POLITECHNIKA WROCŁAWSKA

Grafika komputerowa i komunikacja człowiek-komputer

Kurs: INEK00012L

Sprawozdanie z mini projektu

OpenGL – model układu słonecznego

Wykonał:	**Janusz Pelc 252799**
Termin:	**Np. PN/TP 7:30-10:30**
Data wykonania ćwiczenia:	**17 Stycznia 2022**
Data oddania sprawozdania:	**24 Stycznia 2022**
Ocena:	

Uwagi prowadzącego:

1. **Wstęp**

Ćwiczenie polegało na stworzeniu modelu układu słonecznego za pomocą biblioteki OpenGL. Model powinien zamierać słońce i wszystkie planety, powinien się poruszać oraz być oświetlony.

2. **Model Planety**

Do narysowania planet oraz słońca użyto przekształconej funkcji rysującej jajko. Na początku przekształcono funkcję określającą współrzędne wierzchołków na:

```
x[i][k] = radius * sin(2 * M_PI * u) * cos(M_PI * v);
y[i][k] = -radius * cos(2 * M_PI * u);
z[i][k] = radius * sin(2 * M_PI * u) * sin(M_PI * v);
```

Zmienne u i v to znormalizowane zmienne i oraz k, natomiast radius to promień planety

W przypadku kuli wektory normlane wierzchołków są równe ich współrzędnym. Ponieważ źródło światła znajduje się wewnątrz słońca, jego wektory normalne są odwrócone. Aby oświetlona powierzchnia słońca była również widoczna z zewnątrz użyto funkcji:

```
glEnable(GLUT_FULLY_COVERED);
```

Dzięki niej tekstura jest widoczna po obu stronach powierzchni.

Każda planeta (w tym słońce) posiada następujące parametry:

- Promień
- Odległość od słońca
- Okres obrotu wokół słońca
- Długość dnia
- Kąt nachylenia

W przypadku Saturna, posiada on pierścień. Jest on rysowany poprzez połączenie dwóch okręgów o różnych promieniach.

Na każdą planetę nakładana jest odpowiednia tekstura. Ta sama tekstura jest nakładana na jej pierścień.

Pozycja oraz obrót planet jest ustawiany za pomocą funkcji drawPlanet() klasy Planet:

```
void drawPlanet() {
	glPushMatrix();

	glRotatef(orbit, 0, 1, 0);
	glTranslatef(distance, 0.0, 0.0);
	glRotatef(-orbit, 0, 1, 0);
	glRotatef(tilt, 0, 0, 1);
	glRotatef(rotation, 0, 1, 0);

	PlanetModel();

	if(ring) Ring(1.5 * radius, 2* radius);

	glPopMatrix();
	}
```

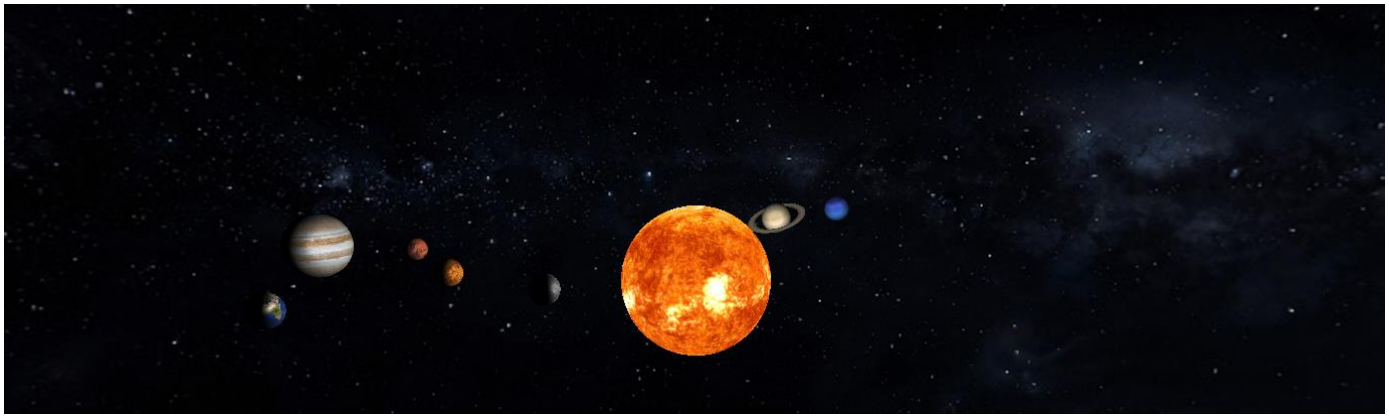
Na początku planeta jest obracana o kąt orbity oraz na nią przesunięta. Następnie jest obracana w przeciwną stronę o kąt orbity. Zapobiega to obrocie osi planety. Na końcu planeta jest pochylona o kąt nachylenia oraz obrócona o kąt rotacji wokół własnej osi.

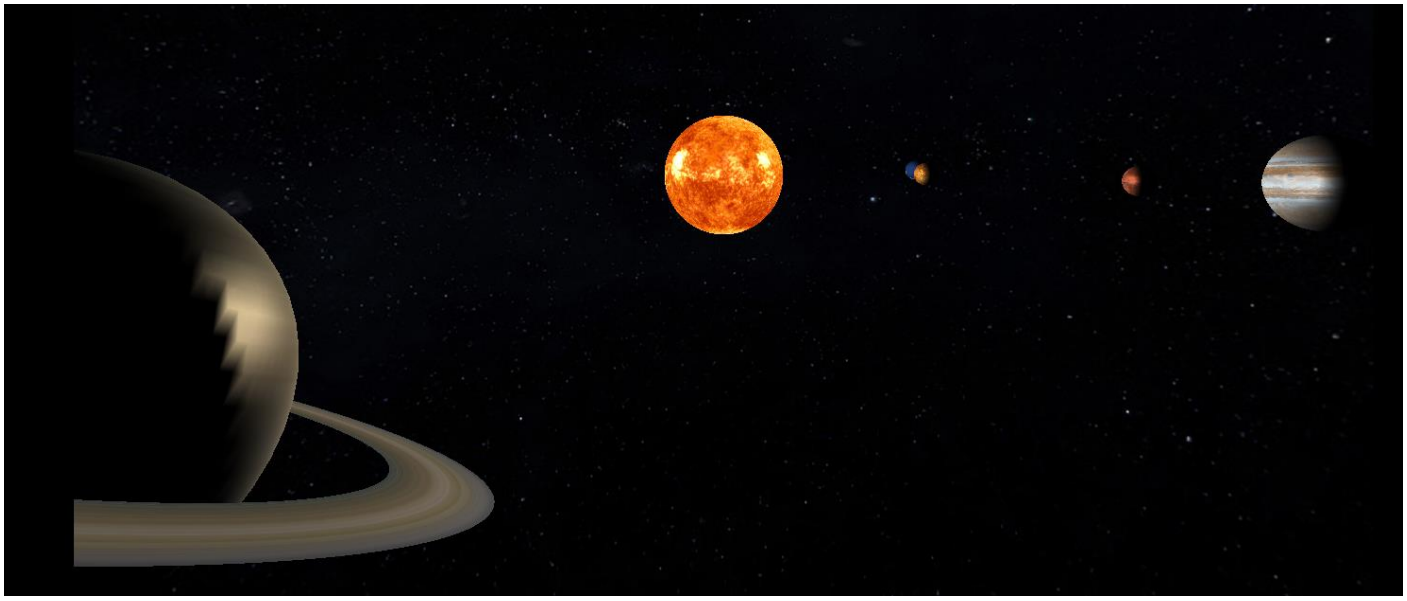
Kąty obrotu są zmieniane za pomocą funkcji spinPlanet klasy Planet oraz funkcji zwrotnej spin(). Kąty są zmieniane proporcjonalnie do prędkości globalnej Gspeed, którą użytkownik może zmienić za pomocą klawiatury.

Dodano również kontrolę kamery za pomocą myszy. Kamera może być obracana wokół słońca oraz przybliżana i oddalana.

Aby tle widoczne były gwiazdy, dodaną dodatkowy element klasy Planet, którego promień jest większy niż maksymalny promień kamery. Dzięki temu kamera zawsze znajduje się wewnątrz kuli, na której powierzchnię wewnętrzną założono teksturę gwiazd.

Ostatecznie dostajemy interaktywną symulację układu słonecznego.





3. **Źródła**
    - Tekstury - https://www.solarsystemscope.com/textures/
    - http://www.zsk.ict.pwr.wroc.pl/zsk/dyd/intinz/gk/lab/
4. **Program**

```
#define _USE_MATH_DEFINES

#include <windows.h>
#include <gl/gl.h>
#include <gl/glut.h>
#include <math.h>
#include <iostream>
```

```
using namespace std;

typedef float point3[3];

static GLfloat viewer[] = { 0.0, 0.0, 10.0 };
// inicjalizacja położenia obserwatora

static GLfloat thetax = 0.0;   // kąt obrotu obiektu
static GLfloat thetay = 0.0;   // kąt obrotu obiektu
static GLfloat thetax1 = 0.0;   // kąt obrotu obiektu
static GLfloat thetay1 = 0.0;   // kąt obrotu obiektu

static GLfloat thetax2 = 0.0;   // kąt obrotu obiektu
static GLfloat thetay2 = 0.0;   // kąt obrotu obiektu

static GLfloat pix2angle;     // przelicznik pikseli na stopnie
static GLfloat cameraz = 0.0;

static GLint status1 = 0;       // stan klawiszy myszy
static GLint status2 = 0;               // 0 - nie naciśnięto żadnego klawisza
                                                        // 1 - naciśnięty zostać lewy klawisz

static float x_pos_old = 0;       // poprzednia pozycja kursora myszy
static float y_pos_old = 0;

static float delta_x = 0;        // różnica pomiędzy pozycją bieżącą
static float delta_y = 0;                       // i poprzednią kursora myszy

static float R1 = 10;

const int N = 51; //rozmiar tablicy wierchołków NxN
static GLfloat theta[] = { 0.0, 0.0, 0.0 }; // trzy kąty obrotu

//tablice współrzędnych wierzchołków
float x[N][N];
float y[N][N];
float z[N][N];

float Nx[N][N];
float Ny[N][N];
float Nz[N][N];

float t[N][N][2];

float Gspeed=8; //prędkosć globalna


//nazwy plików tekstur
const char tearth[] = "2k_earth_daymap.tga";
const char tmercury[] = "2k_mercury.tga";
const char tvenus[] = "2k_venus_surface.tga";
const char tmars[] = "2k_mars.tga";
const char tjupiter[] = "2k_jupiter.tga";
const char tsaturn[] = "2k_saturn.tga";
const char turanus[] = "2k_uranus.tga";
const char tneptune[] = "2k_neptune.tga";
const char tsun[] = "2k_sun.tga";
const char tstars[] = "2k_stars_milky_way.tga";


/*********************************************************************************/
 // Funkcja wczytuje dane obrazu zapisanego w formacie TGA w pliku o nazwie
 // FileName, alokuje pamięć i zwraca wskaźnik (pBits) do bufora w którym
 // umieszczone są dane.
 // Ponadto udostępnia szerokość (ImWidth), wysokość (ImHeight) obrazu
 // tekstury oraz dane opisujące format obrazu według specyfikacji OpenGL
 // (ImComponents) i (ImFormat).
 // Jest to bardzo uproszczona wersja funkcji wczytującej dane z pliku TGA.
 // Działa tylko dla obrazów wykorzystujących 8, 24, or 32 bitowy kolor.
 // Nie obsługuje plików w formacie TGA kodowanych z kompresją RLE.
/*********************************************************************************/

GLbyte* LoadTGAImage(const char* FileName, GLint* ImWidth, GLint* ImHeight, GLint* ImComponents,
GLenum* ImFormat)
```

```
{
	/*****************************************************************************/
	// Struktura dla nagłówka pliku  TGA

#pragma pack(1)
	typedef struct
	{
		GLbyte	idlength;
		GLbyte	colormaptype;
		GLbyte	datatypecode;
		unsigned short	colormapstart;
		unsigned short	colormaplength;
		unsigned char	colormapdepth;
		unsigned short	x_orgin;
		unsigned short	y_orgin;
		unsigned short	width;
		unsigned short	height;
		GLbyte	bitsperpixel;
		GLbyte	descriptor;
	}TGAHEADER;
#pragma pack(8)

	FILE* pFile;
	TGAHEADER tgaHeader;
	unsigned long lImageSize;
	short sDepth;
	GLbyte* pbitsperpixel = NULL;

	/*****************************************************************************/
	// Wartości domyślne zwracane w przypadku błędu

	*ImWidth = 0;
	*ImHeight = 0;
	*ImFormat = GL_BGR_EXT;
	*ImComponents = GL_RGB8;


	errno_t err = fopen_s(&pFile, FileName, "rb");
	if (pFile == NULL)
		return NULL;

	/*****************************************************************************/
	// Przeczytanie nagłówka pliku

	fread(&tgaHeader, sizeof(TGAHEADER), 1, pFile);

	/*****************************************************************************/

	// Odczytanie szerokości, wysokości i głębi obrazu

	*ImWidth = tgaHeader.width;
	*ImHeight = tgaHeader.height;
	sDepth = tgaHeader.bitsperpixel / 8;

	/*****************************************************************************/
	// Sprawdzenie, czy głębia spełnia założone warunki (8, 24, lub 32 bity)

	if (tgaHeader.bitsperpixel != 8 && tgaHeader.bitsperpixel != 24 && tgaHeader.bitsperpixel !=
32)
		return NULL;
	/*****************************************************************************/

	// Obliczenie rozmiaru bufora w pamięci
	lImageSize = tgaHeader.width * tgaHeader.height * sDepth;

	/*****************************************************************************/
	// Alokacja pamięci dla danych obrazu

	pbitsperpixel = (GLbyte*)malloc(lImageSize * sizeof(GLbyte));

	if (pbitsperpixel == NULL)
		return NULL;

	if (fread(pbitsperpixel, lImageSize, 1, pFile) != 1)
```

```
	{
		free(pbitsperpixel);
		return NULL;
	}

	/*****************************************************************************/
	// Ustawienie formatu OpenGL
	switch (sDepth)
	{
	case 3:
		*ImFormat = GL_BGR_EXT;
		*ImComponents = GL_RGB8;
		break;
	case 4:
		*ImFormat = GL_BGRA_EXT;
		*ImComponents = GL_RGBA8;
		break;
	case 1:
		*ImFormat = GL_LUMINANCE;
		*ImComponents = GL_LUMINANCE8;
		break;
	};
	fclose(pFile);
	return pbitsperpixel;
}

class Planet {
public:
	float x[N][N];
	float y[N][N];
	float z[N][N];

	float Nx[N][N];
	float Ny[N][N];
	float Nz[N][N];

	float radius; //promień
	float distance; //odległość od słońca
	float orbit = rand() % 361;  //kąt orbity
	float speed; //prędkość oboru wokół słońca
	float day; //dłuość dnia
	float rotation=0; //kąt obrotu wokół własnej osi
	float tilt; //kąt nachylenia
	bool type; //czy gwiazda
	bool ring; //czy pierścień
	const char* file; //plik tekstury

	GLbyte* pBytes;
	GLint ImWidth, ImHeight, ImComponents;
	GLenum ImFormat;


	void Ring(float R1, float R2) { //funkcja rysuje pierścień wokół planety
		float kat, kat1;
		for (int i = 0; i < 359; i++) {
			kat = i * M_PI / 180;
			kat1 = (i + 1) * M_PI / 180;

			glBegin(GL_POLYGON);  //funkcja rysuje wielokąt
			glNormal3f(0, 1, 0);
			glTexCoord2f(i / 360, 0);
			glVertex3f(R1 * cos(kat), 0, R1 * sin(kat));

			glTexCoord2f((i + 1) / 360, 0);
			glVertex3f(R1 * cos(kat1), 0, R1 * sin(kat1));

			glTexCoord2f((i + 1) / 360, 1);
			glVertex3f(R2 * cos(kat1), 0, R2 * sin(kat1));

			glTexCoord2f(i / 360, 1);
			glVertex3f(R2 * cos(kat), 0, R2 * sin(kat));

			glEnd();
		}
```

```
		kat = 359 * M_PI / 180;
		kat1 = (0) * M_PI / 180;

		glBegin(GL_POLYGON);  //funkcja rysuje wielokąt
		glTexCoord2f(359 / 360, 0);
		glVertex3f(R1 * cos(kat), 0, R1 * sin(kat));

		glTexCoord2f((0) / 360, 0);
		glVertex3f(R1 * cos(kat1), 0, R1 * sin(kat1));

		glTexCoord2f((0) / 360, 1);
		glVertex3f(R2 * cos(kat1), 0, R2 * sin(kat1));

		glTexCoord2f(359 / 360, 1);
		glVertex3f(R2 * cos(kat), 0, R2 * sin(kat));

		glEnd();
	}


	void PlanetModel() {
		// Zdefiniowanie tekstury 2-D
		glTexImage2D(GL_TEXTURE_2D, 0, ImComponents, ImWidth, ImHeight, 0, ImFormat,
GL_UNSIGNED_BYTE, pBytes);

		for (int i = 0; i < N-1 ; i++) {
			for (int k = 0; k < N-1 ; k++) {

				glBegin(GL_POLYGON);  //funkcja rysuje wielokąt
				glNormal3f(Nx[i][k], Ny[i][k], Nz[i][k]);
				glTexCoord2f(t[i][k][0], t[i][k][1]);
				glVertex3f(x[i][k], y[i][k], z[i][k]);

				glNormal3f(Nx[i + 1][k], Ny[i + 1][k], Nz[i + 1][k]);
				glTexCoord2f(t[i + 1][k][0], t[i + 1][k][1]);
				glVertex3f(x[i + 1][k], y[i + 1][k], z[i + 1][k]);

				glNormal3f(Nx[i][k + 1], Ny[i][k + 1], Nz[i][k + 1]);
				glTexCoord2f(t[i][k + 1][0], t[i][k + 1][1]);
				glVertex3f(x[i][k + 1], y[i][k + 1], z[i][k + 1]);
				glEnd();

				glBegin(GL_POLYGON);
				glNormal3f(Nx[i + 1][k + 1], Ny[i + 1][k + 1], Nz[i + 1][k + 1]);
				glTexCoord2f(t[i + 1][k + 1][0], t[i + 1][k + 1][1]);
				glVertex3f(x[i + 1][k + 1], y[i + 1][k + 1], z[i + 1][k + 1]);

				glNormal3f(Nx[i + 1][k], Ny[i + 1][k], Nz[i + 1][k]);
				glTexCoord2f(t[i + 1][k][0], t[i + 1][k][1]);
				glVertex3f(x[i + 1][k], y[i + 1][k], z[i + 1][k]);

				glNormal3f(Nx[i][k + 1], Ny[i][k + 1], Nz[i][k + 1]);
				glTexCoord2f(t[i][k + 1][0], t[i][k + 1][1]);
				glVertex3f(x[i][k + 1], y[i][k + 1], z[i][k + 1]);
				glEnd();

			}
		}
	}

	void initPlanet( ) {
		float u = 0, v = 0; //zmienne u i v wykorzystywane w funkcjach określających
wierzchołki;
		float fN = N; //wartość stałej N jako float
		float l = 0;
		float xu;
		float yu;
		float zu;

		float xv;
		float yv;
		float zv;

		//  Przeczytanie obrazu tekstury z pliku
```

```
		pBytes = LoadTGAImage(file, &ImWidth, &ImHeight, &ImComponents, &ImFormat);

		for (int i = 0; i < N; i++) {
			u = float(i / (fN -1)); //zmienna zmniejszana proporcjonalnie do stałej N, aby znajadowała się w zakresie użytej funkcji [0,1]
			for (int k = 0; k < N; k++) {
				v = float(k / (fN -1)); //zmienna jest zmniejszana proporcjonalnie do stałej N, aby znajadowała się w zakresie użytej funkcji [0,1]
				x[i][k] = radius * sin(2 * M_PI * u) * cos(M_PI * v);
				y[i][k] = -radius * cos(2 * M_PI * u);
				z[i][k] = radius * sin(2 * M_PI * u) * sin(M_PI * v);

				Nx[i][k] = x[i][k];
				Ny[i][k] = y[i][k];
				Nz[i][k] = z[i][k];

				l = sqrt(Nx[i][k] * Nx[i][k] + Ny[i][k] * Ny[i][k] + Nz[i][k] * Nz[i][k]);
				Nx[i][k] /= l;
				Ny[i][k] /= l;
				Nz[i][k] /= l;

				if (type) {
					Nx[i][k] *= -1;
					Ny[i][k] *= -1;
					Nz[i][k] *= -1;
				}
				if (x[i][k] == 0 && z[i][k] == 0) {
					Nx[i][k] = 0;
					if (y[i][k] == -5)Ny[i][k] = -1;
					else Ny[i][k] = 1;
					Nz[i][k] = 0;
				}

			}
			if(type )Ny[0][i] = 1;
			else Ny[0][i] = -1;
		}
		float fi, fk;
		int i;
		for (int k = 0; k < N; k++) {
			fk = (k);
			fk /= 2;
			for (i = 0; i < N/2+1; i++) {
				fi = (i);
				fi *= 2;

				t[i][k][0] = ((fN-1)/2-fk) / (fN-1);
				t[i][k][1] = (fN-1-fi) / (fN-1);
			};

			for (; i < N; i++) {
				fi = (i) -(fN-1)/2;
				fi *= 2;

				t[i][k][0] = (fN-1-fk) / (fN-1);
				t[i][k][1] = (fi) / (fN-1);
			}
		}
	}

	Planet(float _radius,float _distance, float _speed, float _day, bool _type, const char*  _file, float _tilt, bool _ring) {

		radius = _radius;
		distance = _distance;
		if (_speed != 0) speed = 1 / _speed;
		else speed = 0;
		day = 1/_day;
		type = _type;
		file = _file;
		tilt = _tilt;
		ring = _ring;
```

```
                initPlanet();
        }

Planet() {}
void drawPlanet() {
                glPushMatrix();

                glRotatef(orbit, 0, 1, 0);
                glTranslatef(distance, 0.0, 0.0);
                glRotatef(-orbit, 0, 1, 0);
                glRotatef(tilt, 0, 0, 1);
                glRotatef(rotation, 0, 1, 0);

                PlanetModel();

                if(ring) Ring(1.5 * radius, 2* radius); //rysowanie promieni wokół planety

                glPopMatrix();
        }
        void spinPlanet() {  //obrót planety wokół słońca i własnej osi

                orbit -= speed*Gspeed;
                rotation -= day * Gspeed;
                if (orbit <0 ) orbit += 360.0;
                if (rotation <0 ) rotation += 360.0;


        }

};
Planet stars;
Planet sun;
Planet mercury;
Planet venus;
Planet earth;
Planet mars;
Planet jupiter;
Planet saturn;
Planet uranus;
Planet neptune;


void spin() {  //funkcja obraca wszystkie planety
        sun.spinPlanet();
        mercury.spinPlanet();
        venus.spinPlanet();
        earth.spinPlanet();
        mars.spinPlanet();
        jupiter.spinPlanet();
        saturn.spinPlanet();
        uranus.spinPlanet();
        neptune.spinPlanet();
        //odświeżenie zawartości aktualnego okna
        glutPostRedisplay();

}


GLfloat light_position[] = { 0.0, 0.0, 0.0, 1.0 };
// położenie źródła światła
/*************************************************************************************/

/*************************************************************************************/


/*************************************************************************************/
// 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
                y_pos_old = y;                                  // jako pozycji poprzedniej
                status1 = 1;          // wcięnięty został lewy klawisz myszy
```

```
        }

        else if (btn == GLUT_RIGHT_BUTTON && state == GLUT_DOWN) {

                x_pos_old = x;         // przypisanie aktualnie odczytanej pozycji kursora
                y_pos_old = y;                                  // jako pozycji poprzedniej
                status2 = 1;          // wcięnięty został lewy klawisz myszy
        }

        else    status1 = status2 = 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

        delta_y = y - y_pos_old;     // obliczenie różnicy położenia kursora myszy

        y_pos_old = y;            // podstawienie bieżącego położenia jako poprzednie

        glutPostRedisplay();     // przerysowanie obrazu sceny
}


// 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
        // Narysowanie osi przy pomocy funkcji zdefiniowanej powyżej

        if (status1 == 1) {                                             // jeśli lewy klawisz myszy
wcięnięty
                thetax += delta_x * pix2angle / 20;     // modyfikacja kąta obrotu o kat
proporcjonalny do różnicy położeń kursora myszy
                thetay += delta_y * pix2angle / 20;
                if (thetay > M_PI / 2 - 0.000001) {
                        thetay = M_PI / 2 - 0.000001;
                }
                else if (thetay < -M_PI / 2 + 0.000001) {
                        thetay = -M_PI / 2 + 0.000001;
                }
        }
        else if (status2 == 1) {
                R1 += delta_y / 10;
                if (R1 < sun.radius+1) R1 = sun.radius+1;
                else if (R1 > 100) R1 = 100;
        }

        viewer[0] = R1 * cos(thetax) * cos(thetay);
        viewer[1] = R1 * sin(thetay);
        viewer[2] = R1 * sin(thetax) * cos(thetay);

        glLightfv(GL_LIGHT0, GL_POSITION, light_position);

        glColor3f(1.0f, 1.0f, 1.0f);

        stars.drawPlanet();
```

```
        sun.drawPlanet();
        mercury.drawPlanet();
        venus.drawPlanet();
        earth.drawPlanet();
        mars.drawPlanet();
        jupiter.drawPlanet();
        saturn.drawPlanet();
        uranus.drawPlanet();
        neptune.drawPlanet();


        glFlush();
        // Przekazanie poleceń rysujących do wykonania
        glutSwapBuffers();
}
/*************************************************************************************/

// Funkcja ustalająca stan renderowania

void MyInit(void)
{
        /*********************************************************************************/
        // Definicja materiału z jakiego zrobiony jest czajnik

        GLfloat mat_ambient[] = { 0, 0, 0, 1.0 };
        // współczynniki ka =[kar,kag,kab] dla światła otoczenia

        GLfloat mat_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
        // współczynniki kd =[kdr,kdg,kdb] światła rozproszonego

        GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
        // współczynniki ks =[ksr,ksg,ksb] dla światła odbitego

        GLfloat mat_shininess = { 20.0 };
        // współczynnik n opisujący połysk powierzchni


/*************************************************************************************/
// Definicja źródła światła
GLfloat light_position[] = { 0.0, 0.0, 0.0, 0.0 };
        // położenie źródła


        GLfloat light_ambient[] = { 0, 0, 0, 1.0 };
        // składowe intensywności świecenia źródła światła otoczenia
        // Ia = [Iar,Iag,Iab]

        GLfloat light_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
        // składowe intensywności świecenia źródła światła powodującego
        // odbicie dyfuzyjne Id = [Idr,Idg,Idb]

        GLfloat light_specular[] = { 1.0, 1.0, 1.0, 1.0 };
        // składowe intensywności świecenia źródła światła powodującego
        // odbicie kierunkowe Is = [Isr,Isg,Isb]

        GLfloat att_constant = { 1.0 };
        // składowa stała ds dla modelu zmian oświetlenia w funkcji
        // odległości od źródła

        GLfloat att_linear = { 0.05 };
        // składowa liniowa dl dla modelu zmian oświetlenia w funkcji
        // odległości od źródła

        GLfloat att_quadratic = { 0.001 };
        // składowa kwadratowa dq dla modelu zmian oświetlenia w funkcji
        // odległości od źródła

/*************************************************************************************/
// Ustawienie parametrów materiału i źródła światła

/*************************************************************************************/
// Ustawienie patrametrów materiału
        glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
        glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
```

```
        glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
        glMaterialf(GL_FRONT, GL_SHININESS, mat_shininess);

        /*************************************************************************************/
        // Ustawienie parametrów źródła

        glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
        glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
        glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
        glLightfv(GL_LIGHT0, GL_POSITION, light_position);

        glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, att_constant);


        /*************************************************************************************/
// Definicja źródła światła
/*************************************************************************************/
        // Ustawienie opcji systemu oświetlania sceny

        glShadeModel(GL_SMOOTH); // właczenie łagodnego cieniowania
        glEnable(GL_LIGHTING);   // właczenie systemu oświetlenia sceny
        glEnable(GL_LIGHT0);     // włączenie źródła o numerze 0

        glEnable(GL_DEPTH_TEST); // włączenie mechanizmu z-bufora

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

        glEnable(GLUT_FULLY_COVERED);

        //Okreslenie parametrów planet
        stars = Planet(200, 0, 0, 0, true, tstars, 0, false);
        sun = Planet(5, 0, 0, 25, true, tsun, 7.25, false);
        mercury = Planet(1, 10,58.5, 88, false, tmercury, 0.03, false);
        venus = Planet(1, 18, 224,243, false, tvenus, 2.64, false);
        earth = Planet(1, 25, 365,1, false, tearth, 23.44, false);
        mars = Planet(1, 30, 686,1,false, tmars, 25.19, false);
        jupiter = Planet(3, 40, 4333,0.41, false, tjupiter, 3.13, false);
        saturn = Planet(2, 50, 10765,0.45, false, tsaturn, 26.73, true);
        uranus = Planet(2, 60, 30707,0.7, false, turanus, 82.23, false);
        neptune = Planet(2, 70, 60223,0.66, false, tneptune, 28.32, false);
/*************************************************************************************/
        // Włączenie mechanizmu teksturowania
        glEnable(GL_TEXTURE_2D);

        /*************************************************************************************/
        // Ustalenie trybu teksturowania
        glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);

        /*************************************************************************************/
        // Określenie sposobu nakładania tekstur
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);


}
/*************************************************************************************/
// 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)
{
        pix2angle = 360.0 / (float)horizontal;  // przeliczenie pikseli na stopnie

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

        glLoadIdentity();
        // Czyszcznie macierzy bieżącej

        gluPerspective(90, 1.0, 1.0, 400.0);
```

```
        // 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

}

//fynkcja zwortna wyznaczają prędkość
void keys(unsigned char key, int x, int y)
{
        if (key == ',' ) Gspeed = Gspeed/2;
        if (Gspeed < 1) Gspeed = 1;
        if (key == '.' ) Gspeed =Gspeed * 2;
        RenderScene(); // przerysowanie obrazu sceny
}

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

void main(void)
{
        cout << "Lewy przycisk myszy - obrot kamery" << endl
                << "Prawy przycisk myszy - przyblizenie kamery" << endl;
        cout << "Przyciski: " << endl
                << ", - zwolnij" << endl
                << ". - przyspiesz" << endl;

        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);

        glutInitWindowSize(1000, 1000);

        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
        glutKeyboardFunc(keys);
        glutMouseFunc(Mouse);
        // Ustala funkcję zwrotną odpowiedzialną za badanie ruchu myszy
        glutMotionFunc(Motion);

        glutIdleFunc(spin);

        glutMainLoop();
        // Funkcja uruchamia szkielet biblioteki GLUT
}

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