Animacja w OpenGL używając biblioteki GLUT

1. *Zasady ogólne*

   W scenach animowanych wrażenie ruchu może być wywołane przez:

   (a) ruch obserwatora,

   (b) ruch obiektów.

   Najprostsza implementacja ruchu przy pomocy biblioteki okienkowej GLUT polega na uruchomieniu nieskończonej pętli odrysowywania sceny, zmieniając przed kolejnym odrysowaniem parametry położenia obiektów i (lub) obserwatora. Potrzebna jest do tego funkcja

   ```
   glutIdleFunc(nazwa),
   ```

   gdzie `nazwa` jest nazwą funkcji odpowiedzialnej za odrysowywanie sceny.

   Funkcja `glutIdleFunc()`, która powinna być wywołana w funkcji `main()` działa analogicznie jak funkcja `glutDisplayFunc()`, tzn. rejestruje nazwę innej funkcji.

   Typowa postać funkcji `nazwa()` jest następująca:

   ```
   nazwa()
   {

     Wywołania zmieniające parametry ruchu;

     glutPostRedisplay(); // automatyczne wywołanie funkcji rysującej scenę,
                          // tzn. funkcji której nazwa jest parametrem funkcji
                          // glutDisplayFunc();
   }
   ```

   Zmienne odpowiedzialne za zmianę parametrów ruchu muszą być **globalne**, gdyż poruszające się obiekty są zdefiniowane w funkcji rysującej scenę[1], wobec czego ich ruch też jest tam opisany.

   Ponadto, jak było wspomniane na ćwiczeniach nr 1, dla uniknięcia efektu "migotnia sceny" (wywołanego głównie przez konieczność przerysowania tła okna przed kolejną klatką animacyjną) stosuje się dwie płaszczyzny obrazu (dwa bufory koloru): scena jest rysowana w płaszczyźnie niewidocznej w oknie (tzw. **tylnej** (ang. BACK)) i dopiero po zakończeniu procesu rysowania danej klatki animacyjnej płaszczyzna ta jest przełączana do postaci widocznej (staje się tzw. płaszczyzną **przednią** (ang. FRONT)). Jednocześnie dotychczasowa płaszczyzna przednia staje się tylną i w niej jest rysowana kolejna klatka animacyjna. Używanie dwóch plaszczyzn obrazowych wymaga przydzielenia pamięci dla obu z nich. Jak wiadomo po ćwiczeniach

[1] tzn. funkcji której nazwa jest parametrem funkcji **glutDisplayFunc()**

nr 1, odpowiada za to funkcja `glutDisplayMode()`. Dotychczas przydzielaliśmy dla okna pamięć dla pojedynczej płaszczyzny obrazu, za co odpowiadał parametr `GLUT_SINGLE`), a tym razem musimy go zastąpić parametrem `GLUT_DOUBLE`. Więc potrzebne na wywołanie ma postać

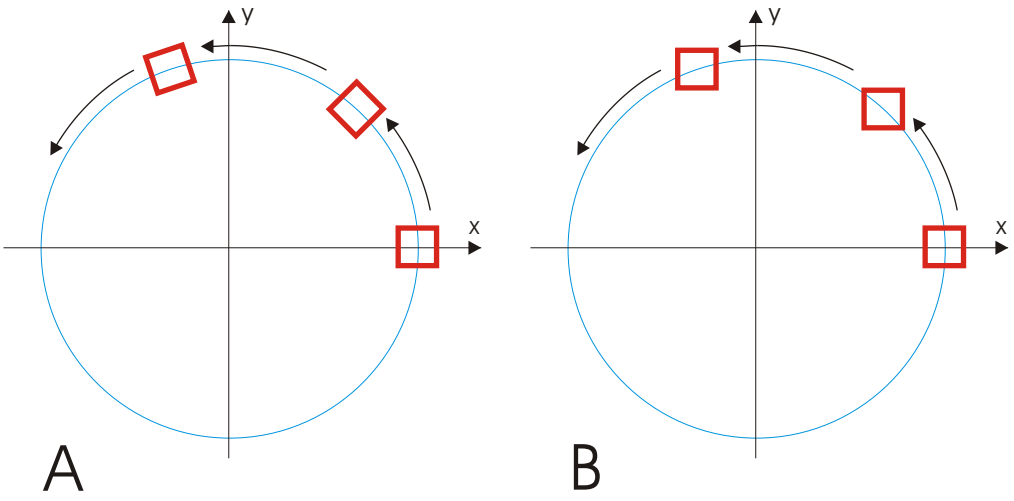
```
glutDisplayMode( GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH );
```

Dodatkowo musimy wymusić, aby po narysowaniu danej klatki animacyjnej w tylnym buforze, bufor ten został zamieniony rolą z dotychczasowym przednim, czyli musimy przełączyć bufory na końcu funkcji rysującej kolejne klatki animacyjne, (czyli po prostu funkcji rysujacej scenę z bieżącymi ustawieniami). Służy do tego wywołanie

```
glutSwapBuffers();
```

2. *Przykład: ruch obiektów po okręgu*

   Załóżmy, że chcemy zrealizować ruch po okręgu o promieniu $r = 5$ dookoła osi Z. Obiektem obracanym ma być kwadrat o boku długości 1, leżący w płaszczyźnie XY. Zadanie rozwiążemy w dwóch przypadkach określonych rysunkami poniżej.



   Wiadomo (zob. plik `opengl2.pdf`), że za obroty w OpenGL odpowiada np. funkcja `glRotatef()`, której parametrami są: kąt obrotu i oś obrotu. Zakładając, że początkowo środek kwadratu znajduje się w punkcie $(5, 0, 0)$ oraz że jego krawędzie są równoległe do osi X, Y, obrót będzie polegał na zwiększaniu pierwszego parametru tej funkcji, począwszy od wartości zero. Trzeba przypomnieć, że kąt obrotu w funkcji `glRotatef()` mierzony jest w stopniach. Oznaczmy kat obrotu przez α.

   (a) *Parametry animacyjne*

   Przy ustalonym promieniu $r = 5$ wystarczy zmieniać wartość kąta α o ustalony przedział $\Delta\alpha$, począwszy od wartości początkowej zero. Oznacza to że potrzebujemy zmiennej globalnej `alfa`, która będzie miała wartość początkową 0

stopni i która będzie się zmieniała z klatki na klatkę o wartość `delta_alfa` (równą np. 0.5 stopnia). W kodzie C++ możemy to zapisać jako:

```
const float delta_alfa=0.5;
float alfa=0.0;
```

(b) *Funkcja animacyjna.*

Załózmy, że funkcja animacyjna (tzn. parametr funkcji `glutIdleFunc()`) nazywa się `anim()`. Wtedy, jak wynika z punktu 1 powinna mieć ona postać

```
void anim()
{
  alfa+=delta_alfa;
  glutPostRedisplay();
}
```

(c) *Uwagi dotyczące rzutu i obserwatora*

Proszę zwrócić uwagę, że sformułowanie zagadnienia implikuje, że przez cały czas animacji zarówno rzut jak i obserwator pozostają stałe, tzn. stałe pozostają ich macierze. W związku z tym nie ma sensu inicjować ich przed każdą klatką animacyjną, tzn. na początku kodu funkcji rysującej scenę. Lepiej zrobić to przed uruchomieniem animacji, tzn. przed pierwszym wywołaniem funkcji rysującej scenę. Wykorzystamy do tego dodatkową funkcję, którą nazwiemy `init()`. Bedzie ona wywołana w funkcji `main()` przed funkcją `glutMainLoop()`, tzn. przed uruchomieniem pętli obsługi zdarzeń (zob. plik `opengl1.pdf`). Mając taką funkcję można w niej umieścić również wszystkie inne operacje, które nie bedą się zmieniały z klatki na klatkę. Zgodnie z zasadą działania stosów macierzowych jej rozsądna postać powinna być następująca:

```
void init()
{
  glClearColor(1.0,1.0,1.0,1.0); //kolor tła okna
  glEnable(GL_DEPTH_TEST); // włączenie testu głębokości

  glMatrixMode(GL_PROJECTION);
  gluPerspective(); // z właściwymi parametrami
  gluLookAt(); // z właściwymi parametrami
  glMatrixMode(GL_MODELVIEW); // nic więcej na stosie rzutowania
                              // się nie znajdzie
}
```

(d) *Funkcja rysująca układ XYZ*

Dla lepszego zaprezentowania ruchu wyświetlimy schematycznie osie układu XYZ. Posłużymy się poniższą funkcją `uklad()`.

```
uklad()
{

  glBegin(GL_LINES);
    glColor3f(0.0, 1.0, 0.0); //zielona oś X
    glVertex3f(-5.0, 0.0, 0.0);
    glVertex3f(5.0, 0.0, 0.0);

    glColor3f(0.0, 0.0, 1.0); // niebieska oś Y
    glVertex3f(0.0, -5.0, 0.0);
    glVertex3f(0.0,  5.0, 0.0);

    glColor3f(0.0, 0.0, 0.0); // czarna oś Z
    glVertex3f(0.0, 0.0, -5.0);
    glVertex3f(0.0, 0.0,  5.0);

  glEnd();

}
```

(e) *Funkcja rysująca scenę:*

Załóżmy, że funkcja rysująca scenę nazywa się `scena()`. Załóżmy ponadto, że mamy funkcję `kwadrat()` rysującą taki kwadrat o środku w punkcie $(0, 0, 0)$, tzn.

```
void kwadrat()
{
  glBegin(GL_QUADS);
    glVertex3f( 0.5,  0.5, 0.0);
    glVertex3f(-0.5,  0.5, 0.0);
    glVertex3f(-0.5, -0.5, 0.0);
    glVertex3f( 0.5, -0.5, 0.0);
  glEnd();

}
```

- **Przypadek A - plik** `ogl31.cpp`
  W każdej klatce animacyjnej wykonamy następujące czynności:
  i. załadujemy na stos modelowania macierz identycznościową - krok ten jest bez znaczenia dla klatki pierwszej (zob. plik `opengl2.pdf`) - w efekcie usuniemy macierz z poprzedniej klatki,
  ii. wyświetlimy układ współrzędnych
  iii. przesuniemy środek kwadratu do punktu $(5, 0, 0)$
  iv. obrócimy przesunięty kwadrat wokół osi Z o zwiększoną wartość kąta `alfa` mierzonego zawsze od osi X. Ponieważ obracany będzie kazdy wierzchołek oddzielnie, więc w miarę obrotu krawędzie będą zmieniały

nachylenie w stosunku do osi X (wierzchołki kwadratu obracają się o taki sam kąt jak jego środek).

Zgodnie z zasadami obsługi stosów macierzowych (operacje muszą być odkładane w kolejności odwrotnej do ich wykonania) kod funkcji `scena()` musi być następujący :

```
void scena()
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  glLoadIdentity();// przed kolejną klatką animacyjną
                   // na stosie modelowania nie mogą
                   // znajdować się żadne macierze z
                   // poprzedniej klatki animacyjnej

  uklad();

  glRotatef(alfa, 0.0, 0.0, 1.0);

  glTranslatef(5.0, 0.0, 0.0);

  kwadrat();

  glFlush();
  glutSwapBuffers(); // zamiana buforów koloru
}
```

- **Przypadek B - plik** `ogl32.cpp`
  W tym przypadku krawędzie kwadratu mają pozostać równoległe do osi X, Y. Trzeba wykonać korekcję obrotu wierzchołków kwadratu (ale nie jego środka), który dokonuje się po przesunięciu środka do punktu $(5, 0, 0)$. Zrobimy to wykonując obrót o taki sam kąt, w przeciwnym kierunku i w takim położeniu, że środek pozostaje bez zmian, a wierzchołki się obracają, tzn. w sytuacji gdy środek kwadratu jest w punkcie $(0, 0, 0)$, czyli leży na osi obrotu.
  W każdej klatce animacyjnej wykonamy następujące czynności:
  i. załadujemy na stos modelowania macierz identycznościową - krok ten jest bez znaczenia dla klatki pierwszej (zob. plik `opengl2.pdf`) - w efekcie usuniemy macierz z poprzedniej klatki,
  ii. wyświetlimy układ współrzędnych
  iii. obrócimy kwadrat wokół osi Z o zwiększoną wartość kąta `alfa` mierzonego zawsze od osi X, ale w kierunku przeciwnym niż dodatni kąt obrotu - tzn. wykonamy obrót o `-alfa`.
  iv. przesuniemy środek kwadratu do punktu $(5, 0, 0)$
  v. obrócimy obrócony i przesuniety kwadrat wokół osi Z o taką samą jak w punkcie 2(e)iii wartość kąta `alfa` ale w przeciwnym (tym razem

dodatnim) kierunku.

Kod funkcji `scena()` musi być następujący:

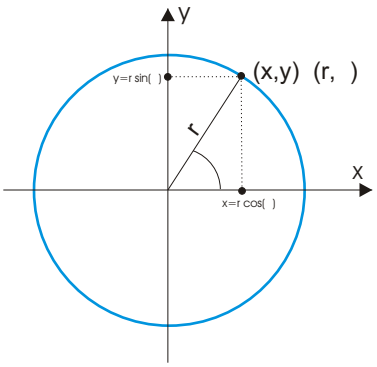
```
void scena()
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  glLoadIdentity();// przed kolejną klatką animacyjną
                   // na stosie modelowania nie mogą
                   // znajdować się żadne macierze z
                   // poprzedniej klatki animacyjnej

  uklad()

  glRotatef(alfa, 0.0, 0.0, 1.0);// srodek kwadratu w punkcie (5,0,0);

  glTranslatef(5.0, 0.0, 0.0);

  glRotatef(-alfa, 0.0, 0.0, 1.0);// środek kwadratu w punkcie (0,0,0)

  kwadrat();

  glFlush();
  glutSwapBuffers(); // zamiana buforów koloru
}
```

W efekcie działania funkcji `glutIdleFunc(anim)` funcja `scena()` będzie wywoływana w nieskończonej pętli.

3. *Przykład: ruch obserwatora po okręgu*

   Rozpatrzmy ruch obserwatora w płaszczyźnie XY po okręgu o promieniu $r = 5$ dookoła osi Z. W środku układu współrzędnych znajduje się sześcian o krawędzi długości 1 i ścianach równoległych do ścian układu. Przez cały czas działania animacji wzrok obserwatora ma być skierowany na środek układu. Zakładamy, że sześcian opisany jest funkcją `kostka()`. Kod definiujący taki sześcian znajduje się w programach `ogl21.cpp` lub `ogl22.cpp`. Będziemy się posługiwali analogicznie nazwanymi funkcjami jak w przykładzie ruchu obiektów.

   (a) *Użyteczna definicja okręgu:*



Równanie $x^2+y^2=r^2$ przedstawia okrąg o środku w punkcie $(0,0)$ i promieniu r. Nazwijmy go $S((0,0),r)$. Wtedy

$$S((0,0),r) = \{(x,y) \in \mathbb{R}^2 : \ x^2+y^2=r^2\}.$$

Równoważnie każdy punkt tego okręgu może być przedstawiony w tzw. *współrzędnych biegunowych*, tzn. przy pomocy pary (r,α), gdzie α jest kątem nachylenia promienia okręgu w stosunku do osi X. Zbiór wszystkich punktów okręgu jest wtedy wyznaczony przez wszystkie pary (r,α), gdzie α przebiega zbiór kątów $[0°, 360°)$. Ponieważ funkcje trygonometryczne w typowych bibliotekach matematycznych (np. `math.h`) posługują się pojęciem *radiana*[2], więc ostatecznie

$$S((0,0),r) = \{(r,\alpha) : \ \alpha \in [0, 2\pi)\}.$$

(b) *Funkcja pomocnicza* `init()`.

Tym razem sformułowanie zadania wymusza, żeby funkcja generująca obserwatora zmieniała swoje parametry przed wygenerowaniem każdej klatki animacyjnej, więc jedynie rzut pozostanie stały, przez cały czas działania animacji. Zatem jasne jest, że kod pomocniczej funkcji `init()` musi być następujący:

```
void init()
{
  glClearColor(1.0,1.0,1.0,1.0); //kolor tła okna
  glEnable(GL_DEPTH_TEST); // włączenie testu głębokości

  glMatrixMode(GL_PROJECTION);
  gluPerspective(); // z właściwymi parametrami
  glMatrixMode(GL_MODELVIEW); // nic więcej na stosie rzutowania
                              // się nie znajdzie
}
```

(c) *Parametry animacyjne*

Podobnie jak w poprzednim przykładzie będziemy potrzebowali zmiennej określającej aktualny kat obrotu obserwatora. Ponownie nazywamy ją `alfa` i inicjujemy wartością zero. Zakładamy interpretacyjnie, że `alfa` jest mierzona w

[2]Ilość radianów odpowiadająca α stopniom wynosi $\frac{\pi\,\alpha}{180}$.

stopniach. Dla ruchu obserwatora użyjemy biegunowej postaci równania okręgu. Wówczas potrzebna bedzie nam stała `pi`. Ostatecznie globalny blok deklaracji ma postać:

```
const float pi=3.1415926535897;
const float delta_alfa=0.5;
float alfa=0.0;
```

(d) *Funkcja* `scena()`.

Dla każdej klatki animacyjnej wykonamy następujące czynności:

i. załadujemy na stos modelowania macierz identycznościową - krok ten jest bez znaczenia dla klatki pierwszej (zob. plik `opengl2.pdf`) - w efekcie usuniemy macierz z poprzedniej klatki,

ii. zdefiniujemy aktualne położenie obserwatora

iii. wyświetlimy układ współrzędnych

iv. wygenerujemy sześcian w środku układu współrzędnych

Kod funkcji `scena()` musi być następujący - plik `ogl33.cpp`:

```
void scena()
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  glLoadIdentity();// przed kolejną klatką animacyjną
                   // na stosie modelowania nie mogą
                   // znajdować się żadne macierze z
                   // poprzedniej klatki animacyjnej


  gluLookAt(5*cos(pi*alfa/180.0), 5*cos(pi*alfa/180.0),
            6.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0);
  //obserwator znajduje się na wysokości 6.

  uklad();

  kostka();

  glFlush();
  glutSwapBuffers(); // zamiana buforów koloru
}
```

Prosty efekt złożenia ruchów obserwatora i obiektu uzyskamy np. obracając sześcian dookoła osi X, w czasie gdy obserwator obraca się wokół osi Z. Wystarczy przed wywołaniem funkcji `kostka()` dodać wywołanie

```
glRotatef(alfa, 1.0, 0.0, 0.0);
```

4. *Przykład - ruch cykliczny po odcinku*

Rozważmy cykliczny ruch obiektu pomiędzy dwoma punktami P_1 i P_2 po najkrótszej drodze, tzn. po odcinku łączącym te punkty. Załóżmy, że przed rozpoczęciem ruchu obiekt znajduje się w środku tego odcinka. Najpierw przesuwa się w kierunku punku P_1, a następnie "odbija od niego" kierując się w kierunku punktu P_2. Po jego osiągnięciu wraca do P_1 itd.

(a) *Ogólne obserwacje*

Jasne jest, że bez zmiany ogólności możemy założyć, że odcinek ma długość 1, mieści się na osi X i jego środek znajduje się w punkcie $(0, 0, 0)$. W przeciwnym przypadku zawsze możemy użyć przesunięć, obrotów i skalowań w celu przekształcenia takiego odcinka do dowolnej długości, orientacji i położenia. Zatem potrzebujemy tylko jednej zmiennej animacyjnej, oznaczającej bieżące położenie obiektu na osi X. Załóżmy, że zmienna ta ma nazwę `pozycja`. Zatem na początku

```
float pozycja=0.0;
```

(b) *Funkcja animacyjna - rozwiązanie pierwsze: ruch jednostajny*

Zagadnienie rozwiążemy wprowadzając drugą zmienną globalną oznaczającą wielkość kroku, o który zmienia się pozycja pomiędzy klatkami animacyjnymi. Nazwijmy tą zmienną `przyrost` i zainicjujmy wartością 0.01. Wówczas obiekt będzie się początkowo poruszał w kierunku punktu 0.5 na osi X. Ale w tym położeniu powinien się "odbić", co wykonamy zmieniając znak przyrostu na przeciwny - ujemmy w tym przypadku. Po osiągnięciu punktu -0.5 na osi X ponownie zmieniamy znak zmiennej `przyrost`. Zatem deklaracja zmiennej globalnej `przyrost` ma postać

```
przyrost=0.01;
```

Kod funkcji animacyjnej jest następujący:

```
void anim()
{

  pozycja+=przyrost;
  if (pozycja >= 0.5) przyrost*=-1;
  else (pozycja <= -0.5) przyrost*=-1;

  glutPostRedisplay();

}
```

Program `ogl34.cpp` pokazuje schemat tego ruchu używając poruszającego się wierzchołka - używamy do tego wywołania `glBegin(GL_POINTS)`. Normalnie pojedyncze wierzchołki są reprezentowane przez pojedyncze piksele. Dla sensownej wizualizacji zwiększamy obraz wierzchołka 10 razy, używając funkcji

```
glPointSize(10);
```

(c) *Funkcja animacyjna - rozwiązanie pierwsze: ruch niejednostajny*

Jeżeli nie zależy nam na jednostajności ruchu, a jedynie na jego cykliczności, to możemy użyć np. funkcji `sin()`, której wartości zmieniają się cyklicznie pomiędzy -1 i 1. Startując od `pozycja=0.0` wartość `sin(pozycja)` rośnie do 1.0, a potem maleje. Dla realizacji ruchu pomiędzy -0.5 o 0.5 wystarczy pomnożyć wartość tej funkcji przez 0.5. Zatem w tym przypadku kod funkcji animacyjnej jest następujący:

```
void anim()
{

  pozycja+=przyrost;

  glutPostRedisplay();
}
```

Poruszający się wierzchołek będzie miał kod

```
glVertex3f(0.5*sin(pozycja))
```

Pokazuje to program `ogl35.cpp`.

5. *Zadanie*

Wykonać animację wahadła zegarowego. Wahdało składa się z dwóch prostopadłościanów, oś obrotu też jest przybliżana przez prostopadłościan. Nie trzeba uwzględniać fizyki, tzn. wystarczy zrealizować jednostajny ruch wahadła.