Grafika komputerowa

Sprawozdanie z laboratorium 4 i 5

Tematy: Modelowanie 3D; Modelowanie oświetlenia w OpenGL

Student: Bartosz Baniak
Nr albumu: 80512
Nr w grupie: 1

Grupa: WCY21IJ2S1
Daty laboratoriów: 6.12.2023r.; 14.12.2023 r.
Prowadzący: dr inż. Marek Salamon

1. Treść zadania

Laboratorium 4

1. Wykorzystując projekt „**szescian**” napisać fragment programu:
   a) generujący osie układu współrzędnych *XYZ* w kolorach (*r, g, b)* ; 0.5 pkt.
   b) umożliwiający zmianę odległości obserwatora od obiektu w zakresie (*odlmin, odlmax*); 0.5 pkt.
   c) obrót obserwatora w pełnym zakresie (od 0 do 360 stopni) wokół osi *0X, 0Y i 0Z*. 0.5 pkt.
2. Wykorzystując funkcje ***GL_QUAD_STRIP*** napisać program przedstawiający obraz perspektywiczny stożka ściętego o promieniu dolnej podstawy ***R*** , promieniu górnej podstawy ***r*** i wysokości ***h*** . Podstawy stożka wygenerować za pomocą funkcji ***GL_TRIANGLE_FAN***. Podstawa stożka leży na płaszczyźnie *XZ*, oś stożka łączy punkty (*0,0,0*) i (*0,h,0).* 1.5 pkt.
3. Wprowadzić możliwość interakcyjnej zmiany liczby podziałów pionowych bryły w zakresie od 4 do 64. 1 pkt.
4. /zadanie indywidualne/ 4 pkt.

Ocena końcowa: 4 pkt. DST
5 pkt. DST+
6 pkt. DB
7 pkt. DB+
8 pkt. BDB

1.(X2) sfera w trybie GL_TRIANGLES o promieniu 3 i środku w punkcie (0, 0, 0),

Laboratorium 5

Wykorzystując biblioteki *OpenGL* i *GLUT* oraz plik *materials.h* napisać program przedstawiający perspektywiczny obraz **bryły z ćwicz. 4** pokrytej materiałem o podanych właściwościach. Bryłę należy oświetlić dwoma źródłami światła o podanych parametrach.

Program powinien umożliwiać:

1. Zobrazowanie modelu geometrycznego bryły; (*glPolygonMode*) 0.5 pkt.
2. Wyznaczenie wektorów normalnych; (*zobrazowanie w celu weryfikacji poprawności)* 1 pkt.
3. Niezależne włączanie i wyłączanie źródeł światła (2 światła); 1 pkt.
4. Zmianę materiału bryły (3 materiały); 1 pkt.
5. Zmianę trybu cieniowania; (*glShadeModel*) 0.5 pkt.
6. Zmianę liczby podziałów bryły; /ćwicz.4 brak -0.5 pkt./
7. Zmianę położenia obserwatora; /ćwicz.4 brak -0.5 pkt./
8. Zobrazowanie menu programu. 0.5 pkt.

Zadanie 1

Materiały:
1. Właściwości materiału nr 1: **fioletowy błyszczący** (widziany w białym świetle),
2. Właściwości materiału nr 2: **żółty matowy** (widziany w białym świetle),
3. Właściwości materiału nr 3: **mosiądz**

Źródła światła:

Źródło nr 1:
- typ: **reflektor** (ang. *spot*),
- kolor: **biały**,
- natężenie: **1**,
- kąt odcięcia: **30°**,
- 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: **zielony**,
- natężenie: **0.7**,
- położenie: **stałe** w punkcie P(10,10, 10) **układu współrzędnych obserwatora**.
- kierunek świecenia: **na obiekt**.

2. Sposób rozwiązania

Laboratorium 4

1) Funkcja RysujOsie() służy do rysowania trójwymiarowego układu współrzędnych. W tej funkcji, za pomocą prostych linii, reprezentowane są trzy osie: X, Y i Z. Każda oś ma inny kolor: oś X jest czerwona, Y – zielona, a Z – niebieska, co pomaga w ich rozróżnieniu. Osie rozciągają się zarówno w dodatnich, jak i ujemnych kierunkach, tworząc krzyż w centrum układu współrzędnych. Długość każdej osi to 160 jednostek, z centrum układu współrzędnych umieszczonym w punkcie (0,0,0).
   Instrukcje case 's': i case 'w': są częścią instrukcji warunkowej znajdującej się w funkcji ObslugaKlawiatury, która kontroluje zmienną odlObserwatora, odpowiadającą za odległość obserwatora. Kiedy użytkownik naciśnie klawisz 's', odległość obserwatora jest zwiększana o 1.0, ale nie więcej niż do maksymalnego limitu odlmax. Analogicznie, naciśnięcie 'w' zmniejsza tę odległość o 1.0, ale nie poniżej minimalnego limitu odlmin. To pozwala na interaktywne zbliżanie się lub oddalanie od sceny 3D.
   W celu umożliwienia obrotu obserwatora w pełnym zakresie zostały usunięte ograniczenia na klawisze w funkcji ObslugaKlawiszySpecjalnych.
2) Funkcja RysujScietyStozek, wizualizuje ścięty stożek, który składa się z dwóch płaskich podstaw i ściennego boku. Podstawy są kreślone jako koła, wykorzystując tryb GL_TRIANGLE_FAN, co oznacza, że tworzone są wachlarze trójkątów z jednego centralnego punktu. Dolna podstawa ma większy promień R (połowa liter w nazwisku), a górna mniejszy r (połowa liter w imieniu), co nadaje stożkowi ścięty kształt. Boki stożka są tworzone przez GL_QUAD_STRIP, czyli serię czworokątów, które łączą się ze sobą wzdłuż krawędzi stożka, tworząc jego powierzchnię boczną. Każdy czworokąt jest tworzony przez połączenie dwóch punktów na dolnej podstawie i dwóch na górnej, przy czym każdy nowy czworokąt dzieli jeden bok z poprzednim, tworząc płynną, ciągłą powierzchnię.
3) Liczba trójkątów użyta do narysowania podstaw (numTriangles) bezpośrednio wpływa na płynność i okrągłość tych podstaw. Większa liczba trójkątów sprawia, że podstawy są bardziej okrągłe i mniej wielokątne.
4) Funkcja RysujSfere() służy do tworzenia sfery przez rysowanie wielu trójkątów. Sfera jest podzielona na poziome warstwy (stacks), i pionowe segmenty (slices). Dzięki temu podziałowi, cała powierzchnia sfery jest pokryta siatką trójkątów.
   Wewnątrz dwóch zagnieżdżonych pętli for, program oblicza współrzędne wierzchołków dla każdego trójkąta sfery. Współrzędne te są wyznaczane na podstawie kątów theta i phi, które odpowiadają położeniu punktu na sferze w koordynatach biegunowych. Kąty te są proporcjonalnie rozmieszczone w zależności od liczby stosów i wycinków.
   Każda iteracja wewnętrznej pętli tworzy dwa trójkąty, które razem formują czworobok (fragment sfery).

Laboratorium 5

1) Program pozwala na zmianę trybu wyświetlania brył geometrycznych między pełnym wypełnieniem a wyświetlaniem tylko krawędzi. Zmiana ta jest realizowana poprzez użycie funkcji glPolygonMode(GL_FRONT_AND_BACK, mode), gdzie mode może przyjmować wartości GL_FILL (pełne wypełnienie) lub GL_LINE (tylko krawędzie). Tryb jest zmieniany przez użytkownika za pomocą klawiszy 'f' i 'g', co aktualizuje zmienną polygonMode.
2) W programie obliczane są wektory normalne dla powierzchni sfery, co jest kluczowe dla poprawnego oświetlenia i renderowania bryły. W funkcji RysujSfere, dla każdego punktu na powierzchni sfery, obliczany jest wektor normalny. W funkcji RysujWektoryNormalne możliwe jest wyświetlenie tych wektorów normalnych, co pomaga w zrozumieniu, jak są one rozłożone na bryle.
3) Program obsługuje dwa niezależne źródła światła. Użytkownik może włączać i wyłączać każde światło osobno, korzystając z klawiszy '1'/'!' dla pierwszego światła oraz '2'/'@' dla drugiego światła. Funkcja WlaczOswietlenie odpowiednio reaguje na te zmiany, aktywując lub deaktywując źródła światła w scenie.
4) Użytkownik ma możliwość wyboru spośród trzech różnych materiałów dla renderowanej bryły. Każdy materiał ma inne właściwości odbicia światła. Funkcja ZmienMaterial realizuje te zmiany, pozwalając na dynamiczną zmianę wyglądu bryły w zależności od wybranego materiału.
5) Tryb cieniowania określa, jak kolory i oświetlenie są interpolowane na powierzchni bryły. Użytkownik może przełączać między trybem płaskim (GL_FLAT) a gładkim (GL_SMOOTH) za pomocą klawiszy '6' i '^'. Funkcja ObslugaKlawiatury obsługuje tę zmianę.
6) Program umożliwia zmianę liczby podziałów sfery, co wpływa na jej szczegółowość i gładkość. Więcej podziałów oznacza bardziej szczegółową i gładką bryłę. Użytkownik może zwiększać lub zmniejszać liczbę podziałów za pomocą klawiszy ',' i '.'.
7) Ustawienia kamery są kontrolowane przez użytkownika, który może zmieniać jej położenie i orientację. To pozwala na lepsze oglądanie bryły z różnych perspektyw. Klawisze 'w', 's', 'a', 'd', 'u', 'o', '=' i '-' służą do manipulowania położeniem i orientacją kamery.
8) Funkcja RysujMenu renderuje tekstowe menu na ekranie. Menu to informuje użytkownika o dostępnych opcjach sterowania. Tekst jest renderowany jako nakładka na scenę 3D.

3. Wyniki

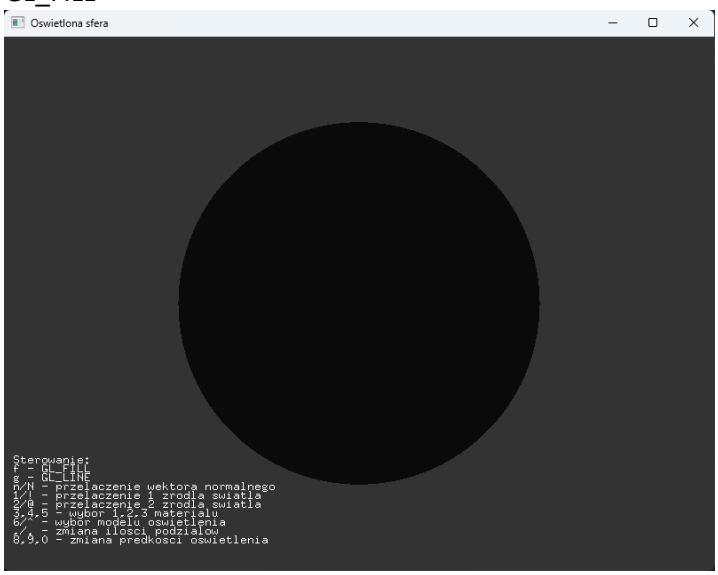
1. Zobrazowanie modelu geometrycznego bryły

Kod:

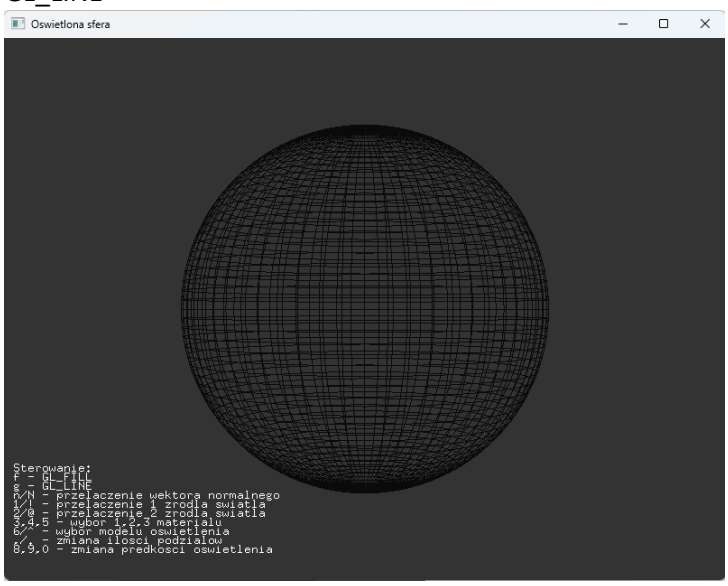
```
void RysujSfere(float promien, int liczbaPodzialow) {


    for (int i = 0; i < liczbaPodzialow; ++i) {
        double theta1 = i * PI / liczbaPodzialow;
        double theta2 = (i + 1) * PI / liczbaPodzialow;

        glBegin(GL_QUAD_STRIP);
        for (int j = 0; j <= liczbaPodzialow; ++j) {
            double phi = j * 2 * PI / liczbaPodzialow;

            double phi2 = (j + 1) * 2 * PI / liczbaPodzialow;

            float x = promien * cos(phi) * sin(theta1);
            float y = promien * cos(theta1);
            float z = promien * sin(phi) * sin(theta1);

            float x0 = promien * cos(phi) * sin(theta1);
            float y0 = promien * cos(theta1);
            float z0 = promien * sin(phi) * sin(theta1);

            float x1 = promien * cos(phi) * sin(theta2);
            float y1 = promien * cos(theta2);
            float z1 = promien * sin(phi) * sin(theta2);

            float x2 = promien * cos(phi2) * sin(theta1);
            float y2 = promien * cos(theta1);
            float z2 = promien * sin(phi2) * sin(theta1);

            float ax = x2 - x0;
            float ay = y2 - y0;
            float az = z2 - z0;

            float bx = x1 - x0;
            float by = y1 - y0;
            float bz = z1 - z0;

            float cx = ay * bz - az * by;
            float cy = az * bx - ax * bz;
            float cz = ax * by - ay * bz;

            glNormal3f(x, y, z);
            glVertex3f(x, y, z);

            x = promien * cos(phi) * sin(theta2);
            y = promien * cos(theta2);
            z = promien * sin(phi) * sin(theta2);


            glVertex3f(x, y, z);

        }
        glEnd();

        double theta = PI / 2;
        glBegin(GL_TRIANGLE_FAN);
        glNormal3f(0, -1, 0);

        glVertex3f(0.0, 0.0, 0.0);

        for (int i = 0; i <= liczbaPodzialow; ++i) {
            double phi = i * 2 * PI / liczbaPodzialow;
            double x = promien * cos(phi) * sin(theta);
            double y = promien * cos(theta);
            double z = promien * sin(phi) * sin(theta);

            glVertex3f(x, y, z);

        }
        glEnd();
    }
}
```

```
case 'f':
    strcpy(polygonMode, "GL_FILL");
    break;
case 'g':
    strcpy(polygonMode, "GL_LINE");
    break;
```

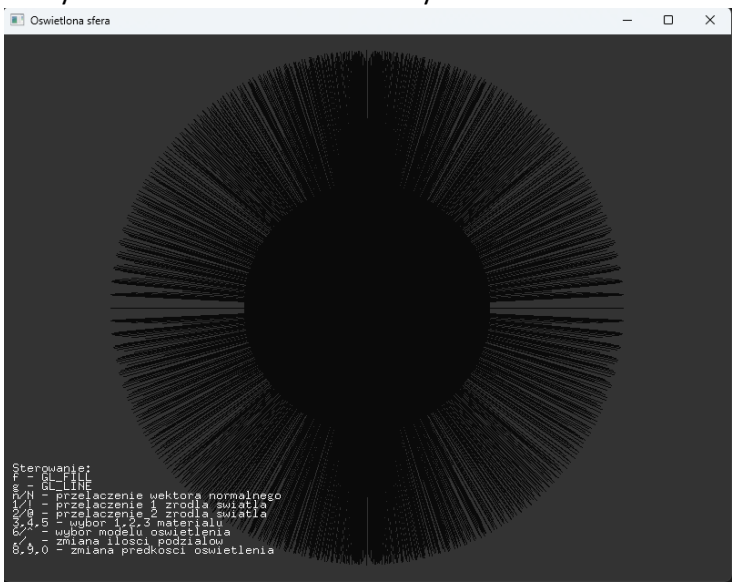
GL_FILL



GL_LINE



2. Wyznaczenie wektorów normalnych



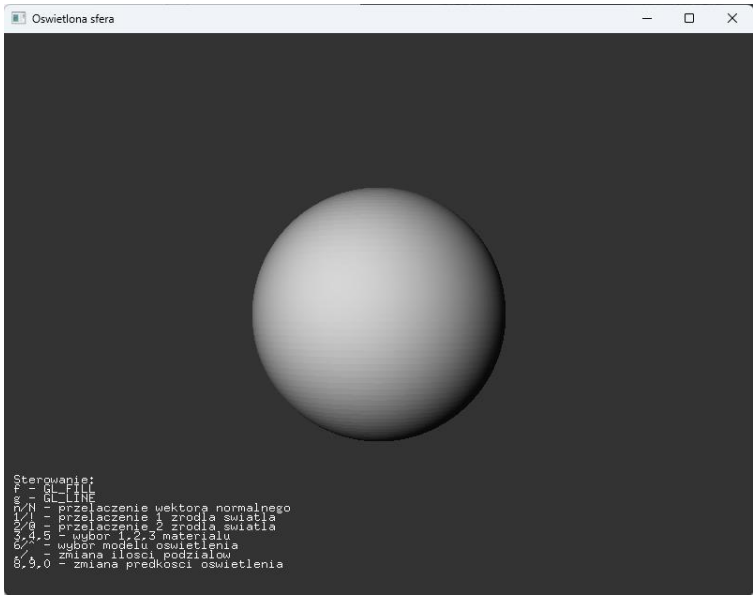
Kod:

```
void RysujWektoryNormalne(float promien, int liczbaPodzialow) {

    if (normalVector == 1) glColor3b(1, 1, 1);
    if (normalVector == 0) glColor3b(0, 0, 0);

        for (int i = 0; i < (liczbaPodzialow) + 1; ++i) {
            double theta1 = i * PI / liczbaPodzialow;
            double theta2 = (i + 1) * PI / liczbaPodzialow;


            for (int j = 0; j <= liczbaPodzialow; ++j) {
                glBegin(GL_LINES);
                double phi = j * 2 * PI / liczbaPodzialow;

                double phi2 = (j + 1) * 2 * PI / liczbaPodzialow;

                float x = promien * cos(phi) * sin(theta1);
                float y = promien * cos(theta1);
                float z = promien * sin(phi) * sin(theta1);

                float x0 = promien * cos(phi) * sin(theta1);
                float y0 = promien * cos(theta1);
                float z0 = promien * sin(phi) * sin(theta1);

                float x1 = promien * cos(phi) * sin(theta2);
                float y1 = promien * cos(theta2);
                float z1 = promien * sin(phi) * sin(theta2);

                float x2 = promien * cos(phi2) * sin(theta1);
                float y2 = promien * cos(theta1);
                float z2 = promien * sin(phi2) * sin(theta1);

                float ax = x2 - x0;
                float ay = y2 - y0;
                float az = z2 - z0;

                float bx = x1 - x0;
                float by = y1 - y0;
                float bz = z1 - z0;

                float cx = ay * bz - az * by;
                float cy = az * bx - ax * bz;
                float cz = ax * by - ay * bz;


                glVertex3f(x, y, z);
                //glVertex3f(cx + x, cy + y, cz + z);
                glVertex3f(2 * x, 2 * y, 2 * z);

                x = promien * cos(phi) * sin(theta2);
                y = promien * cos(theta2);
                z = promien * sin(phi) * sin(theta2);
                glEnd();
            }

            glBegin(GL_LINES);
            glVertex3f(0, 0, 0);
            glVertex3f(0, -1, 0);
            glVertex3f(0, promien, 0);
            glVertex3f(0, promien+1, 0);
            glEnd();
            double theta = PI / 2;
        }
}
```

```
case 'n':
    normalVector = 1;
    break;
case 'N':
    normalVector = 0;
    break;
```

3. Definicje źródeł światła z zadania indywidualnego

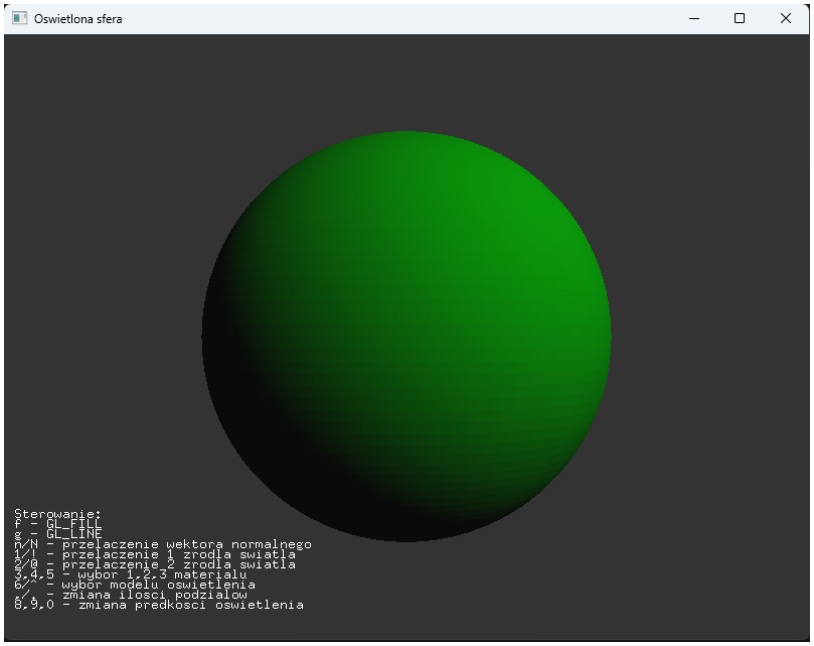
Światło nr 1:



Kod:

```
static float timeElapsed = 0.0f;
static float light1TimeElapsed = 0.0f;
float light1TimeStep = orbitVelocity;

timeElapsed += angularVelocity; // Aktualizacja czasu globalnego
light1TimeElapsed += light1TimeStep; // Aktualizacja czasu dla ruchu orbity światła 1

// Światło 1 - Reflektor
if (swiatlo1 == 1) {
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT1);

    GLfloat light1Diffuse[] = { 1.0f, 1.0f, 1.0f, 1.0f }; // Kolor biały
    GLfloat light1Position[] = {
        orbitRadius * cos(light1TimeElapsed),
        orbitRadius * sin(light1TimeElapsed) * cos(Deg2Rad(orbitInclination)),
        orbitRadius * sin(light1TimeElapsed) * sin(Deg2Rad(orbitInclination)),
        1.0f
    };
    GLfloat light1Direction[] = { -light1Position[0], -light1Position[1], -light1Position[2] };

    glLightfv(GL_LIGHT1, GL_DIFFUSE, light1Diffuse);
    glLightfv(GL_LIGHT1, GL_POSITION, light1Position);
    glLightfv(GL_LIGHT1, GL_SPOT_DIRECTION, light1Direction);
    glLightf(GL_LIGHT1, GL_SPOT_CUTOFF, 30.0f); // Kąt odcięcia 30 stopni

    glLightf(GL_LIGHT1, GL_CONSTANT_ATTENUATION, 1.0f);
    glLightf(GL_LIGHT1, GL_LINEAR_ATTENUATION, 0.0f);
    glLightf(GL_LIGHT1, GL_QUADRATIC_ATTENUATION, 0.0f);

    glEnable(GL_DEPTH_TEST);
}
else {
    glDisable(GL_LIGHT1);
}
```

Światło nr 2:

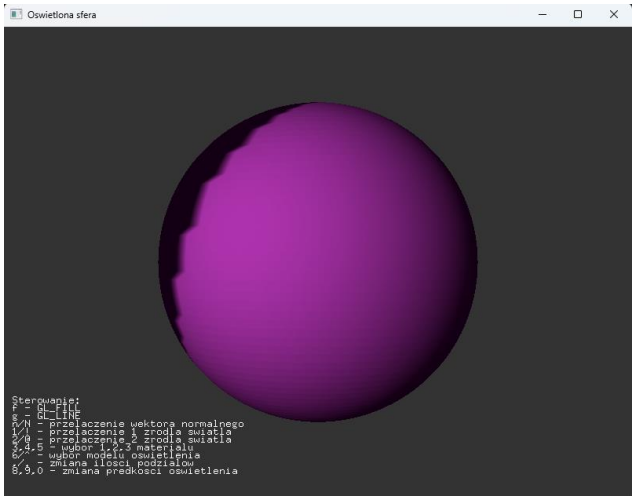


Kod:

```
// Światło 2 - Kierunkowe
if (swiatlo2 == 1) {
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT2);

    GLfloat light2Diffuse[] = { 0.0f, 0.7f, 0.0f, 1.0f }; // Kolor zielony
    GLfloat light2Position[] = { 10.0f, 10.0f, 10.0f, 0.0f }; // Światło kierunkowe
    GLfloat light2Direction[] = { -10.0f, -10.0f, -10.0f };

    glLightfv(GL_LIGHT2, GL_DIFFUSE, light2Diffuse);
    glLightfv(GL_LIGHT2, GL_POSITION, light2Position);
    glLightfv(GL_LIGHT2, GL_SPOT_DIRECTION, light2Direction);

    glLightf(GL_LIGHT2, GL_CONSTANT_ATTENUATION, 0.7f);
    glLightf(GL_LIGHT2, GL_LINEAR_ATTENUATION, 0.0f);
    glLightf(GL_LIGHT2, GL_QUADRATIC_ATTENUATION, 0.0f);

    glEnable(GL_DEPTH_TEST);
}
else {
    glDisable(GL_LIGHT2);
}
```

4. Definicje materiałów z zadania indywidualnego:

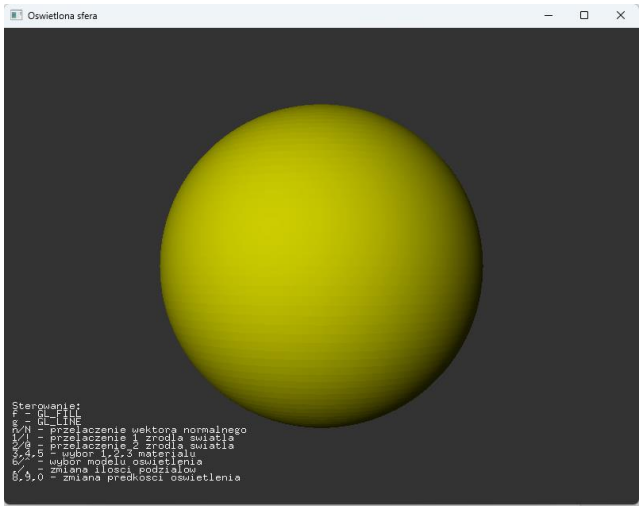
Materiał 1:



Kod:

```
case 1:

    // fioletowy błyszczący (widziany w białym świetle)
    GLfloat firstAmbient[] = { 0.4, 0.0, 0.4, 1.0 };
    GLfloat firstDiffuse[] = { 0.6, 0.2, 0.6, 1.0 };
    GLfloat firstSpecular[] = { 1.0, 0.7, 1.0, 1.0 };
    GLfloat firstShininess = 76.8;

    glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, firstAmbient);
    glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, firstDiffuse);
    glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, firstSpecular);
    glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, firstShininess);
    break;
```

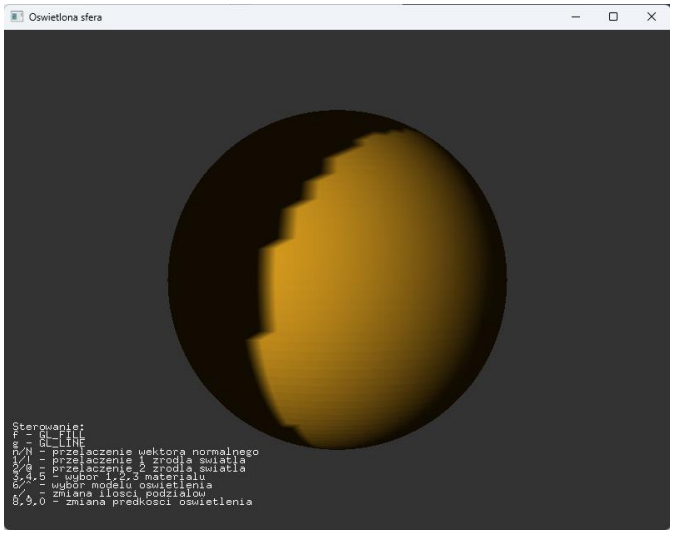
Materiał 2:



Kod:

```
case 2:
    // żółty matowy (widziany w białym świetle)
    GLfloat secondAmbient[] = { 0.5, 0.5, 0.0, 1.0 };
    GLfloat secondDiffuse[] = { 0.7, 0.7, 0.0, 1.0 };
    GLfloat secondSpecular[] = { 0.0, 0.0, 0.0, 1.0 };
    GLfloat secondShininess = 0.0;

    glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, secondAmbient);
    glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, secondDiffuse);
    glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, secondSpecular);
    glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, secondShininess);
    break;
```

Materiał 3:

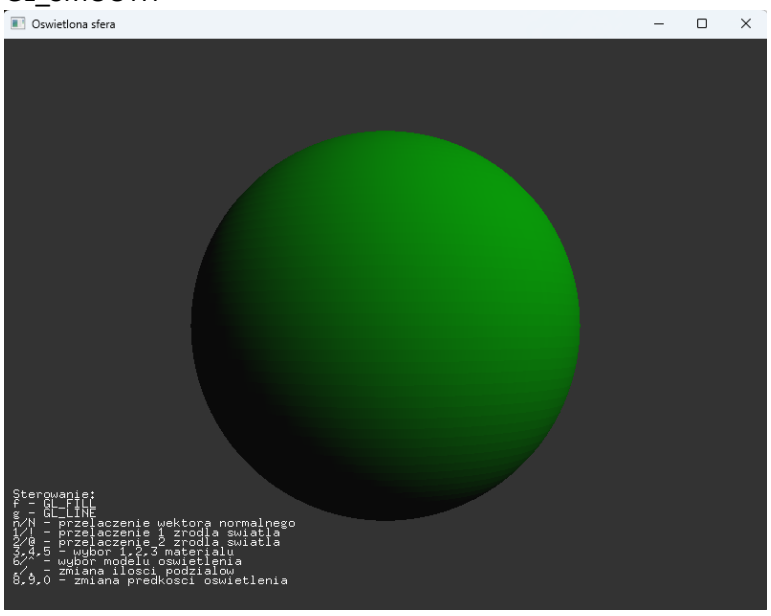


Kod:

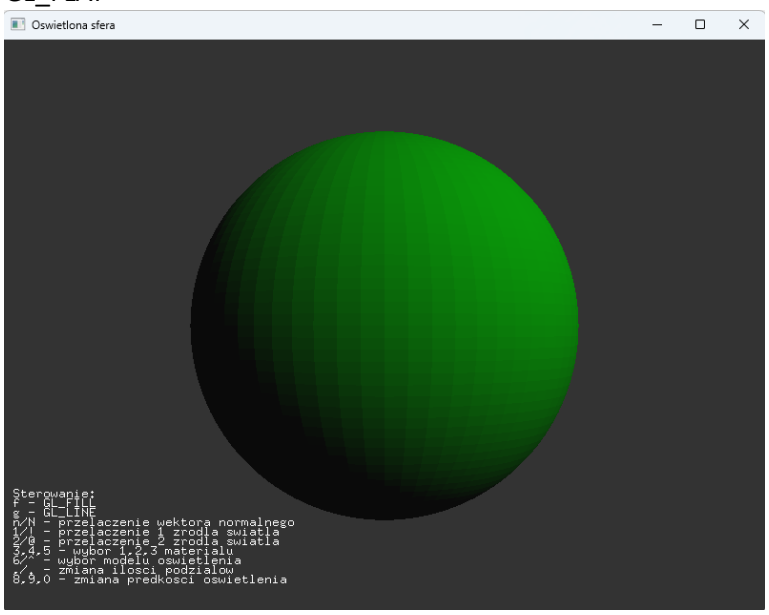
```
case 3:
    // mosiadz
    GLfloat thirdAmbient[] = { 0.329412, 0.223529, 0.027451, 1.000000 };
    GLfloat thirdDiffuse[] = { 0.780392, 0.568627, 0.113725, 1.000000 };
    GLfloat thirdSpecular[] = { 0.992157, 0.941176, 0.807843, 1.000000 };
    GLfloat thirdShininess = 27.8974;;

    glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, thirdAmbient);
    glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, thirdDiffuse);
    glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, thirdSpecular);
    glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, thirdShininess);
    break;
```

5. Zmiana trybu cieniowania

GL_SMOOTH



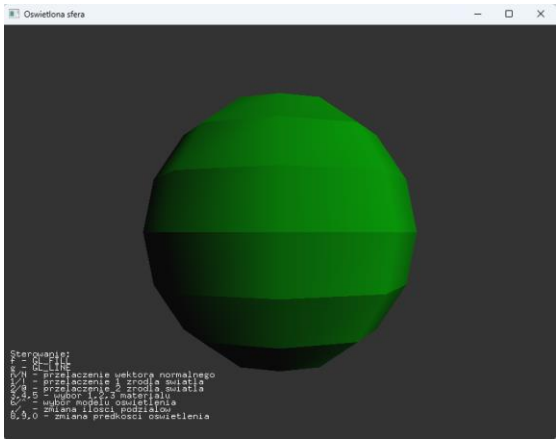
GL_FLAT

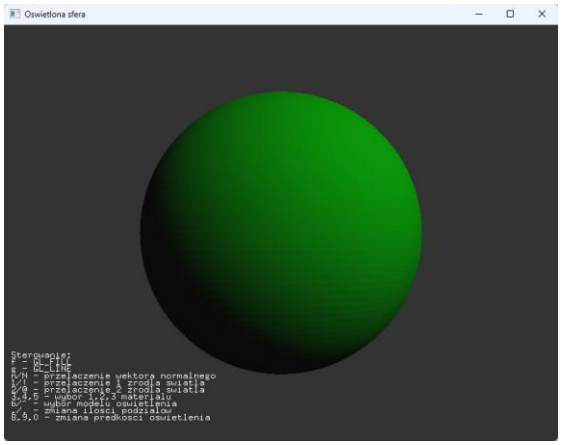


Kod:

```
case '6':
    shadeModel = GL_SMOOTH;
    break;
case '^':
    shadeModel = GL_FLAT;
    break;
```

6. Zmiana liczby podziałów

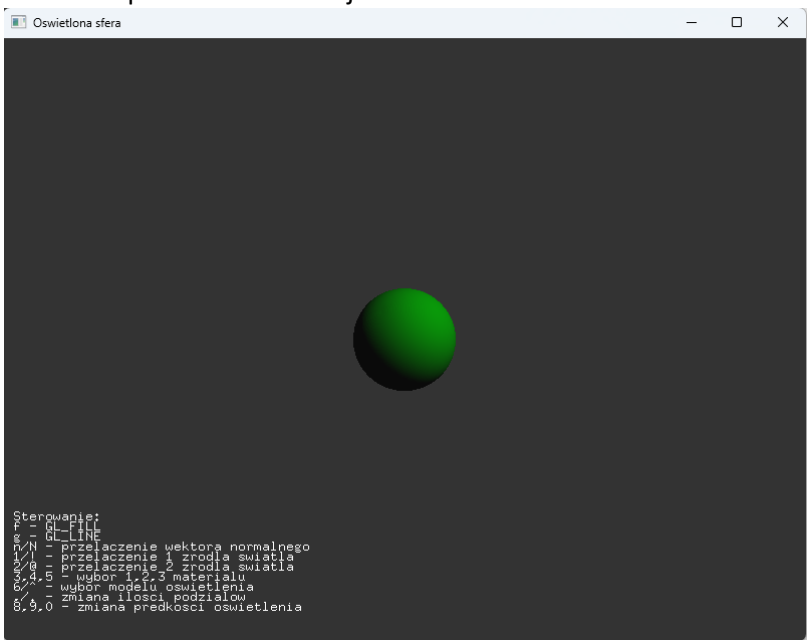




Kod:

```
case '.':
    podzialN = (podzialN < podzialNmax) ? podzialN + 2.0 : podzialN;
    break;
case ',':
    podzialN = (podzialN > podzialNmin) ? podzialN - 2.0 : podzialN;
    break;
```

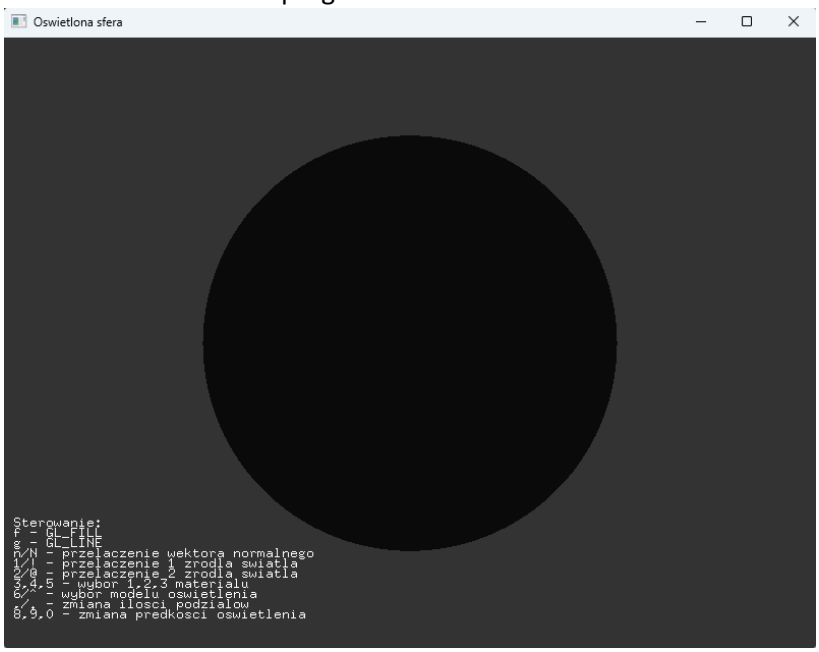
7. Zmiana położenia orientacji obserwatora



Kod:

```
case 'w':
    rotObsX = rotObsX + 1.0;
    break;
case 's':
    rotObsX = rotObsX - 1.0;
    break;
case 'a':
    rotObsY = rotObsY - 1.0;
    break;
case 'd':
    rotObsY = rotObsY + 1.0;
    break;
case 'u':
    rotObsZ = rotObsZ - 1.0;
    break;
case 'o':
    rotObsZ = rotObsZ + 1.0;
    break;
case '=':
    odl = (odl > odlmin) ? odl - 1.0 : odl;
    break;
case '-':
    odl = (odl < odlmax) ? odl + 1.0 : odl;
    break;
```

8. Zobrazowanie menu programu



Kod:

```
sprintf(buf, "Sterowanie: ");
glRasterPos2i(X_OFFSET_SWIATLO, Y_OFFSET_SWIATLO);
RysujTekstRastrowy(GLUT_BITMAP_8_BY_13, buf);


sprintf(buf, "f - GL_FILL");
glRasterPos2i(X_OFFSET_SWIATLO, Y_OFFSET_SWIATLO - 10);
RysujTekstRastrowy(GLUT_BITMAP_8_BY_13, buf);


sprintf(buf, "g - GL_LINE");
glRasterPos2i(X_OFFSET_SWIATLO, Y_OFFSET_SWIATLO - 20);
RysujTekstRastrowy(GLUT_BITMAP_8_BY_13, buf);


sprintf(buf, "n/N - przelaczenie wektora normalnego");
glRasterPos2i(X_OFFSET_SWIATLO, Y_OFFSET_SWIATLO - 30);
RysujTekstRastrowy(GLUT_BITMAP_8_BY_13, buf);


sprintf(buf, "1/! - przelaczenie 1 zrodla swiatla");
glRasterPos2i(X_OFFSET_SWIATLO, Y_OFFSET_SWIATLO - 40);
RysujTekstRastrowy(GLUT_BITMAP_8_BY_13, buf);


sprintf(buf, "2/@ - przelaczenie 2 zrodla swiatla");
glRasterPos2i(X_OFFSET_SWIATLO, Y_OFFSET_SWIATLO - 50);
RysujTekstRastrowy(GLUT_BITMAP_8_BY_13, buf);


sprintf(buf, "3,4,5 - wybor 1,2,3 materialu");
glRasterPos2i(X_OFFSET_SWIATLO, Y_OFFSET_SWIATLO - 60);
RysujTekstRastrowy(GLUT_BITMAP_8_BY_13, buf);
```

```
sprintf(buf, "6/^ - wybor modelu oswietlenia");
glRasterPos2i(X_OFFSET_SWIATLO, Y_OFFSET_SWIATLO - 70);
RysujTekstRastrowy(GLUT_BITMAP_8_BY_13, buf);



sprintf(buf, ",/. - zmiana ilosci podzialow");
glRasterPos2i(X_OFFSET_SWIATLO, Y_OFFSET_SWIATLO - 80);
RysujTekstRastrowy(GLUT_BITMAP_8_BY_13, buf);


sprintf(buf, "8,9,0 - zmiana predkosci oswietlenia");
glRasterPos2i(X_OFFSET_SWIATLO, Y_OFFSET_SWIATLO - 90);
RysujTekstRastrowy(GLUT_BITMAP_8_BY_13, buf);
```

4. Wnioski

Podczas pracy nad tym kodem w środowisku OpenGL zgłębiłem różnorodne aspekty renderowania grafiki trójwymiarowej. Skupiłem się na zrozumieniu i implementacji systemów oświetlenia, co pozwoliło mi na dokładne kontrolowanie efektów wizualnych w scenie. Nauczyłem się konfigurować różne typy świateł, takie jak reflektory i światła kierunkowe, oraz zastosowałem te techniki do realistycznego oświetlenia modeli 3D. Eksperymentowałem również z wektorami normalnymi i technikami cieniowania, co pozwoliło mi lepiej zrozumieć, jak światło wpływa na wygląd i percepcję obiektów w przestrzeni 3D. Praca ta pozwoliła mi również na praktyczne zastosowanie i zrozumienie animacji w czasie rzeczywistym, reagując na interakcje użytkownika i aktualizując stan sceny.