Światło i materiały w OpenGL

March 27, 2019

1 Introduction

Celem jest stosowanie światła oraz materiałów biblioteki OpenGL.

Oświetlenie jest jednym z najważniejszych problemów dla realistycznej grafiki 3D. Celem jest symulowanie źródeł światła i sposobu, w jaki emitowane przez nie światło oddziałuje z obiektami na scenie. Obliczenia oświetlenia są domyślnie wyłączone w OpenGL. Oznacza to, że gdy OpenGL stosuje kolor do wierzchołka, po prostu używa bieżącej wartości koloru ustawionej przez jedną z funkcji glColor*. Aby OpenGL wykonał obliczenia oświetlenia, musisz włączyć oświetlenie, wywołując `glEnable(GL_LIGHTING)`. Jeśli to wszystko, co robisz, przekonasz się, że wszystkie obiekty są całkowicie czarne. Jeśli chcesz je zobaczyć, musisz włączyć niektóre światła.

Właściwości powierzchni, które określają, w jaki sposób oddziałuje ona ze światłem, określa się jako materiał powierzchni. Powierzchnia może mieć kilka różnych właściwości materiału.

Aby oświetlić scenę, oprócz włączenia `GL_LIGHTING`, musisz skonfigurować co najmniej jedno źródło światła. W przypadku bardzo podstawowego oświetlenia często wystarczy

```
glEnable(GL_LIGHT0);
```

To polecenie włącza światło kierunkowe, które świeci od kierunku widza do sceny. (Zauważ, że ostatni znak w `GL_LIGHT0` to zero.) Ponieważ świeci od kierunku widza, oświetli wszystko, co użytkownik może zobaczyć. Światło jest białe, bez składnika specular; to znaczy, zobaczysz rozproszony kolor obiektów, bez żadnych odbłysków światła.

2 Materiały

Właściwości materiału są atrybutami wierzchołków w ten sam sposób, w jaki kolor jest atrybutem wierzchołka. Oznacza to, że stan OpenGL zawiera bieżącą wartość dla każdej właściwości materiału. Gdy wierzchołek jest generowany przez wywołanie jednej z funkcji `glVertex *`, kopia każdej z bieżących właściwości materiału jest zapisywana wraz ze współrzędnymi wierzchołków. Gdy renderowany jest prymityw zawierający wierzchołek, właściwości materiału związane

z wierzchołkiem są używane wraz z informacją o oświetleniu, aby obliczyć kolor wierzchołka.

Jest to skomplikowane przez fakt, że wielokąty są dwustronne, a przednia powierzchnia i tylna powierzchnia wielokąta mogą mieć różne materiały. Oznacza to, że w rzeczywistości dla każdego wierzchołka przechowywane są dwa zestawy wartości właściwości materiału: materiał przedni i materiał tylny. (Materiał z tyłu nie jest w rzeczywistości używany, chyba że włączysz oświetlenie dwustronne, które zostanie omówione poniżej).

Mając to wszystko na uwadze, przyjrzymy się funkcjom ustawiania bieżących wartości właściwości materiału.

Dla ustawienia kolorów materiału otoczenia, rozproszonego, lustrzanego i emisyjnego, funkcja jest

```
void glMaterialfv (int side, int property, float * valueArray)
```

Pierwszym parametrem może być `GL_FRONT_AND_BACK, GL_FRONT` lub `GL_BACK`. Informuje, czy ustawiasz wartość właściwości materiału dla powierzchni przedniej, tylnej lub obu.

Drugi parametr informuje, która właściwość materiału jest ustawiana. Może to być `GL_AMBIENT`, GL_DIFFUSE, GL_SPECULAR, GL_EMISSION lub `GL_AMBIENT_AND_DIFFUSE`. Zauważ, że możliwe jest ustawienie kolorów ambient i diffuse na tę samą wartość za pomocą jednego wywołania `glMaterialfv`, używając `GL_AMBIENT_AND_DIFFUSE` jako nazwy właściwości.

Ostatnim parametrem `glMaterialfv` jest tablica zawierająca cztery liczby zmiennoprzecinkowe. Liczby dają kolorowym komponentom RGBA wartości w zakresie od 0,0 do 1,0; wartości spoza tego zakresu są faktycznie dozwolone i będą używane w obliczeniach oświetlenia, ale takie wartości są nietypowe. Należy zauważyć, że wymagany jest komponent alfa, ale jest on używany tylko w przypadku koloru rozproszonego: Gdy obliczany jest kolor wierzchołka, jego komponent alfa jest ustawiony jako składnik alfa koloru rozproszonego materiału.

Właściwość materiału połysk (shininess) jest pojedynczą liczbą, a nie tablicą, i istnieje inna funkcja do ustawiania jej wartości (bez „v" na końcu nazwy):

```
void glMaterialf (int side, int property, float value)
```

Ponownie, stroną może być `GL_FRONT_AND_BACK, GL_FRONT` lub `GL_BACK`. Właściwość musi być `GL_SHININESS`. A wartość jest zmiennoprzecinkowa w zakresie od 0,0 do 128,0.

W porównaniu z dużą liczbą wersji `glColor *` i `glVertex *` opcje ustawiania materiału są ograniczone. W szczególności nie można ustawić koloru materiału bez zdefiniowania tablicy zawierającej wartości składowych koloru. Załóżmy na przykład, że chcemy ustawić kolory otoczenia i rozproszone na niebieskawo zielony. W C można to zrobić za pomocą

```
float bgcolor [4] = {0,0, 0,7, 0,5, 1,0};
glMaterialfv (GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, bgcolor);
```

Z symulatorem JavaScript dla OpenGL to wyglądałoby

```
var bgcolor = [0.0, 0.7, 0.5, 1.0];
glMaterialfv (GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, bgcolor);
```

A w JOGL API dla Javy, gdzie metody z parametrami tablicowymi mają dodatkowy parametr dający indeks początkowy danych w tablicy, staje się

```
float [] bgcolor = {0.0F, 0.7F, 0.5F, 1.0F};
gl.glMaterialfv (GL2.GL_FRONT_AND_BACK, GL2.GL_AMBIENT_AND_DIFFUSE, bgcolor, 0);
```

W C trzeci parametr jest w rzeczywistości wskaźnikiem do float, który pozwala na elastyczność przechowywania wartości dla kilku właściwości materiału w jednej tablicy. Załóżmy na przykład, że mamy tablicę C

```
float gold [13] = {0.24725, 0.1995, 0.0745, 1.0, / * ambient * /
                   0,75164, 0,60648, 0,22648, 1,0, / * rozproszone * /
                   0,628281, 0,555802, 0,366065, 1,0, / * specular * /
                   50,0 / * po␣lysk * /
    };
```

gdzie pierwsze cztery liczby w tablicy określają kolor otoczenia; następne cztery, kolor rozproszony; następne cztery, kolor lustrzany; i ostatni numer, wskaźnik połysku. Ta tablica może być użyta do ustawienia wszystkich właściwości materiału:

```
glMaterialfv (GL_FRONT_AND_BACK, GL_AMBIENT, gold);
glMaterialfv (GL_FRONT_AND_BACK, GL_DIFFUSE, &gold[4]);
glMaterialfv (GL_FRONT_AND_BACK, GL_SPECULAR, &gold[8]);
glMaterialf (GL_FRONT_AND_BACK, GL_SHININESS, gold[12]);
```

Zauważ, że ostatnią funkcją jest `glMaterialf`, a nie `glMaterialfv`, a jej trzecim parametrem jest liczba, a nie wskaźnik. Coś podobnego można zrobić w Javie

```
float [] gold = {0.24725F, 0.1995F, 0.0745F, 1.0F, / * ambient * /
                 0,75164F, 0,60648F, 0,22648F, 1,0F, / * rozproszone * /
                 0,628281F, 0,555802F, 0,366065F, 1,0F, / * specular * /
                 50.0F / * po␣lysk * /
    };


gl.glMaterialfv(GL2.GL_FRONT_AND_BACK, GL2.GL_AMBIENT, gold, 0);
gl.glMaterialfv( GL2.GL_FRONT_AND_BACK, GL2.GL_DIFFUSE, gold, 4 );
gl.glMaterialfv( GL2.GL_FRONT_AND_BACK, GL2.GL_SPECULAR, gold, 8 );
gl.glMaterialf( GL2.GL_FRONT_AND_BACK, GL2.GL_SHININESS, gold[12] );
```

Funkcje `glMaterialfv` i `glMaterialf` można wywołać w dowolnym momencie, w tym między wywołaniami `glBegin` i `glEnd`. Oznacza to, że różne wierzchołki prymitywu mogą mieć różne właściwości materiału.

Więc może podoba Ci się `glColor *` lepiej niż `glMaterialfv`? Jeśli tak, możesz użyć go do pracy z materiałem, jak również zwykłym kolorem. Jeśli użyjesz

```
glEnable (GL_COLOR_MATERIAL);
```

następnie niektóre właściwości koloru materiału będą śledzić kolor. Domyślnie ustawienie koloru spowoduje również ustawienie bieżących właściwości przedniego i tylnego, otoczenia i rozproszonego materiału. To jest na przykład użycie

```
glColor3f (1, 0, 0,);
```

będzie, jeśli oświetlenie jest włączone, miało taki sam efekt jak wywołanie

```
glMaterialfv (GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, tablica);
```

gdzie tablica zawiera wartości 1, 0, 0, 1. Możesz zmienić właściwość materiału, która śledzi kolor za pomocą

```
void glColorMaterial (side, property);
```

gdzie side może być `GL_FRONT_AND_BACK, GL_FRONT` lub `GL_BACK`, a property może być `GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR, GL_EMISSION` lub `GL_AMBIENT_AND_DIFFUSE`. Ani `glEnable`, ani `glColorMaterial` nie mogą być wywoływane między wywołaniami `glBegin` i `glEnd`, więc wszystkie wierzchołki prymitywu muszą używać tego samego ustawienia.

Przypomnijmy, że gdy `glDrawArrays` lub `glDrawElements` jest używane do rysowania prymitywu, wartości kolorów wierzchołków prymitywu mogą być pobierane z tablicy kolorów, jak określono za pomocą `glColorPointer`.

Nie ma podobnych tablic właściwości materiałów. Jeśli jednak używana jest tablica kolorów, gdy włączone jest oświetlenie, a `GL_COLOR_MATERIAL` jest również włączone, wówczas tablica kolorów będzie używana jako źródło wartości właściwości materiału, które śledzą kolor.

3 Definiowanie normalnych wektorów

Wektory normalne są niezbędne do obliczeń oświetlenia. Podobnie jak kolor i materiał, wektory normalne są atrybutami wierzchołków. Stan OpenGL zawiera bieżący wektor normalny, który jest ustawiany za pomocą funkcji z rodziny `glNormal *`. Gdy wierzchołek jest określony za pomocą `glVertex *`, kopia bieżącego wektora normalnego jest zapisywana jako atrybut wierzchołka i jest używana jako normalny wektor dla tego wierzchołka, gdy kolor wierzchołka jest obliczany przez równanie oświetlenia. Zauważ, że normalny wektor dla wierzchołka musi być określony zanim `glVertex *` zostanie wywołany dla tego wierzchołka.

Funkcje w rodzinie `glNormal *` obejmują `glNormal3f, glNormal3d, glNormal3fv` i `glNormal3dv`. Jak zwykle „v" oznacza, że wartości są w tablicy, „f" oznacza, że wartości są zmiennoprzecinkowe, a „d" oznacza, że wartości są double. (Wszystkie normalne wektory mają trzy składniki). Kilka przykładów:

```
glNormal3f( 0, 0, 1 );  // (This is the default value.)
glNormal3d( 0.707, 0.707, 0.0 );
float normalArray[3] = { 0.577, 0.577, 0.577 };
glNormal3fv( normalArray );
```

Dla wielokąta, który ma wyglądać płasko, ten sam wektor normalny jest używany dla wszystkich wierzchołków wielokąta. Na przykład, aby narysować jedną stronę sześcianu, powiedzmy stronę „górną", zwróconą w kierunku dodatniej osi y:

```
glNormal3f (0, 1, 0); // Punkty wzd␣luż dodatniej osi y
glBegin (GL_QUADS);
glVertex3fv (1,1,1);
glVertex3fv (1,1, -1);
glVertex3fv (-1,1, -1);
glVertex3fv (-1,1,1);
glEnd ();
```

Pamiętaj, że normalny wektor powinien wskazywać przednią powierzchnię wielokąta, a przednia powierzchnia jest określona przez kolejność generowania wierzchołków. (Można by pomyśleć, że przednia ściana powinna być określona przez kierunek, w którym wskazuje normalny wektor, ale to nie jest sposób, w jaki się to robi. Jeśli normalny wektor wierzchołka wskazuje niewłaściwy kierunek, obliczenia oświetlenia nie zapewnią poprawnego koloru dla tego wierzchołka.)

Podczas modelowania gładkiej powierzchni, wektory normalne powinny być wybierane prostopadle do powierzchni, a nie do wielokątów, które przybliżają powierzchnię. Załóżmy, że chcemy narysować bok cylindra o promieniu 1 i wysokości 2, gdzie środek cylindra znajduje się na (0,0,0), a oś leży wzdłuż osi z . Możemy przybliżyć powierzchnię za pomocą pojedynczego paska trójkąta. Górne i dolne krawędzie boku cylindra są okręgami. Wierzchołki wzdłuż górnej krawędzi będą miały współrzędne $(\cos(a), \sin(a), 1)$, a wierzchołki wzdłuż dolnej krawędzi będą miały współrzędne $(\cos(a), \sin(a), -1)$, gdzie a jest pewnym kątem . Wektor normalny wskazuje w tym samym kierunku co promień, ale jego współrzędna Z wynosi zero, ponieważ wskazuje bezpośrednio na bok cylindra. Zatem normalny wektor po stronie cylindra w obu tych punktach będzie $(\cos(a), \sin(a), 0)$. Kiedy rysujemy bok cylindra jako trójkątny pasek, musimy generować pary wierzchołków na przemiennych krawędziach. Wektor normalny jest taki sam dla dwóch wierzchołków w parze, ale jest różny dla różnych par. Oto kod:

```
glBegin(GL_TRIANGLE_STRIP);
for (i = 0; i <= 16; i++) {
   double angle = 2*3.14159/16 * i;  // i 16-ths of a full circle
   double x = cos(angle);
   double y = sin(angle);
   glNormal3f( x, y, 0 );  // Normal for both vertices at this angle.
   glVertex3f( x, y, 1 );  // Vertex on the top edge.
   glVertex3f( x, y, -1 ); // Vertex on the bottom edge.
}
glEnd();
```

Z drugiej strony, gdy rysujemy górę i dół cylindra, chcemy płaskiego wielokąta,

z normalnym wektorem wskazującym kierunek (0,0,1) dla góry i kierunku (0,0, - 1) na dole:

```
glNormal3f( 0, 0, 1);
glBegin(GL_TRIANGLE_FAN);  // Draw the top, in the plane z = 1.
for (i = 0; i <= 16; i++) {
   double angle = 2*3.14159/16 * i;
   double x = cos(angle);
   double y = sin(angle);
   glVertex3f( x, y, 1 );
}
glEnd();

glNormal3f( 0, 0, 1 );
glBegin(GL_TRIANGLE_FAN);  // Draw the bottom, in the plane z = -1
for (i = 16; i >= 0; i--) {
   double angle = 2*3.14159/16 * i;
   double x = cos(angle);
   double y = sin(angle);
   glVertex3f( x, y, -1 );
}
glEnd();
```

Zauważ, że wierzchołki na dnie są generowane w odwrotnej kolejności niż wierzchołki na górze, aby uwzględnić fakt, że górna i dolna powierzchnia są w przeciwnych kierunkach. Jak zawsze, wierzchołki muszą być wyliczone w kolejności przeciwnej do ruchu wskazówek zegara, jak widać z przodu.

Równanie oświetlenia zakłada, że wektory normalne są normalnymi jednostkami, to znaczy mają długość równą jeden. Domyślnie w OpenGL używa się normalnych wektorów. Jeśli jednak użyjesz

```
glEnable (GL_NORMALIZE);
```

wtedy OpenGL automatycznie konwertuje każdy normalny wektor do normalnej jednostki, która wskazuje ten sam kierunek.

4 Światła

OpenGL obsługuje co najmniej osiem źródeł światła, które są identyfikowane przez stałe `GL_LIGHT0, GL_LIGHT1, ..., GL_LIGHT7`. (Implementacja OpenGL może zezwalać na dodatkowe światła.) Każde źródło światła może być skonfigurowane jako światło kierunkowe lub punktowe, a każde światło może mieć własną **intensywność rozproszoną, lustrzaną i otoczenia**.

Domyślnie wszystkie źródła światła są wyłączone. Aby włączyć światło, wywołaj `glEnable (świat␣lo)`, gdzie `świat␣lo` jest jedną ze stałych `GL_LIGHT0, GL_LIGHT1, ....` Jednak włączenie światła nie daje żadnego oświetlenia, z wyjątkiem przypadku `GL_LIGHT0`, ponieważ wszystkie natężenia światła są domyślnie

zero, z jednym wyjątkiem rozproszonego koloru światła numer 0. Aby uzyskać trochę światła z innych źródeł światła, musisz zmienić niektóre z ich właściwości. Właściwości światła można ustawić za pomocą funkcji

```
void glLightfv (int light, int property, float * valueArray);
```

Pierwszym parametrem jest jedna ze stałych `GL_LIGHT0, GL_LIGHT1, ..., GL_LIGHT7`. Określa, które światło jest konfigurowane. Drugi parametr informuje, która właściwość światła jest ustawiona. Może to być `GL_DIFFUSE, GL_SPECULAR, GL_AMBIENT` lub `GL_POSITION`. Ostatni parametr to tablica zawierająca co najmniej cztery liczby zmiennoprzecinkowe, podające wartość właściwości.

Dla właściwości koloru cztery liczby w tablicy określają składowe koloru czerwonego, zielonego, niebieskiego i alfa koloru. (Komponent alfa nie jest naprawdę używany.) Wartości od 0,0 do 1,0, ale mogą znajdować się poza tym zakresem; w rzeczywistości wartości większe niż 1,0 są czasami przydatne. Pamiętaj, że kolory rozproszone i lustrzane światła określają, jak światło oddziałuje z rozproszonymi i lustrzanymi kolorami materiału, a kolor otoczenia jest po prostu dodawany do globalnego światła otoczenia, gdy światło jest włączone. Na przykład, możesz użyć:

```
float blue1 [4] = {0,4, 0,4, 0,6, 1};
float blue2 [4] = {0,0, 0, 0,8, 1};
float blue3 [4] = {0,0, 0, 0,15, 1};
glLightfv (GL_LIGHT1, GL_DIFFUSE, blue1);
glLightfv (GL_LIGHT1, GL_SPECULAR, blue2);
glLightfv (GL_LIGHT1, GL_AMBIENT, blue3);
```

Właściwość `GL_POSITION` światła jest nieco inna. Służy zarówno do ustalenia, czy światło jest światłem punktowym, czy kierunkowym, oraz do ustawienia jego pozycji lub kierunku. Wartość właściwości dla `GL_POSITION` jest tablicą czterech liczb `(x, y, z, w)`, z których co najmniej jedna musi być niezerowa.

Gdy czwarta liczba, `w`, jest równa zero, to światło jest **kierunkowe**, a punkt `(x, y, z)` określa kierunek światła: promienie świetlne świecą w kierunku linii od punktu (x, y, z) w kierunku początku współrzędnych. Jest to związane z jednorodnymi współrzędnymi: Źródło światła można uznać za punkt w nieskończoności w kierunku `(x, y, z)`.

Z drugiej strony, jeśli czwarta liczba, `w`, jest niezerowa, to światło jest światłem **punktowym** i znajduje się w punkcie `(x / w, y / w, z / w)`. Zazwyczaj `w` wynosi 1. Wartość `(x, y, z, 1)` daje światło punktowe w `(x, y, z)`. Ponownie, jest to naprawdę jednorodne współrzędne.

Domyślne położenie dla wszystkich świateł to (0,0,1,0), reprezentujące światło kierunkowe świecące od dodatniego kierunku osi Z, w kierunku ujemnego kierunku osi Z.

Ważnym i potencjalnie mylącym faktem dotyczącym świateł jest to, że pozycja określona dla światła jest przekształcana przez transformację widoku modelu, która obowiązuje w momencie ustawienia pozycji za pomocą `glLightfv`. Innym sposobem powiedzenia tego jest ustawienie pozycji we współrzędnych

oka, a nie we współrzędnych światowych. Wywołanie `glLightfv` z właściwością ustawioną na `GL_POSITION` jest bardzo podobne do wywołania `glVertex *`. Pozycja światła jest przekształcana w taki sam sposób, w jaki zostaną przekształcone współrzędne wierzchołków. Na przykład

```
float position[4] = { 1,2,3,1 }
glLightfv(GL_LIGHT1, GL_POSITION, position);
```

umieszcza światło w tym samym miejscu co

```
glTranslatef(1,2,3);
float position[4] = { 0,0,0,1 }
glLightfv(GL_LIGHT1, GL_POSITION, position);
```

Dla światła kierunkowego kierunek światła jest przekształcany przez część obrotową transformacji widoku modelu.

5 Właściwości oświetlenia globalnego

Oprócz właściwości poszczególnych źródeł światła system oświetlenia OpenGL wykorzystuje kilka właściwości globalnych. W OpenGL 1.1 są tylko trzy takie właściwości. Jedną z nich jest globalne światło otoczenia, które jest światłem otoczenia, które nie pochodzi z właściwości koloru otoczenia żadnego źródła światła. Globalne światło otoczenia będzie obecne w środowisku, nawet jeśli wszystkie `GL_LIGHT0, GL_LIGHT1, ...` są wyłączone. Domyślnie globalne światło otoczenia jest czarne (wszystkie komponenty RGB są zerowe). Wartość można zmienić za pomocą funkcji

```
void glLightModelfv( int property, float* value )
```

gdzie `property` musi być `GL_LIGHT_MODEL_AMBIENT`, a `value` jest tablicą zawierającą cztery liczby, które odpowiadają kolorowym komponentom RGBA globalnego światła otoczenia jako liczby z zakresu od 0,0 do 1,0. Poziom oświetlenia globalnego otoczenia powinien być dość niski. Na przykład w C:

```
float ambientLevel[] = { 0.15, 0.15, 0.15, 1 };
glLightModelfv( GL_LIGHT_MODEL_AMBIENT, ambientLevel );
```

Składnik alfa koloru jest zwykle ustawiony na 1, ale nie jest używany do niczego. W przypadku JOGL, jak zwykle, istnieje dodatkowy parametr określający początkowy indeks danych w tablicy:

```
float[] ambientLevel = { 0.15F, 0.15F, 0.15F, 0 };
gl.glLightModelfv( GL2.GL_LIGHT_MODEL_AMBIENT, ambientLevel, 0 );
```

Pozostałe dwie opcje, które mogą być wyłączone lub włączone. Właściwości to `GL_LIGHT_MODEL_TWO_SIDE` i `GL_LIGHT_MODEL_LOCAL_VIEWER`. Można je ustawić za pomocą funkcji

```
void glLightModeli( int property, int value )
```

o wartości równej 0 lub 1, aby wskazać, czy opcja powinna być wyłączona, czy włączona. Możesz użyć stałych symbolicznych `GL_FALSE` i `GL_TRUE` dla tej wartości, ale są to tylko nazwy dla 0 i 1.

`GL_LIGHT_MODEL_TWO_SIDE` służy do włączania oświetlenia dwustronnego. Przypomnij sobie, że wielokąt może mieć dwa zestawy właściwości materiału, materiał przedni i materiał tylny. Gdy oświetlenie dwustronne jest wyłączone, co jest domyślne, używany jest tylko materiał przedni; Służy zarówno do przedniej, jak i tylnej powierzchni wielokąta. Ponadto ten sam wektor normalny jest używany dla obu ścian. Ponieważ te wektory wskazują lub przynajmniej mają wskazywać przednią powierzchnię, nie dają poprawnego wyniku dla tylnej powierzchni. W efekcie tylna strona wygląda jak oświetlona przez źródła światła, które leżą przed wielokątem, ale tylna strona powinna być oświetlona światłami, które leżą za wielokątem.

Z drugiej strony, gdy oświetlenie dwustronne jest włączone, materiał tylny jest używany na tylnej powierzchni, a wektor normalny jest odwracany, gdy jest używany do oświetlenia w obliczeniach dla tylnej ściany.

Powinieneś używać oświetlenia dwustronnego za każdym razem, gdy na twojej scenie są widoczne tylne powierzchnie. (W przypadku oświetlenia dwustronnego można użyć tego samego materiału na obu powierzchniach lub określić różne materiały dla dwóch powierzchni. Na przykład, aby umieścić lśniący purpurowy materiał na przednich ścianach i matowy żółty materiał na tylnych ścianach:

```
glLightModeli( GL_LIGHT_MODEL_TWO_SIDE, 1 ); // Turn on two-sided lighting.

float purple[] = { 0.6, 0, 0.6, 1 };
float yellow[] = { 0.6, 0.6, 0, 1 };
float white[] = { 0.4, 0.4, 0.4, 1 }; // For specular highlights.
float black[] = { 0, 0, 0, 1 };


glMaterialfv( GL_FRONT, GL_AMBIENT_AND_DIFFUSE, purple );  // front material
glMaterialfv( GL_FRONT, GL_SPECULAR, white );
glMaterialf( GL_FRONT, GL_SHININESS, 64 );

glMaterialfv( GL_BACK, GL_AMBIENT_AND_DIFFUSE, yellow );  // back material
glMaterialfv( GL_BACK, GL_SPECULAR, black );  // no specular highlights
```

Trzecia właściwość materiału, `GL_LIGHT_MODEL_LOCAL_VIEWER`, jest znacznie mniej ważna. Ma to związek z kierunkiem od powierzchni do widza w równaniu oświetlenia. Domyślnie kierunek ten jest zawsze kierowany bezpośrednio na ekran, co jest prawdą dla rzutu ortograficznego, ale nie zapewnia dokładnej precyzji dla projekcji perspektywicznej. Jeśli włączysz opcję widoku lokalnego, użyty zostanie prawdziwy kierunek do widoku. W praktyce różnica zwykle nie jest zauważalna.

Literatura

Uwaga!

Światło i materiały w języku C
https://drive.google.com/open?id=1EQBH5FawTNUFFo1l99Iv8kxz7oEtS1nkeGHY7S2pSd8
W języku angielskim

- książka interakcyjna: Lighting and Materials http://math.hws.edu/graphicsbook/c4/s1.html, http://math.hws.edu/graphicsbook/c4/s2.html (rozdziały 4.1-4.2)

6 Zadanie

Celem jest stworzenie piramidy z użyciem różnych materiałów i umieszczenie jej na „podstawie". Użytkownik może obracać podstawę wokół osi Y, przeciągając mysz w poziomie. Scena wykorzystuje oświetlenie. Początkowo włączone jest tylko podstawowe oświetlenie. W ramach laboratorium będziesz musiał poprawić oświetlenie.

Możesz wykonać laboratorium w Javie lub C. Aby wykonać laboratorium w Javie, potrzebujesz plików Lab6.java. Aby wykonać laboratorium w C, potrzebujesz plik lab6.c.