WOJSKOWA AKADEMIA TECHNICZNA

Grafika komputerowa

Sprawozdanie z pracy laboratoryjnej Nr 4-5

Temat: Modelowanie oświetlenia 3D w OpenGL

Prowadzący: dr inż. Marek Salamon

Autor: Mateusz Jasiński WCY20IJ1S1

Data wykonania: 15.01.2023

1. Treść zadania

Lab 4:

Zadanie laboratoryjne – zestaw 8

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_TRIANGLES*** 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.

Laboratorium GK- ćwiczenie 4

Temat: Modelowanie obiektów 3D

Wykorzystując wybrane funkcje modelowania geometrycznego biblioteki OpenGl napisać program przedstawiający obraz perspektywiczny bryły o zadanych parametrach:

Użytkownik za pomocą klawiatury powinien mieć możliwość:
- wprowadzania liczby podziałów pionowych i poziomych bryły w zakresie od 4 do 64,
- zmiany odległości obserwatora od obiektu,
- zmiany orientacji obserwatora (obroty względem osi X, Y i Z).

8.(X2) ¾ walca w trybie GL_QUADS o promieniu podstawy 5 i środku w punkcie (0, 0, 0), wysokości 4,

Lab 5:

Zadanie laboratoryjne

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.

Modelowanie oświetlenia – Zestaw 3

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. Zobrazowanie wektorów normalnych; (*weryfikacja poprawności obliczeń*) **1 pkt.**
3. Niezależne włączanie i wyłączanie źródeł światła; **1 pkt.**
4. Zmianę materiału bryły; **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 8

Materiały:

1. Właściwości materiału nr 1: **niebieski błyszczący** (widziany w białym świetle),
2. Właściwości materiału nr 2: **szary matowy** (widziany w białym świetle),
3. Właściwości materiału nr 3: **polerowane złoto**

Źródła światła:

Źródło nr 1:

- typ: **reflektor** (ang. *spot*),
- kolor: **żółty**,
- natężenie: **1**,
- kąt odcięcia: **45°**,
- 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**.

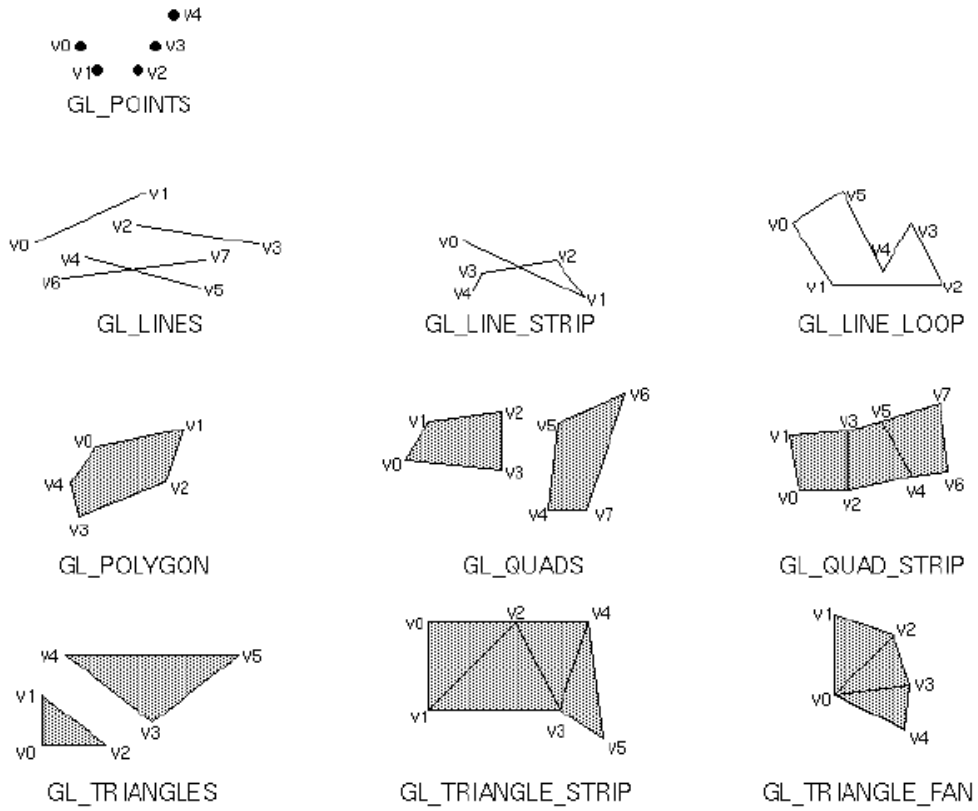
Źródło nr 2:

- typ: **kierunkowe**,
- kolor: **fioletowy**,
- natężenie: **0.7**,
- położenie: **stałe** w punkcie P(10,-5, 10) **układu współrzędnych obserwatora**.
- kierunek świecenia: **na obiekt**.

2. Wstęp teoretyczny

Laboratorium 4

Tworzenie obiektów 3D w OpenGL polega na tworzeniu punktów w przestrzeni za pomocą funkcji glVertex3f oraz wybraniu odpowiedniego trybu do ich połączenia. Jest wiele trybów, a każdy z nich łączy punkty na swój sposób. Niektóre tryby tworzą linię tylko między 2 (GL_LINES) lub 3 (GL_TRIANGLES) punktami, inne mogą łączyć potencjalnie nieskończenie wiele (ograniczenia sprzętowe, zasobów) punktów np. GL_TRIANGLE_FAN.



Laboratorium 5

Do tworzenia wektorów w OpenGL używana jest funkcja glNormal3f, która służy do określenia wektora normalnego dla powierzchni trójkąta lub innego typu geometrii. Wektor normalny jest wektorem prostopadłym do powierzchni, który jest używany do określenia kierunku oświetlenia i cieniowania powierzchni.
glLight - służy do konfiguracji oświetlenia, takiego jak położenie, kolor i natężenie źródła światła.
glMaterial - służy do konfiguracji właściwości materiałów, takich jak kolor, połysk i odpowiedź na światło.

3. Cel ćwiczenia i sposób rozwiązania

Laboratorium 4

Cel ćwiczenia

Głównymi celami ćwiczenia było:

- dodanie osi XYZ w kolorach (r, g, b)
- zmiana pozycji obserwatora
- stworzyć obiekt 3D stożka ściętego z możliwością zmiany liczby podziałów pionowych
- stworzyć obiekt 3D ¾ walca z możliwością zmiany liczby podziałów pionowych i poziomych

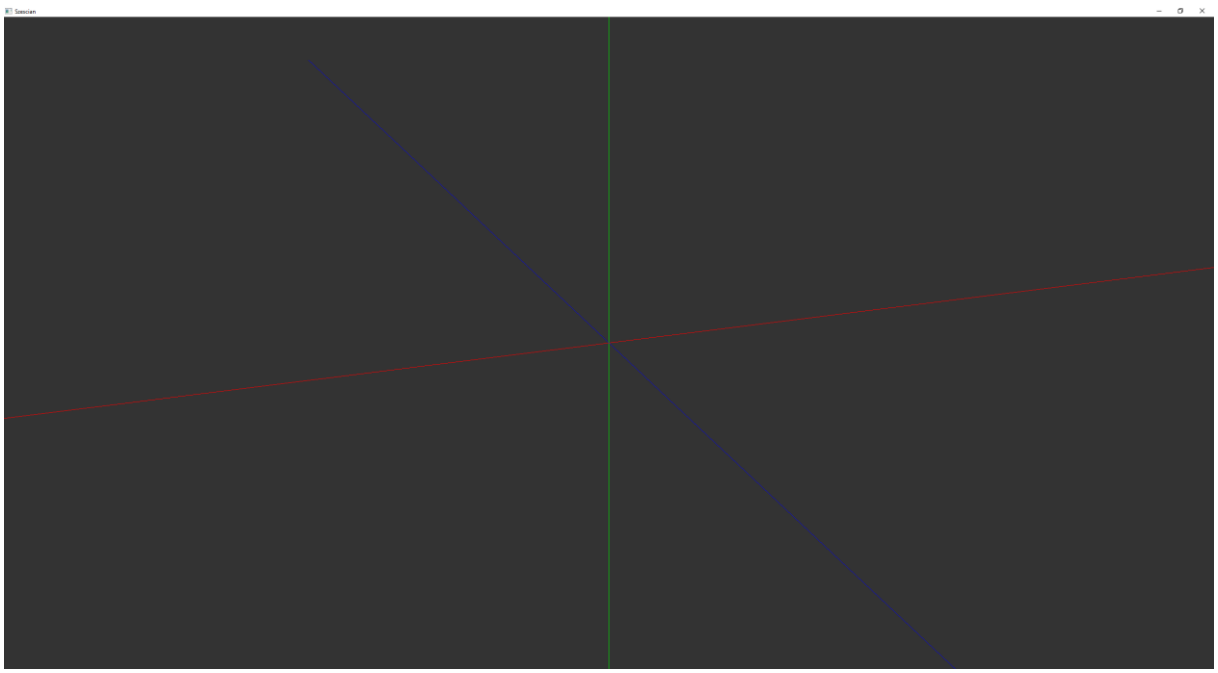
Dodanie osi XYZ

Dodanie osi polegało na użyciu trybu GL_LINES do stworzenia 3 linii na odpowiednich osiach o danym kolorze. Do uzyskania wyniku wystarczyło dodać 2 punkty na jednej z osi przy pomocy funkcji glVertex3f, które w trybie GL_LINES zostają połączone i tworzą linie.

Kod programu

```
void RysujOxOyOz()
{
    // Pocztaek tworzenia ukladu wspolrzednych
    glBegin(GL_LINES);

    // Os X
    glColor3f(1.0, 0.0, 0.0);
    glVertex3f(-100.0, 0.0, 0.0);
    glVertex3f(100.0, 0.0, 0.0);

    // Os Y
    glColor3f(0.0, 1.0, 0.0);
    glVertex3f(0.0, -100.0, 0.0);
    glVertex3f(0.0, 100.0, 0.0);

    // Os Z
    glColor3f(0.0, 0.0, 1.0);
    glVertex3f(0.0, 0.0, -100.0);
    glVertex3f(0.0, 0.0, 100.0);

    // Koniec tworzenia ukladu wspolrzednych
    glEnd();
}
```

Efekt



Zmiana pozycji obserwatora

Do zmiany pozycji obserwatora potrzebujemy zmiennych GLfloat przechowujących położenie obserwatora:
- OBSERWATOR_ODLEGOSC – przechowuje odległość obserwatora od początku układu
- OBSERWATOR_OBROT_X – przechowuje kąt obrotu obserwatora względem osi X
- OBSERWATOR_OBROT_Y – przechowuje kąt obrotu obserwatora względem osi Y
- OBSERWATOR_OBROT_Z – przechowuje kąt obrotu obserwatora względem osi Z

Dodatkowo dodać możliwość modyfikacji wartości tych zmiennych przez użytkownika, można to zrobić poprzez dodanie obsługi klawiszy i przypisanie modyfikacji zmiennych pod konkretne przyciski np. strzałki.

Kod programu

```
void WyswietlObraz(void)
{
  // Wyczyszczenie bufora koloru i bufora glebokosci
  glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);

  // Przejscie w tryb modyfikacji macierzy przeksztalcen geometrycznych
  glMatrixMode(GL_MODELVIEW);

  // Zastapienie aktywnej macierzy macierza jednostkowa
  glLoadIdentity();

  // Ustalenie polozenia obserwatora
  glTranslatef(0, 0, -OBSERWATOR_ODLEGLOSC);
  glRotatef(OBSERWATOR_OBROT_X, 1, 0, 0);
  glRotatef(OBSERWATOR_OBROT_Y, 0, 1, 0);
  glRotatef(OBSERWATOR_OBROT_Z, 0, 0, 1);

  // Narysowanie osi x,y,z
  RysujOxOyOz();

  // Narysowanie szescianu
  // RysujSzescian(bok);
  RysujStozekSciety(N, R, h, r);

  // Przelaczenie buforow ramki
  glutSwapBuffers();
}


void ObslugaKlawiszySpecjalnych(int klawisz, int x, int y)
{
    switch (klawisz)
    {
    case GLUT_KEY_UP:
        OBSERWATOR_OBROT_X = OBSERWATOR_OBROT_X + 1.0;
        break;

    case GLUT_KEY_DOWN:
        OBSERWATOR_OBROT_X = OBSERWATOR_OBROT_X - 1.0;
        break;

    case GLUT_KEY_LEFT:
        OBSERWATOR_OBROT_Y = OBSERWATOR_OBROT_Y - 1.0;
        break;

    case GLUT_KEY_RIGHT:
```

```
        OBSERWATOR_OBROT_Y = OBSERWATOR_OBROT_Y + 1.0;
        break;

    case GLUT_KEY_HOME:
        OBSERWATOR_OBROT_Z = OBSERWATOR_OBROT_Z + 1.0;
        break;

    case GLUT_KEY_END:
        OBSERWATOR_OBROT_Z = OBSERWATOR_OBROT_Z - 1.0;
        break;
    }
}


void ObslugaKlawiatury(unsigned char klawisz, int x, int y)
{
    switch (klawisz)
    {
    case '1':
        if (OBSERWATOR_ODLEGLOSC - 1 < odlmin) break;
        else OBSERWATOR_ODLEGLOSC--;
        break;
    case '!':
        if (OBSERWATOR_ODLEGLOSC + 1 > odlmax) break;
        else OBSERWATOR_ODLEGLOSC++;
        break;
    case '2':
        if (N > minN)
            N -= 1.0;
        break;
    case '@':
        if (N < maxN)
            N += 1.0;
        break;
    case 27:
        exit(0);
        break;
    }
}
```

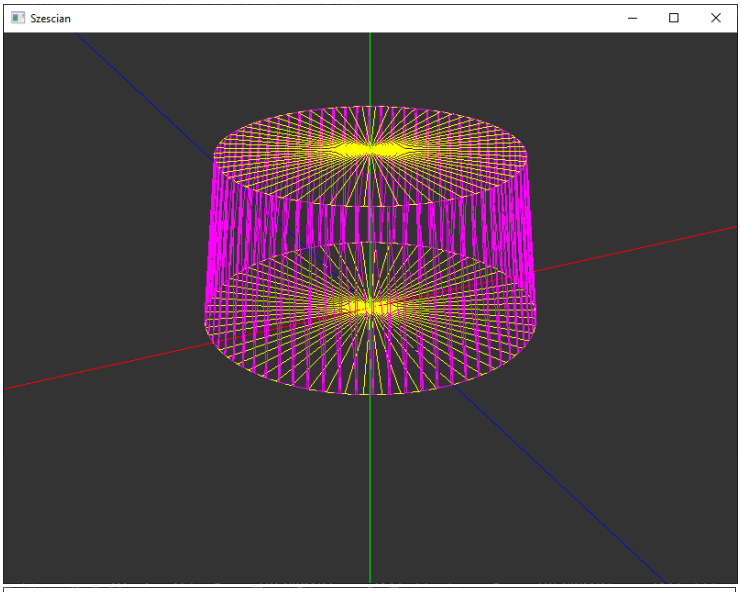
Stożek ścięty

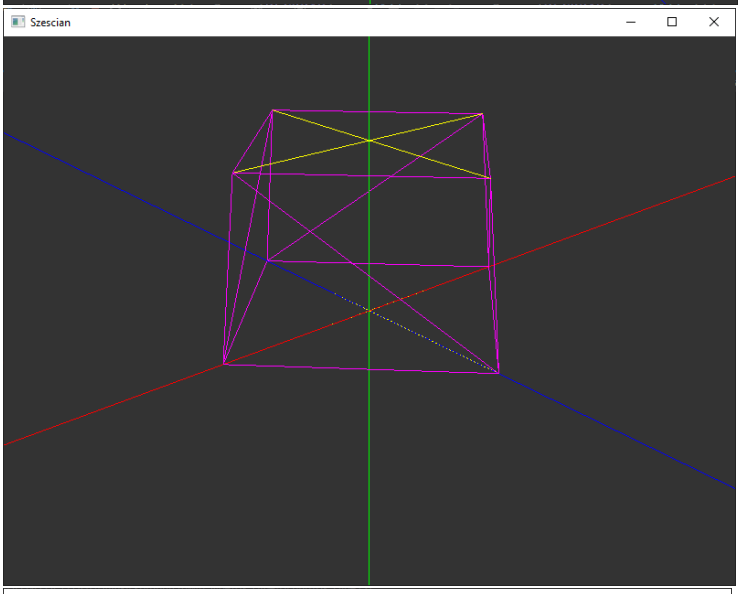
Stożek ścięty składa się z 2 postaw i boków, gdzie:
R = 8 – promień dolnej podstawy
r = 7 – promień górnej podstawy
h = 8 – wysokość boków
Do stworzenia boków wykorzystujemy tryb GL_TRIANGLES, a do podstaw GL_TRIANGLE_FAN. Boki powstają poprzez łączenie 2 trójkątów stworzonych przez funkcję glVertex3f w kwadraty, które stanowią ściany stożka ściętego. Podstawa powstaje przy pomocy trybu GL_TRIAGNLE_FAN, która łączy punkt początkowy z każdym kolejnym dodanym punktem, dodatkowo łączy dwa kolejno dodane punkty ze sobą. Funkcja ta łączy punkty tworząc coś przypominające wachlarz, stąd też dopisek _FAN.

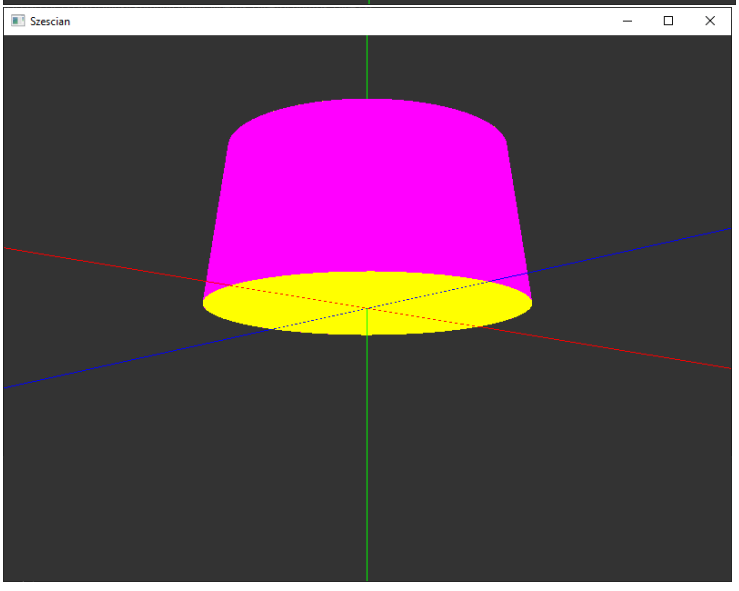
Kod programu

```
void podstawa(int n, float r, float h)
{
    double dAlfa = 360.0 / (double)n;
    glColor3f(1.0, 1.0, 0.0);
    glBegin(GL_TRIANGLE_FAN);
    glVertex3f(0.0, h, 0.0);
    for (int i = 0; i * dAlfa <= 360.0; i++)
        glVertex3f(r * cos(DEG2RAD(i * dAlfa)), h, r * sin(DEG2RAD(i * dAlfa)));
    glEnd();
}

void RysujStozekSciety(int n, float R, float h, float r)
{
    double dAlfa = 360.0 / (double)n;
    glColor3f(1.0, 0.0, 1.0);
    glBegin(GL_TRIANGLES);
    for (int i = 0; i < 360; i = i + dAlfa) {
        glVertex3f(R * cos(DEG2RAD(i)), 0, R * sin(DEG2RAD(i)));
        glVertex3f(r * cos(DEG2RAD(i)), h, r * sin(DEG2RAD(i)));
        glVertex3f(r * cos(DEG2RAD(i + dAlfa)), h, r * sin(DEG2RAD(i + dAlfa)));

        glVertex3f(R * cos(DEG2RAD(i)), 0, R * sin(DEG2RAD(i)));
        glVertex3f(r * cos(DEG2RAD(i + dAlfa)), h, r * sin(DEG2RAD(i + dAlfa)));
        glVertex3f(R * cos(DEG2RAD(i + dAlfa)), 0, R * sin(DEG2RAD(i + dAlfa)));
    }
    glEnd();
    podstawa(n, R, 0);
    podstawa(n, r, h);
}
```

Efekt







¾ walca

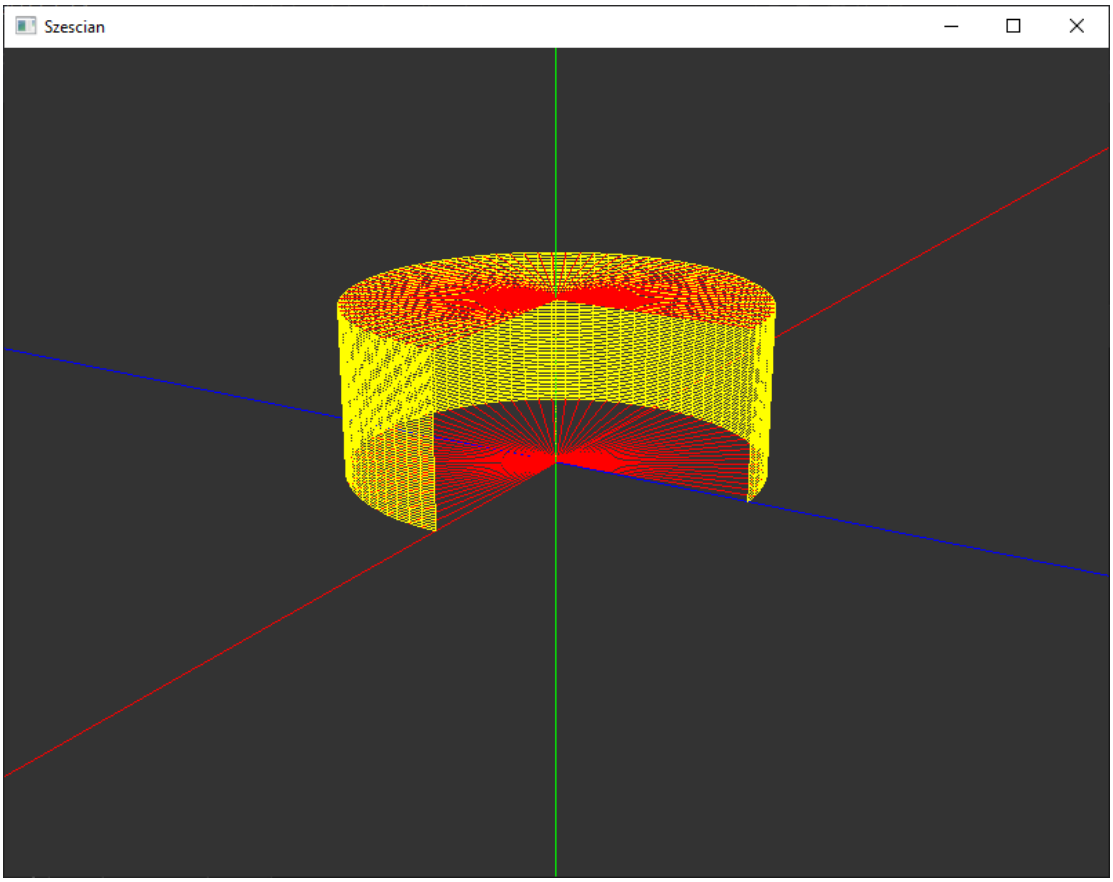
Zadanie indywidualne polega na stworzenia walca o promieniu 5 i wysokości 4 używając trybu GL_QUADS do stworzenia boków. Dodatkowo należało umożliwić zmianę liczby podziałów pionowych i poziomych (zmienna n).

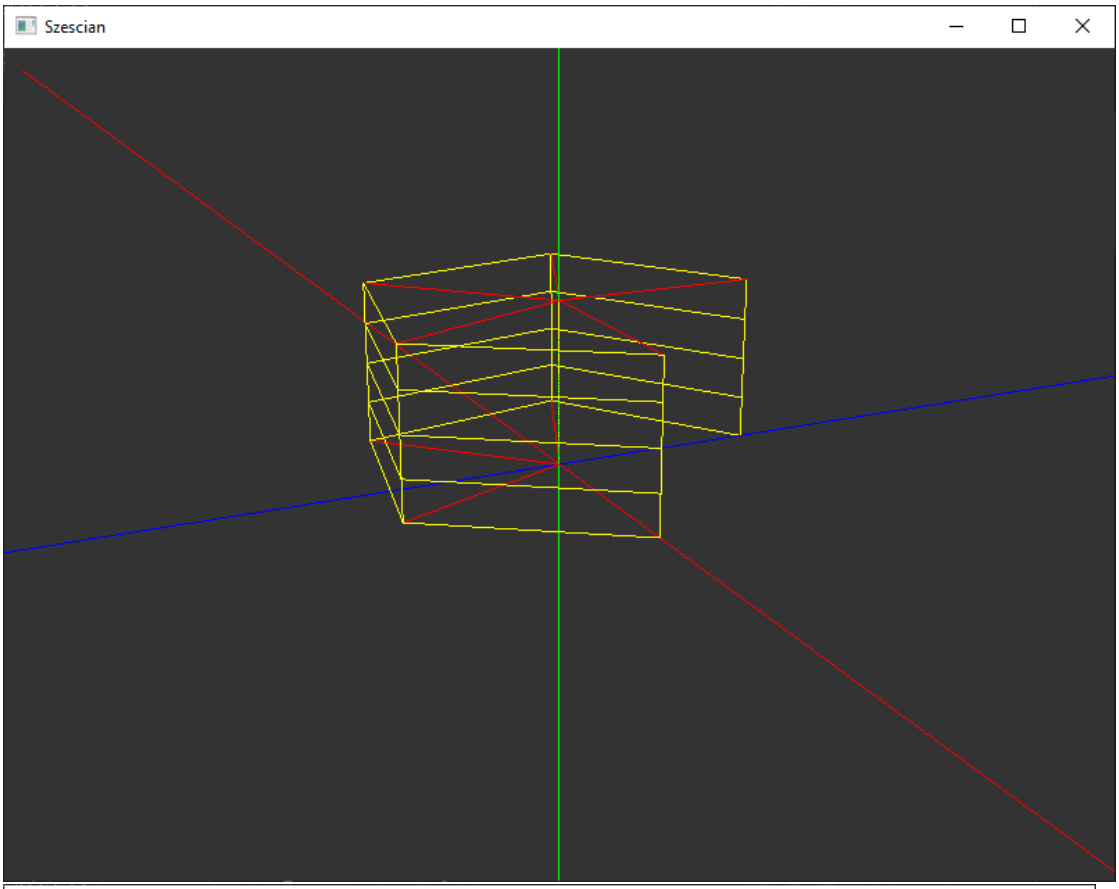
Kod programu

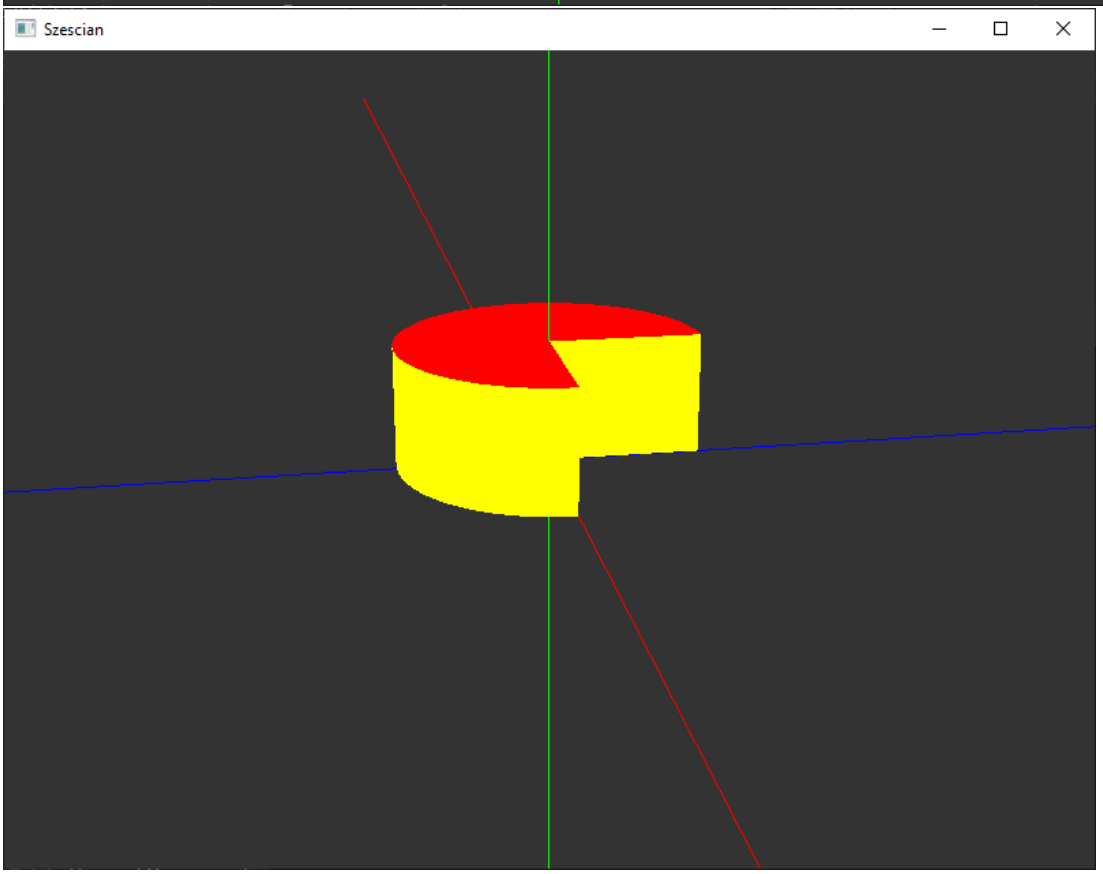
```
void RysujWalec(float n)
{
    float h = 4.0, r = 5.0;
    float hDelta = (h / (float)n);
    float delta = (1.5 * M_PI / (float)n);
    // boki
    glColor3f(1.0, 1.0, 0.0);
    for (float hTemp = 0.0; hTemp < h; hTemp += hDelta) {
        glBegin(GL_QUADS);
        for (float angle = 0.0; angle < (1.5 * M_PI); angle += delta) {
            glVertex3f(r * cos(angle), hTemp, r * sin(angle));
            glVertex3f(r * cos(angle), ((hTemp + hDelta) > h) ? h : (hTemp +
hDelta), r * sin(angle));
            angle += delta;
            if (angle > 1.5 * M_PI) {
                angle = 1.5 * M_PI;
                glVertex3f(r * cos(angle), ((hTemp + hDelta) > h) ? h : (hTemp +
hDelta), r * sin(angle));
                glVertex3f(r * cos(angle), hTemp, r * sin(angle));
                angle -= delta;
                break;
            }
            glVertex3f(r * cos(angle), ((hTemp + hDelta) > h) ? h : (hTemp +
hDelta), r * sin(angle));
            glVertex3f(r * cos(angle), hTemp, r * sin(angle));
            angle -= delta;
        }
        glEnd();
    }

    // góra
    glColor3f(1.0, 0.0, 0.0);
    glBegin(GL_TRIANGLE_FAN);
    glVertex3f(0.0, h, 0.0);
    for (float angle = 0; angle <= ((1.5 * M_PI) + delta); angle += delta)
        glVertex3f(r * cos((angle > 1.5 * M_PI) ? 1.5 * M_PI : angle), h, r *
sin((angle > 1.5 * M_PI) ? 1.5 * M_PI : angle));
    glEnd();

    // dół
    glBegin(GL_TRIANGLE_FAN);
    glVertex3f(0.0, 0.0, 0.0);
    for (float angle = 0; angle <= ((1.5 * M_PI) + delta); angle += delta)
        glVertex3f(r * cos((angle > 1.5 * M_PI) ? 1.5 * M_PI : angle), 0, r *
sin((angle > 1.5 * M_PI) ? 1.5 * M_PI : angle));
    glEnd();

    // domknięcie
    glColor3f(1.0, 1.0, 0.0);
    glBegin(GL_QUADS);
    glVertex3f(0.0, 0, 0.0);
    glVertex3f(0.0, h, 0.0);
    glVertex3f(r, h, 0);
    glVertex3f(r, 0, 0);
```

```
    glVertex3f(0.0, 0, 0.0);
    glVertex3f(0.0, h, 0.0);
    glVertex3f(0, h, -r);
    glVertex3f(0, 0, -r);
    glEnd();
}
```

Efekt




Szescian


Szescian

Laboratorium 5

Cel ćwiczenia

Głównymi celami ćwiczenia było:
- Wyznaczenie wektorów normalnych
- Dodanie dwóch niezależnych źródeł światła
- Zmiana materiału bryły
- Zmiana trybu cieniowania
- Zobrazowanie menu programu

Wyznaczenie wektorów normalnych i zobrazowanie modelu geometrycznego bryły

W OpenGl, wektory normalne są używane do określenia jak światło będzie odbijać się od powierzchni trójkątów. Aby wyznaczyć wektory normalne, należy obliczyć wektor normalny dla każdego trójkąta, który tworzy powierzchnię. Można to zrobić przez obliczenie iloczynu wektorowego dwóch wektorów tworzących trójkąt. Następnie, aby wyświetlić wektory normalne, należy ustawić odpowiedni tryb rysowania (np. GL_LINES) i rysować linie z położenia środka każdego trójkąta w kierunku obliczonego wektora normalnego.

Kod programu

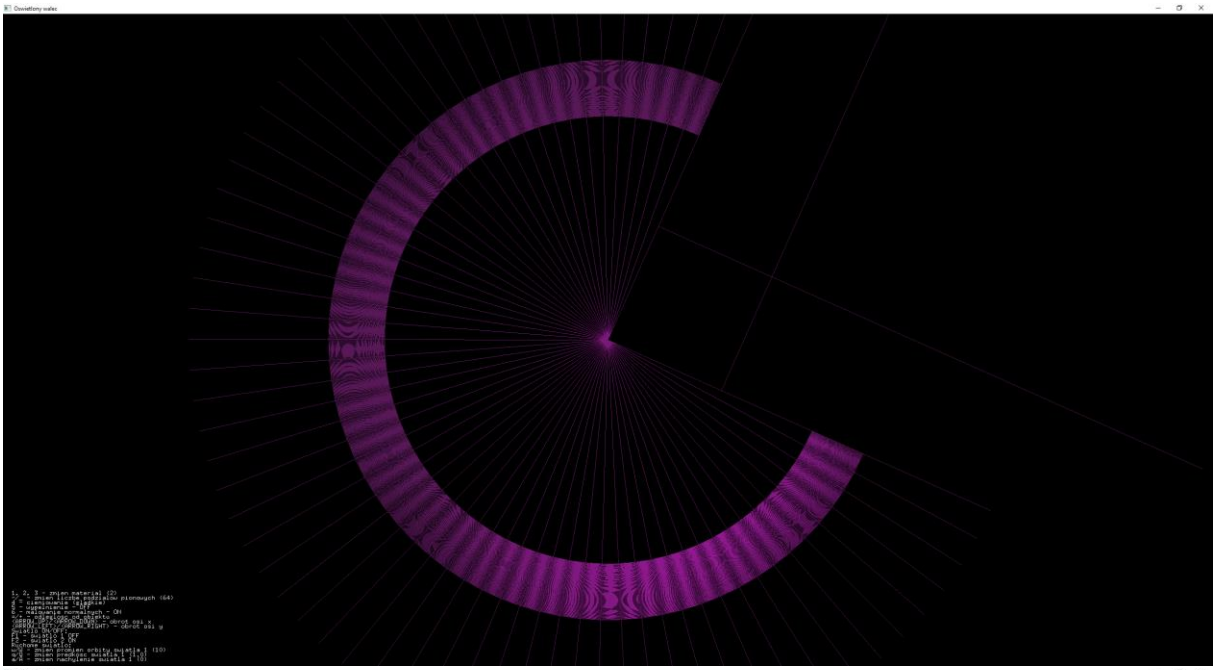
```
void RysujWalec() {
    delta = r / ilosc_podzialow;
    beta = h / ilosc_podzialow;
    alfa = kat / ilosc_podzialow;
    double temp_r, temp_r_2, temp_h, temp_h_2;

    //dolna podstawa
    glColor3f(1.0, 0.0, 0.0);
    glBegin(GL_TRIANGLE_FAN);
    glVertex3f(0.0, 0.0, 0.0);
    for (int i = 0.0; i * alfa <= kat; i++)
    {
        glNormal3f(0, -1, 0);
        glVertex3f(r * cos(DEG2RAD(i * alfa)), 0, r * sin(DEG2RAD(i * alfa)));
    }
    glEnd();

    //gorna podstawa
    glColor3f(1.0, 0.0, 0.0);
    glBegin(GL_TRIANGLE_FAN);
    glVertex3f(0.0, h, 0.0);
    for (int i = 0.0; i * alfa <= kat; i++)
    {
        glNormal3f(0, 1, 0);
        glVertex3f(r * cos(DEG2RAD(i * alfa)), h, r * sin(DEG2RAD(i * alfa)));
    }
    glEnd();

    //boki
    glColor3f(1.0, 1.0, 0.0);
    for (int j = 0; j < ilosc_podzialow; j++)
    {
        temp_h = j * beta;
        temp_h_2 = (j + 1) * beta;
        glBegin(GL_TRIANGLES);
```

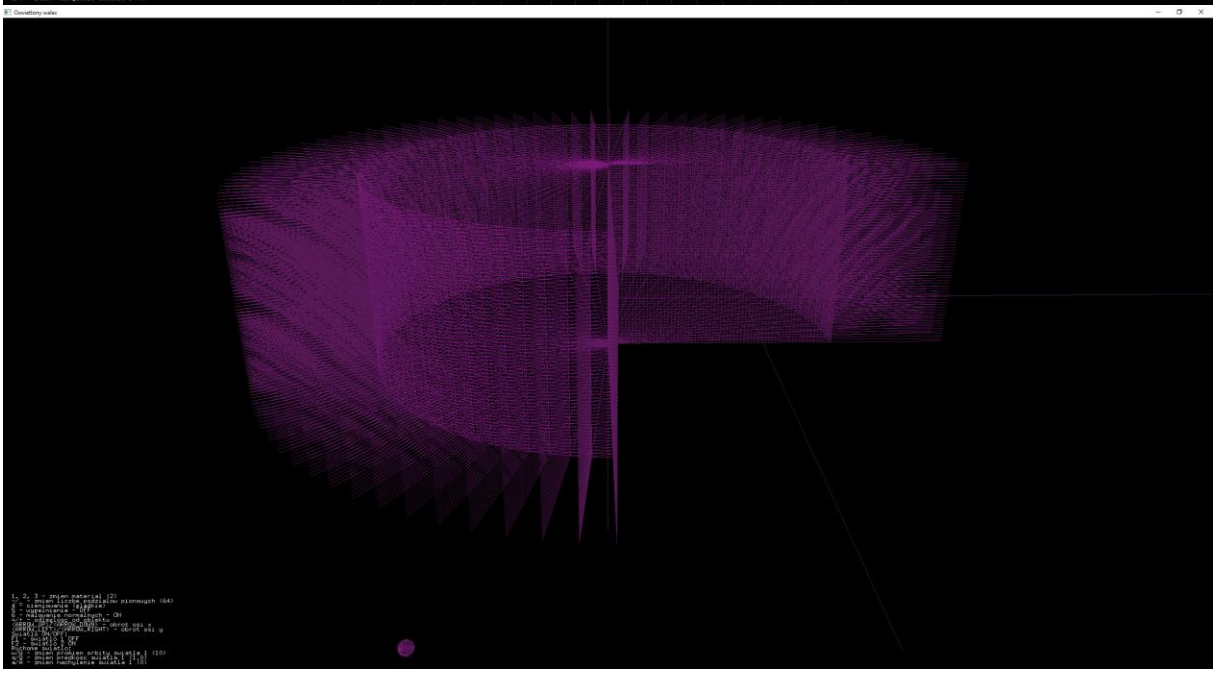
```
        for (int i = 0.0; i * alfa < kat; i++)
        {
            glNormal3f(cos(DEG2RAD(i * alfa)), 0, sin(DEG2RAD(i * alfa)));
            glVertex3f(r * cos(DEG2RAD(i * alfa)), temp_h, r * sin(DEG2RAD(i *
alfa)));
            glNormal3f(cos(DEG2RAD(i * alfa)), 0, sin(DEG2RAD(i * alfa)));
            glVertex3f(r * cos(DEG2RAD(i * alfa)), temp_h_2, r * sin(DEG2RAD(i *
alfa)));
            glNormal3f(cos(DEG2RAD((i + 1) * alfa)), 0, sin(DEG2RAD((i + 1) *
alfa)));
            glVertex3f(r * cos(DEG2RAD((i + 1) * alfa)), temp_h, r *
sin(DEG2RAD((i + 1) * alfa)));
            glNormal3f(cos(DEG2RAD(i * alfa)), 0, sin(DEG2RAD(i * alfa)));
            glVertex3f(r * cos(DEG2RAD(i * alfa)), temp_h_2, r * sin(DEG2RAD(i *
alfa)));
            glNormal3f(cos(DEG2RAD((i + 1) * alfa)), 0, sin(DEG2RAD((i + 1) *
alfa)));
            glVertex3f(r * cos(DEG2RAD((i + 1) * alfa)), temp_h, r *
sin(DEG2RAD((i + 1) * alfa)));
            glNormal3f(cos(DEG2RAD((i + 1) * alfa)), 0, sin(DEG2RAD((i + 1) *
alfa)));
            glVertex3f(r * cos(DEG2RAD((i + 1) * alfa)), temp_h_2, r *
sin(DEG2RAD((i + 1) * alfa)));
        }
        glEnd();
    }

    //domkniecie z
    glColor3f(1.0, 1.0, 0.0);
    for (int j = 0; j < ilosc_podzialow; j++)
    {
        temp_h = j * beta;
        temp_h_2 = (j + 1) * beta;
        glBegin(GL_TRIANGLES);
        for (int i = 0.0; i * alfa < kat; i++)
        {
            glNormal3f(-1, 0, 0);
            temp_r = i * delta;
            temp_r_2 = (i + 1) * delta;
            glVertex3f(0, temp_h, -temp_r_2);
            glVertex3f(0, temp_h_2, -temp_r_2);
            glVertex3f(0, temp_h, -temp_r);
            glVertex3f(0, temp_h_2, -temp_r);
            glVertex3f(0, temp_h, -temp_r);
            glVertex3f(0, temp_h_2, -temp_r_2);
        }
        glEnd();
    }
    //domkniecie x
    for (int j = 0; j < ilosc_podzialow; j++)
    {
        temp_h = j * beta;
        temp_h_2 = (j + 1) * beta;
        glBegin(GL_TRIANGLES);
        for (int i = 0.0; i * alfa < kat; i++)
        {
            glNormal3f(0, 0, -1);
            temp_r = i * delta;
            temp_r_2 = (i + 1) * delta;
            glVertex3f(temp_r_2, temp_h, 0);
            glVertex3f(temp_r_2, temp_h_2, 0);
            glVertex3f(temp_r, temp_h, 0);
```

```
            glVertex3f(temp_r, temp_h_2, 0);
            glVertex3f(temp_r, temp_h, 0);
            glVertex3f(temp_r_2, temp_h_2, 0);
        }
        glEnd();
    }
}

void CzyNormalne() {
    delta = r / ilosc_podzialow;
    beta = h / ilosc_podzialow;
    alfa = kat / ilosc_podzialow;
    double temp_r, temp_r_2, temp_h, temp_h_2;
    //zaobrazowanie wektorow normalmych sciany bocznej
    glColor3f(0.0, 0.0, 0.0);
    for (int j = 0; j < ilosc_podzialow; j++)
    {
        temp_h = j * beta;
        temp_h_2 = (j + 1) * beta;
        glBegin(GL_LINES);
        for (int i = 0.0; i * alfa < kat; i++)
        {
            glVertex3f(r * cos(DEG2RAD(i * alfa)), temp_h, r * sin(DEG2RAD(i *
alfa)));
            glVertex3f(1.5 * r * cos(DEG2RAD(i * alfa)), temp_h, 1.5 * r *
sin(DEG2RAD(i * alfa)));

            glVertex3f(r * cos(DEG2RAD(i * alfa)), temp_h_2, r * sin(DEG2RAD(i *
alfa)));
            glVertex3f(1.5 * r * cos(DEG2RAD(i * alfa)), temp_h_2, 1.5 * r *
sin(DEG2RAD(i * alfa)));

            glVertex3f(r * cos(DEG2RAD((i + 1) * alfa)), temp_h, r *
sin(DEG2RAD((i + 1) * alfa)));
            glVertex3f(1.5 * r * cos(DEG2RAD((i + 1) * alfa)), temp_h, 1.5 * r *
sin(DEG2RAD((i + 1) * alfa)));

            glVertex3f(r * cos(DEG2RAD(i * alfa)), temp_h_2, r * sin(DEG2RAD(i *
alfa)));
            glVertex3f(1.5 * r * cos(DEG2RAD(i * alfa)), temp_h_2, 1.5 * r *
sin(DEG2RAD(i * alfa)));

            glVertex3f(r * cos(DEG2RAD((i + 1) * alfa)), temp_h, r *
sin(DEG2RAD((i + 1) * alfa)));
            glVertex3f(1.5 * r * cos(DEG2RAD((i + 1) * alfa)), temp_h, 1.5 * r *
sin(DEG2RAD((i + 1) * alfa)));

            glVertex3f(r * cos(DEG2RAD((i + 1) * alfa)), temp_h_2, r *
sin(DEG2RAD((i + 1) * alfa)));
            glVertex3f(1.5 * r * cos(DEG2RAD((i + 1) * alfa)), temp_h_2, 1.5 * r
* sin(DEG2RAD((i + 1) * alfa)));
        }
        //wektory podstaw do zaobrazowania
        glVertex3f(0, -5, 0);
        glVertex3f(0, 9, 0);

        //zaobrazowanie wektorow scianek lezacych na osi x/z
        glVertex3f(0, 2, -2.5);
        glVertex3f(12, 2, -2.5);
        glVertex3f(2.5, 2, -12);
        glVertex3f(2.5, 2, 0);
```

```
        glEnd();
    }
}
```

Efekt





Definicja i zmiana położenia źródła światła

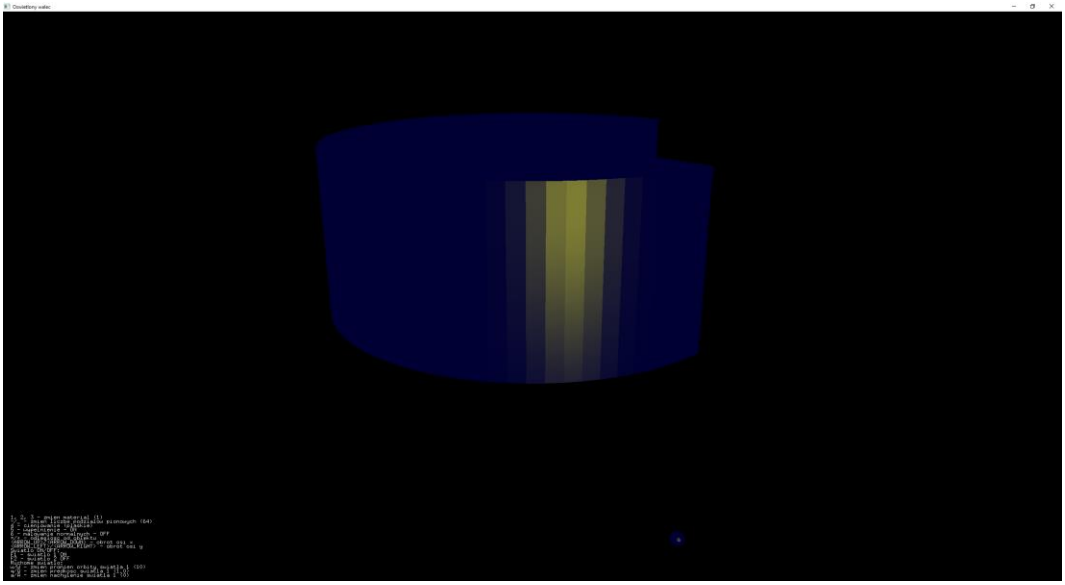
W OpenGL, aby dodać źródło światła, należy skonfigurować jego parametry i ustawić jego pozycję, można to zrobić za pomocą funkcji glLight.
OpenGL obsługuje kilka różnych źródeł światła (np. GL_LIGHT0, GL_LIGHT1 itd.) Można skonfigurować parametry i pozycję każdego z nich osobno.
Ważne jest, aby pamiętać, że aby światło mogło być widoczne na rysowanych bryłach, trzeba włączyć oświetlenie globalne za pomocą glEnable(GL_LIGHTING).

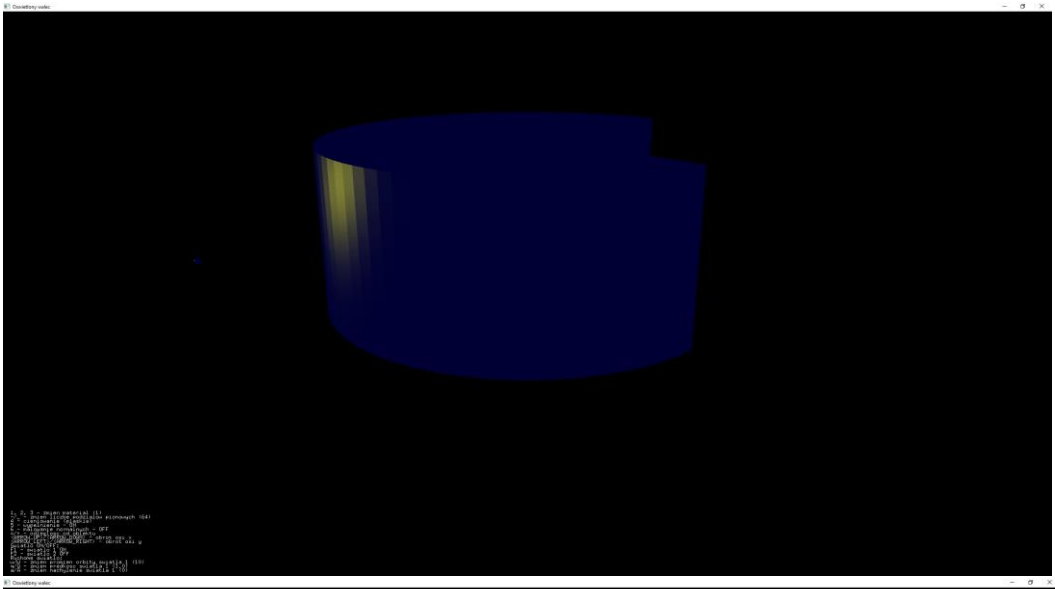
Kod programu

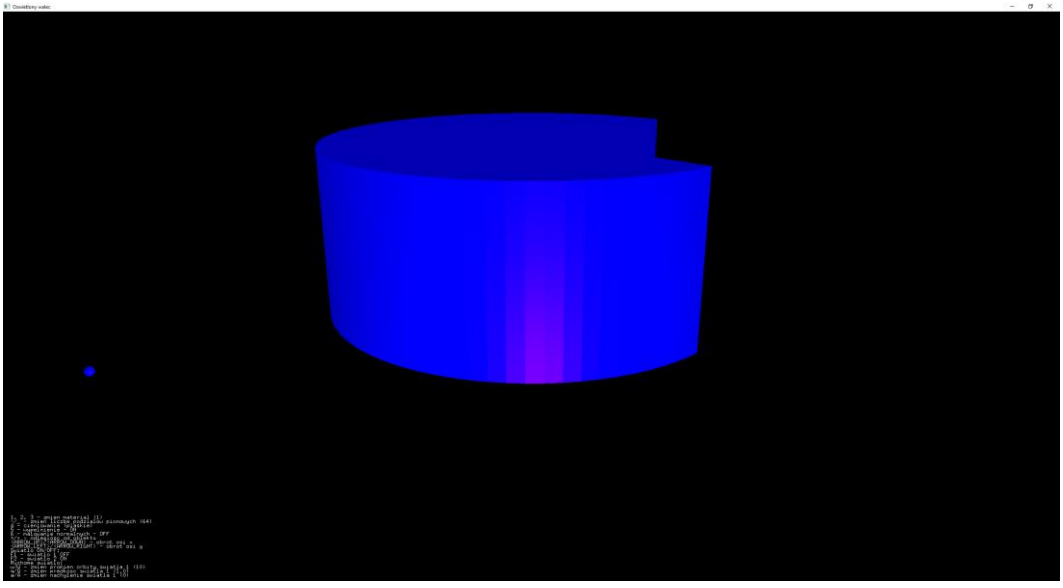
```
    GLfloat param_swiatla[10][4] = {
    {0.5, 0.5, 0.0, 1.0},   // [0] otoczenie
    {0.5, 0.5, 0.0, 1.0},   // [1] rozproszenie
    {0.5, 0.5, 0.0, 1.0},   // [2] lustrzane
    {0.0, 0.0, 0.0, 0.0},   // [3] polozenie
    {0.0, 0.0, 0.0, 0.0},   // [4] kierunek swiecenia
    {0.0, 0.0, 0.0, 0.0},   // [5] tlumienie katowe swiatla
    {45.0, 0.0, 0.0, 0.0},  // [6] kat odciecia swiatla
    {1.0, 0.0, 0.0, 0.0},   // [7] stale tlumienie
    {0.0, 0.0, 0.0, 0.0},   // [8] tlumienie liniowe
    {0.0, 0.0, 0.0, 0.0} }; // [9] tlumienie kwadratowe

    GLfloat param_swiatla2[10][4] = {
    {0.5, 0.0, 0.5, 0.7},     // [0] otoczenie
    {0.5, 0.0, 0.5, 0.7},     // [1] rozproszenie
    {0.5, 0.0, 0.5, 0.7},     // [2] lustrzane
    {10.0, -5.0, 10.0, 1.0},  // [3] polozenie
    {-1.0, 1.0, -1.0, 1.0},     // [4] kierunek swiecenia
    {0.0, 0.0, 0.0, 0.0},     // [5] tlumienie katowe swiatla
    {180.0, 0.0, 0.0, 0.0},   // [6] kat odciecia swiatla
    {1.0, 0.0, 0.0, 0.0},     // [7] stale tlumienie
    {0.0, 0.0, 0.0, 0.0},     // [8] tlumienie liniowe
    {0.0, 0.0, 0.0, 0.0} }; // [9] tlumienie kwadratowe

    memcpy(swiatlo, param_swiatla, LPOZ_MENU_SWIATLA * 4 * sizeof(GLfloat));
    memcpy(swiatlo2, param_swiatla2, LPOZ_MENU_SWIATLA * 4 * sizeof(GLfloat));


void WlaczOswietlenie(void)
{
    GLfloat pozycja_swiatla[] = { promien_swiatla * cos(DEG2RAD(kat_swiatla)) ,
0.01 , promien_swiatla * sin(DEG2RAD(kat_swiatla)), 1.0 };
    GLfloat kierunek_swiatla[] = { -(promien_swiatla *
cos(DEG2RAD(kat_swiatla))) , 0.0 , -(promien_swiatla *
sin(DEG2RAD(kat_swiatla))), 0.0 };

    // Odblokowanie zerowego zrodla swiatla
    if (wlacznik1)glEnable(GL_LIGHT0);
    else  glDisable(GL_LIGHT0);
    if (wlacznik2) glEnable(GL_LIGHT1);
    else  glDisable(GL_LIGHT1);


    // Inicjowanie swiatla 1 + mala sferka

    glPushMatrix();
    glRotatef(nachylenie_swiatla, 1, 0, 0);
```

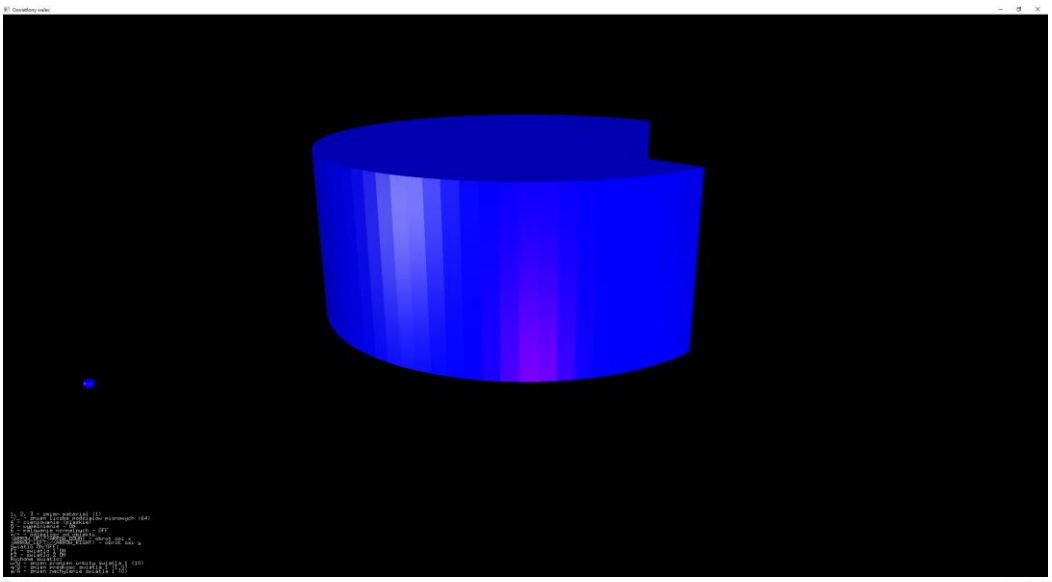
```
    glTranslatef(promien_swiatla * cos(DEG2RAD(kat_swiatla)), 0.01,
promien_swiatla * sin(DEG2RAD(kat_swiatla)));
    glColor3f(0.0, 0.0, 0.0);
    glutSolidSphere(0.1, 20, 20);

    glLightfv(GL_LIGHT0, GL_AMBIENT, swiatlo[0]);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, swiatlo[1]);
    glLightfv(GL_LIGHT0, GL_SPECULAR, swiatlo[2]);
    glLightfv(GL_LIGHT0, GL_POSITION, pozycja_swiatla);
    glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, kierunek_swiatla);
    glLightf(GL_LIGHT0, GL_SPOT_EXPONENT, swiatlo[5][0]);
    glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, swiatlo[6][0]);
    glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, swiatlo[7][0]);
    glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, swiatlo[8][0]);
    glLightf(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, swiatlo[9][0]);

    glPopMatrix();

    // Inicjowanie  wiat a 2
    glLightfv(GL_LIGHT1, GL_AMBIENT, swiatlo2[0]);
    glLightfv(GL_LIGHT1, GL_DIFFUSE, swiatlo2[1]);
    glLightfv(GL_LIGHT1, GL_SPECULAR, swiatlo2[2]);
    glLightfv(GL_LIGHT1, GL_POSITION, swiatlo2[3]);
    glLightfv(GL_LIGHT1, GL_SPOT_DIRECTION, swiatlo2[4]);
    glLightf(GL_LIGHT1, GL_SPOT_EXPONENT, swiatlo2[5][0]);
    glLightf(GL_LIGHT1, GL_SPOT_CUTOFF, swiatlo2[6][0]);
    glLightf(GL_LIGHT1, GL_CONSTANT_ATTENUATION, swiatlo2[7][0]);
    glLightf(GL_LIGHT1, GL_LINEAR_ATTENUATION, swiatlo2[8][0]);
    glLightf(GL_LIGHT1, GL_QUADRATIC_ATTENUATION, swiatlo2[9][0]);

}
```

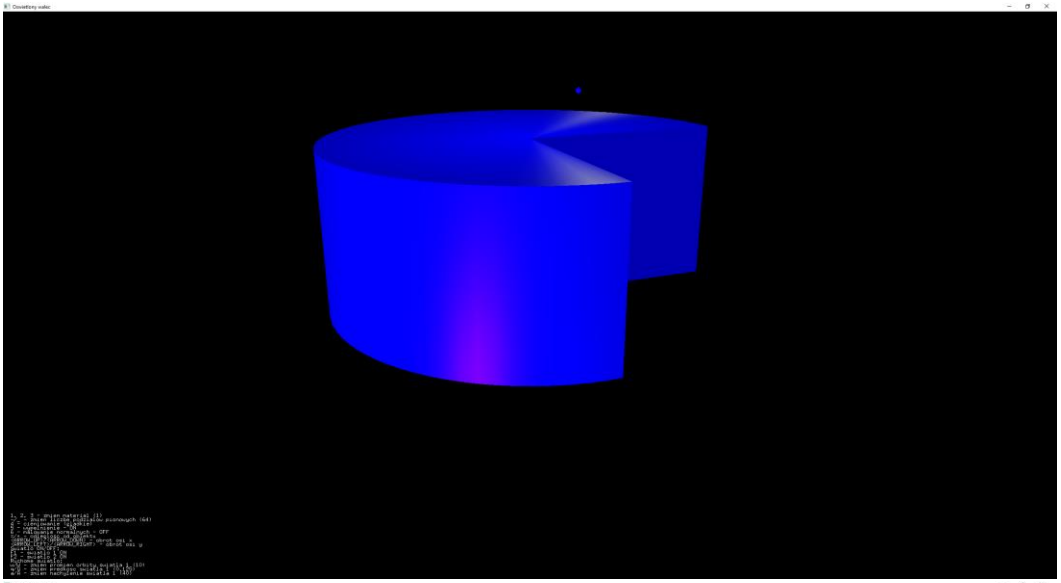
Efekt

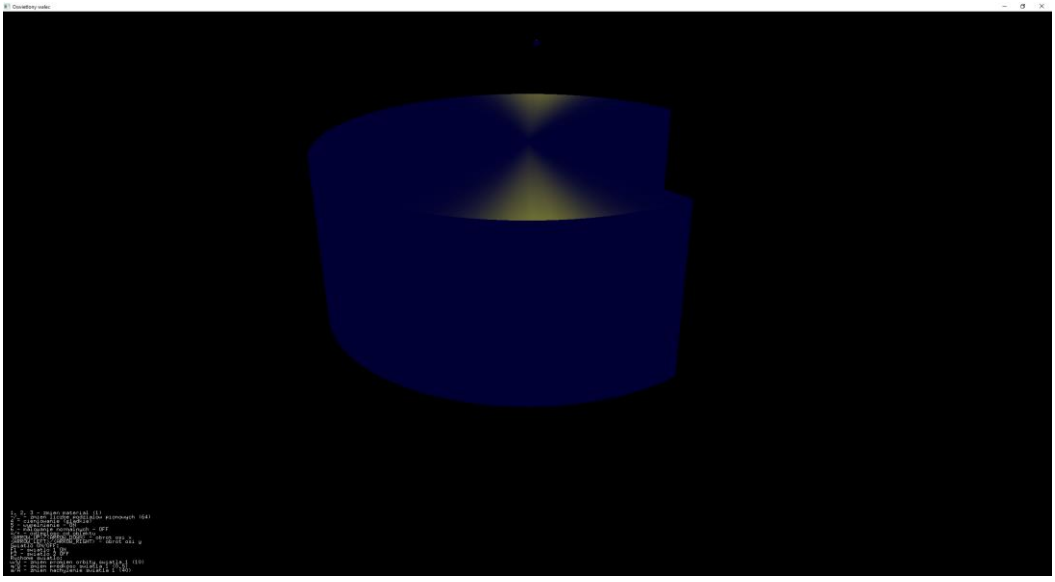












Definicja i zmiana materiału bryły

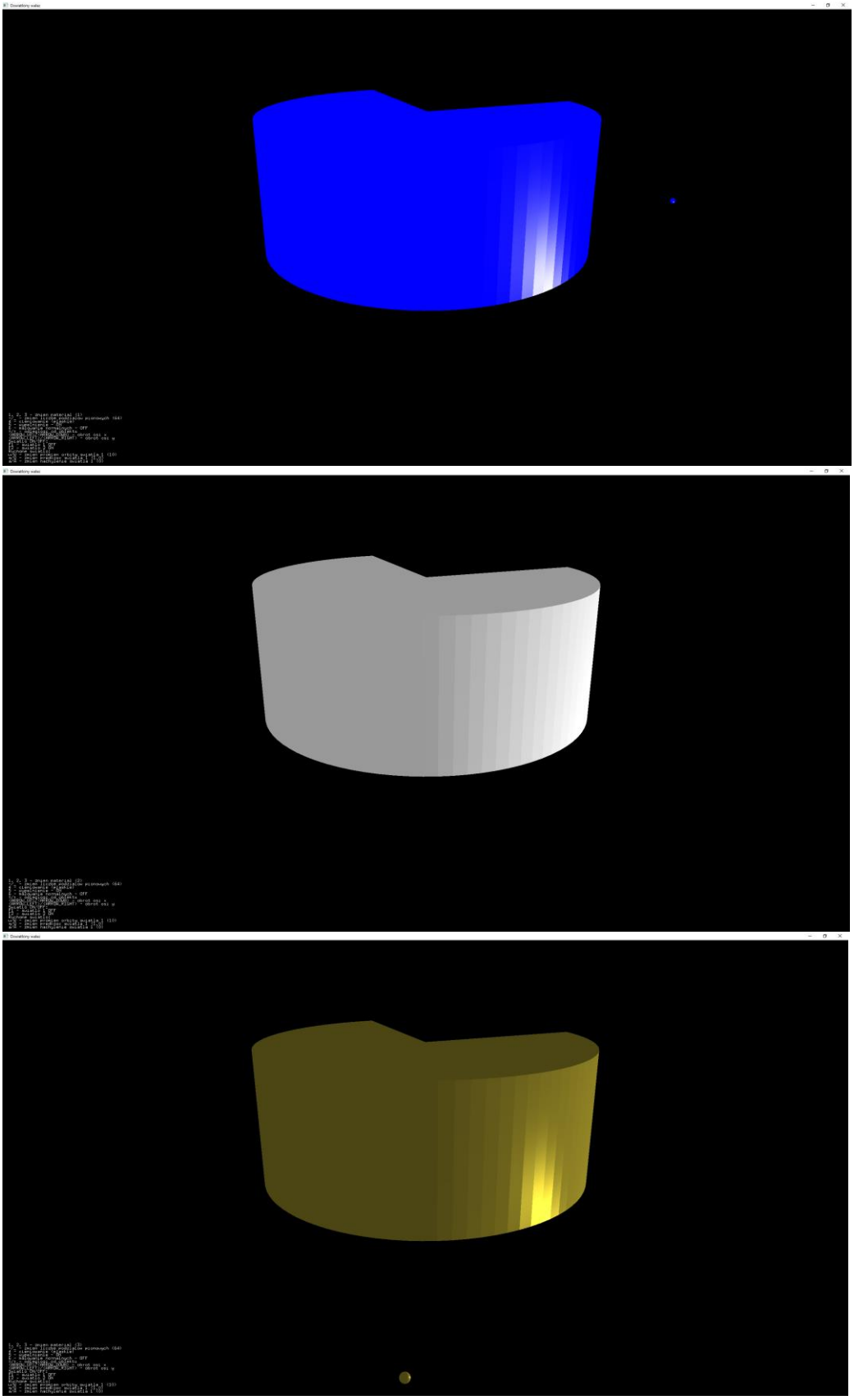
W OpenGl można zmienić wygląd bryły poprzez zmianę jej parametrów materiałowych. Aby to zrobić, należy skonfigurować parametry materiału przed rysowaniem bryły. Na przykład, aby ustawić kolor bryły na czerwony, należy ustawić parametr "ambient" i "diffuse" na (1.0, 0.0, 0.0, 1.0) za pomocą funkcji glMaterialfv.

Kod programu

```
GLfloat param_materialu[6][4] = {//niebieski blyszczacy widziany w bialym
swietle
    {0.0, 0.0, 1.0, 1.0},  // [0] wspolczynnik odbicia swiatla otoczenia
    {0.0, 0.0, 1.0, 1.0},  // [1] wspolczynnik odbicia swiatla rozproszonego
    {1.0, 1.0, 1.0, 1.0},  // [2] wspolczynnik odbicia swiatla lustrzanego
    {50.0, 0.0, 0.0, 0.0},  // [3] polysk
    {0.0, 0.0, 0.0, 0.0} }; // [4] kolor swiatla emitowanego

    GLfloat param_materialu2[6][4] = {//szary matowy widziany w bialym swietle
    {0.5, 0.5, 0.5, 1.0},  // [0] wspolczynnik odbicia swiatla otoczenia
    {0.5, 0.5, 0.5, 1.0},  // [1] wspolczynnik odbicia swiatla rozproszonego
    {0.0, 0.0, 0.0, 1.0},  // [2] wspolczynnik odbicia swiatla lustrzanego
    {0.0, 0.0, 0.0, 0.0},  // [3] polysk
    {0.0, 0.0, 0.0, 0.0} }; // [4] kolor swiatla emitowanego

    GLfloat param_materialu3[6][4] = {//polerowane zloto
    {0.247250, 0.224500, 0.064500, 1.000000},  // [0] wspolczynnik odbicia
swiatla otoczenia
    {0.346150, 0.314300, 0.090300, 1.000000},  // [1] wspolczynnik odbicia
swiatla rozproszonego
    {0.797357, 0.723991, 0.208006, 1.000000},  // [2] wspolczynnik odbicia
swiatla lustrzanego
    {83.2, 0.0, 0.0, 0.0},  // [3] polysk
    {0.0, 0.0, 0.0, 0.0} }; // [4] kolor swiatla emitowanego


void Material1(void) {
    glMaterialfv(GL_FRONT, GL_AMBIENT, material[0]);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, material[1]);
    glMaterialfv(GL_FRONT, GL_SPECULAR, material[2]);
    glMaterialfv(GL_FRONT, GL_SHININESS, material[3]);
    glMaterialfv(GL_FRONT, GL_EMISSION, material[4]);
}

void Material2(void) {
    glMaterialfv(GL_FRONT, GL_AMBIENT, material2[0]);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, material2[1]);
    glMaterialfv(GL_FRONT, GL_SPECULAR, material2[2]);
    glMaterialfv(GL_FRONT, GL_SHININESS, material2[3]);
    glMaterialfv(GL_FRONT, GL_EMISSION, material2[4]);
}

void Material3(void) {
    glMaterialfv(GL_FRONT, GL_AMBIENT, material3[0]);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, material3[1]);
    glMaterialfv(GL_FRONT, GL_SPECULAR, material3[2]);
    glMaterialfv(GL_FRONT, GL_SHININESS, material3[3]);
    glMaterialfv(GL_FRONT, GL_EMISSION, material3[4]);
}
```

```
void WyswietlObraz(void)
{
    // Wyczyszczenie bufora ramki i bufora glebokosci
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // Okreslenie wielkosci widoku, wlaczenie rzutowania perspektywicznego
    // i zainicjowanie stosu macierzy modelowania
    UstawParametryWidoku(szerokoscOkna, wysokoscOkna);

    // Ustalenie polozenia obserwatora
    glTranslatef(0, 0, odleglosc);
    glRotatef(rotObsX, 1, 0, 0);
    glRotatef(rotObsY, 0, 1, 0);
    glRotatef(rotObsZ, 0, 0, 1);

    // Generacja obrazu
    if (kolor == 0) Material1();
    else if (kolor == 1) Material2();
    else Material3();

    RysujWalec();

    if (normalne == 1) CzyNormalne();
    WlaczOswietlenie();

    // Narysowanie tekstow z opisem parametrow oswietlenia i materialu
    RysujNakladke();

    // Przelaczenie buforow ramki
    glutSwapBuffers();

    // Wy wietlanie siatki/materialow
    if (malowanie == 1) glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    else glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);

    if (cieniowanie == 1) glShadeModel(GL_SMOOTH);
    else glShadeModel(GL_FLAT);

    if (szybkosc_swiatla == 0) kat_swiatla = kat_swiatla + 0.5;
    else if (szybkosc_swiatla == 1) kat_swiatla = kat_swiatla + 0.25;
    else if (szybkosc_swiatla == 2) kat_swiatla = kat_swiatla + 0.125;

}
```

Efekt (kolor światła został zmieniony na biały)



Zmiana trybu cieniowania

W OpenGL istnieje kilka trybów cieniowania, które można ustawić, aby zmienić sposób, w jaki cienie są rysowane na bryłach.

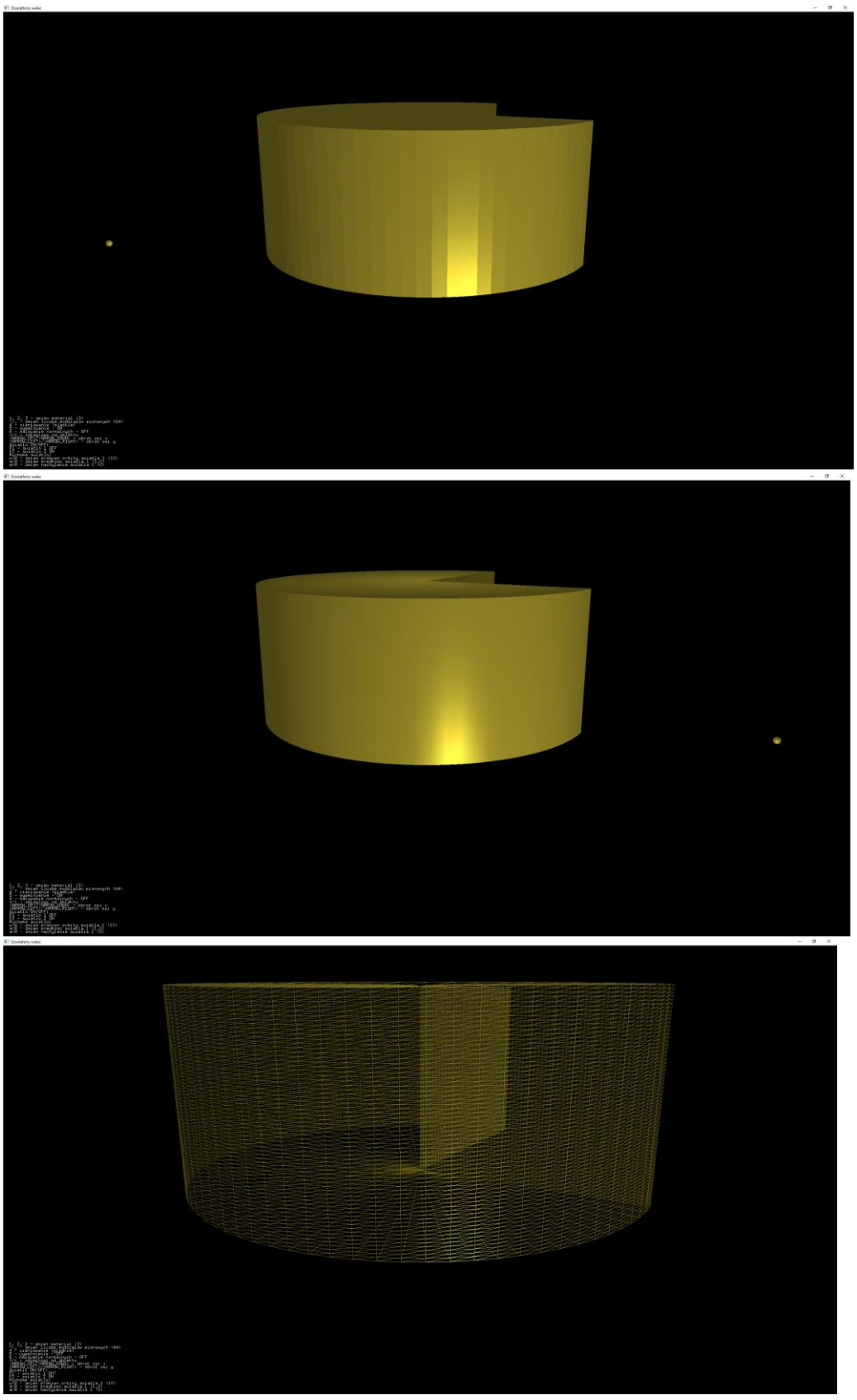
- GL_FLAT: Cienie są obliczane tylko dla pierwszego trójkąta tworzącego powierzchnię, co powoduje, że powierzchnie wyglądają płasko.
- GL_SMOOTH: Cienie są obliczane dla każdego punktu na powierzchni, co powoduje, że powierzchnie wyglądają bardziej naturalnie i z wygładzonymi przejściami kolorów.

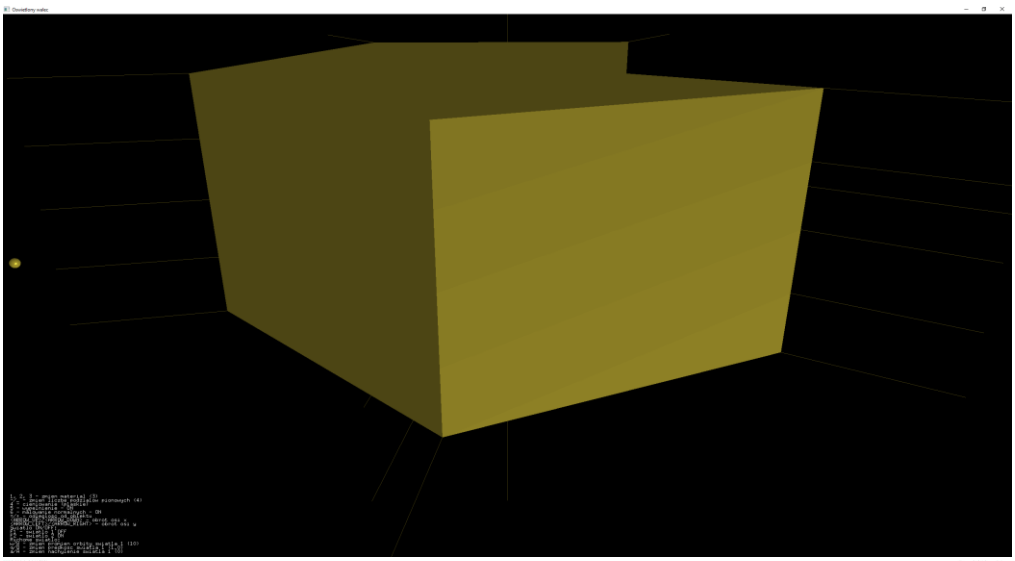
Aby zmienić tryb cieniowania, należy użyć funkcji glShadeModel i przekazać do niej odpowiedni argument (GL_FLAT lub GL_SMOOTH).

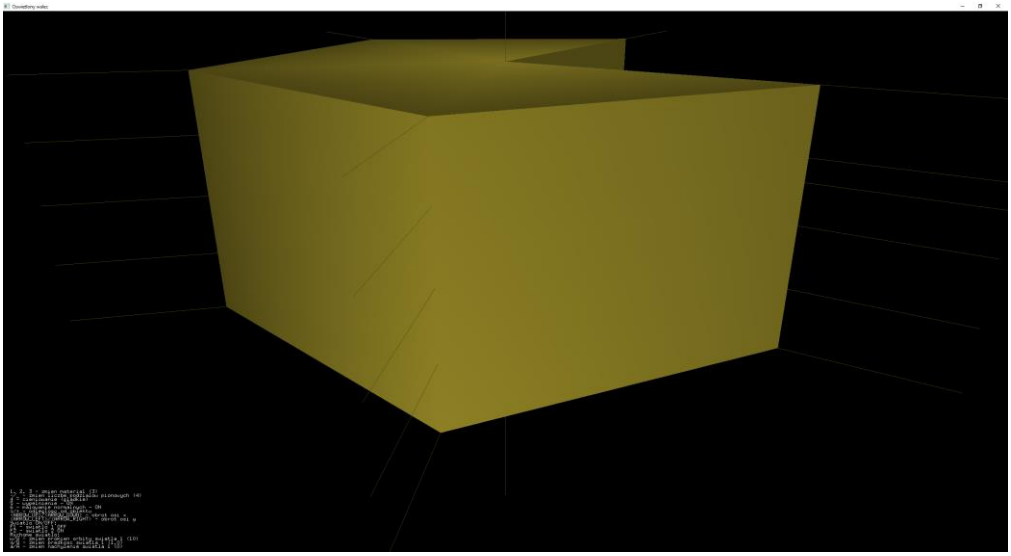
Kod programu

```
void WyswietlObraz(void)
{
    // Wyczyszczenie bufora ramki i bufora glebokosci
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // Okreslenie wielkosci widoku, wlaczenie rzutowania perspektywicznego
    // i zainicjowanie stosu macierzy modelowania
    UstawParametryWidoku(szerokoscOkna, wysokoscOkna);

    // Ustalenie polozenia obserwatora
    glTranslatef(0, 0, odleglosc);
    glRotatef(rotObsX, 1, 0, 0);
    glRotatef(rotObsY, 0, 1, 0);
    glRotatef(rotObsZ, 0, 0, 1);

    // Generacja obrazu
    if (kolor == 0) Material1();
    else if (kolor == 1) Material2();
    else Material3();

    RysujWalec();

    if (normalne == 1) CzyNormalne();
    WlaczOswietlenie();

    // Narysowanie tekstow z opisem parametrow oswietlenia i materialu
    RysujNakladke();

    // Przelaczenie buforow ramki
    glutSwapBuffers();

    // Wy wietlanie siatki/materialow
    if (malowanie == 1) glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    else glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);

    if (cieniowanie == 1) glShadeModel(GL_SMOOTH);
    else glShadeModel(GL_FLAT);

    if (szybkosc_swiatla == 0) kat_swiatla = kat_swiatla + 0.5;
    else if (szybkosc_swiatla == 1) kat_swiatla = kat_swiatla + 0.25;
    else if (szybkosc_swiatla == 2) kat_swiatla = kat_swiatla + 0.125;
}
```

Efekt







Zobrazowanie menu programu

Aby dodać menu do programu OpenGL za pomocą funkcji glutBitmapCharacter i glRasterPos2i, należy najpierw skonfigurować pozycję tekstu, a następnie wyświetlić tekst na ekranie. Tekst jest wyświetlany na ekranie za pomocą funkcji glutBitmapCharacter(), która przyjmuje jako argument czcionkę (np. GLUT_BITMAP_9_BY_15) i znak do wyświetlenia. Aby zrobić menu, należy po prostu dodać kolejne wywołania glutBitmapCharacter() i ustawić odpowiednią pozycję tekstu.

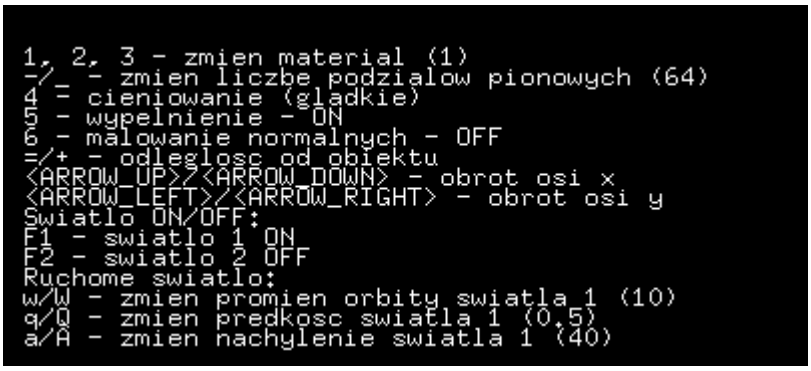
Kod programu

```
void RysujTekstRastrowy(void* font, char* tekst)
{
    for (int i = 0; i < (int)strlen(tekst); i++)
        glutBitmapCharacter(font, tekst[i]);
}

void RysujNakladke(void)//slasz piekne menu, nie trzaeba interaktywnie
{
    char buf[255];
    // Zmiana typu rzutu z perspektywicznego na ortogonalny
    glMatrixMode(GL_PROJECTION);
    glPushMatrix();
    glLoadIdentity();
    glOrtho(0.0, szerokoscOkna, 0.0, wysokoscOkna, -100.0, 100.0);

    // Modelowanie sceny 2D (zawartosci nakladki)
    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();
    glLoadIdentity();

    // Zablokowanie oswietlenia (mialoby ono wplyw na kolor tekstu)
    glDisable(GL_LIGHTING);

    // Okreslenie koloru tekstu
    glColor3f(1.0, 1.0, 1.0);

    // RYSOWANIE MENU
    glColor3f(1.0, 1.0, 1.0);

    sprintf(buf, " 1, 2, 3 - zmien material (%d)", kolor + 1);
    glRasterPos2i(X_OFFSET_OBIEKT, Y_OFFSET_OBIEKT - 150);
    RysujTekstRastrowy(GLUT_BITMAP_8_BY_13, buf);

    sprintf(buf, " -/_ - zmien liczbe podzialow pionowych (%.f)",
ilosc_podzialow);
    glRasterPos2i(X_OFFSET_OBIEKT, Y_OFFSET_OBIEKT - 160);
    RysujTekstRastrowy(GLUT_BITMAP_8_BY_13, buf);

    if (cieniowanie == 0) sprintf(buf, " 4 - cieniowanie (plaskie)");
    else  sprintf(buf, " 4 - cieniowanie (gladkie)");
    glRasterPos2i(X_OFFSET_OBIEKT, Y_OFFSET_OBIEKT - 170);
    RysujTekstRastrowy(GLUT_BITMAP_8_BY_13, buf);

    if (malowanie == 0) sprintf(buf, " 5 - wypelnienie - ON");
    else  sprintf(buf, " 5 - wypelnienie - OFF");
    glRasterPos2i(X_OFFSET_OBIEKT, Y_OFFSET_OBIEKT - 180);
    RysujTekstRastrowy(GLUT_BITMAP_8_BY_13, buf);
```

```
    if (normalne == 1) sprintf(buf, " 6 - malowanie normalnych - ON");
    else  sprintf(buf, " 6 - malowanie normalnych - OFF");
    glRasterPos2i(X_OFFSET_OBIEKT, Y_OFFSET_OBIEKT - 190);
    RysujTekstRastrowy(GLUT_BITMAP_8_BY_13, buf);

    sprintf(buf, " =/+ - odleglosc od obiektu");
    glRasterPos2i(X_OFFSET_OBIEKT, Y_OFFSET_OBIEKT - 200);
    RysujTekstRastrowy(GLUT_BITMAP_8_BY_13, buf);

    sprintf(buf, " <ARROW_UP>/<ARROW_DOWN> - obrot osi x");
    glRasterPos2i(X_OFFSET_OBIEKT, Y_OFFSET_OBIEKT - 210);
    RysujTekstRastrowy(GLUT_BITMAP_8_BY_13, buf);

    sprintf(buf, " <ARROW_LEFT>/<ARROW_RIGHT> - obrot osi y");
    glRasterPos2i(X_OFFSET_OBIEKT, Y_OFFSET_OBIEKT - 220);
    RysujTekstRastrowy(GLUT_BITMAP_8_BY_13, buf);

    sprintf(buf, " Swiatlo ON/OFF:");
    glRasterPos2i(X_OFFSET_OBIEKT, Y_OFFSET_OBIEKT - 230);
    RysujTekstRastrowy(GLUT_BITMAP_8_BY_13, buf);

    if (wlacznik1 == 1) sprintf(buf, " F1 - swiatlo 1 ON");
    else sprintf(buf, " F1 - swiatlo 1 OFF");
    glRasterPos2i(X_OFFSET_OBIEKT, Y_OFFSET_OBIEKT - 240);
    RysujTekstRastrowy(GLUT_BITMAP_8_BY_13, buf);

    if (wlacznik2 == 1) sprintf(buf, " F2 - swiatlo 2 ON");
    else sprintf(buf, " F2 - swiatlo 2 OFF");
    glRasterPos2i(X_OFFSET_OBIEKT, Y_OFFSET_OBIEKT - 250);
    RysujTekstRastrowy(GLUT_BITMAP_8_BY_13, buf);

    sprintf(buf, " Ruchome swiatlo:");
    glRasterPos2i(X_OFFSET_OBIEKT, Y_OFFSET_OBIEKT - 260);
    RysujTekstRastrowy(GLUT_BITMAP_8_BY_13, buf);

    sprintf(buf, " w/W - zmien promien orbity swiatla 1 (%.f)",
promien_swiatla);
    glRasterPos2i(X_OFFSET_OBIEKT, Y_OFFSET_OBIEKT - 270);
    RysujTekstRastrowy(GLUT_BITMAP_8_BY_13, buf);

    if (szybkosc_swiatla == 0) sprintf(buf, " q/Q - zmien predkosc swiatla 1
(1.0)");
    else if (szybkosc_swiatla == 1) sprintf(buf, " q/Q - zmien predkosc swiatla
1 (0.5)");
    else if (szybkosc_swiatla == 2) sprintf(buf, " q/Q - zmien predkosc swiatla
1 (0.125)");
    else sprintf(buf, " q/Q - zmien predkosc swiatla 1 (STOP)");
    glRasterPos2i(X_OFFSET_OBIEKT, Y_OFFSET_OBIEKT - 280);
    RysujTekstRastrowy(GLUT_BITMAP_8_BY_13, buf);

    sprintf(buf, " a/A - zmien nachylenie swiatla 1 (%.f)", nachylenie_swiatla);
    glRasterPos2i(X_OFFSET_OBIEKT, Y_OFFSET_OBIEKT - 290);
    RysujTekstRastrowy(GLUT_BITMAP_8_BY_13, buf);

    // Przywrocenie macierzy sprzed wywolania funkcji
    glMatrixMode(GL_PROJECTION);
    glPopMatrix();
    glMatrixMode(GL_MODELVIEW);
    glPopMatrix();

    // Odblokowanie oswietlenia
    glEnable(GL_LIGHTING);
```

}

Efekt


1, 2, 3 - zmien material (1)
-/_ - zmien liczbe podzialow pionowych (64)
4 - cieniowanie (gladkie)
5 - wypelnienie - ON
6 - malowanie normalnych - OFF
=/+ - odleglosc od obiektu
<ARROW_UP>/<ARROW_DOWN> - obrot osi x
<ARROW_LEFT>/<ARROW_RIGHT> - obrot osi y
Swiatlo ON/OFF:
F1 - swiatlo 1 ON
F2 - swiatlo 2 OFF
Ruchome swiatlo:
w/W - zmien promien orbity swiatla 1 (10)
q/Q - zmien predkosc swiatla 1 (0.5)
a/A - zmien nachylenie swiatla 1 (40)

4. Podsumowanie

Wszystkie postawione przede mną zadania zostały pomyślnie zrealizowane.
Podczas oddawania prac na zajęciach błędnie wykonałem wyznaczanie wektorów normalnych i co za tym idzie oświetlenie. Jak widać po powyższym opracowaniu niedociągnięcia te zostały poprawione.
Na zajęciach dowiedziałem, że OpenGL jest zaawansowaną i trudną do opanowania biblioteką graficzną, która pozwala na tworzenie trójwymiarowych aplikacji graficznych. W sprawozdaniu opisałem różne sposoby wykorzystania ten biblioteki takie jak: wyznaczanie i wyświetlanie wektorów normalnych, zmiana materiału bryły, zmiana trybu cieniowania, dodanie źródeł światła oraz tworzenie menu. Jest to tylko mała część możliwości OpenGl, posiada on bowiem jeszcze wiele różnych funkcji i opcji, takich jak animacje, efekty specjalne, itp., które pozwalają na dostosowanie wyglądu i zachowania aplikacji do indywidualnych potrzeb. Aby poznać je wszystkie trzeba poświęcić wiele czasu na naukę i praktykę, o czym przekonałem się na zajęciach, podczas poprawiania błędów i robienia zadania indywidualnego.