## WOJSKOWA AKADEMIA TECHNICZNA

# *LABORATORIUM*

# *GRAFIKA KOMPUTEROWA*

Stopień, imię i nazwisko słuchacza

Grupa szkoleniowa

Stopień, imię i nazwisko prowadzącego

***Grzegorz Pol***

***I7X3S1***

***dr inż. Marek Salamon***

Data wykonania ćwiczenia

***2.02.2009 r.***

***SPRAWOZDANIE***

***Z***

***PRACY LABORATORYJNEJ***

***NR 4***

|  |  |
| --- | --- |
| **Temat:** | Modelowanie oświetlenia. |

* 1. **Zadanie**

Podczas zajęć laboratoryjnych należało wykonać zadanie o poniższej treści (zestaw15):

Wykorzystując biblioteki OpenGL i GLUT napisać program przedstawiający perspektywiczny obraz obiektu o następujących parametrach:

1. Typ obiektu: *walec o zmiennej podzielnej przez 3 liczbie podziałów pionowych i poziomych*,
2. Właściwości materiału 1: *czerwony matowy* (widziany w białym świetle),
3. Właściwości materiału 2: *czarny błyszczący* (widziany w białym świetle),
4. Właściwości materiału 3: *żółty emitujący* (widziany w białym świetle),
5. Sposób przyporządkowywania materiałów do obiektu zgodnie ze wzorem: *pasy pionowe* z uwzględnieniem podziałów pionowych.

Obiekt należy oświetlić dwoma źródłami światła o następujących parametrach:

*Źródło nr 1:*

* typ: *reflektor* (ang. *spot*),
* kolor: zielony,
* natężenie: *1*,
* kąt odcięcia: *300*,
* położenie: *zmienne* po orbicie kołowej o środku w punkcie S(0, 0, 0) z możliwością interaktywnej zmiany następujących parametrów:
	+ promienia orbity,
	+ prędkości kątowej (3 różne prędkości),
	+ kąta nachylenia orbity do osi OX,
* kierunek świecenia: *na obiekt*.

*Źródło nr 2:*

* typ: *kierunkowe*,
* kolor: żółty,
* natężenie: *0.5*,
* położenie: *stałe* w punkcie P(5,5,20) *układu współrzędnych obserwatora,*
* kierunek świecenia: *na obiekt*.

Program powinien umożliwiać:

* 1. interaktywne, niezależne włączanie i wyłączanie źródeł światła;
	2. interaktywną zmianę liczby podziałów pionowych i poziomych bryły;
	3. interaktywną zmianę wielkości bryły;
	4. interaktywną zmianę położenia obserwatora poprzez podanie następujących parametrów:

– odległości obserwatora od środka układu współrzędnych sceny;

– kąta obrotu wokół osi OY w zakresie [00, 3600] z krokiem 10.

Oświetlony obiekt powinien zawsze znajdować się w centralnej części okna.

* 1. **Opis rozwiązania:**

Realizację treści zadania rozpocząłem od zdefiniowania parametrów materiałów:

// Materiał nr 1 – czerwony matowy

const GLfloat ambient\_m1[4] = { 1.0, 0.0, 0.0, 1.0 };

const GLfloat diffuse\_m1[4] = { 0.5, 0.0, 0.0, 1.0 };

const GLfloat specular\_m1[4] = { 0.0, 0.0, 0.0, 0.0 };

const GLfloat emission\_m1[4] = { 0.0, 0.0, 0.0, 0.0 };

// Materiał nr 2 – czarny błyszczący

const GLfloat ambient\_m2[4] = { 0.0, 0.0, 0.0, 1.0 };

const GLfloat diffuse\_m2[4] = { 0.5, 0.5, 0.5, 1.0 };

const GLfloat specular\_m2[4] = { 1.0, 1.0, 1.0, 1.0 };

const GLfloat emission\_m2[4] = { 0.0, 0.0, 0.0, 0.0 };

// Materiał nr 3 – żółty emitujący

const GLfloat ambient\_m3[4] = { 0.0, 0.0, 0.0, 1.0 };

const GLfloat diffuse\_m3[4] = { 0.0, 0.0, 0.0, 1.0 };

const GLfloat specular\_m3[4] = { 0.0, 0.0, 0.0, 1.0 };

const GLfloat emission\_m3[4] = { 1.0, 1.0, 0.0, 1.0 };

Jak widać na powyższym listingu całość sprowadza się do utworzenia tablic, które to zawierają odpowiednio ustawione składowe RGBA (Red Green Blue Alpha) poszczególnych parametrów.

Składowe parametru *ambient* określają stopień odbicia światła otaczającego, parametru *diffuse* – stopień rozproszenia światła rozproszonego, parametru *specular* – stopień odbicia światła odbitego, zaś składowe RGBA parametru *emission* określają światło emitowane przez obiekt. Ten ostatni parametr domyślnie jest ustawiony tak, że materiał nie jest emitujący. Dlatego pierwszy i drugi materiał ma tablicę emission ustawioną na 0.

Dobranie powyższych parametrów nie jest zadaniem łatwym i przysporzyło mi kłopotów, jednakże po jakimś czasie udało się osiągnąć oczekiwany efekt.

Przy znajdowaniu właściwości dla pierwszego materiału ustawiłem oprócz oczywistego parametru *ambient* parametr *diffuse* gdzie ustaliłem rozpraszanie światła czerwonego na wartość *0.5*. Natomiast podczas ustawień drugiego materiału pamiętałem, że powierzchnie błyszczące powinny mieć mniejsze składowe parametru *diffuse*, od składowych parametru *specular*. Ostatni materiał wymagał dodatkowo dobrania odpowiednich wartości dla parametru *emission*.

Po ustawieniu naszych kolorów przystąpiłem do napisania funkcji, które to będą korzystały z właśnie przed chwilą to zdefiniowanych materiałów w celu ustawienia ich w odpowiednim momencie modelowania obiektu. Funkcje te przedstawiają się następująco:

// Definiowanie wl. materiału walca na podstawie zapisanych w tablicy 'material'

// parametrow (material obowiazuje tylko do scian skierowanych przodem do obs.)

// Funkcja ustawiajaca material nr 1

void Material1() {

 glMaterialfv(GL\_FRONT, GL\_AMBIENT, ambient\_m1);

 glMaterialfv(GL\_FRONT, GL\_DIFFUSE, diffuse\_m1);

 glMaterialfv(GL\_FRONT, GL\_SPECULAR, specular\_m1);

 glMaterialf(GL\_FRONT, GL\_SHININESS, 0);

 glMaterialfv(GL\_FRONT, GL\_EMISSION, emission\_m1);

}

// Definiowanie wl. materiału walca na podstawie zapisanych w tablicy 'material'

// parametrow (material obowiazuje tylko do scian skierowanych przodem do obs.)

// Funkcja ustawiajaca material nr 2

void Material2() {

 glMaterialfv(GL\_FRONT, GL\_AMBIENT, ambient\_m2);

 glMaterialfv(GL\_FRONT, GL\_DIFFUSE, diffuse\_m2);

 glMaterialfv(GL\_FRONT, GL\_SPECULAR, specular\_m2);

 glMaterialf(GL\_FRONT, GL\_SHININESS, 128);

 glMaterialfv(GL\_FRONT, GL\_EMISSION, emission\_m2);

}

// Definiowanie wl. materiału walca na podstawie zapisanych w tablicy 'material'

// parametrow (material obowiazuje tylko do scian skierowanych przodem do obs.)

// Funkcja ustawiajaca material nr 3

void Material3() {

 glMaterialfv(GL\_FRONT, GL\_AMBIENT, ambient\_m3);

 glMaterialfv(GL\_FRONT, GL\_DIFFUSE, diffuse\_m3);

 glMaterialfv(GL\_FRONT, GL\_SPECULAR, specular\_m3);

 glMaterialf(GL\_FRONT, GL\_SHININESS, 0);

 glMaterialfv(GL\_FRONT, GL\_EMISSION, emission\_m3);

}

W powyższym listingu opisałem sposób zachowania się powierzchni obiektów na poszczególne rodzaje światła za pomocą funkcji *glMaterial*. Tutaj dodatkowo musiałem ustawić parametr *shininess*. Jego wartości możemy wybrać z zakresu liczb od 0 do 128. Liczba 0 oznacza, że dany materiał jest matowy – natomiast 128 że jest błyszczący.

Po ustawieniu kolorów, którymi to będzie pokolorowana nasza przyszła bryła przystąpiłem do ustawienia parametrów świateł.

// Reflektor

const GLfloat ambient\_s1[4] = { 0.0, 1.0, 0.0, 1.0 };

const GLfloat diffuse\_s1[4] = { 1.0, 1.0, 0.0, 1.0 };

GLfloat specular\_s1[4] = { 1.0, 1.0, 1.0, 1.0 };

GLfloat position\_s1[4] = { 0.0, 0.0, 0.0, 1.0 };

GLfloat direction\_s1[4] = { -90, 0, 0 };

// Swiatlo kierunkowe

const GLfloat ambient\_s2[4] = {1.0, 1.0, 0.0, 0.5 };

const GLfloat diffuse\_s2[4] = {0.3, 0.3, 0.3, 0.5 };

GLfloat specular\_s2[4] = {1.0, 1.0, 1.0, 1.0 };

GLfloat position\_s2[4] = {5.0, 5.0, 20.0,0.0 };

Ustawianie świateł jest procesem podobnym do ustawiania materiałów. Dodatkowo występują tutaj parametry: *position*, który to określa położenie źródła światła oraz parametr *direction, który jest* trój-współrzędnym znormalizowanym wektorem określającym kierunek reflektora. Ponadto mamy tutaj okazję, wykorzystać dobrodziejstwo modeli przestrzeni barw RGBA. RGBA jest wzbogaconą wersją o kanał alfa znanego nam wszystkim modelu RGB. Wspomniany przeze mnie [kanał](http://pl.wikipedia.org/wiki/Kana%C5%82_alfa) używany jest jako odpowiednik [współczynnika pochłaniania światła](http://pl.wikipedia.org/wiki/Prawo_Bouguera#Wsp.C3.B3.C5.82czynnik_poch.C5.82aniania_.C5.9Bwiat.C5.82a). Gdy kanał alfa skojarzony z danym elementem ma wartość 0, to taki piksel staje się całkowicie przezroczysty (czyli przybiera kolor tła pod nim - w rezultacie wydaje się nam, że nie widzimy go). Natomiast, gdy wartość zapisana we współczynniku alfa wyniesie 1, element będzie całkowicie widoczny (jak w zwykłym obrazie cyfrowym bez kanału alfa). Także ustawiając światło kierunkowe wartość *alfa* ustawiłem na *0.5*. To jest nasze natężenie światła.

Następnie postanowiłem zdefiniować źródła naszych świateł za pomocą funkcji *Swiatlo1()* oraz funkcji *Swiatlo2()*:

void Swiatlo1() {

 glPushMatrix();

 glEnable(GL\_LIGHTING);

 glRotatef(kat\_y, 1, 0, 0);

 glRotatef(kat\_x, 0, 1, 0);

 glRotatef(kat\_z, 0, 0, 1);

 glTranslatef(R\_L2, R\_L1, 0);

 glPushMatrix();

 glutWireSphere(0.25, 50, 50);

 glPopMatrix();

 glLightfv(GL\_LIGHT1, GL\_DIFFUSE, diffuse\_s1);

 glLightfv(GL\_LIGHT1, GL\_SPECULAR, specular\_s1);

 glLightfv(GL\_LIGHT1, GL\_POSITION, position\_s1);

 glLightf(GL\_LIGHT1, GL\_SPOT\_CUTOFF, 30);

 glLightfv(GL\_LIGHT1, GL\_SPOT\_DIRECTION, direction\_s1);

 glPopMatrix();

}

void Swiatlo2() {

 glLightfv(GL\_LIGHT2, GL\_DIFFUSE, diffuse\_s2);

 glLightfv(GL\_LIGHT2, GL\_SPECULAR, specular\_s2);

 glLightfv(GL\_LIGHT2, GL\_POSITION, position\_s2);

}

Funkcja *Swiatlo1()* odpowiada w naszym programie za krążący wokół naszego obiektu reflektor. Zgodnie z treścią zadania mamy możliwość modyfikacji oddalenia naszego światła od obiektu poprzez parametry *R\_L1* i *R\_L2*. Nasz reflektor jest symbolizowany przez kulę, którą stworzyłem za pomocą funkcji GLUT: *glutWireSphere* o liczbie południków i równoleżników 50. Ponadto zwrócę uwagę na parametr *spot\_cutoff*, który reprezentuje kąt odcięcia. Ustawiam go zgodnie z treścią zadania na 30˚. Funkcja *Swiatlo2()* odpowiada tylko za przypisaniu światłu kierunkowemu wcześniej to zdefiniowanych parametrów *diffuse*, *specu lar* oraz *position*.

Po wymienionych wyżej czynnościach przystąpiłem do modelowania figury. Naszą figurą jest walec w pasy pionowe. Każdy z tych pasów ma być wykonany z innego materiału (materiały zostały już przez nas wcześniej w programie zdefiniowane). Rysowanie w/w figury podzieliłem na 3 etapy:

* *Rysowanie ściany bocznej* – do modelowania ściany bocznej użyłem prymitywu *GL\_TRIANGLE\_STRIP.* Każdy element ściany bocznej tworzę oddzielnie, a następnie sprawdzam wynik dzielenia modulo przez 3 liczby, która odpowiada za numer (ściana 1, 2, 3, 4…) danego podziału.

 // Wyznaczenie kata wyznaczajacego pojedynczy wycinek pionowy

 dAlfa = 360.0L/(double)nh;

 // Wyznaczenie wysokosci pojedynczego wycinka poziomego

 dH = h/(double)nv;

// Wyznaczanie wierzcholkow i wektorow normalnych powierzchni bocznych

for (i = 0; floor((i+1)\*dH\*1.0E10) <= floor(h\*1.0E10); i++) {

 glBegin(GL\_TRIANGLE\_STRIP);

 glNormal3f(0.0, 0.0, 1.0);

 glVertex3f(0.0, (i + 1)\*dH, r);

 glVertex3f(0.0, i\*dH, r);

 for (j = 1; j\*dAlfa <= 360.0L + dAlfa; j++){

 if (j%3 == 0) {

 Material1();

 } else if (j%3 ==1){

 Material3();

 } else {

 Material2();

 }

 glNormal3f(sin(DEG2RAD(j\*dAlfa)), 0.0, cos(DEG2RAD(j\*dAlfa)));

 glVertex3f(r\*sin(DEG2RAD(j\*dAlfa)), (i + 1)\*dH, r\*cos(DEG2RAD(j\*dAlfa)));

 glVertex3f(r\*sin(DEG2RAD(j\*dAlfa)), i\*dH, r\*cos(DEG2RAD(j\*dAlfa)));

 }

 glEnd();

}

* *Rysowanie dolnej podstawy* – tutaj z wyborem prymitywu nie miałem żadnych problemów, gdyż prymityw *GL\_TRIANGLE\_FAN* jest wręcz do tego stworzony. Wybór materiału, z którego jest wykonany dany trójkąt działa na identycznej zasadzie co przy rysowaniu ściany bocznej.

// Wyznaczenie wierzcholkow i wektorow normalnych dolnej podstawy

 glBegin(GL\_TRIANGLE\_FAN);

 glNormal3f(0.0, -1.0, 0.0);

 glVertex3f(0.0, 0.0, 0.0);

 for (i = 0; i \* dAlfa <= 360.0L + dAlfa; i++){

 if (i%3 == 1) {

 Material1();

 } else if (i%3 ==0){

 Material2();

 } else{

 Material3();

 }

 glVertex3f(r\*sin(DEG2RAD(i\*dAlfa)), 0.0, r\*cos(DEG2RAD(i\*dAlfa)));

 }

 glEnd();

* *Rysowanie górnej podstawy* wygląda identycznie jak w przypadku rysowania dolnej podstawy. Jedyną różnicą jest ustawienie drugiego parametru na *h* w obu poleceniach *glVertex3f()* odpowiadającego za wysokość naszego punktu rysowanego trójkąta.

 // Wyznaczenie wierzcholkow i wektorow normalnych gornej podstawy

 glBegin(GL\_TRIANGLE\_FAN);

 glNormal3f(0.0, 1.0, 0.0);

 glVertex3f(0.0, h, 0.0);

 for (i = 0; i \* dAlfa <= 360.0L + dAlfa; i++){

 if (i%3 == 1) {

 Material1();

 } else if (i%3 ==0){

 Material2();

 } else{

 Material3();

 }

 glVertex3f(r\*sin(DEG2RAD(i\*dAlfa)), h, r\*cos(DEG2RAD(i\*dAlfa)));

 }

 glEnd();

* 1. **Działanie programu:**

Program uruchamia się i wyświetla model figury o parametrach początkowych (parametry te możemy zmieniać za pomocą klawiszy specjalnych – o tym jednak później), a więc nasza figura składa się zarówno z 12 podziałów poziomych jak i pionowych.



Jednak do zaobserwowania lepszego efektu odbicia światła trzeba stanowczo zwiększyć liczbę podziałów pionowych oraz poziomych - mimo tego, że zmiana podziałów poziomów nie jest teraz dla nas w ogóle widoczna (widoczna byłaby np. przy ścianie bocznej imitującej szachownicę).





Na powyższym i poniższym screenie można bez większego wysiłku zobaczyć różniće w jakości oświetlenia naszych brył. Poniższy model jest oświetlony w sposób o wiele bardziej realistycznie niż górny.



Na obu obrazkach oprócz brył widzimy nasz reflektor w postaci sfery. Poniżej przedstawię jak wygląda nasz model po włączeniu drugiego kierunkowego światła:



Róznica nie jest aż nadto zauważalna. Spowodowane jest to przede wszystkim dość słabym światłem kierunkowym (natężenie 0.5) oraz jego kolorem (jeden z materiałów pasów pionowych jest żółty). Także użycie tego reflektora w sposób minimalny zmienia kolor naszej figury (np. czarny staje się ciemnoszary).

Patrząc na wszystkie powyższe rysunki możemy zaobserwować, że na pasy pionowe wykonane z żółtego materiału oświetlenie nie ma wpływu na ich kolor. Oznacza to, że kolor został poprawnie przeze mnie zdefiniowany – wg treści zadania miał być emitujący.

* 1. **Klawisze specjalne:**

Niestety na statycznym obrazie nie jestem w stanie zademonstrować wszystkich funkcji mojego programu. A są to min:

// Funkcja obslugi klawiatury

void ObslugaKlawiatury(unsigned char klawisz, int x, int y)

{

 switch(klawisz){

- przybliżanie i oddalanie naszego reflektora od obserwowanego obiektu

 case 'q': R\_L2+=0.1;

 break;

 case 'Q': R\_L2-=0.1;

 break;

- zwiększanie i zmniejszanie liczby pionowych podziałów

 case '8': lPionowych = (lPionowych == LPION\_MAX)? LPION\_MAX : lPionowych + 3;

 break;

 case '\*': lPionowych = (lPionowych == LPION\_MIN)? LPION\_MIN : lPionowych - 3;

 break;

- zwiększanie i zmniejszanie liczby poziomych podziałów

 case '9': lPoziomych = (lPoziomych == LPOZ\_MAX)? LPOZ\_MAX : lPoziomych + 3;

 break;

 case '(': lPoziomych = (lPoziomych == LPOZ\_MIN)? LPOZ\_MIN : lPoziomych - 3;

- zwiększanie oraz zmniejszanie wysokości bryły

 break;

 case 'w': wysokosc = (wysokosc == WYS\_MAX) ? WYS\_MAX : wysokosc + 1;

 break;

 case 'W': wysokosc = (wysokosc == 1) ? wysokosc : wysokosc - 1;

 break;

- zwiększanie oraz zmniejszanie promienia bryły

 case 'p': promien = (promien == R\_MAX) ? R\_MAX : promien + 1;

 break;

 case 'P': promien = (promien == 1) ? promien : promien - 1;

 break;

- ustawienie parametrów domyślnych (początkowych)

 case 'r': UstawDomyslneWartosciParametrow();

 break;

- przyszpieszanie i spowalnianie naszego reflektora

 case '3': if(roznica<3) roznica=roznica+0.5;

 break;

 case '#' : if(roznica>1) roznica=roznica-0.5;

 break;

- włączanie i wyłączanie reflektora

 case '4': glEnable(GL\_LIGHT1);

 break;

 case '$': glDisable(GL\_LIGHT1);

 break;

- włączanie i wyłączanie światła kierunkowego

 case '6': glEnable(GL\_LIGHT2);

 break;

 case '^': glDisable(GL\_LIGHT2);

 break;

 }

}

* 1. **Ekran:**

Aby ułatwić użytkownikowi korzystanie z mojego programu postanowiłem klawiszologie wyświetlić na ekranie wraz z aktualną liczbą podziałów poziomych i pionowych oraz wysokością i promieniem bryły. Do zrealizowania powyższego przedwsięzięcia musiałem najpierw zmienić typ rzutu z perspektywicznego na ortogonalny, zamodelować scenę dwuwymiarową, zablokować oświetlenie aby nie oświetlało mi tekstu, oraz wybrać kolor tekstu. Wymienione powyżej przeze mnie zadania realizuje poniższy kod:

void RysujNakladke(void){

 char buf[255];

 glMatrixMode(GL\_PROJECTION);

 glPushMatrix();

 glLoadIdentity();

 glOrtho(0.0, szerokoscOkna, 0.0, wysokoscOkna,-100.0, 100.0);

 glMatrixMode(GL\_MODELVIEW);

 glPushMatrix();

 glLoadIdentity();

 glDisable(GL\_LIGHTING);

 glColor3f(0.0, 0.0, 0.25);

W tym momencie muszę przenieść wybrany przeze mnie tekst do wcześniej zainicjowanego bufora *buf*, wybrać pozycję tekstu (zawsze używałem stałych *X\_OFFSET\_OBIEKT* oraz *Y\_OFFSET\_OBIEKT* jako pozycji startowej następnie dodając lub odejmując odpowiednią liczbę pikseli), a następnie narysować tekst rastrowy zawarty w *buf* w miejsce przeze mnie wskazane. Poniżej przedstawiam wybrane z programu napisy:

 sprintf(buf, "Grzegorz Pol");

 glRasterPos2i(X\_OFFSET\_OBIEKT, Y\_OFFSET\_OBIEKT+280);

 RysujTekstRastrowy(GLUT\_BITMAP\_8\_BY\_13, buf);

 sprintf(buf, "Grupa: I7X3S1");

 glRasterPos2i(X\_OFFSET\_OBIEKT+650, Y\_OFFSET\_OBIEKT+280);

 RysujTekstRastrowy(GLUT\_BITMAP\_8\_BY\_13, buf);

 sprintf(buf, "Parametry bryly:");

 glRasterPos2i(X\_OFFSET\_OBIEKT, Y\_OFFSET\_OBIEKT);

 RysujTekstRastrowy(GLUT\_BITMAP\_8\_BY\_13, buf);

 sprintf(buf, " - promien (%.1f)", promien);

 glRasterPos2i(X\_OFFSET\_OBIEKT, Y\_OFFSET\_OBIEKT - 10);

 RysujTekstRastrowy(GLUT\_BITMAP\_8\_BY\_13, buf);

 ...

Po zakończeniu wypisywania na ekran resetuje macierze do stanu sprzed wywołania funkcji, a następnie odblokowuje oświetlenie (w przeciwnym razie w następnej klatce by ono nie działało):

 glMatrixMode(GL\_PROJECTION);

 glPopMatrix();

 glMatrixMode(GL\_MODELVIEW);

 glPopMatrix();

 glEnable(GL\_LIGHTING);

}

Ekran wraz z dodanymi napisami wygląda następująco:



* 1. **Wnioski:**

Zadanie, zostało zrealizowane w pełni pomyślnie. Powyższe ćwiczenia miało na celu zapoznanie nas z technikami, które są potrzebne do zamodelowania oświetlenia za pomocą biblioteki OpenGL. W trakcie wykonywania ćwiczeń musiałem skorzystać z wiedzy nabytej na poprzednich zajęciach laboratoryjnych, a także wykazać się nowymi informacjami na temat oświetlenia. Podczas użytkowania programu zarejestrowałem kilka ciekawych spostrzeżeń oraz wniosków:

* Zarówno definiowanie oraz utworzenie materiałów budujących naszą figurę jest dość proste (o ile się zna i rozumie poszczególne parametry opisujące materiał) . Ponadto wypadałoby w sposób umiejętny obsługiwać się paletą RGB i umieć szybko liczyć odwrotności danej liczby reprezentującej kolor.
* Definiowanie i tworzenie światła jest wręcz podobne do tej samej operacji dotyczącej materiałów.
* Aby zaobserwować bardziej realistyczne zachowanie oświetlenia trzeba zwiększyć liczbę podziałów pionowych jak i poziomych.
* Nie można jednak z liczbą podziałów „przesadzić”, gdyż zbyt duża ich liczba powoduje zwiększanie zapotrzebowanie programu na pamięć i procesor, które służą mu do obliczenia. W takim momencie wiele zależy od wydajności komputera, na którym pracujemy.
* Przy innym świetle kierunkowym (mocniejszym) można zaobserwować nakładanie się świateł co powoduje, że kolor, który pokrywa nasz materiał potrafi zmienić kolor na całkowicie inny.

Ponadto dziś patrząc na wysiłek jaki włożyłem w to aby bardzo dobrze zrozumieć daną pracę laboratoryjną dochodzę do wniosku, że modelowanie oświetlenia mimo używania naprawdę prostych i jasnych (z definciji – gorzej z praktyką) funkcji OpenGL jest czynnością bardzo trudną, wymagjącą poświęcenia znacznie większej ilości czasu niż się wydaję na początki w celu nabycia doświadczenia, które jest wręcz niezbędne do trudniejszych zadań dotyczących tematu oświetlenia.