Ćwiczenie 4

OpenGL - interakcja z użytkownikiem

1. Cel ćwiczenia

Ćwiczenie ma za zadanie pokazać, jak przy pomocy funkcji biblioteki *OpenGL* z rozszerzeniem *GLUT* można zrealizować prostą interakcję, polegające na sterowaniu ruchem obiektu i położeniem obserwatora w przestrzeni 3D. Do sterowania służyla będzie mysz. Ponadto zostaną zilustrowanie sposoby prezentacji obiektów trójwymiarowych w rzucie perspektywicznym.

2. Rzutowanie perspektywiczne

W poprzednim ćwiczeniu stosowano rzutowanie równoległe, a dokładniej jego szczególny przypadek zwany rzutem ortograficznym. Do realizacji rzutowania ortograficznego służyfunkcja `glOrtho()`. W rzucie ortograficznym rzutnia, czyli płaszczyzna na której powstawał obraz, była równoległa do płaszczyzny tworzonej przez osie `x` i `y`, a proste rzutowania biegły równoległe do osi `z`. Narysowany w ten sposób czajnik wygląda tak jak na rysunku 1.

Rys. 1. Czajnik w rzucie ortograficznym

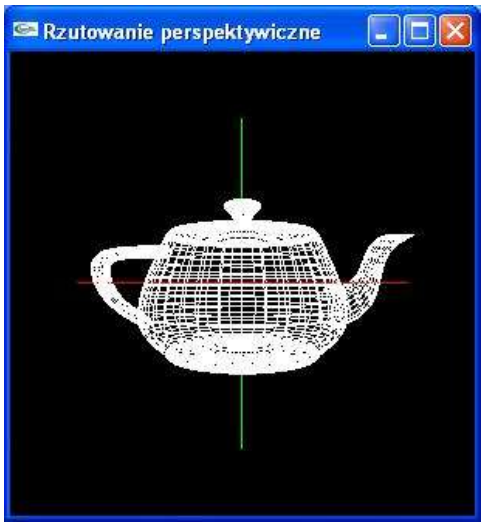
Ze względu na to, że proste rzutowania są równoległe do osi ***z***, przesuwanie obiektu wzdłuż tej osi nie spowoduje żadnego efektu na obrazie. Aby umożliwić pokazanie efektów przemieszczeń obiektu we wszystkich osiach nalely zastosować rzutowanie perspektywiczne. Rzut perspektywiczny jest lepszy od równoległego nie tylko ze względu na możliwość prezentacji przemieszczeń, pozwala także lepiej pokazać na płaszczyźnie geometrii trójwymiarowego obiektu.
Biblioteka *OpenGL* daje kilka możliwości definiowania rzutu perspektywicznego. Jedną z nich prezentuje szablon programu zamieszczony niżej.

```
/*************************************************************************************/
//  Szkielet programu do tworzenia modelu sceny 3-D z wizualizacją osi
//  układu współrzędnych dla rzutowania perspektywicznego
/*************************************************************************************/
#include <windows.h>
#include <gl/gl.h>
#include <gl/glut.h>

typedef float point3[3];
static GLfloat viewer[]= {0.0, 0.0, 10.0};
// inicjalizacja położenia obserwatora
/*************************************************************************************/
// Funkcja rysująca osie układu współrz?dnych

void Axes(void)
{

      point3  x_min = {-5.0, 0.0, 0.0};
      point3  x_max = { 5.0, 0.0, 0.0};
      // pocz?tek i koniec obrazu osi x

      point3  y_min = {0.0, -5.0, 0.0};
      point3  y_max = {0.0,  5.0, 0.0};
      // pocz?tek i koniec obrazu osi y

      point3  z_min = {0.0, 0.0, -5.0};
      point3  z_max = {0.0, 0.0,  5.0};
```

```
      //  pocz?tek i koniec obrazu osi y

      glColor3f(1.0f, 0.0f, 0.0f);  // kolor rysowania osi - czerwony
      glBegin(GL_LINES); // rysowanie osi x


            glVertex3fv(x_min);
            glVertex3fv(x_max);

      glEnd();

      glColor3f(0.0f, 1.0f, 0.0f);  // kolor rysowania - zielony
      glBegin(GL_LINES);  // rysowanie osi y

            glVertex3fv(y_min);
            glVertex3fv(y_max);

      glEnd();

      glColor3f(0.0f, 0.0f, 1.0f);  // kolor rysowania - niebieski
      glBegin(GL_LINES); // rysowanie osi z

            glVertex3fv(z_min);
            glVertex3fv(z_max);

      glEnd();

}
/*************************************************************************************/
// Funkcja określająca co ma być rysowane (zawsze wywoływana, gdy trzeba
// przerysować scenę)

void RenderScene(void)
{

       glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
      // Czyszczenie okna aktualnym kolorem czyszczącym

      glLoadIdentity();
      // Czyszczenie macierzy bie??cej

      gluLookAt(viewer[0],viewer[1],viewer[2], 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
      // Zdefiniowanie położenia obserwatora

      Axes();
      // Narysowanie osi przy pomocy funkcji zdefiniowanej powyżej


      glColor3f(1.0f, 1.0f, 1.0f);
      // Ustawienie koloru rysowania na biały

      glutWireTeapot(3.0);
      // Narysowanie czajnika

      glFlush();
      // Przekazanie poleceń rysujących do wykonania

     glutSwapBuffers();

 }
/*************************************************************************************/
// Funkcja ustalająca stan renderowania

void MyInit(void)
{
       glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
       // Kolor czyszczący (wypełnienia okna) ustawiono na czarny
}
/*************************************************************************************/

// Funkcja ma za zadanie utrzymanie stałych proporcji rysowanych
// w przypadku zmiany rozmiarów okna.
// Parametry vertical i horizontal (wysokość i szerokość okna) są
// przekazywane do funkcji za każdym razem gdy zmieni się rozmiar okna.

void ChangeSize(GLsizei horizontal, GLsizei vertical)
{

    glMatrixMode(GL_PROJECTION);
    // Przełączenie macierzy bieżącej na macierz projekcji

    glLoadIdentity();
    // Czyszcznie macierzy bieżącej

    gluPerspective(70, 1.0, 1.0, 30.0);
    // Ustawienie parametrów dla rzutu perspektywicznego


    if(horizontal <= vertical)
        glViewport(0, (vertical-horizontal)/2, horizontal, horizontal);

    else
        glViewport((horizontal-vertical)/2, 0, vertical, vertical);
    // Ustawienie wielkości okna okna widoku (viewport) w zależności
    // relacji pomiędzy wysokością i szerokością okna

    glMatrixMode(GL_MODELVIEW);
    // Przełączenie macierzy bieżącej na macierz widoku modelu

    glLoadIdentity();
    // Czyszczenie macierzy bieżącej

}
/*************************************************************************************/
// Główny punkt wejścia programu. Program działa w trybie konsoli
```

```
void main(void)
{

      glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB |GLUT_DEPTH);


      glutInitWindowSize(300, 300);

      glutCreateWindow("Rzutowanie perspektywiczne");

      glutDisplayFunc(RenderScene);
      // Określenie, że funkcja RenderScene będzie funkcją zwrotną
      // (callback function).  Będzie ona wywoływana za każdym razem
      // gdy zajdzie potrzeba przerysowania okna



      glutReshapeFunc(ChangeSize);
      // Dla aktualnego okna ustala funkcję zwrotną odpowiedzialną
      // za zmiany rozmiaru okna



      MyInit();
      // Funkcja MyInit() (zdefiniowana powyżej) wykonuje wszelkie
      // inicjalizacje konieczne  przed przystąpieniem do renderowania

      glEnable(GL_DEPTH_TEST);
      // Włączenie mechanizmu usuwania niewidocznych elementów sceny

      glutMainLoop();
      // Funkcja uruchamia szkielet biblioteki GLUT

}
/*************************************************************************************/
```

Po wykonaniu programu uzyskuje się obraz, który pokazano na rysunku 2. Obiekt jak widać wygląda trochę inaczej niż ten z poprzedniego rysunku i chyba bardziej przypomina swój trójwymiarowy pierwowzór.



Rys. 2. Czajnik w rzucie perspektywicznym

Zamieszczony powyżej kod programu jast bardzo podobny do tego, który być używany dla przypadku rzutowania ortograficznego w poprzednim ćwiczeniu. Różnią się od niego jednak w trzech miejscach:

- Na początku programu wprowadzono nową zmienną globalną `viewer[]`.
  Jest to tablica składająca się z trzech liczb, które określają współrz?dne `x`, `y`, `z`, położenia obserwatora.

- W funkcji `RenderScene()`, przed wywołaniem funkcji definiujących rysowane obiekty, umieszczono nową funkcję `gluLookAt()`. Jest to funkcja, która w prosty sposób pozwala na definiowanie położenia obserwatora trzymającego hipotetyczną kamerę. Składnia jej jest następująca:

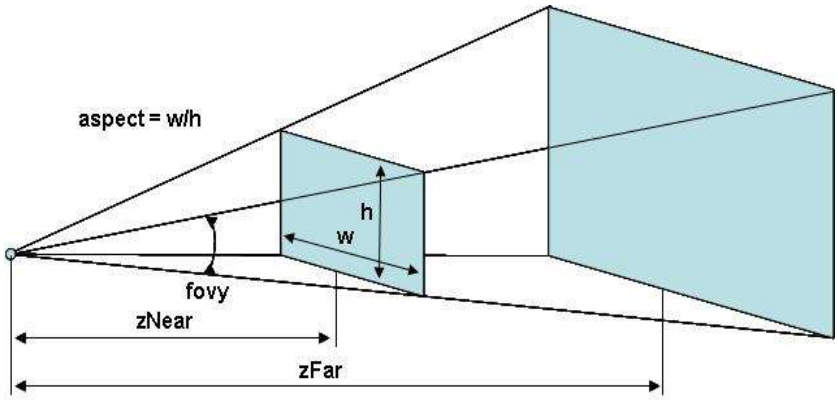
  ```
  void gluLookAt(GLdouble eyeX, GLdouble eyeY, GLdouble eyeZ,
                                GLdouble centerX, GLdouble centerY, GLdouble centerZ,
                                GLdouble upX , GLdouble upY, GLdouble upZ);
  ```

  Jak widać, funkcja ma 9 argumentów. Pierwsze trzy określają współrzędne punktu, w którym umieszczony jest obserwator. Trzy kolejne definiują punkt, na który obserwator patrzy, a ostatnia trójka wektor pozwalający na podanie skrócenia trzymanej przez obserwatora kamery. W przykładzie obserwator znajduje się w punkcie `(viewer[0], viewer[1], viewer[2]) = (0, 0, 10)` i patrzy na środek układu współrzędnych, czyli punkt (0, 0, 0). Hipotetyczna kamera jest ustawiona natomiast tak, że jej oś `y` pokrywa się z osią y układu współrzędnych. Ostatnie argumenty funkcji mają więc wartości `(upX, upY, upZ) = (0, 1, 0)`.

- W funkcji `ChangeSize()`, w miejsce `glOrtho()` zastosowano funkcję `gluPerspective()`, która służy do definiowania obszaru (bryły) widoczności dla rzutu perspektywicznego. Funkcja zdefiniowana jest tak:

  ```
  void gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble zNear, GLdouble zFar)
  ```

  Widok bryły i interpretację poszczególnych argumentów funkcji pokazano na rys. 3. Dwa widoczne na nim prostokąty, określają granice w jakich elementy wchodzące w skład opisu sceny będą widoczne na obrazie. Te fragmenty sceny, które leżą przed mniejszym i za większym prostokątem nie zostaną wyświetlone.



Rys. 3. Bryła widoczności dla rzutu perspektywicznego

Jak widać, pełne zdefiniowanie rzutu perspektywicznego nie jest aż takie proste. Należy podać `13` parametrów, `9` dla określenia sposobu patrzenia (ustawienia syntetycznej kamery) w funkcji `gluLookAt()` i `4` dla zdefiniowania właściwości kamery (kąta patrzenia i zakresu obcinania) w funkcji `gluPerspective()`. Jednak przy pewnej wprawie i odrobinie wyobraźni, przyjęta koncepcja pozwala na intuicyjne i w miarę wygodne manipulowanie widokiem.

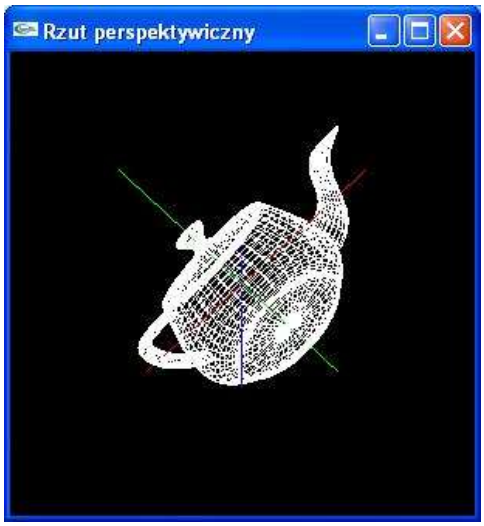
Dla przykładu jeśli zmienić ustawienie obserwatora z:

```
gluLookAt(0.0, 0.0, 10.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0);
```

na

```
gluLookAt(3.0, 3.0, 10.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0);
```

czyli lekko unieść i przesunąć kamerę w prawo `((viewer[0], viewer[1], viewer[2]) = (0, 0, 10))` z jednoczesnym jej obrotem o 45 stopni `(upX, upY, upZ) = (1, 1, 0)` uzyskany na ekranie obraz będzie wyglądał tak jak na rys. 4.



Rys. 4. Czajnik widziany z innego punktu obserwacji i przy przekr?conej kamerze

Na rysunku pojawiła się oś ***z*** (niebieska). Oś ta zdefiniowana była od początku w funkcji `Axes()` ale na żadnym z poprzednich rysunków nie było jej widać.

3. Przykład sposobu realizacji prostej interakcji

Na początek zostanie pokazane, jak zrealizować zadanie, które polega na obracaniu czajnika przy pomocy myszy. Zadanie można sformułować tak:

- Narysować czajnik w położeniu początkowym, tak aby wyglądać jak na rys. 2.
- W przypadku wciśnięcia lewego klawisza myszy, przesuwanie kursora myszy w prawo powinno powodować obrót czajnika wokół osi ***y*** w prawo.
- W przypadku wciśnięcia lewego klawisza myszy, przesuwanie kursora myszy w lewo powinno powodować obrót czajnika wokół osi ***y*** w lewo.

Trzeba wyraźnie zaznaczyć, że pożądany efekt można uzyskać na **dwa** sposoby. (1) Można obracać obiekt wokół osi przy ustalonym położeniu przy ustalonym położeniu obserwatora. (2) Można także obracać obserwatora wokół obiektu. W tym przypadku zastała wybrana (1) metoda.

Aby osiągnąć założony efekt, należy dokonać kilku modyfikacji podanego na początku opisu ćwiczenia kodu.

Aby osiągnąć założony efekt, należy dokonać kilku modyfikacji podanego na początku opisu ćwiczenia kodu:

1. W sekcji opisującej zmienne globalne dodać nowe zmienne:

```
static GLfloat theta = 0.0;   // kąt obrotu obiektu
static GLfloat pix2angle;     // przelicznik pikseli na stopnie

static GLint status = 0;       // stan klawiszy myszy
                               // 0 - nie naciśnięto żadnego klawisza
                               // 1 - naciśnięty został lewy klawisz

static int x_pos_old=0;       // poprzednia pozycja kursora myszy

static int delta_x = 0;        // różnica pomiędzy pozycją bieżącą
                                      // i poprzednią kursora myszy
```

2. Dodać dwie nowe funkcje zwrotne:

```
/*************************************************************************************/
// Funkcja "bada" stan myszy i ustawia wartości odpowiednich zmiennych globalnych

void Mouse(int btn, int state, int x, int y)
{


    if(btn==GLUT_LEFT_BUTTON && state == GLUT_DOWN)
    {
        x_pos_old=x;         // przypisanie aktualnie odczytanej pozycji kursora
                             // jako pozycji poprzedniej
        status = 1;          // wcięnięty został lewy klawisz myszy
    }
    else

        status = 0;          // nie został wcięnięty żaden klawisz
}

/*************************************************************************************/
// Funkcja "monitoruje" położenie kursora myszy i ustawia wartości odpowiednich
// zmiennych globalnych

void Motion( GLsizei x, GLsizei y )
{

    delta_x=x-x_pos_old;     // obliczenie różnicy położenia kursora myszy

    x_pos_old=x;            // podstawienie bieżącego położenia jako poprzednie

    glutPostRedisplay();     // przerysowanie obrazu sceny
}

/*************************************************************************************/
```

3. Zarejestrować dodane funkcje zwrotne w funkcji `main()`, dopisując dwie linie kodu:

```
glutMouseFunc(Mouse);
// Ustala funkcję zwrotną odpowiedzialną za badanie stanu myszy

glutMotionFunc(Motion);
// Ustala funkcję zwrotną odpowiedzialną za badanie ruchu myszy
```

4. W funkcji `ChangeSize()` (najlepiej na samym początku) dodać linię:

```
pix2angle = 360.0/(float)horizontal;  // przeliczenie pikseli na stopnie
```

5. W funkcji `RenderScene()` po wywołaniu funkcji rysującej osie a przed funkcję rysującą czajnik dodać:

```
if(status == 1)                     // jeśli lewy klawisz myszy wcięnięty
{
   theta += delta_x*pix2angle;    // modyfikacja kąta obrotu o kat proporcjonalny
}                                  // do różnicy położeń kursora myszy

glRotatef(theta, 0.0, 1.0, 0.0);  //obrót obiektu o nowy kąt
```

Tak zmodyfikowany kod pozwoli na zrealizowanie zadania sterowania obrotami w lewo i prawo przy pomocy myszy. Należy zauważyć, że sterowanie odbywa się przez modyfikację kąta obrotu nie przez aktualne położenie myszy, a różnicę pomiędzy położeniem bieżącym i poprzednim. Podane rozwiązanie umożliwia lepszą stabilizację ruchu obiektu i uniknięcie skokowych zmian w generowanym obrazie w przypadku nieciągłego ruchu myszy. Jest to rozwiązanie godne polecenia także w innych zadaniach polegających na przenoszeniu ruchu myszy na ruch obiektu na scenie.

4. Zadania do wykonania

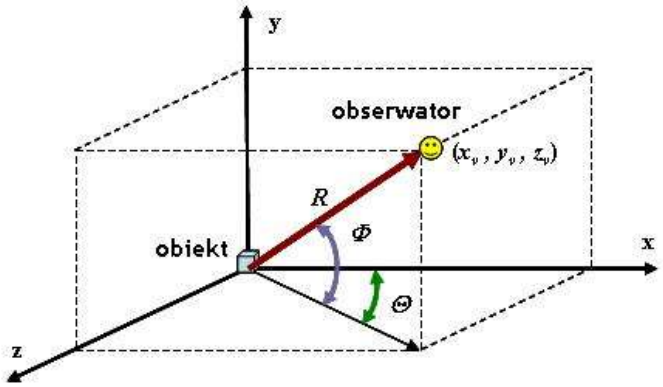
Zadanie 1. Należy zmodyfikować poprzednio napisany program tak aby:

- Po uruchomieniu programu czajnik był rysowany w położeniu początkowym.
- Przy wciśniętym lewym klawiszu myszy, ruch kursora myszy w kierunku poziomym powodować obroty czajnika wokół osi ***y*** (tak jak w poprzednim przykładzie).

- Przy wciśniętym lewym klawiszu myszy, ruch kursora myszy w kierunku pionowym powodować obroty czajnika wokół osi ***x*** .
- Przy wciśniętym prawym klawiszu myszy, ruchy kursora myszy w kierunku pionowym wywoływać zbliżanie i oddalanie się obserwatora od czajnika (swoisty rodzaj funkcji zoom).

Zadanie 2. Jako obiektu w miejsce czajnika, użyć zbudowanego w poprzednim ćwiczeniu modelu jajka a następnie zmodyfikować napisany wcześniej kod tak, aby można było oglądać jajko ze zmieniającego się punktu widzenia.
Sterowanie położeniem obserwatora powinno odbywać się przy pomocy myszy, przy następujących założeniach:

- Jajko znajduje się jest w środku układu współrzędnych.
- Punkt, w którym umieszczony jest obserwator może poruszać się po powierzchni sfery o promieniu ***R*** i środku leżącym w środku układu współrzędnych.
- Sterowanie położeniem obserwatora odbywa się powinno przy pomocy dwóch kątów.
  Pierwszy z nich określa kierunek patrzenia na obiekt i nosi nazwę **azymutu** i oznaczany będzie dalej jako ***Θ***. Drugi oznaczony przez ***Φ***, określa pośrednio wysokość położenia obserwatora nad hipotetycznym horyzontem i nazywa się kątem **elewacji** (elewacja określa w astronomii wysokość na jakiej gwiazda jest położona nad horyzontem). Odpowiedni układ geometryczny został zilustrowany na rys. 5.



Rys. 5. Obiekt, obserwator i kąty azymutu oraz elewacji

Na podstawie rys. 5 można dość łatwo wyznaczyć zależności wiążące położenie obserwatora z azymutem, kątem elewacji i promieniem sfery na powierzchni, której znajduje się obserwator. Są one następujące:

$$x_s(\Theta,\Phi) = R\cos(\Theta)\cos(\Phi)$$
$$y_s(\Theta,\Phi) = R\sin(\Phi)$$
$$z_s(\Theta,\Phi) = R\sin(\Theta)\cos(\Phi)$$
$$0 \le \Theta \le 2\pi$$
$$0 \le \Phi \le 2\pi$$

Ostatecznie zadanie brzmi tak:

- Po uruchomieniu programu, jajko powinno zostać narysowane w położeniu początkowym.
- Przy wciśniętym lewym klawiszu myszy, ruch kursora myszy w kierunku poziomym powinien powodować proporcjonalną zmianę azymutu kąta ***Θ***.
- Przy wciśniętym lewym klawiszu myszy, ruch kursora myszy w kierunku pionowym powinien powodować proporcjonalną zmianę kąta elewacji ***Φ***.
- Przy wciśniętym prawym klawiszu myszy, ruchy kursora myszy w kierunku pionowym winien realizować zmianę promienia ***R***.

W ten sposób, jeśli parametry zmieniające położenie kursora myszy na kąty i promień zostaną prawidłowo dobrane, obserwator będzie mógł zobaczyć jak wygląda jajko z każdej strony.

Zadania domowe

Ćwiczenie opracował: Jacek Jarnicki, korekta Marek Woda (2017-11-08)

Back