vertex Array Object), tablicę indeksów z utworzonym buforem indeksów i ustalenie prymitywu rysowania.



Rysunek 30 Kolejno od góry: połączenie VAO, IBO z buforami, ustalenie prymitywu, narysowanie modelu i wyczyszczenie bufora

Źródło 30 Opracowanie własne

Po połączeniu buforów z potokiem graficznym OpenGL, trafiają one do shadera, w którym odczytywane są za pomocą ustalonej wcześniej numeracji buforów, jak pokazane to zostało na poniższym zdjęciu.



Rysunek 31 Odczyt buforów w shaderze wierzchołków

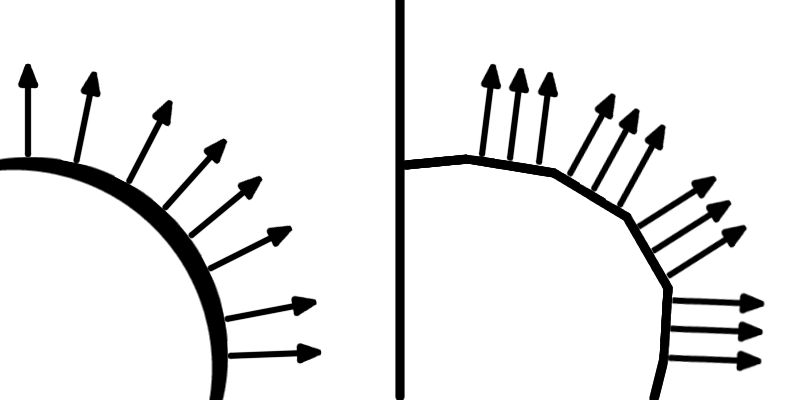
Źródło 31 Opracowanie własne

Dane te są dalej wykorzystywane do obliczeń związanych z pozycją wierzchołka w generowanym świecie, jak i później na ekranie komputera.

**Wektory normalne**

Aplikacja oferuje dwa sposoby definicji wektorów normalnych:

* Uśrednione – każdy wierzchołek ma przypisany wektor normalny, który nie jest prostopadły do żadnej z sąsiadujących powierzchni dla uzyskania efektu obłej powierzchni. W obrębie fragmentu żaden z wektorów normalnych nie jest równoległy do sąsiadującego ze względu na wykorzystanie mechanizmu interpolacji, jaki zapewnie potok graficzny OpenGL,
* Równoległe – każdy fragment posiada wierzchołek wiodący, który jest prostopadły do powierzchni i decyduje o zwrocie wszystkich wektorów normalnych w obrębie danego fragmentu. Wszystkie wektory normalne w obrębie danego fragmentu są równoległe do siebie i mechanizm interpolacji dla granicznych wektorów normalnych jest wyłączany specjalnie aby uzyskać efekt „kanciastej” bryły



Rysunek 32 Schemat wyznaczania wektorów normalnych, od lewej: Uśredniony i Równoległy

Źródło 32 Opracowanie własne

Z perspektywy shadera, przełączanie się pomiędzy trybami cieniowania nie jest trudne. Tak jak inne bufory, tak i wektory normlane przekazywane są do shadera wierzchołków w postaci zmiennej o podanej lokacji. Domyślnie, wartości pomiędzy poszczególnymi wartościami buforów poddawane są mechanizmowi interpolacji.

Interpolacja definiowana jest przez książkę „OpenGL Księga Eksperta” w poniżej zamieszczony sposób[[9]](#footnote-9):

*Interpolacja to termin wykorzystywany do opisania procesu określania wartości znajdujących się między dwoma znanymi punktami.*

Równanie obrazujące proces interpolacji wygląda następująco[[10]](#footnote-10):

Gdzie:

Punkt początkowy

Punkt końcowy

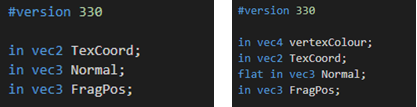
Wartość wynikowa pomiędzy punktami i

Argument, zmienna sterująca miejscem uśrednienia (bliżej do punktu A lub do punktu B) w dziedzinie wartości od 0 do 1

Proces interpolacji w shaderach OpenGL można zaaplikować do wartości w dowolnym kontekście, od wartości składowych kolorów, po uśrednione pozycje wektorów, tak jak to ma miejsce w przypadku uśrednionych wektorów normalnych.

W obu trybach wyznaczania wektorów normalnych ich pozycje przekazywane są do shaderów w ten sam sposób. Różnica polega na użyciu shadera z odpowiednią zmienną przyjmującą te wektory.

Dodając przedrostek „flat” przed typem przekazywanej zmiennej możliwe jest wyłączenie mechanizmu interpolacji, co zastosowałem w omawianej aplikacji.



Rysunek 33 Dodanie przedrostka „flat” do zmiennej blokuje dla niej interpolację

Źródło 33 Opracowanie własne

Wykorzystując blokadę interpolacji dla zmiennej w shaderze należy wyznaczyć wierzchołek wiodący, którego wartości będą naśladowane przez pozostałe wierzchołki znajdujące się w tym samym fragmencie. Można wybrać wierzchołek pierwszy, bądź ostatni w buforze wierzchołka. W omawianej aplikacji wierzchołkiem wiodącym jest pierwszy wierzchołek w buforze spośród wierzchołków danego fragmentu.

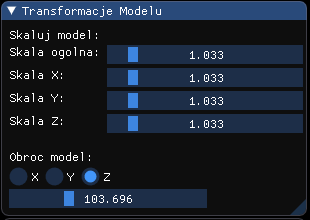


Rysunek 34 Ustawienie wierzchołka wiodącego

Źródło 34 Opracowanie własne

**Transformacje modelu**

Wśród funkcji, które oferuje omawiany program, można znaleźć również transformacje modelu. Skalowanie i obracanie to dostępne formy transformowania modelu.



Rysunek 35 Okno „Transformacje Modelu”

Źródło 35 Opracowanie własne

Na każdy model składa się zbiór wierzchołków, które go tworzą. Każdy z wierzchołków ma swoją pozycję w generowanym świecie. Obserwując operacje skalowania i obracania można odnieść wrażenie, że pozycje wierzchołków są modyfikowane. Czy jest to prawdą, zależy od perspektywy, z której na to patrzymy. Biorąc pod uwagę efekt finalny i perspektywę potoku graficznego, odpowiedź ta jest prawdziwa, ponieważ wraz z kolejnymi krokami potoku, obiekt może być przekształcany, a wraz z nim pozycje wierzchołków. Jednak z perspektywy aplikacji, pozycja wierzchołków jest wciąż taka sama. Dlatego definiując bufor wierzchołków mogłem określić jego użycie jako GL\_STATIC\_DRAW. Jest zapisywany tylko raz, w jednej postaci, jaką określiliśmy na początku programu, jeszcze przed uruchomieniem pętli głównej.

Wszelkie transformacje są obliczane „w biegu”, podczas pracy pętli głównej programu. Dzieje się to za pośrednictwem wymnażania pozycji każdego wierzchołka przez macierz przekształcenia. Równanie pozycji wierzchołka z uwzględnieniem tego przekształczenia zostało pokazane poniżej.

Skalowanie:

Gdzie:

– początkowe współrzędne wierzchołka

współczynnik krotności powiększenia

Obrót względem osi X:

Obrót względem osi Y:

Obrót względem osi Z:

Wspólna macierz obrotu[[11]](#footnote-11):

Gdzie:

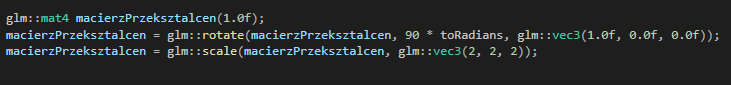
– początkowe współrzędne wierzchołka

kąt obrotu

Chcąc nałożyć na siebie poszczególne obroty, macieże obrotu są przez siebie przemnażane. Podobnie jest z dołączeniem skalowania, wystarczy pomnożyć macież skali przez dotychczas wygenerowaną macierz przekształcenia, aby nanieść skalę.

W zależności od docelowego efektu, mnożenie należy przeprowadzić w odpowiedniej kolejności. Na przykład, jeśli zależy mi na tym, aby obiekt został obrócony, a potem wyskalowany, to w takiej kolejności powinienem wykonać mnożenie.

Implementując macierz przekształceń, nie musiałem wprowadzać manualnie każdego elementu macierzy. W celu wygenerowania jej posłużyłem się odpowiednimi funkcjami zawartymi w bibliotece OpenGL Matematics (w skrócie: GLM).

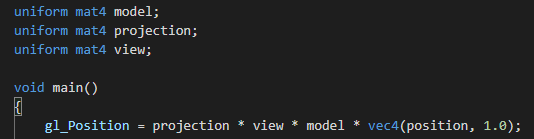


Rysunek 36 Przykład implementacji generowania macierzy przekształceń z użyciem biblioteki GLM

Źródło 36 Opracowanie własne

Na powyższym zdjęciu został pokazany moment (czytając od góry) utworzenia macierzy jednostkowej 4x4, a następnie przekształcenia jej tak, aby obracała obiekt o 90 stopni względem osi X i powiększyła go dwukrotnie w każdym kierunku.

Po otrzymaniu finalnej macierzy przekształceń, przekazałem ją za pomocą obiektu uniform do bufora. Bufor został przekazany do shadera wierzchołków, w którym odczytana macierz została pomnożona przez pozycję wierzchołka w generowanym świecie. Takie pomnożenie sprawia, że pozycja wierzchołka zostaje przekształcona według zamierzonych parametrów w macierzy przekształceń.



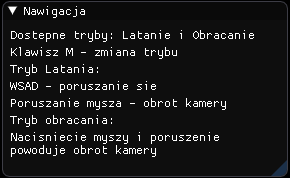
Rysunek 37 Fragment shadera wierzchołków przedstawiający wyliczenie pozycji wierzchołka w generowanym świecie

Źródło 37 Opracowanie własne

Na powyższym zdjęciu macierz przekształceń została nazwana „model” i jest przemnożona przez pozycję wierzchołka w świecie. Pozycja wierzchołka zawarta jest w trzyelementowym wektorze o nazwie „position”.

**Kamera**

Poprzez słowo „kamera” rozumiana jest pozycja, z punktu widzenia której widzimy świat. Program oferuje dwa rodzaje obsługi kamery, co wyjaśnione jest w oknie o nazwie „Nawigacja”.



Rysunek 38 Okno „Nawigacja” z instrukcją użytkowania kamery

Źródło 38 Opracowanie własne

Implementacja kamery

Jak już zostało napisane, pozycja kamery definiuje punkt, z perspektywy kórego widzimy świat. Jednak przekazując do shadera jedynie informacje o pozycji wierzchołków i ewentualnych transformacjach modelu, nie jesteśmy w stanie przekazać informacji o fakcie istnienia kamery. Tak samo, nie jesteśmy w stanie przekazać informacji o tym, że świat, który chcemy generować ma być widziany w ogóle z jakiejś perspektywy, a także o tym, które części świata mają być widziane na ekranie w danej chwili. W celu określenia tych informacji, należy przeprowadzić operację rzutowania generowanej sceny na finalny obraz.

W OpenGL istnieją dwa podstawowe rodzaje rzutowania: ortograficzne i perspektywiczne.

Rzutowanie ortograficzne[[12]](#footnote-12) przypomina nieco obrazki rysowane w komiksach. Charakteryzują się one brakiem zachowania perspektywy i proporcji wielkości obiektu do jego odległości od punktu widzenia. W takim rzutowaniu przyjmujemy, że kamera znajduje się nieskończenie daleko od obserwowanego świata i jednocześnie widzi tylko ten świat (To znaczy, że kraniec generowanego świata to jednocześnie kraniec obszaru „widzenia” kamery).

W moim programie korzystam z tego drugiego rodzaju rzutowania, czyli perspektywicznego. Przeciwnie do ortograficznego, rzutowanie to uwględnia zmniejszanie się wymiarów obiektów oraz zmienianiem się ich położenia, wraz z przemieszczaniem się punktu widzenia, tak jak ma to miejsce w rzeczywistym świecie[[13]](#footnote-13).

Oznacza to, że pozycja wierzchołków względem centrum generowanego świata jest inna, niż względem punktu widzenia. Dlatego należy ją zmienić, aby uzyskac pożądany efekt.

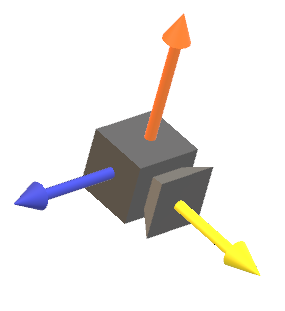
Zmiana ta polega na przemnożeniu początkowej pozycji wierzchołka przez macież perspektywy i macież widoku. Możenie należy wykonać po przemnożeniu początkowej pozycji wierzchołka przez macież przekształcenia.

Macierz widoku

Macierz widoku, jak sama nazwa wskazuje, przekształca pozycję wierzchołka w świecie, na pozycję względem punktu widzenia kamery.

Tworząc tą macierz, należy uwzględnić 3 atrybuty w postaci wektorów, jakie trzeba zdefiniować dla kamery:

* Pozycja – gdzie kamera jest umiejscowiona w świecie,
* Góra – wektor o zwrocie pionowym w górę. Jego zmiana informuje o przechyleniu się punktu widzenia w którąś ze stron,
* Punkt zainteresowania – jest to pozycja, na którą „patrzy” kamera. Odejmując tą pozycję od pozycji kamery w świecie, można uzyskać wektor kierunku, który definiujemy jako przód,
* Wektor boczny – można go uzyskać za pośrednictwem iloczynu wektorowego wektorów „Góry” i „Przodu”.



Rysunek 39 Kamera wraz z wektorami, które ją definują

Źródło 39 Opracowanie własne

Definicja wektora patrzenia w przód[[14]](#footnote-14):

Gdzie:

pozycja punktu zainteresowania

pozycja kamery w świecie

Definicja wektora bocznego:

Gdzie:

wektor patrzenia w przód

wektor patrzenia w górę

Rzutowanie pozycji wierzchołków w świecie na finalny ekran wymaga zarówno zmiany pozycji wierzchołków składających się na obserwowany model, jak i jego obrotu.

W tym celu należy stworzyć macierz, która będzie w stanie dokonać takego przekształcenia. Na macierz tą składają się wspomniana już w tej pracy macierz obrotu oraz macierz przesunięcia[[15]](#footnote-15).

Gdzie:

wektor patrzenia w przód

wektor uzyskany na podstawie iloczynu wektorowego:

wektor boczny

pozycja kamery w świecie

macież widoku

Implementując mój program, nie musiałem obliczać każdego elementu macierzy, tylko posłużyłem się funkcją zawartą w bibliotece GLM o uproszczonej sygnaturze[[16]](#footnote-16):

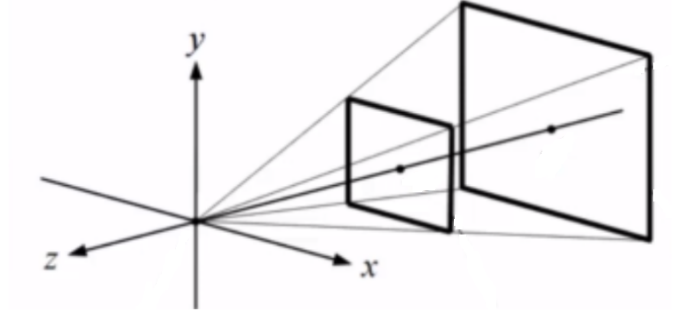
*glm::lookAt(pozycja kamery, punkt zainteresowania, wektor patrzenia w górę)*

Macierz perspektywy

Kolejnym rodzajem macierzy, jaki potrzebny jest do utworzenia widoku z perspektywy umiejscowionej w trójwymiarowym świecie, jest macierz perspektywy.

Główną zaletą zastosowania przekształcenia z użyciem tej macierzy jest uzyskanie efektu zmniejszania się rozmiarów obiektów na ekranie wraz ze wzrastającą odległością od punktu widzenia, co tworzy wrażenie głębi.

Do utworzenia macierzy perspektywy, potrzebne będą współrzędne granic przestrzeni widoku. Przestrzeń ta zawiera wszystkie wierzchołki, które maja zostać wyświetlone na ekranie w danym momencie, czyli można powiedzieć, że są w zasięgu widzenia kamery.



Rysunek 40 Przestrzeń widoku

Źródło 40 Opracowanie własne

Tworzenie macierzy perspektywy[[17]](#footnote-17):

Gdzie:

pozycja krawędzi bliskiej przestrzeni widoku

pozycja krawędzi dalekiej przestrzeni widoku

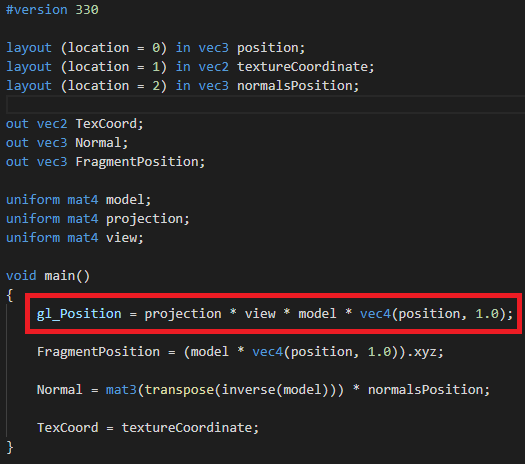
pozycja krawędzi lewej przestrzeni widoku

pozycja krawędzi prawej przestrzeni widoku

Macierz tą przemnażamy przez pozycję wierzchołka i inne macierze przekształceń.

Biblioteka GLM oferuje funkcję generowania macierzy perspektywy, której uproszczona sygnatura wygląda w ten sposób[[18]](#footnote-18):

*glm::perspective(kąt widzenia z perspektywy kamery, proporcje okna, bliska granica przestrzeni widoku, daleka granica przestrzeni widoku)*



Rysunek 41 Shader wierzchołków, w którym wykorzystano macierze przekształceń dla pozycji wierzchołka

Źródło 41 Opracowanie własne

Tryb obracania kamerą

Pierwszym i podstawowym rodzajem obsługi kamery jest „Tryb obracania”, w którym kamera otacza model, a zarazem i centrum układu współrzędnych świata. Ustawiona jest w określonym promieniu w stosunku do środka układu współrzędnych i zawsze zwrócona jest również w kierunku środka. Nawigacja opiera się na kliknięciu kursorem na dowolne miejsce ekranu i przesunięcie myszy w kierunku, w którym chcielibyśmy obrócić obserwowany model.

Po naciśnięciu lewego przycisku myszy, zostaje odmierzona długość, jaką przebył kursor w dwóch wymiarach w stosunku do wymiarów okna aplikacji.

Na podstawie zmierzonych odległości, wyliczany jest kąt i kierunek w którym kamera zostanie przemieszczona.

Dane potrzebne do algorytmu obliczania pozycji kamery:

pozycja kamery

pozycja środka modelu

zmiana pozycji kursora w poziomie

zmiana pozycji kursora w pionie

szerokość okna aplikacji

wysokość okna aplikacji

Algorytm wyliczania pozycji kamery:

1. Obliczenie współczynników proporcjonalności obrotu w stosunku do szerokośći i wysokości okna aplikacji. Zakładam, że przesunięcie kursora wzdłuż całego okna aplikacji poziomo spowoduje obrót bryły o 360 stopni, natomiast pionowo o 180 stopni.
2. Wyliczenie poziomego i pionowego kąta obrotu bryły

Gdzie:

kąt obrotu w poziomie

kąt obrotu w pionie

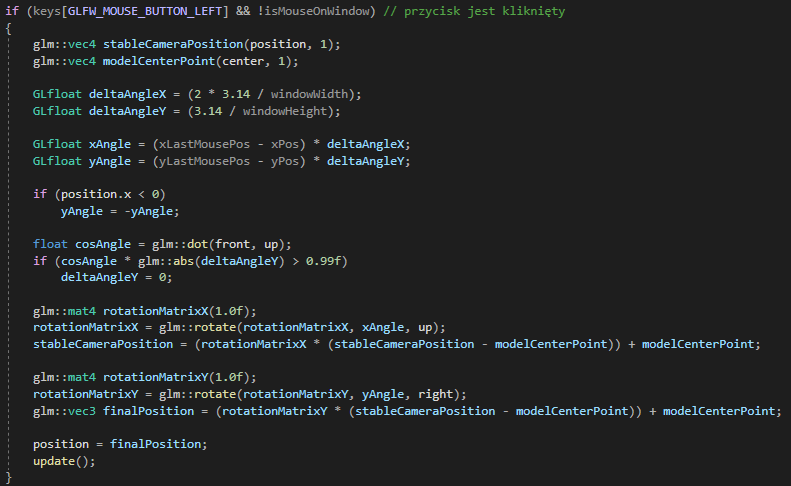
1. Utworzenie macierzy obrotu w poziomie

Wyliczenie pozycji kamery na podstawie poziomego obrotu:

1. Utworzenie macierzy obrotu w pionie

Wyliczenie pozycji kamery na podstawie pionowego obrotu:

Macierze obrotu w takcie pracy programu wraz z danymi będą zmieniały swoje postacie, ponieważ tutaj widoczne są ich postacie dla wektora pionowego równego z osią Y i poziomego równego z osią Z. Jednak wraz ze zmienianiem się pozycji położenia i zwrotu kamery, wektor pionowy i wektor poziomy, względem którego robiony jest obrót będą wskazywać inne kierunki. Macież tą tworzy funkcja rotate() z biblioteki GLM.



Rysunek 42 Implementacja algorytmu kamery otaczającej bryłę

Źródło 42 Opracowanie własne

Po wyliczeniu nowej pozycji kamery po przesunięciu, wystarczy użyć funkcji *lookAt()* z biblioteki GLM, która na podstawie pozycji kamery, pozycji środka modelu i wektora pionowego wyliczy macierz widoku, która zostanie użyta w shaderze wierzchołków.



Rysunek 43 Przykład użycia funkcji lookAt() dla kamery otaczającej

Źródło 43 Opracowanie własne

Tryb latania kamerą

Aplikacja oferuje też drugi tryb pracy kamery, czyli tryb latania. W tym trybie użytkownik może zmieniać kierunek zwrotu kamery ruszając kursorem oraz poruszać nią za pomocą klawiszy WSAD w przestrzeni generowanego świata.

Przemieszczanie się kamery polega na dodaniu odpowiedniego wektora do aktualnej jej pozycji. Ze względu na fakt, że kamerę definują trzy wektory (patrzenia w przód, górę i w prawo), to można wykorzystać je do uzyskania punktu odniesienia i zdefiniowania stron, w którą może poruszyć się kamera.

Praca programu bazuje na pętli głównej, w której wykonywane są funkcje odpowiedzialne za ustawienie modeli rysowanych na ekranie oraz przekazywanie buforów do shaderów. Każdy obrót pętli trwa przez pewien czas, który można zmierzyć. Ten czas będę określał symbolem . Potrzebny jest on do wyliczenia drogi, jaką ma pokonać kamera podczas jednego obrotu pętli głównej, gdy przycisk jest wciśnięty.

Kolejnym parametrem jest prędkość, która zależna jest od preferencji i wyrażona jest w jednostce odległości na układzie współrzędnych generowanego świata na pojedynczy obrót pętli głównej programu. Jej symbolem będzie .

Wyznaczenie drogi pokonywanej przez kamerę w trakcie jednego obrotu pętli głównej:

Zmiana pozycji kamery w osi prawo-lewo:

Zmiana pozycji kamery w osi góra-dół:

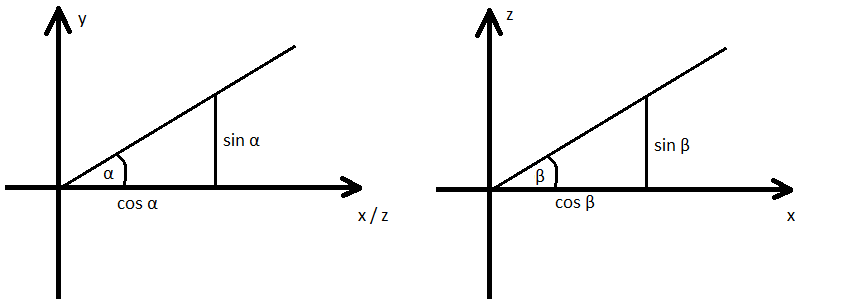
Gdzie:

aktualna pozycja kamery

wektor patrzenia w prawo (znormaliowany)

wektor patrzenia w przód (znormaliowany)

Obrót kamery polega na obliczeniu pozycji wektorów patrzenia w przód i w prawo. Parametrem, który decyduje o wyniku obliczeń jest zmiana położenia kursora w pionie i w poziomie, co rzutuje na zmianę kąta nachylenia kamery w tych samych obu kierunkach. Wartości zmiany położenia w poszczególnych osiach można obliczyć na podstawie wzorów trygonometrycznych.



Rysunek 44 Obliczenie wartości zmian położenia w kierunku (od lewej): pionowym, poziomym

Źródło 44 Opracowanie własne

Biorąc pod uwagę powyższą grafikę, można wyprowadzić wzory:

Gdzie:

wektor patrzenia w przód

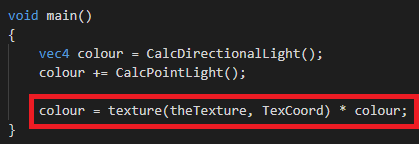
wektor patrzenia w prawo

wektor patrzenia w górę

**Tekstury**

Omawiana aplikacja oferuje też funkcję nałożenia tekstury na wyświetlany obiekt. Zrealizowane to zostało z użyciem tekseli, czyli współrzędnych na obrazie tekstury. Każdy z wierzchołków przypisany ma teksel wyrażony w potaci współrzędnych XY. W shaderze pobierana jest wartość koloru dla teksela o podanych współrzędnych z tekstury i wartość ta mnożona jest przez finalny kolor obliczony w poprzednim podrozdziale.

Do zapisu pliku tekstury w shaderze wykorzystałem obiekt typu sampler2D oferowany w standardowej bibliotece GLSL. Odczyt koloru teksela o zadanych współrzędnych wykonałem za pomocą funkcji texture().



Rysunek 45 Fragment shadera fragmentów odpowiedzialny za nałożenie tekstury na fragment

Źródło 45 Opracowanie własne

# IMPLEMENTACJA APLIKACJI W WYBRANYM ŚRODOWISKU PROGRAMOWYM

Omawiana aplikacja została utworzona, przetestowana i skompilowana na poniżej przedstawionym środowisku sprzętowym:

* Procesor: Intel Core i5-2400 CPU @ 3.10 Ghz
* Pamięć RAM: 8 GB
* Karta graficzna: MSI Geforce GTX 1050
* Pamięć VRAM: 2 GB

Oraz na poniżej przedstawionym środowisku programowym:

* System operacyjny: Windows 10 Education, 64 bitowy
* Środowisko programistyczne: Microsoft Visual Studio 2019

**Wykorzystane biblioteki**

W celu poprawnego skorzystania z dobroci, które oferuje OpenGL, dobrze jest wykorzystać przydatne narzędzia, które zoptymalizują proces tworzenia aplikacji graficznej.

Pierwszym z nich jest biblioteka OpenGL Extension Wrangler Library (**GLEW**). Pomaga ona w wyszukiwaniu rozszerzeń OpenGL w środowisku, w którym uruchamiana jest aplikacja. Potrafi znaleźć odniesienia do implementacji OpenGL w lokalnych sterownikach karty graficznej. Umożliwia to możliwość uruchomienia aplikacji na różnych rodzinach procesorów graficznych[[19]](#footnote-19).

Drugą biblioteką wykorzystywaną w projekcie jest **GLFW**, która zapewnia wsparcie dla API systemu Windows w tworzonej aplikacji. Ułatwia pracę z oknami i umożliwia obsługę urządzeń wejściowych[[20]](#footnote-20).

Trzecią biblioteką jest OpenGL Matematics (**GLM**), która zawiera zbiór funkcji pomagających w obliczeniach wykonywanych na generowanych modelach. Może na przykład wyliczyć macierze widoku, perspektywy, czy przekształceń bryły[[21]](#footnote-21).

Ostatnią użytą biblioteką jest **ImGui**. Umożliwia ona generowanie okien tworzących graficzny interfejs użytkownika w celu łatwego dostępu do dostosowywania parametrów programu[[22]](#footnote-22).

Poniżej przedstawione zostały wyniki uzyskane w utworzonej aplikacji.

## Scena

Aplikacja oferuje wybór pomiędzy czterema rodzajami modeli.



Rysunek 46 Przedstawienie dostępnych brył

Źródło 46 Opracowanie własne

Oto parametry, przy użyciu których wygenerowane zostały powyższe bryły:

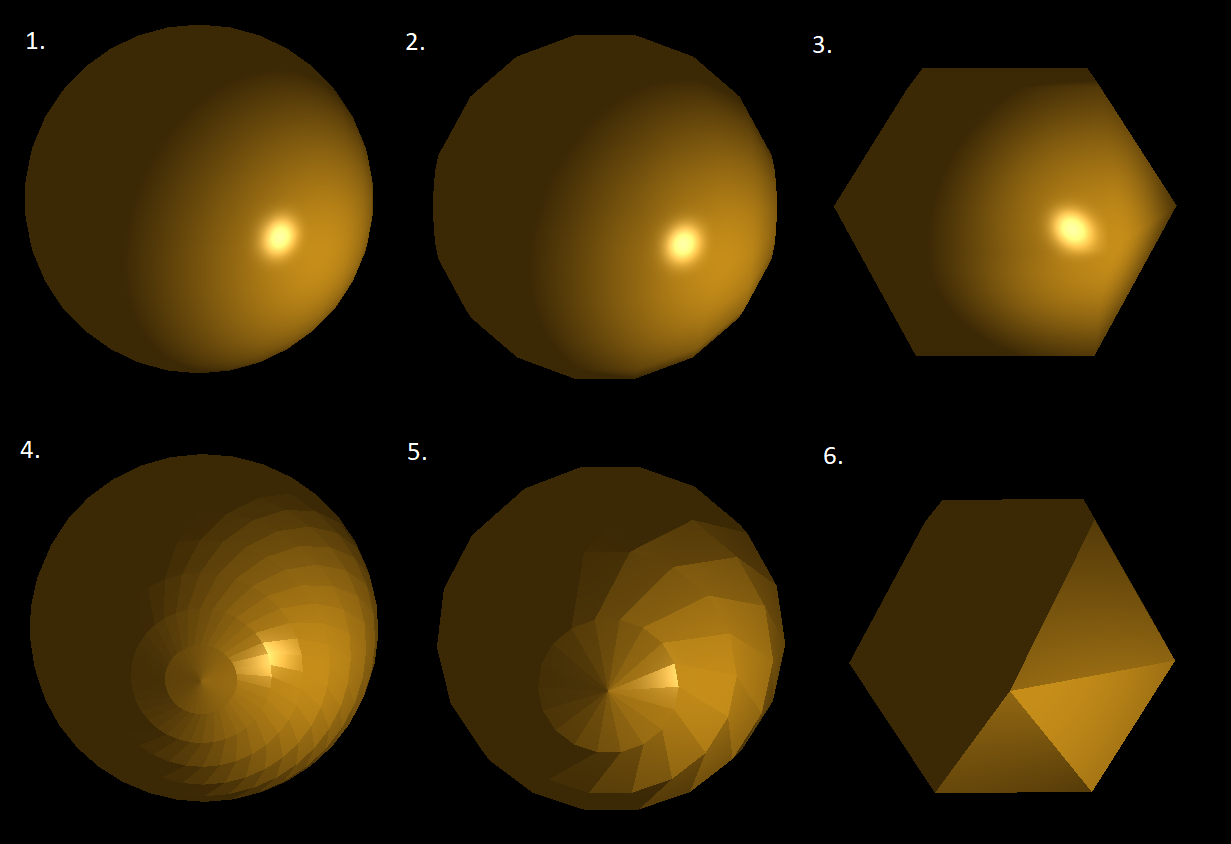
Tabela 2 Parametry brył wygenerowanych na Rysunku 46

Źródło: Opracowanie własne

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nazwa** | Sfera | Prostopadłościan | Ostrosłup | Walec |
| **Materiał** | Mosiądz | Mosiądz | Mosiądz | Mosiądz |
| **Tekstura** | Brak | Brak | Brak | Brak |
| **Wektory normalne** | Uśrednione | Równoległe | Równoległe | Uśrednione |

Zdjęcie zostało zrobione z użyciem tego samego rodzaju źródła światła w tej samej pozycji, natężeniu i barwie. Natężenie wynosi 30% intensywności składowej otoczenia, oraz po 70% dla składowych rozproszonej i odbitej. Obecne jest również dodatkowe światło otaczające o intensywności 10%. Współczynnik odpowiedzialny za zaciemnianie światła wraz z odległością jest stały i ma wartość 100%.

Obserwując model sfery przy ustawieniu wektorów normalnych na równoległe, można zaobserwować, że jest ona zbudowana ze kończonej liczby wierzchołków połączonych liniami prostymi. Aplikacja oferuje możliwość zmiany ilości tych wierzchołków w pionie i w poziomie (wybieramy ich ilość na jednym pasie w tych obu kierunkach).



Rysunek 47 Przedstawienie różnych ilości wierzchołków dla bryły

Źródło 47 Opracowanie własne

Poniższa tabela przedstawia ustawione ilości wierzchołków w powyżej pokazanych sferach:

Tabela 3 Ilości wierzchołków w bryłach wygenerowanych na rysunku 47

Źródło: Opracowanie własne

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Numer bryły** | 1. | 2. | 3. | 4. | 5. | 6. |
| **Liczba wierzchołków pionowo** | 18 | 9 | 3 | 18 | 9 | 3 |
| **Liczba wierzchołków poziomo** | 36 | 18 | 6 | 36 | 18 | 6 |
| **Wektory normalne** | U | U | U | R | R | R |

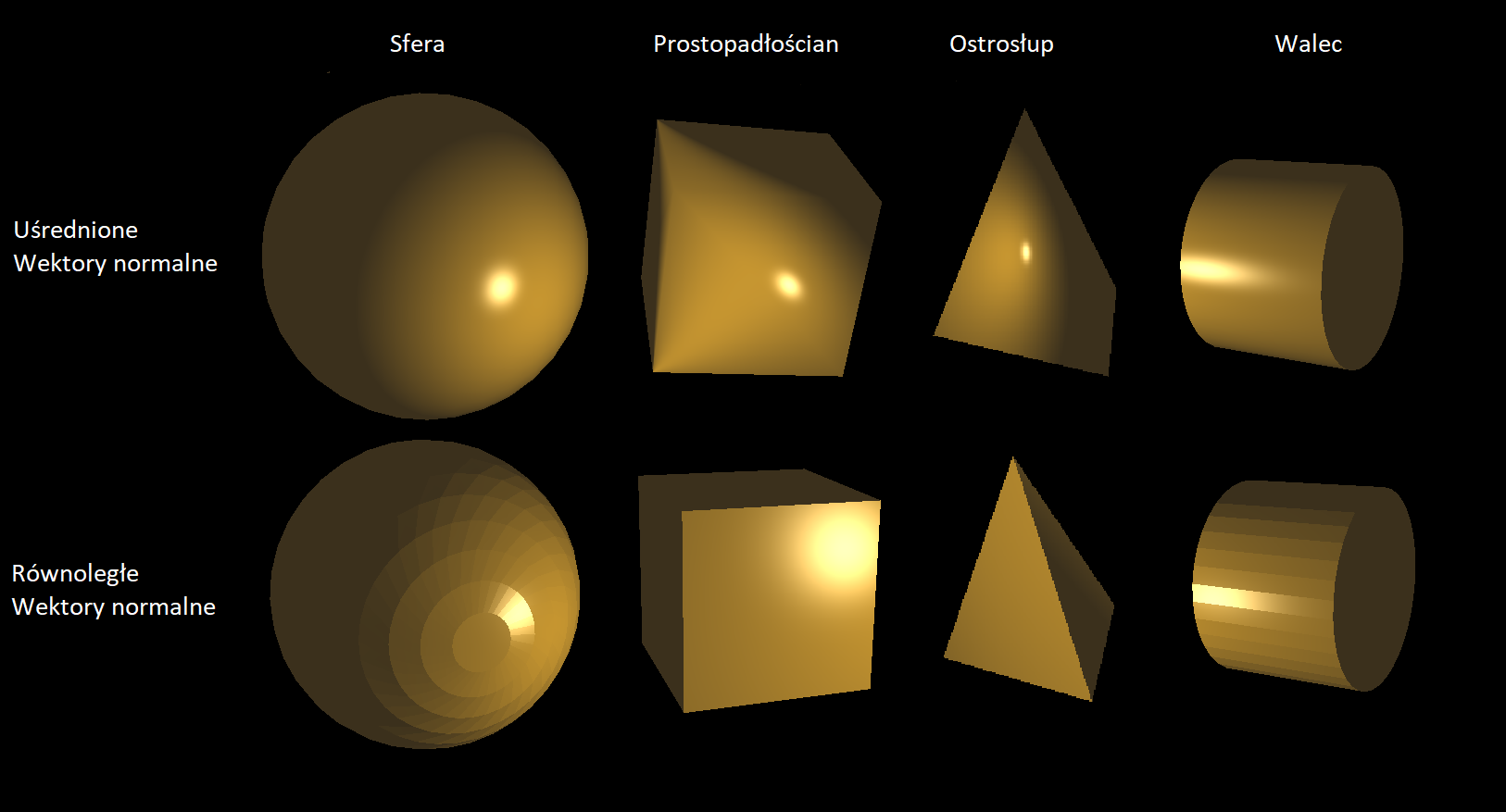
Gdzie:

R – równoległe

U – uśrednione

**Wektory normalne**

Poniżej wygenerowane zostały modele w dwóch trybach generowania wektorów normalnych powierzchni modelu.



Rysunek 48 Prezentacja możliwości ustawień wektorów normalnych

Źródło 48 Opracowanie własne

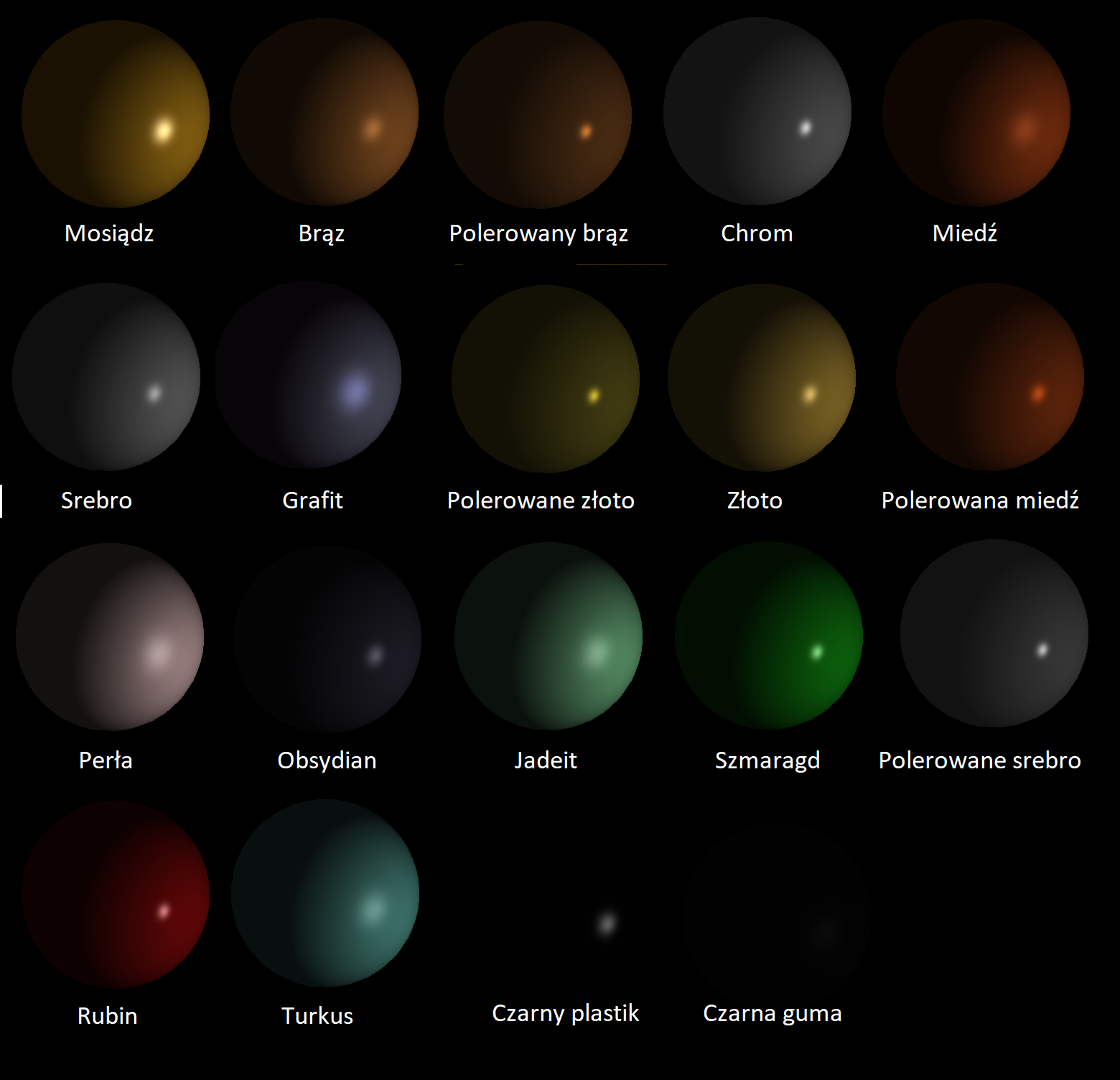
Można zaobserwować na powyższej grafice, że efekty zależą od odpowiedniego dopasowania bryły i sposobu generowania wektorów. Nie ma jednego podejścia, które sprawdza się w przypadku brył zawierających obłe elementy i tych kanciastych.

Wektory generowane równolegle do siebie znajdują zastosowanie w bryłach o ostrych krawędziach, takich jak prostopadłościan i ostrosłup. Całość wygląda wtedy realistycznie. Jednak psują efekt obłości w przypadku sfery i walca. Sprawiają, że widoczne są krawędzie prymitywów.

Przeciwnie sytuacja wygląda w przypadku wektorów normalnych uśrednianych. Te sprawdzają się na sferze i walcu, jednak na prostopadłościanie i ostrosłupie powodują wrażenie zniekształcania bryły. Wynika to z faktu, że po uśrednieniu wektory normalne wcale nie są prostopadłe do powierzchni i oświetlenie padające na nią jest traktowane jakby była ona obła. Nie jest to prawda, więc efekt wychodzi sztuczny.

**Materiały**

Aplikacja oferuje szereg materiałów do wyboru. Wybór materiału wiąże się ze zmianą wrażenia odczucia padającego nań światła. Każdy materiał ma przypisane współczynniki pozwalające odwzorować rzeczywistą swoją postać. Poniżej przedstawione zostały wyniki implementacji różnych materiałów.

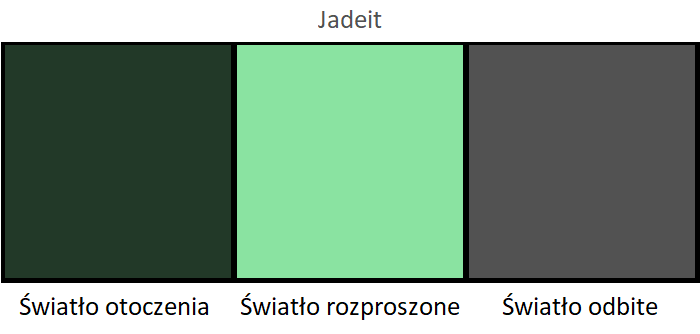


Rysunek 49 Dostępne materiały bez nałożonych tekstur

Źródło 49 Opracowanie własne

Jak można spostrzec na powyższej grafice, każdy z materiałów odróżnia się wielkością „plamki” światła odbitego, co jest bezpośrednim skutkiem zmiany wartości współczynnika odbicia światła w materiale.

Parametrem materiału jest też jego barwa, co rzuca się w oczy na powyższym porównaniu. Najciekawsze efekty daje zastosowanie innych barw dla poszczególnych składowych oświetlenia. Dobrym przykładem jest jadeit, w którym ten ciekawy efekt pokazany na grafice został osiągnięty dzięki zastosowaniu poniżej pokazanych kolorów.

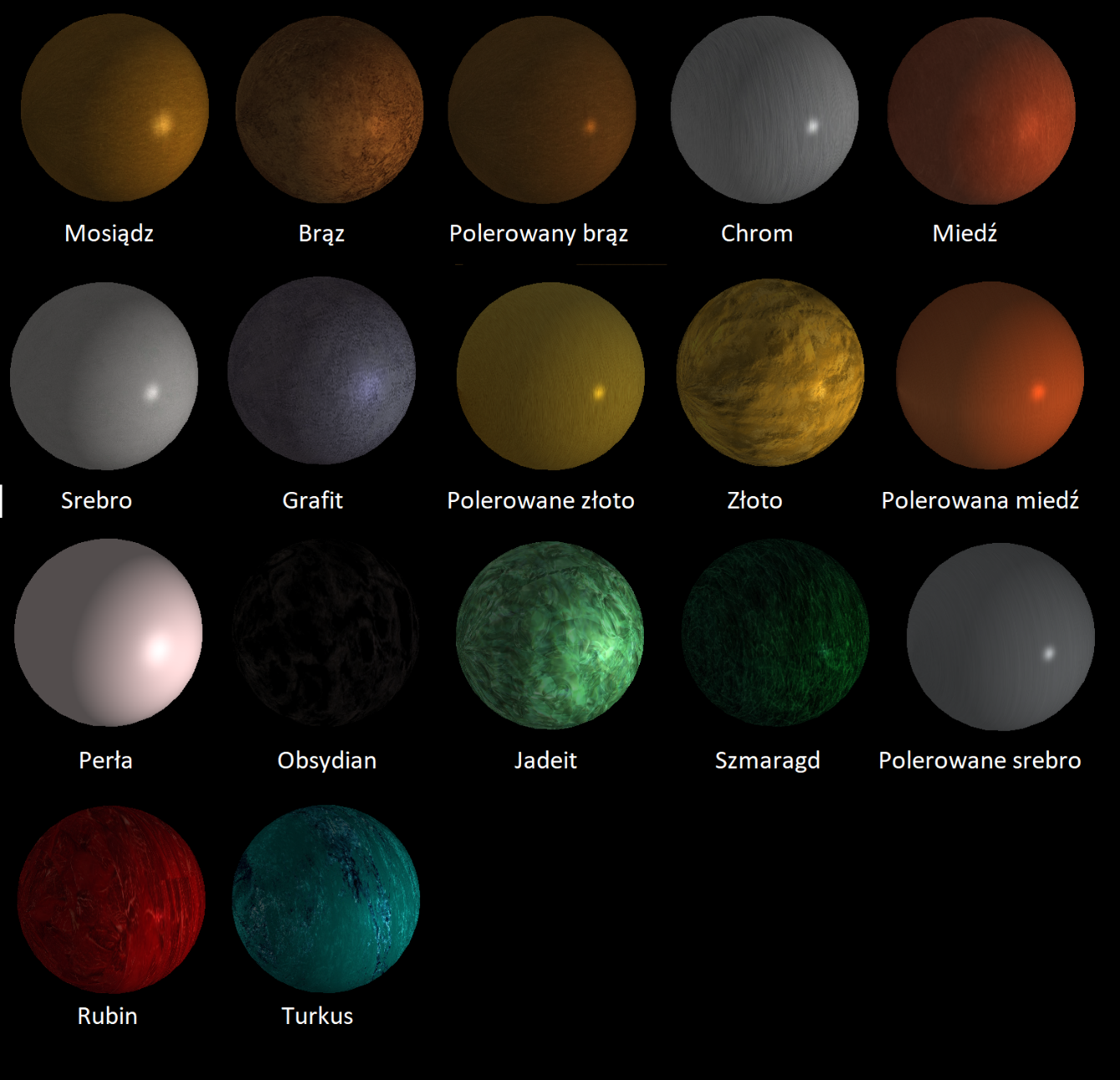


Rysunek 50 Przykład kombinacji kolorów dla różnych składowych światła. W tym przypadku Jadeit.

Źródło 50 Opracowanie własne

**Tekstury**

Funkcjonalność nakładania tekstury na wyświetlany obiekt zwiększa poczucie realizmu bryły. Takie przynajmniej były zamiary. Ciekawie wypada połączenie materiału z teksturą, ponieważ jego barwa często podobna jest do barwy tekstury. Podbija to nieco intensywność danego koloru. W niektórych przypadkach sprawia to, że bryła może sprawiać wrażenie sztucznej, jednak często osiągnięty efekt jest jeszcze bardziej realistyczny. Tak jest na przykład w przypadku rubinu, lub turkusu, których wzór posiada czarne elementy. Czerń w przestrzeni RGB jest po prostu zerami, lub bardzo małą wartością każdej składowej, dlatego nawet po przemnożeniu tego koloru przez jasną barwę światła, pozostaje ona wciąż niemal kompletnie czarna. Powoduje to efekt, w którym jasne elementy tekstury zostają rozjaśnione, natomiast czarne pozostają czarne, co dodaje realizmu poprzez niepowtarzalność wzoru.

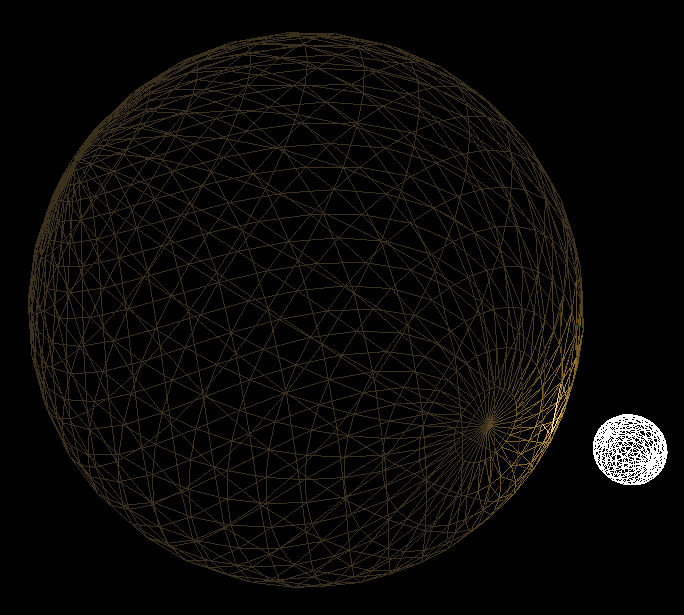


Rysunek 51 Dostępne materiały w komplecie z teksturami (Czarny plastik i czarna guma nie ujęte, ze względu na brak widoczności przy nałożonej teksturze)

Źródło 51 Opracowanie własne

**Siatka wielokątów**

Efekt siatki wielokątów potrafi zadziwić. Można zobaczyć wtedy imitację stelaża bryły, co tak naprawdę jest jedynie konturami prymitywów. Efekt jest więcej niż zadowalający.



Rysunek 52 Przykład wykorzystania trybu widzenia siatki wielokątów. W tym przypadku wykorzystany materiał to mosiądz. Biała kula imituje źródło światła.

Źródło 52 Opracowanie własne

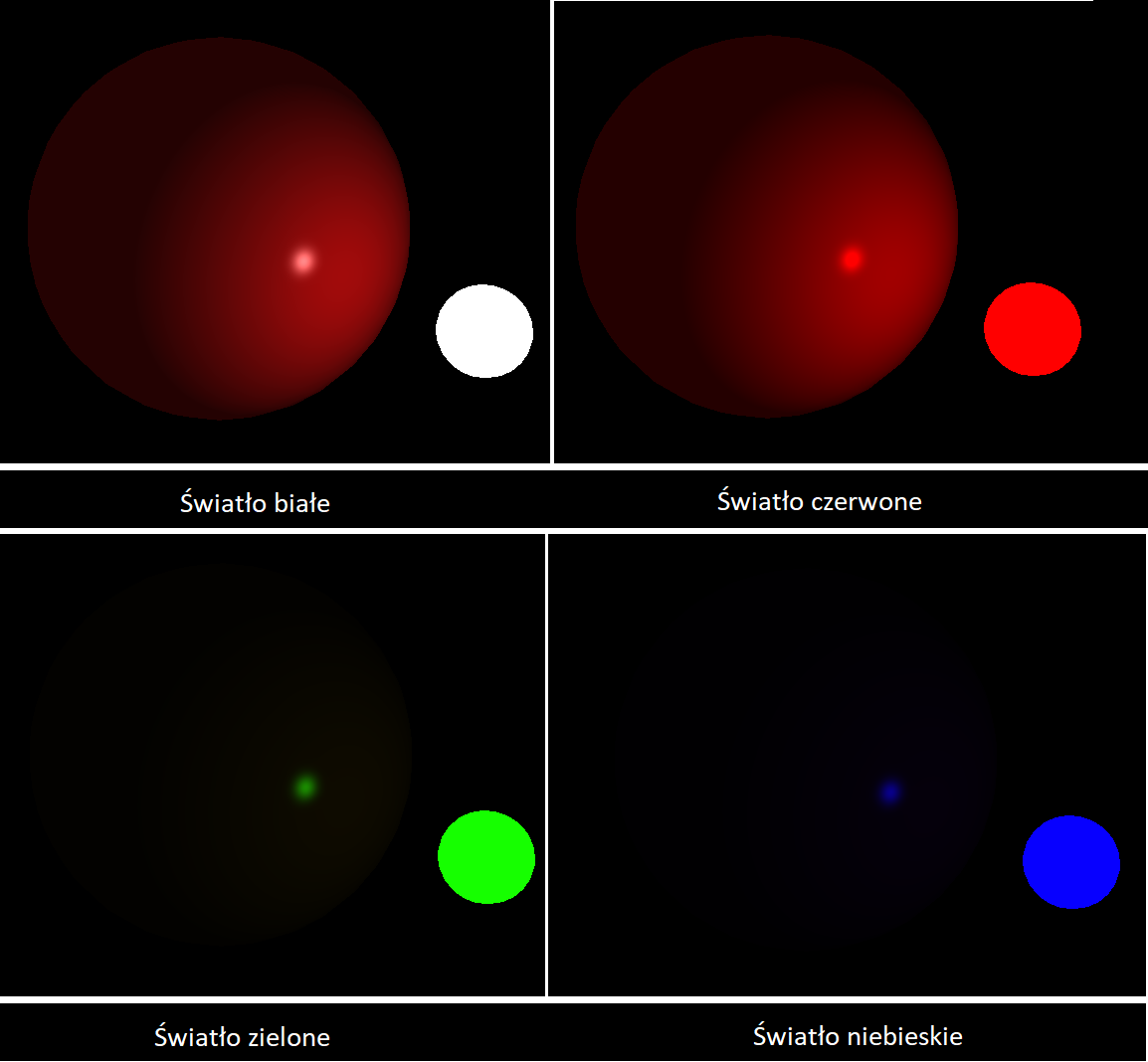
## Źródło światła

**Barwa światła**

Pierwszym dostosowywalnym elementem źródła światła jest jego barwa. Domyślnie ustawiona jest na biel, aby ukazać prawdziwe oblicze wybranego materiału, bądź jego kombinacji z teksturą. Jednak jej zmiana może spowodować ciekawe efekty. Dobry przykład został pokazany na zdjęciu poniżej. Przedstawiona bryła ma na sobie materiał nazwany rubinem, którego kolory to różne odcienie czerwieni. Przy świetle białym wygląda bardzo ładnie, podobnie przy czerwonym. W zasadzie jedyna różnica to lekkie zaczerwienienie oświetlenia rozproszonego i zmiana koloru oświetlenia odbitego na czerowny z białego.

Natomiast gdy zaświecone zostało światło o barwie zielonej, czerwień rubinu zmieniła się w niemal czerń. Stało się tak za sprawą przemnożenia bardzo małej wartości składowej R dla barwy źródła światła przez dużą wartość składowej R materiału. Spowodowało to wyraźną redukcję wartości finalnej składowej R, którą widać na ekranie. Podobnie sprawa przebiegła ze składową G. Co z tego, że źródło światła miało bardzo wysoką wartość składowej oznaczającej zieleń, skoro ta sama składowa dla oświetlanego materiału wynosiła niemal zero? Wynik po przemnożeniu pozostaje nadal bliski zeru, co oznacza, że wynikowy kolor to prawie czerń.

Ten sam proces przebiega dla przypadku z barwą źródła światła niebieską.

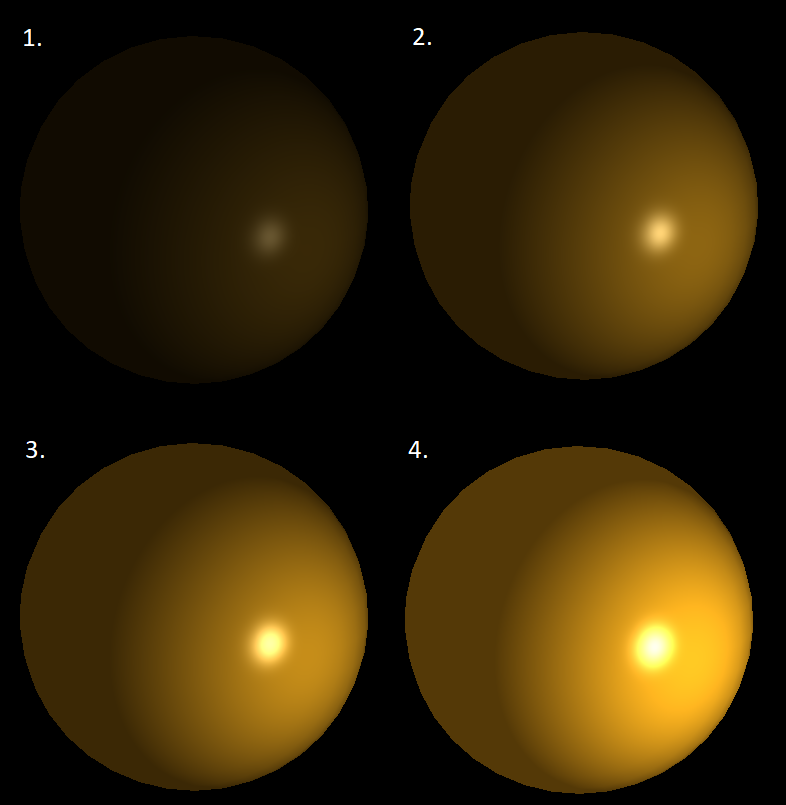
****

Rysunek 53 Prezentacja efektu zmiany barwy źródła światła i jego wpływu na oświetlaną powierzchnię. Na przykładzie materiału imitującego rubin.

Źródło 53 Opracowanie własne

**Intensywność źródła światła i jego składowych**

W aplikacji możliwe jest również sterowanie intensywnością oświetlenia. Podstawowe sterowanie odbywa się poprzez jednoczesną zmianę wartości intensywności wszystkich składowych oświetlenia Phonga.



Rysunek 54 Prezentacja różnych wartości intensywności oświetlenia

Źródło 54 Opracowanie własne

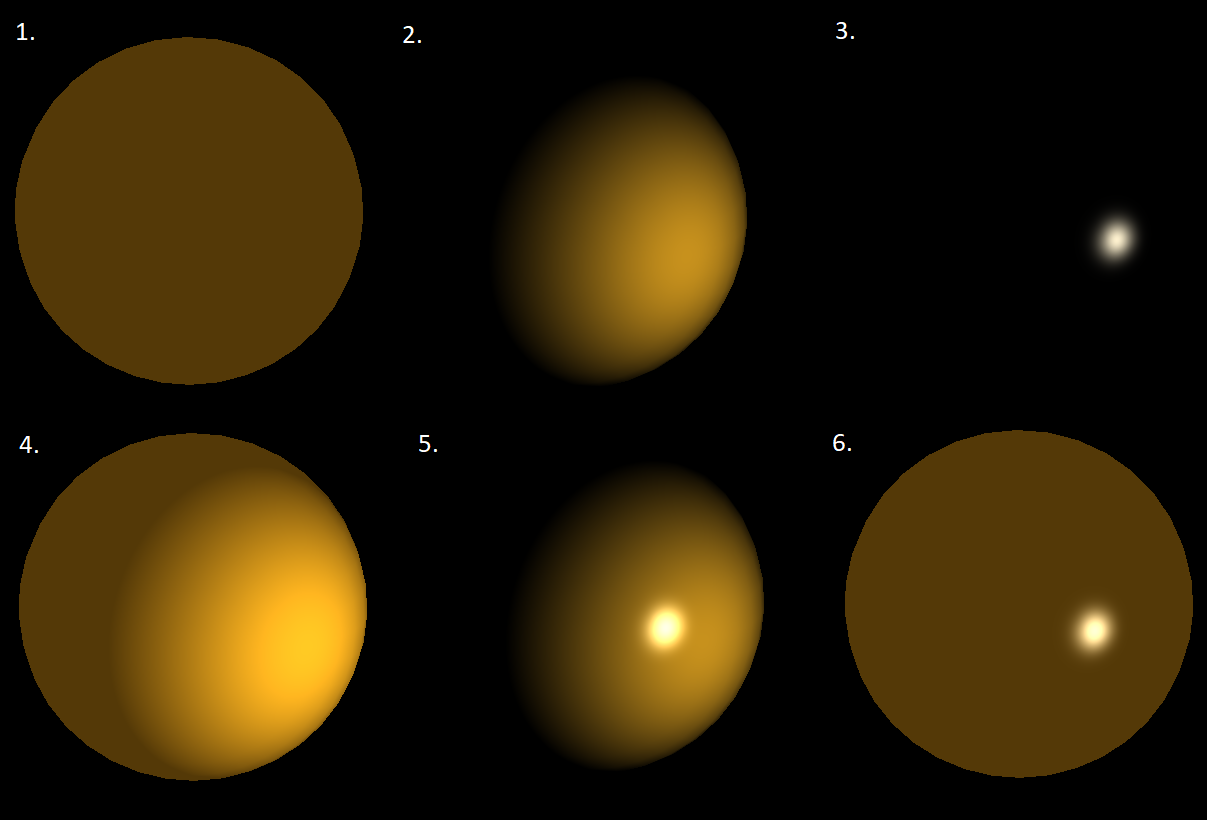
Poniższa tabela przedstawia wartości, jakie zostały użyte do wygenerowania brył pokazanych na powyższej grafice:

Tabela 4 Wartości współczynników intensywności światła brył wygenerowanych na Rysunku 54

Źródło: Opracowanie własne

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Numer bryły** | 1. | 2. | 3. | 4. |
| **Intensywność oświetlenia** | 20% | 50% | 70% | 100% |

Możliwe jest też dostosowanie wartości intensywności pojedynczych składowych oświetlenia Phonga. Dzięki temu można obejrzeć jak wyglądają one osobno, bez nakładania się ich na siebie. Przykłady zostały podane na obrazku poniżej.



Rysunek 55 Prezentacja składowych oświetlenia Phonga niezależnie

Źródło 55 Opracowanie własne

Poniższa tabela przedstawia wartości składowych oświetlenia Phonga dla powyżej przedstawionych sfer:

Tabela 5 Wartości współczynników światła dla brył z Rysunku 55

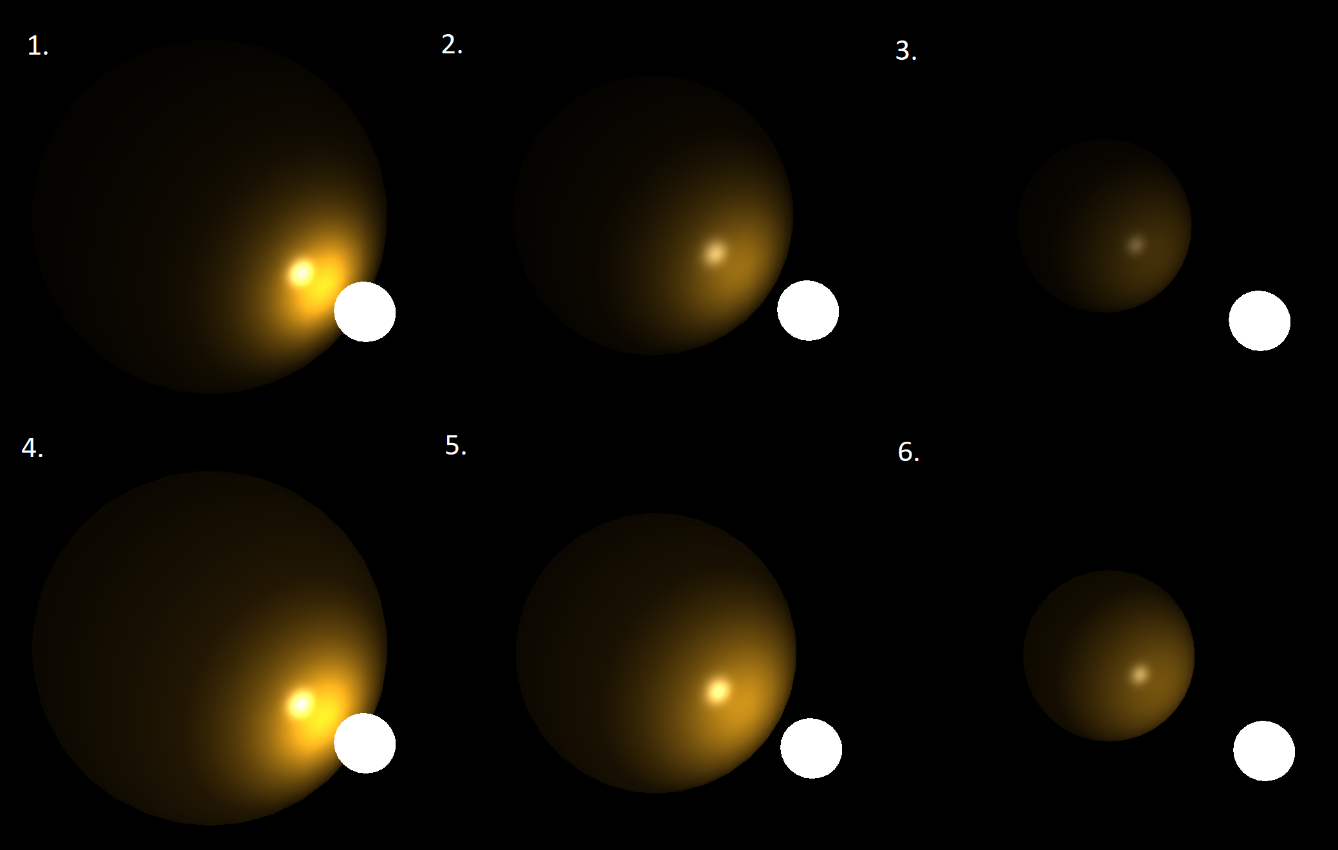
Źródło: Opracowanie własne

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Nr bryły** | 1. | 2. | 3. | 4. | 5. | 6. |
| **Światło otoczenia** | 100% | 0% | 0% | 100% | 0% | 100% |
| **Światło rozproszone** | 0% | 100% | 0% | 100% | 100% | 0% |
| **Światło odbite** | 0% | 0% | 100% | 0% | 100% | 100% |

Wartym spostrzeżenia jest tutaj fakt, że sama obecność wszystkich trzech składowych oświetlenia nie wystarczy do realistycznego odtworzenia wrażenia oglądania rzeczywistej sfery. Sekret osiągnięcia tego tkwi w odpowiednim doborze ich intensywności.

**Współczynniki tłumienia**

Kolejnym ważnym czynnikiem wpływającym na realistyczny odbiór obserwowanej bryły jest wrażenie tłumienia światła wraz ze wzrastającą odległością. Efekt ten można kontrolować poprzez zmianę wartości współczynników funkcji kwadratowej obecnej we wzorze na współczynnik tłumienia. Poniżej przedstawione zostały wyniki eksperymentów z różnymi wartościami wspomnianych współczynników w zależności od odległości sfery od źródła światła.



Rysunek 56 Prezentacja zmian współczynników tłumienia

Źródło 56 Opracowanie własne

Poniższa tabela przedstawia wartości współczynników, które pozoliły uzyskać powyższy efekt:

Tabela 6 Wartości współczynników dla brył z Rysunku 56

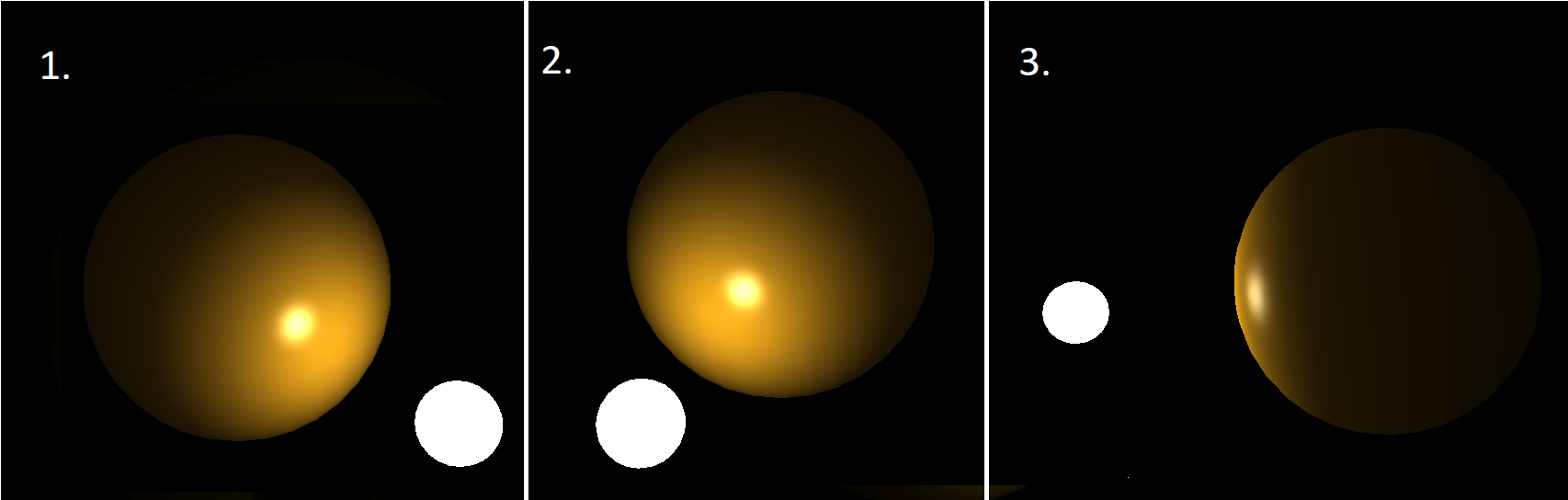
Źródło: Opracowanie własne

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Nr bryły** | 1. | 2. | 3. | 4. | 5. | 6. |
| **Współczynnik Kwadratowy** | 0,4 | 0,4 | 0,4 | 0,1 | 0,1 | 0,1 |
| **Współczynnik Liniowy** | 0,1 | 0,1 | 0,1 | 0,5 | 0,5 | 0,5 |
| **Współczynnik**  **Stały** | 0,3 | 0,3 | 0,3 | 0,3 | 0,3 | 0,3 |
| **Wielkość bryły** | 1x | 0,8x | 0,5x | 1x | 0,8x | 0,5x |

Można spostrzec, że podniesienie wartości współczynnika kwadratowego wpływa na szybsze zaciemnianie się bryły wraz ze zwiększaniem się odległości do bryły. Co ciekawe, bryła 1. i 4. „startowały” na podobnym poziomie podświetlenia, a jednak ta z większym współczynnikiem kwadratowym zaciemniła się szybciej. Oznacza to, że ma on kluczowe znaczenie w zmniejszaniu intensywności śwatła wraz z oddalaniem się jego źródła.

**Zmiana pozycji źródła**

Zmiana pozycji źródła światła odbywa się w programie bardzo prostą metodą. Polega ona na zmianie prędkości poruszającego się punktu w przestrzeni, który zdaje się emitować światło. Można wykorzystać to do przesunięcia go wokół bryły i zatrzymania w interesującym miejscu.



Rysunek 57 Prezentacja zmiany pozycji źródła światła

Źródło 57 Opracowanie własne

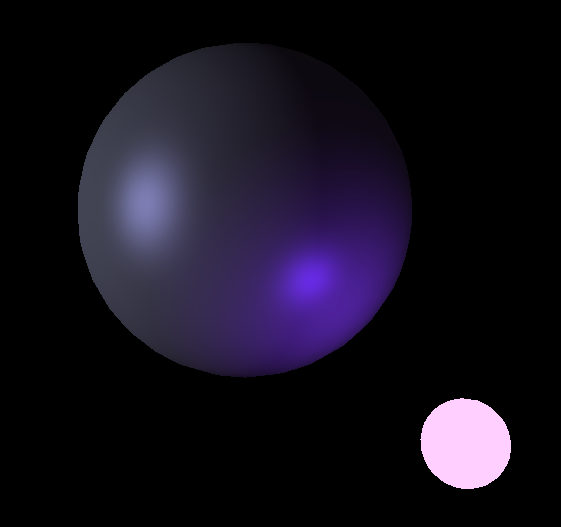
**Dodatkowe światło kierunkowe**

Program oferuje też opcję włączenia dodatkowego światła kierunkowego dla lepszego podświetlenia bryły. Jest to tylko dodatek pomocniczy w analizie światła. Jednak można uzyskać ciekawy efekt zmieniając barwę światła pozycyjnego i włączając białe światło kierunkowe. Efekt widoczny na zdjęciu poniżej. Został użyty materiał symulujący grafit, białe światło kierunkowe i fioletowe światło pozycyjne.

Tabela 7 Tabela wartości współczynników światła dla brył z Rysunku 57

Źródło: Opracowanie własne

|  |  |  |
| --- | --- | --- |
| **Rodzaj światła** | Kierunkowe | Pozycyjne |
| **Barwa** | Biel | Fiolet |
| **Intensywność światła otoczenia** | 2,9% | 100% |
| **Intensywność światła rozproszonego** | 50% | 100% |
| **Intensywność światła odbitego** | 80% | 100% |
| **Wsp. kwadratowy tłumienia** | - | 0,1 |
| **Wsp. liniowy tłumienia** | - | 0,1 |
| **Wsp. stały tłumienia** | - | 0,2 |



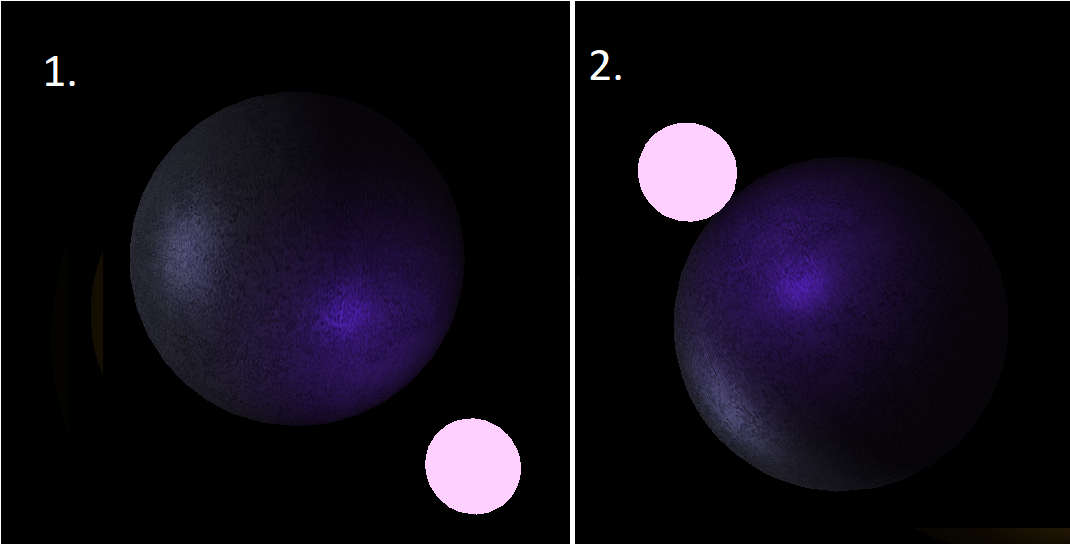
Rysunek 58 Prezentacja efektu zmieszania barw dwóch różnych źródeł światła

Źródło 58 Opracowanie własne

## Kamera

**Tryb obracania**

Kamera pracująca w trybie obracania jest w stanie pokazać bryłę z różnych stron zawsze z tej samej odległości będąc zwróconą w stronę środka bryły.

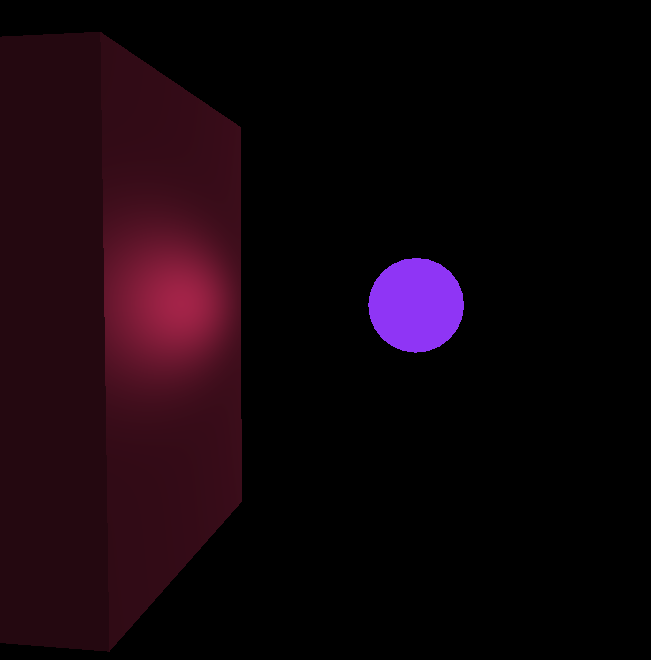


Rysunek 59 Prezentacja efektu obrócenia kamery wokół bryły

Źródło 59 Opracowanie własne

**Tryb latania**

W trybie latania kamera może znaleźć się w każdym miejscu w przestrzeni i może zostać zwrócona w każdym kierunku. Stwarza to możliwość to złapania kilku ciekawych ujęć.



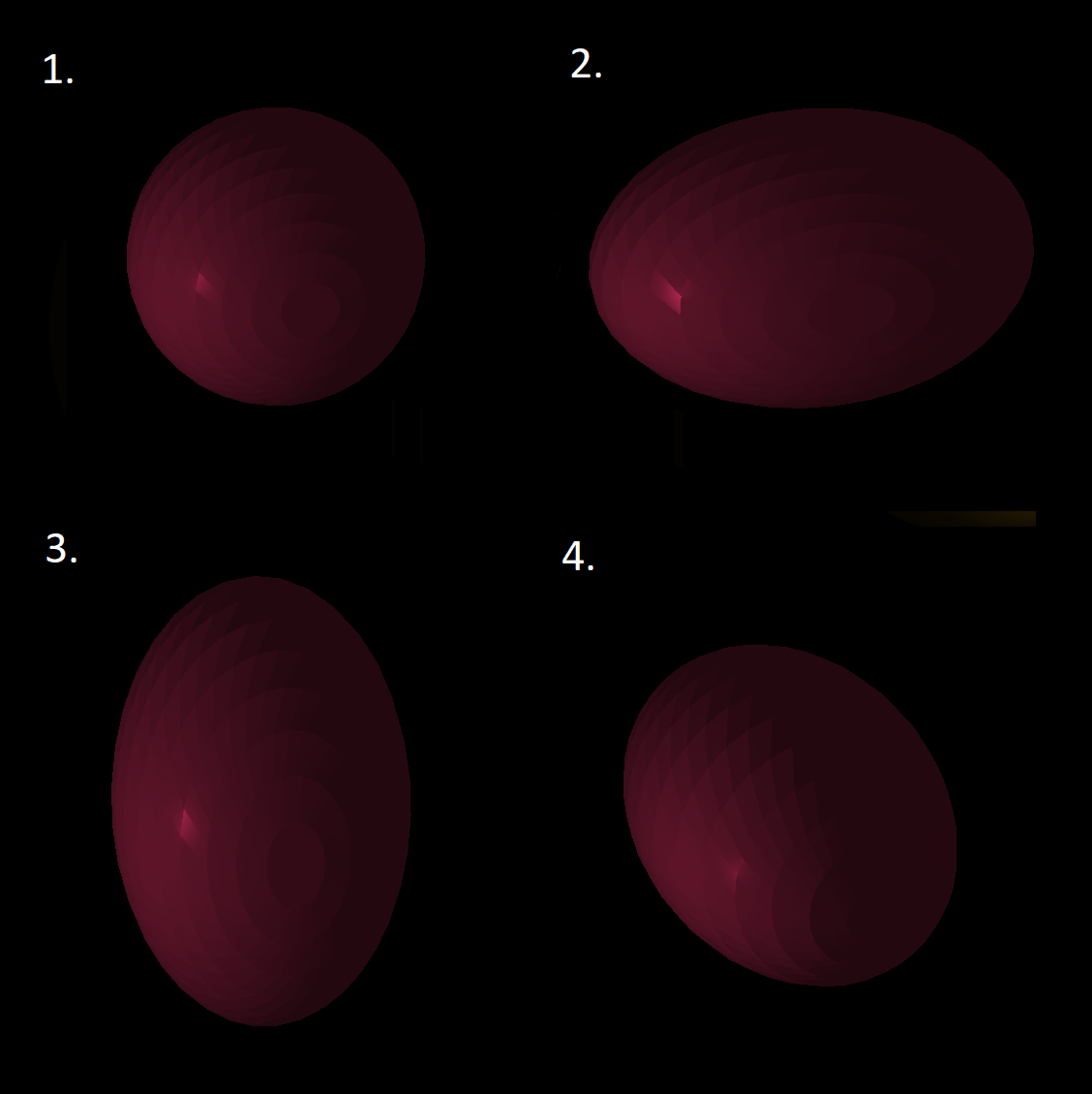
Rysunek 60 Prezentacja bryły z innej perspektywy dzięki trybowi latania

Źródło 60 Opracowanie własne

## Transformacje

**Skalowanie**

Generowane bryły można skalować według podstawowych kierunków X, Y i Z. Możliwe jest powiększenie lub zmniejszenie bryły wykonując czynność nazwaną w moim programie „skalowaniem ogólnym”. Oznacza to tyle, że jednocześnie odbywa się skalowanie wzdłuż wszystkich trzech kierunków.

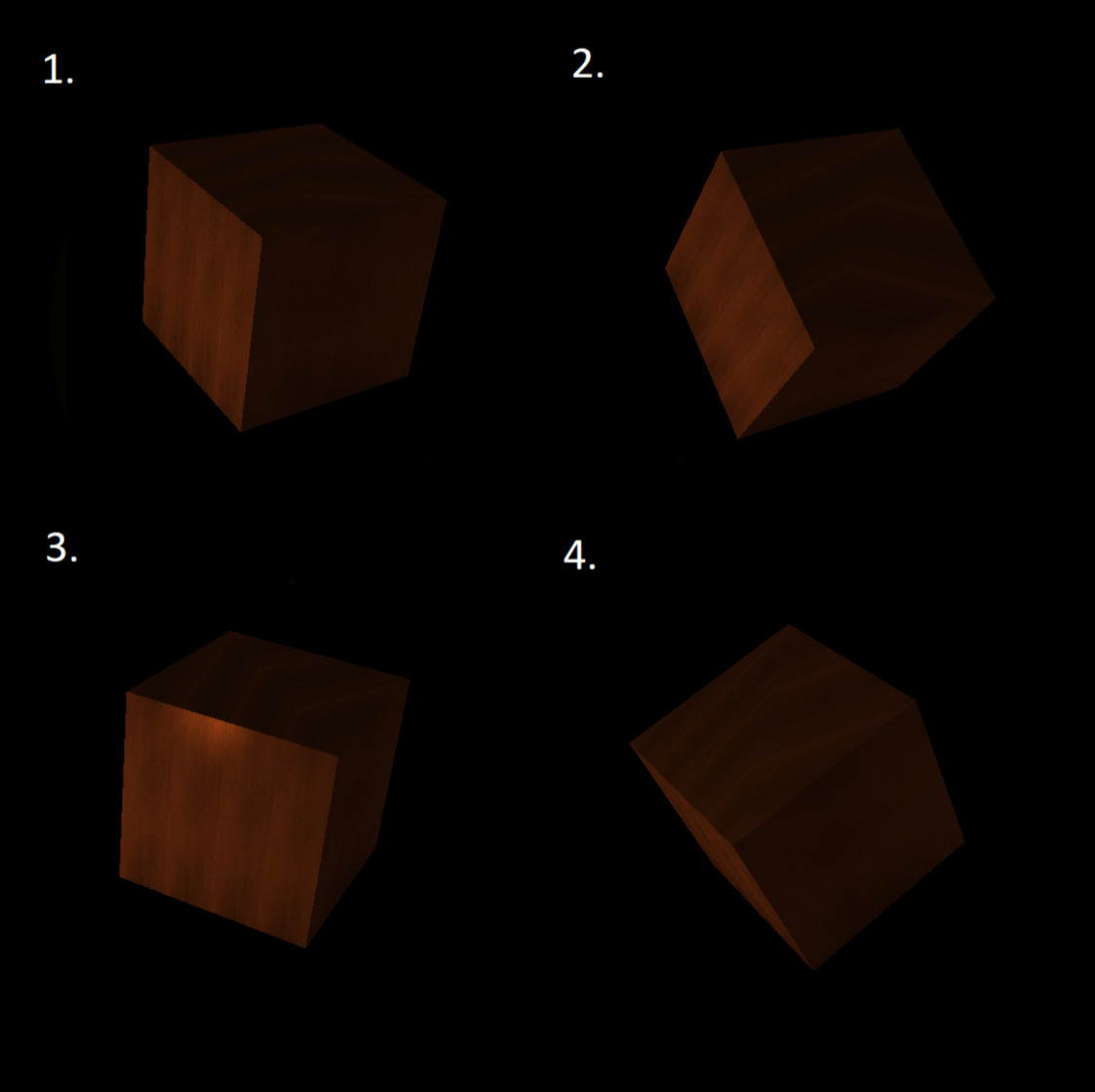


Rysunek 61 Prezentacja bryły wyskalowanej w osi: 1. Stan początkowy, 2. X, 3. Y, 4. Z

Źródło 61 Opracowanie własne

**Obracanie**

Obrócić obserwowany model można względem wszystkich trzech osi współrzędnych o 360 stopni dla każdej z nich.



Rysunek 62 Prezentacja efektu obrotu bryły: 1. Stan początkowy, 2. O 30 stopni względem osi X, 3. O 30 stopni względem osi Y, 4. O 30 stopni względem osi Z

Źródło 62 Opracowanie własne

# ZAKOŃCZENIE

Zaczynając pracę z OpenGL można odnieść wrażenie że próg wstępu do znajomości tej biblioteki jest dosyć wysoki. Istotnie tak jest, ponieważ sama biblioteka OpenGL jest jedynie narzędziem, które pozwala uzyskać zaplanowany cel. Jednak użytkowanie tego narzędzia nie należy do najprostszych, z racji ogromu wiedzy o grafice komputerowej, którą trzeba posiąść. Koszt, który trzeba w ten sposób ponieść wynagradzają jednak uzyskane efekty. Grafika komputerowa jest jedną z tych dziedzin w informatyce, którą śmiało można uznać za jedną z najlepszych form wykorzystania maszyn liczących, jakimi są komputery. W obecnych czasach to właśnie programy wykorzystujące jak najbardziej zaawansowane techniki generowania grafiki komputerowej wyznaczają trendy na rynku konsumenckim sprzętu komputerowego.

Obserwując efekty tej pracy i napisanej aplikacji można w pewien sposób zrozumieć mechanikę stojącą za oświetleniem w generowanych scenach. Nie odbyłoby się to jednak bez wieloletniej pracy poświęconej przez dziesiątki programistów tworzących biblioteki bazujące na tej specyfikacji, a także samego Bui Tuong Phong’a stojącego za opisanym w tej pracy modelem oświetlenia. Dzięki tym ludziom można z powodzeniem uznać, że OpenGL jest bardzo dobrym narzędziem do generowania grafiki komputerowej i mimo jego nie tak młodego wieku, nie można o tej technologii powiedzieć, że jest przestarzała.

## Bibliografia

### Pozycje książkowe

* G. Sellers, R. S. Wright Jr., N. Haemel, *OpenGL. Księga eksperta. Wydanie VII,* Wydawnictwo Helion, 2021
* Janusz Ganczarski*, OpenGL. Podstawy programowania grafiki 3D, wyd. Helion 2015*
* F. Dunn, I. Parberry*, 3D Math Primer For Graphics and Game Development (Wordware Game Math Libra-ry), wyd. Jenson Books Inc*

### Słowniki:

* Lidia Drabik, Aleksandra Kubiak-Sokół, Elżbieta Sobol, *Słownik języka polskiego PWN,* Wydawnictwo Naukowe PWN, 2006

### Źródła internetowe

Wyjaśnienie sformułowania API według Centralnego Ośrodka Infomatyki www.gov.pl/web/popcwsparcie/standard-api-dla-udostepniania-danych

„List of OpenGL applications”: en.wikipedia.org/wiki/List\_of\_OpenGL\_applications

Materiały laboratorium z kierunku Computer Science na Uniwersytecie w Reginie, Kanada. www.cs.uregina.ca/Links/class-info/405/WWW/Lab2/

OpenGL wiki – History of OpenGL. www.khronos.org/opengl/wiki/History\_of\_OpenGL

Fragment rejestru zmian Khronos OpenGL traktujący o funkcjonalności tablicy wierzchołków. www.khronos.org/registry/OpenGL/extensions/EXT/EXT\_vertex\_array.txt

Strona GLFW - www.glfw.org

Strona informacyjna WebGL - www.khronos.org/webgl/

Strona biblioteki SDL - www.libsdl.org

Strona biblioteki SFML - www.sfml-dev.org

Strona biblioteki NVAPI – [www.developer.nvidia.com/nvapi](http://www.developer.nvidia.com/nvapi)

Janusz Ganczarski, *Kurs OpenGL, C++*

Dokumentacja Universal Windows Platform, Fundamentals of DirectX programming

1. G. Sellers, R. S. Wright Jr., N. Haemel, *OpenGL® Księga Eksperta Wydanie VII,* Helion 2016, s. 39 [↑](#footnote-ref-1)
2. Dokumentacja *Universal Windows Platform*, *Fundamentals of DirectX programming,* 08.02.2017 [↑](#footnote-ref-2)
3. Dokumentacja OpenGL autorstwa grupy Khronos, *History of OpenGL,* 2020 [↑](#footnote-ref-3)
4. G. Sellers, R. S. Wright Jr., N. Haemel, *OpenGL® Księga Eksperta Wydanie VII,* wyd. Helion 2016, s. 486 [↑](#footnote-ref-4)
5. Janusz Ganczarski, *OpenGL. Podstawy programowania grafiki 3D,* wyd. Helion 2015, s. 121 [↑](#footnote-ref-5)
6. Janusz Ganczarski, *OpenGL. Podstawy programowania grafiki 3D,* wyd. Helion 2015, s. 32 [↑](#footnote-ref-6)
7. G. Sellers, R. S. Wright Jr., N. Haemel, *OpenGL® Księga Eksperta Wydanie VII,* wyd. Helion 2016, s. 487 [↑](#footnote-ref-7)
8. Janusz Ganczarski, *Kurs OpenGL, C++,* rozdział13. Światła i materiały [↑](#footnote-ref-8)
9. G. Sellers, R. S. Wright Jr., N. Haemel, *OpenGL® Księga Eksperta Wydanie VII,* wyd. Helion 2016, s. 106 [↑](#footnote-ref-9)
10. G. Sellers, R. S. Wright Jr., N. Haemel, *OpenGL® Księga Eksperta Wydanie VII,* wyd. Helion 2016, s. 107 [↑](#footnote-ref-10)
11. G. Sellers, R. S. Wright Jr., N. Haemel, *OpenGL® Księga Eksperta Wydanie VII,* wyd. Helion 2016, s. 97 [↑](#footnote-ref-11)
12. F. Dunn, I. Parberry, *3D Math Primer For Graphics and Game Development (Wordware Game Math Library),* wyd. Jenson Books Inc, s. 148 [↑](#footnote-ref-12)
13. G. Sellers, R. S. Wright Jr., N. Haemel, *OpenGL® Księga Eksperta Wydanie VII,* wyd. Helion 2016, s. 105 [↑](#footnote-ref-13)
14. G. Sellers, R. S. Wright Jr., N. Haemel, *OpenGL® Księga Eksperta Wydanie VII,* Helion 2016, s. 103 [↑](#footnote-ref-14)
15. G. Sellers, R. S. Wright Jr., N. Haemel, *OpenGL® Księga Eksperta Wydanie VII,* Helion 2016, s. 104 [↑](#footnote-ref-15)
16. Dokumentacja biblioteki *OpenGL Matematics (GLM),* rozdział *GLM\_GTC\_matrix\_transform* [↑](#footnote-ref-16)
17. G. Sellers, R. S. Wright Jr., N. Haemel, *OpenGL® Księga Eksperta Wydanie VII,* Helion 2016, s. 105 [↑](#footnote-ref-17)
18. Dokumentacja biblioteki *OpenGL Matematics (GLM),* rozdział *GLM\_GTC\_matrix\_transform* [↑](#footnote-ref-18)
19. Dokumentacja biblioteki *The OpenGL Extension Wrangler Library,* http://glew.sourceforge.net/ [↑](#footnote-ref-19)
20. Dokumentacja biblioteki GLFW, https://www.glfw.org/ [↑](#footnote-ref-20)
21. Dokumentacja biblioteki *OpenGL Matematics,* https://glm.g-truc.net/0.9.5/api/index.html [↑](#footnote-ref-21)
22. Plik README.md dostarczony z biblioteką ImGui, https://github.com/ocornut/imgui [↑](#footnote-ref-22)