Grafika komputerowa

INSTRUKCJA DO LABORATORIUM 2: operacje przestrzenne oraz obsługa klawiatury i myszki

mgr inż. Paweł Stawarz
Rzeszów, 2015

CEL ĆWICZENIA

Celem ćwiczenia jest zapoznanie się z podstawowymi operacjami w grafice trójwymiarowej – przesunięciem, skalowaniem i rotacją oraz ze sposobami pracy z myszką i klawiaturą.

WAŻNA INFORMACJA

NINIEJSZA INSTRUKCJA ZAWIERA LISTĘ FUNKCJI Z BIBLIOTEK OPENGL ORAZ FREEGLUT ORAZ ICH POBIEŻNY OPIS. W CELU DOKŁADNIEJSZEGO ZROZUMIENIA DZIAŁANIA JAK I MOŻLIWOŚCI TYCHŻE METOD, STUDENT POWINIEN ODWOŁAĆ SIĘ DO DOKUMENTACJI WSPOMNIANYCH BIBLIOTEK. NIE NALEŻY SIĘ TAKŻE OGRANICZAĆ DO FUNKCJI WYMIENIONYCH W INSTRUKCJI - UŻYWANIE INNYCH METOD, ZNALEZIONYCH W DOKUMENTACJI, JEST WYSOCE WSKAZANE.

Dokumentacje używanych bibliotek można znaleźć w następujących miejscach:

> OpenGL 2.1 (dostępne na maszynach w pracowni):
> https://www.opengl.org/sdk/docs/man2/
>
> OpenGL 4.5 (najnowsze w chwili pisania instrukcji):
> https://www.opengl.org/sdk/docs/man4/
>
> GLUT (większość funkcji taka sama jak w freeglut, lepiej opisane):
> https://www.opengl.org/resources/libraries/glut/spec3/spec3.html
>
> freeglut:
> http://freeglut.sourceforge.net/docs/api.php#Initialization

DODATKOWE BIBLIOTEKI

Do pracy z grafiką trójwymiarową, przyda nam się biblioteka GLU. Aby z niej korzystać, należy dodać w opcjach projektu flagę `-lglu32` (patrz: instrukcja do poprzednich laboratoriów, strona 9).

Jeżeli chcemy, możemy też ***samodzielnie ściągnąć i dodać inne biblioteki***. Przykładem mogą być biblioteki do ładowania tekstur, czy też modeli. Nie ma potrzeby pytać prowadzącego o pozwolenie. Jeżeli bibliotekę można zainstalować na stanowisku pracy w laboratorium, można jej używać. Nie należy tracić czasu odkrywając koło na nowo.

Nie można natomiast korzystać z gotowych silników graficznych. Nie dlatego, że są złe, lecz dlatego, że nie nauczylibyśmy się przez to ogólnych zasad rządzących grafiką komputerową, gdyż takie silniki często ukrywają przed nami detale implementacji.

TEORIA – OPERACJE MACIERZOWE

W celu zrozumienia rzutowania i transformacji przestrzennych, kluczowym jest, by zrozumieć bardzo istotną rzecz: widok (rzut, projekcja) jest (bardzo upraszczając sprawę) w OpenGL macierzą.

Gdy tworzyliśmy nasz program na poprzednich zajęciach, domyślnie została nam przypisana macierz jednostkowa, która jest macierzą kwadratową, diagonalną, której wszystkie wartości na przekątnej są równe 1. Wygląda więc następująco:

$$I = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Ważną własnością takiej macierzy jest to, że jej iloczyn z dowolną inną macierzą o takich samych wymiarach, daje zawsze w wyniku taką samą macierz jak przed mnożeniem. Czytaj:

$$A \bullet I = A \qquad I \bullet A = A$$

Gdzie `•` oznacza mnożenie macierzy. Mnożenie macierzy jest nieprzemienne (z wyjątkiem mnożenia przez macierz jednostkową i macierz zerową).

Jest to istotne dlatego, że wykonywanie transformacji przestrzennych w bibliotece OpenGL, to w gruncie rzeczy właśnie mnożenie macierzy. Gdy więc stworzymy sześcian i będziemy chcieli go obrócić – pomnożymy aktualną macierz transformacji przez macierz definiującą obrót. Oczywiście ręczne definiowanie takich macierzy byłoby niezmiernie męczące i podatne na błędy użytkownika. Dlatego też biblioteka OpenGL udostępnia nam wiele funkcji, które „ukrywają" przed nami operacje macierzowe i definiują macierze o pożądanych wartościach za nas.

RZUTOWANIE

OpenGL oferuje dwa rodzaje rzutowania – rzutowanie równoległe i perspektywiczne. Technicznie można osiągnąć inne rodzaje rzutowań poprzez odpowiednie operacje macierzowe, aczkolwiek nie jest to zazwyczaj konieczne.

Ortograficzne

Aby włączyć rzutowanie równoległe, należy użyć funkcji glOrtho, której deklaracja wygląda następująco:

```
void glOrtho(       GLdouble    lewa,
                    GLdouble    prawa,
                    GLdouble    dol,
                    GLdouble    gora,
                    GLdouble    przod,
                    GLdouble    tyl
             );
```

Gdzie:

LEWA: pozycja lewej płaszczyzny przycinania,

PRAWA: pozycja prawej płaszczyzny przycinania,

DOL: pozycja dolnej płaszczyzny przycinania,

GORA: pozycja górnej płaszczyzny przycinania,

PRZOD: pozycja przedniej płaszczyzny przycinania (mała wartość **dodatnia**, np. 0.01),

TYL: pozycja tylnej płaszczyzny przycinania (duża wartość dodatnia. np. 100.0)

Perspektywiczne

Aby włączyć rzutowanie perspektywiczne, należy użyć funkcji GLFRUSTUM bądź GLUPERSPECTIVE.

```
void glFrustum(     GLdouble     lewa,
                    GLdouble     prawa,
                    GLdouble     dol,
                    GLdouble     gora,
                    GLdouble     przod,
                    GLdouble     tyl
               );
```

Gdzie:

LEWA: pozycja lewej płaszczyzny przycinania,

PRAWA: pozycja prawej płaszczyzny przycinania,

DOL: pozycja dolnej płaszczyzny przycinania,

GORA: pozycja górnej płaszczyzny przycinania,

PRZOD: pozycja przedniej płaszczyzny przycinania (mała wartość **dodatnia**, np. 0.01),

TYL: pozycja tylnej płaszczyzny przycinania (duża wartość dodatnia. np. 100.0)

```
void gluPerspective(      GLdouble     katWidzenia,
                          GLdouble     proporcje,
                          GLdouble     przod,
                          GLdouble     tyl
                    );
```

Gdzie:

KATWIDZENIA: kąt widzenia w pionie, w stopniach,

PROPORCJE: proporcje obszaru widzenia. Powinny być takie same jak proporcje okna programu (jeżeli nie zmieniono nic od poprzednich zajęć, wartość tego parametru to $[szerokość]/[wysokość]=300.0/300.0=1.0$).

PRZOD: pozycja przedniej płaszczyzny przycinania (mała wartość **dodatnia**, np. 0.01),

TYL: pozycja tylnej płaszczyzny przycinania (duża wartość dodatnia. np. 100.0)

TRANSFORMACJE PRZESTRZENNE

Podstawowe transformacje przestrzenne to translacja (przesunięcie), rotacja (obrót) i skalowanie. Odpowiadają im następujące funkcje biblioteki OpenGL:

Translacja:

```
void glTranslatef(GLfloat x, GLfloat y, GLfloat z);
```

Rotacja:

```
void glRotatef(GLfloat kat, GLfloat x, GLfloat y, GLfloat z);
```

Skalowanie:

```
void glScalef(GLfloat x, GLfloat y, GLfloat z);
```

Gdzie, w każdej z tych funkcji:

X: zmiana w osi X,

Y: zmiana w osi Y,

Z: zmiana w osi Z,

KAT: kąt o który obrócić

Kolejność wywołania tych funkcji ***ma znaczenie***, tak jak i w rzeczywistości. Osobną klasę transformacji stanowią transformacje bezpośrednio na macierzach. Aby powiadomić maszynę stanów na której macierzy właśnie wykonujemy operacje, należy wywołać funkcję GLMATRIXMODE:

```
void glMatrixMode(GLenum mode);
```

Gdzie argument MODE może przyjmować wartości:

GL_MODELVIEW: jeżeli chcemy pracować z macierzą modelu,

GL_PROJECTION: jeżeli chcemy pracować z macierzą widoku,

GL_TEXTURE: jeżeli chcemy pracować z macierzą tekstury

GL_COLOR: jeżeli chcemy pracować z macierzą koloru

W praktycznych zastosowaniach – będziemy używać tylko dwie pierwsze wartości. *GL_PROJECTION* tuż przed wywołaniem którejkolwiek z funkcji dokonujących rzutowania, zaś *GL_MODELVIEW* we wszelkich innych wypadkach.

Macierze – niezależnie od ich rodzaju – tworzą stos (jego maksymalny rozmiar jest zależny od implementacji i rodzaju macierzy). Podczas obliczania wyjściowej pozycji kamery, czy obiektu, wszystkie macierze są mnożone przez siebie, w kolejności FIFO. Mówiąc „aktualna macierz”,

będziemy mieć na mysli ostatnio dodaną macierz (a więc tą na górze stosu). Aby położyć nową macierz na stos, użyjemy funkcji:

```
void glPushMatrix(void);
```

Funkcja ta tworzy ***kopię*** aktualnej macierzy. Aby ściągnąć macierz ze stosu użyjemy funkcji:

```
void glPopMatrix(void);
```

Aby podmienić aktualną macierz na macierz jednostkową (której zalety omówiliśmy wcześniej), użyjemy funkcji GLLOADIDENTITY:

```
void glLoadIdentity(void);
```

Aby podmienić aktualną macierz na dowolną, przez nas wybraną, użyjemy funkcji GLLOADMATRIXF:

```
void glLoadMatrixf(const GLfloat *tablicaWartosci);
```

Gdzie tablica wartości wyglądająca następująco:

```
float m[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15};
```

reprezentuje następującą macierz:

$$m = \begin{bmatrix} 0 & 4 & 8 & 12 \\ 1 & 5 & 9 & 13 \\ 2 & 6 & 10 & 14 \\ 3 & 7 & 11 & 15 \end{bmatrix}$$

Ostatnią przydatną dla nas funkcją jest funkcja mnożąca aktualną macierz przez inną, zdefiniowaną przez nas macierz:

```
void glMultMatrixf(const GLfloat *tablicaWartosci);
```

W której tablicę wartości definiujemy w identyczny sposób jak w GLLOADMATRIXF.

TRANSFORMACJE W PRAKTYCE

Uproszczony algorytm działania jest następujący:

```
1. Podczas każdej klatki:
      1.1. Rozpocznij pracę ze stosem macierzy widoku,
      1.2. Załaduj macierz jednostkową,
      1.3. Ustaw wybrany sposób rzutowania,
      1.4. Rozpocznij pracę ze stosem macierzy modelu,
      1.5. Dokonaj transformacji przestrzennych,
      1.6. Narysuj obiekt.
```

Przykładowa implementacja powyższego algorytmu (korzystająca z bibliotek FREEGLUT i GLU) może wyglądać następująco:

```
#include <iostream>
#include <GL/freeglut.h>

using namespace std;

void rysuj();

int main(int argc, char **argv)
{
	glutInit(&argc, argv);
	glutCreateWindow("Hello World");
	glutDisplayFunc(rysuj);
	glutMainLoop();
	return 0;
}

void rysuj(){                                          // 1
	static float kat=0.0f;
	kat+=0.01f;
	if(kat>=360.0) kat-=360.0;

	glMatrixMode(GL_PROJECTION);                       // 1.1
	glLoadIdentity();                                  // 1.2
	gluPerspective(45.0f, 300.0f/300.0f, 0.1, 100.0);  // 1.3

	glMatrixMode(GL_MODELVIEW);                        // 1.4
	glLoadIdentity();

	glClearColor(0.97f, 0.24f, 0.71f, 1.0f);
	glClear(GL_COLOR_BUFFER_BIT);

	glTranslatef( 0, 0, -5);                           // \
	glRotatef(kat, 0, 1, 0);                           // --> 1.5
	glBegin(GL_POLYGON);
		glVertex3f( -1.0f, -1.0f, 0);
		glVertex3f(  1.0f, -1.0f, 0);
		glVertex3f(  0.0f,  1.0f, 0);
	glEnd();

	glutPostRedisplay();
	glutSwapBuffers();
}
```

MYSZKA I KLAWIATURA

Biblioteka FREEGLUT oferuje kilka różnych funkcji umożliwiających ustawienie własnych metod obsługi zdarzeń pochodzących od klawiatury i myszki. W przypadku klawiatury osobno interpretowane są klawisze zwykłe (a więc litery, cyfry i tym podobne), specjalne (klawisze funkcyjne, strzałki i inne klawisze sterowania kursorem) oraz modyfikatory (ALT, CONTROL, SHIFT). Rozróżniane są także obsługiwane zdarzenia wciśnięcia i puszczenia klawiszy, dzięki czemu możemy na przykład rozróżnić „kliknięcie” klawisza, od jego przytrzymania.

Obsługa zwykłych klawiszy:

```
void glutKeyboardFunc(void (*funkcjaPoNacisnieciu)(unsigned char klawisz, int xMyszki, int yMyszki));
```

```
void glutKeyboardUpFunc(void (*funkcjaPoPuszczeniu)(unsigned char klawisz, int xMyszki, int yMyszki));
```

Obsługa specjalnych klawiszy:

```
void glutSpecialFunc(void (*funkcjaPoNacisnieciu)(int idKlawisza, int xMyszki, int yMyszki));
```

```
void glutSpecialUpFunc(void (*funkcjaPoPuszczeniu)(int idKlawisza, int xMyszki, int yMyszki));
```

Obsługa modyfikatorów:

```
void glutGetModifiers(void);
```

Gdzie, w każdej z tych funkcji:

FUNKCJAPONACISNIECIU: wskaźnik do funkcji o odpowiedniej liczbie i typie argumentów oraz zwracającej VOID. Patrz: instrukcja do poprzednich laboratoriów, strona 8,

FUNKCJAPOPUSZCZENIU: wskaźnik do funkcji o odpowiedniej liczbie i typie argumentów oraz zwracającej VOID. Patrz: instrukcja do poprzednich laboratoriów, strona 8,

KLAWISZ: kod ASCII klawisza (np. 'a' dla klawisza a),

IDKLAWISZA: identyfikator klawisza. Wartości wyjaśnione w dokumentacji funkcji,

XMYSZKI: Pozycja x (w pikselach) kursora myszki, w momencie wystąpienia zdarzenia,

YMYSZKI: Pozycja y (w pikselach) kursora myszki, w momencie wystąpienia zdarzenia,

W wypadku myszki, możliwe jest obsłużenie zdarzeń zarówno przycisków jak i kółka, a także wykrycie poruszania myszką (osobne zdarzenie jest generowane gdy myszka jest przesunięta z przyciśniętym przyciskiem, a osobne, gdy zmieni pozycję i wszystkie jej przyciski są puszczone).

Obsługa naciśnięcia i puszczenia przycisków:

```
void glutMouseFunc(void (*funkcja)(int idPrzycisku, int stan, int x, int y));
```

Obsługa kółka myszki:

```
void glutMouseWheelFunc(void (*funkcja)(int kolko, int kierunek, int x, int y));
```

Obsługa ruchu z wciśniętym przyciskiem:

```
void glutMotionFunc(void (*funkcja)(int x, int y));
```

Obsługa ruchu z puszczonymi przyciskami:

```
void glutPassiveMotionFunc(void (*funkcja)(int x, int y));
```

Gdzie, w każdej z tych funkcji:

FUNKCJA: wskaźnik do funkcji o odpowiedniej liczbie i typie argumentów oraz zwracającej VOID.

IDPRZYCISKU: identyfikator przycisku. Wartości wyjaśnione w dokumentacji funkcji,

STAN: identyfikator stanu. Wartość to GLUT_UP lub GLUT_DOWN,

KOLKO: numer kółka. W przypadku normalnych myszek można zignorować,

KIERUNEK: kierunek scrollowania; -1 lub +1,

X: Pozycja x (w pikselach) kursora myszki, w momencie wystąpienia zdarzenia,

Y: Pozycja y (w pikselach) kursora myszki, w momencie wystąpienia zdarzenia,

INFORMACJE DO ZAPAMIĘTANIA

Informacje te pojawiły się w trakcie wykonywania ćwiczeń i są bardzo istotne, aby móc uczestniczyć w dalszych laboratoriach i rozumieć je.

- Wygląd i cechy macierzy jednostkowej w OpenGL,
- Rodzaje i cechy rzutowań stosowanych w bibliotece OpenGL (ortograficzne, perspektywiczne),
- Podstawowe operacje przestrzenne – czym się cechują, czy są przemienne,
- Po każdych laboratoriach należy wysłać projekt na maila, bądź skopiować na jakiś nośnik danych.