POLITECHNIKA WROCŁAWSKA

Instytut Informatyki, Automatyki i Robotyki

Zakład Systemów Komputerowych

Grafika komputerowa i komunikacja człowiek – komputer

Kurs: INEK00012L

Sprawozdanie z ćwiczenia nr 4

„OpenGL – interakcja z użytkownikiem”

Wykonał:	**Adrian Frydmański**
Termin:	**WT/P 12:00-15:00**
Data wykonania ćwiczenia:	**10 XI 2015**
Data oddania sprawozdania:	**24 XI 2015**
Ocena:	

Uwagi prowadzącego:

WSTĘP TEORETYCZNY

Celem ćwiczenia było pokazanie, jak przy pomocy OpenGL można sterować ruchem obiektu i położeniem obserwatora w przestrzeni trójwymiarowej.

KOD ŹRÓDŁOWY

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

/*************************************************************************************/
// zmienne globalne
typedef float point3[3];
static GLfloat viewer[] = { 0.0, 0.0, 10.0 };          // inicjalizacja położenia obserwatora
static GLfloat focus[] = { 0.0, 0.0, 0.0 };            // inicjalizacja punktu, na który patrzy obserwator
static GLfloat cam[] = { 0.0, 1.0, 0.0 };              // inicjalizacja pozycji kamery

const float pi = 3.14159265;                           // pi
point3 **pTab;                                         // tablica na punkty
point3 **pRGB;                                         // tablica na kolory
int n = 50;
                                                       // do posziału kwadratu jednostkowego
int model = 3;                                         // 1-punkty, 2-siatka, 3-kolorowe trojkaty
float squareLen = 1.0;                                 // długość boku kwadratu jednostkowego
float len = 2;                                         // długość osi
point3 v = { 0.05, 0.05, 0.05 };                       // szybkość obracania się

static GLfloat theta_x = 0.0;                          // kąt obrotu obiektu
static GLfloat theta_y = 0.0;                          // kąt obrotu obiektu
static GLfloat pix2angle_x;                            // przelicznik pikseli na stopnie
static GLfloat pix2angle_y;                            // przelicznik pikseli na stopnie

static GLint status = 0;                               // stan klawiszy myszy (o - brak, 1 - lewy)

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

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

/*************************************************************************************/
// Funkcja rysująca osie układu współrzędnych
void Axes(float len)
{
        // początek i koniec obrazu osi x
        point3  x_min = { -len, 0.0, 0.0 };
        point3  x_max = { len, 0.0, 0.0 };

        // początek i koniec obrazu osi y
        point3  y_min = { 0.0, -len, 0.0 };
        point3  y_max = { 0.0,  len, 0.0 };

        // początek i koniec obrazu osi z
        point3  z_min = { 0.0, 0.0, -len };
        point3  z_max = { 0.0, 0.0,  len };

        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();
}
```

```
/*************************************************************************************/
// Funkcje wyliczaj?ce wspó?rz?dne punktu (u,v) w przestrzeni 3D
float transformTo3D_x(float u, float v)
{
	float x, a = v * pi;
	x = (-90 * pow(u, 5) + 225 * pow(u, 4) - 270 * pow(u, 3) + 180 * pow(u, 2) - 45 * u) * cos(a);
	return x;
}

float transformTo3D_y(float u, float v)
{
	float y;
	y = 160 * pow(u, 4) - 320 * pow(u, 3) + 160 * pow(u, 2);
	return y - 5;                                                                    //
obniżenie jajka, żeby si? ładnie mieściło
}

float transformTo3D_z(float u, float v)
{
	float z, a = v * pi;
	z = (-90 * pow(u, 5) + 225 * pow(u, 4) - 270 * pow(u, 3) + 180 * pow(u, 2) - 45 * u) * sin(a);
	return z;
}

/*************************************************************************************/
// Generowanie siatki punktów:
void eggGenerate()
{
	float step = squareLen / n;
	// współrz?dne 2D -> 3D
	float u, v;
	for (int i = 0; i<n + 1; i++)
		for (int j = 0; j<n + 1; j++)
		{
			u = j * step;
			v = i * step;
			pTab[i][j][0] = transformTo3D_x(u, v);
			pTab[i][j][1] = transformTo3D_y(u, v);
			pTab[i][j][2] = transformTo3D_z(u, v);
		}
}

/*************************************************************************************/
// Renderowanie jajka
void Egg()
{
	// Generowanie siatki
	eggGenerate();
	// Ustawienie koloru białego
	glColor3f(1.0, 1.0, 1.0);
	// switch w zale?ności od modelu
	switch (model)
	{
	case 1:
	// punkty
	{
		glBegin(GL_POINTS);
		for (int i = 0; i<n; i++)
			for (int j = 0; j<n; j++)
				glVertex3fv(pTab[i][j]);
		glEnd();
	}
	break;
	case 2:
	// siatka
	{
		for (int i = 0; i < n; i++)
			for (int j = 0; j < n; j++)
			{
				glBegin(GL_LINES);
				// pionowo
				glVertex3fv(pTab[i][j]);
				glVertex3fv(pTab[i][j + 1]);
				// poziomo
				glVertex3fv(pTab[i][j]);
				glVertex3fv(pTab[i + 1][j]);
				// w prawo w dół
				glVertex3fv(pTab[i][j]);
				glVertex3fv(pTab[i + 1][j + 1]);
				glEnd();
			}
	}
	break;
```

```
	case 3:
	// trójk?ty
	{
		for (int i = 0; i < n; i++)
			for (int j = 0; j < n; j++)
			{
				// w jedn? stron?
				glBegin(GL_TRIANGLES);
				glColor3fv(pRGB[i][j + 1]);
				glVertex3fv(pTab[i][j + 1]);
				glColor3fv(pRGB[i + 1][j]);
				glVertex3fv(pTab[i + 1][j]);
				glColor3fv(pRGB[i + 1][j + 1]);
				glVertex3fv(pTab[i + 1][j + 1]);
				// w drug? stron?
				glColor3fv(pRGB[i][j]);
				glVertex3fv(pTab[i][j]);
				glColor3fv(pRGB[i + 1][j]);
				glVertex3fv(pTab[i + 1][j]);
				glColor3fv(pRGB[i][j + 1]);
				glVertex3fv(pTab[i][j + 1]);
				glEnd();
			}
	}
	break;
	}
}

void allocate()
{
	//Dynamiczna alokacja tablicy punktow
	pTab = new point3*[n + 1];
	for (int i = 0; i < n + 1; i++)
		pTab[i] = new point3[n + 1];
	//Dynamiczna alokacja tablicy i wygenerowanie kolorow losowych dla punktow
	pRGB = new point3*[n + 1];
	for (int i = 0; i < n + 1; i++)
		pRGB[i] = new point3[n + 1];
	for (int i = 0; i < n + 1; i++)
		for (int j = 0; j < n + 1; j++)
		{
			pRGB[i][j][0] = ((float)(rand() % 10) + 1) / 10;
			pRGB[i][j][1] = ((float)(rand() % 10) + 1) / 10;
			pRGB[i][j][2] = ((float)(rand() % 10) + 1) / 10;
		}
}
void relase()
{
	//Zwolnienie pami?ci
	for (int i = 0; i < n + 1; i++)
	{
		delete[] pTab[i];
		delete[] pRGB[i];
		pTab[i] = 0;
		pRGB[i] = 0;
	}
	delete[] pTab;
	delete[] pRGB;
	pTab = 0;
	pRGB = 0;
}

/*************************************************************************************/
// 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 bieżącej
	gluLookAt(viewer[0], viewer[1], viewer[2], focus[0], focus[1], focus[2], cam[0], cam[1], cam[3]);
	// Zdefiniowanie położenia obserwatora
	if (status == 1)                                    // jeśli lewy klawisz myszy wcięnięty
	{
		theta_x += delta_x*pix2angle_x;                 // modyfikacja kąta obrotu o kat proporcjonalny
		theta_y += delta_y*pix2angle_y;                 do różnicy położeń kursora myszy
	}
	else if (status == 2)                       // jeśli prawy klawisz myszy wciśnięty
	{
		viewer[2] += delta_y;                   // modyfikacja polozenia Z obserwatora (zoom)
		if (viewer[2] <= 7)                     // ograniczenie zblizenia
			viewer[2] = 7;
		if (viewer[2] >= 200)                   // ograniczenie oddalenia
			viewer[2] = 200;
	}
```

```
	glRotatef(theta_x, 0.0, 1.0, 0.0);		//obrót obiektu o nowy kąt
	glRotatef(theta_y, 1.0, 0.0, 0.0);		//obrót obiektu o nowy kąt
	Axes(2);					// Narysowanie osi przy pomocy funkcji zdefiniowanej powyżej
	Egg();						// Narysowanie jajka
	glFlush();					// Przekazanie poleceń rysujących do wykonania
	glutSwapBuffers();
}

/*************************************************************************************/
// Funkcja zwrotna dla klawiszy
void Keys(unsigned char key, int x, int y)
{
	// zmiana modelu wy?wietlania
	if (key == 'p' || key == '1') model = 1;
	if (key == 's' || key == '2') model = 2;
	if (key == 't' || key == '3') model = 3;
	// zmiana "rozdzielczo?ci" jajka
	if (key == '+' || key == '-' || key == 72 || key == 80)
	{
		relase();
		if (key == '+' || key == 72) n += 5;
		if ((key == '-' || key == 80) && n > 0) n -= 5;
		allocate();
	}
	RenderScene();
}

/*************************************************************************************/
// Funkcja ustalająca stan renderowania
void MyInit(void)
{
	glClearColor(0.0f, 0.0f, 0.0f, 1.0f);		// Kolor czyszcący (wypełnienia okna) ustawiono na czarny
}

/*************************************************************************************/
// Funkcja ma za zadanie utrzymanie stałych proporcji rysowanych obiektów w przypadku zmiany rozmiarów okna.
// Parametry vertical i horizontal są przekazywane do funkcji za każdym razem gdy zmieni się rozmiar okna.
void ChangeSize(GLsizei horizontal, GLsizei vertical)
{
	pix2angle_x = 360.0 / (float)horizontal;	// przeliczenie pikseli na stopnie
	pix2angle_y = 360.0 / (float)vertical;		// przeliczenie pikseli na stopnie

	// Deklaracja zmiennej AspectRatio  określającej proporcję wymiarów okna
	GLfloat AspectRatio;

	if (vertical == 0)				// Zabezpieczenie przed dzieleniem przez 0
		vertical = 1;

	// Ustawienie wielkościokna okna widoku (viewport)
	// W tym przypadku od (0,0) do (horizontal, vertical)
	glViewport(0, 0, horizontal, vertical);

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

	glLoadIdentity();				// Czyszcznie macierzy bieżącej

	// Wyznaczenie współczynnika  proporcji okna
	// Gdy okno nie jest kwadratem wymagane jest określenie tak zwanej
	// przestrzeni ograniczającej pozwalającej zachować właściwe
	// proporcje rysowanego obiektu.
	// Do okreslenia przestrzeni ograniczającej służy funkcja
	// glOrtho(...)
	AspectRatio = (GLfloat)horizontal / (GLfloat)vertical;

	gluPerspective(80, AspectRatio, 1.0, 200.0);	// Ustawienie parametrów dla rzutu perspektywicznego
							// (fovy, aspect, zNear, zFar)
							// (kąt, stosunek szerokości do wysokości, odległość od okna,
odległość od rzutni)
	// Ustawienie wielkości okna okna widoku (viewport) w zależności
	// relacji pomiędzy wysokością i szerokością okna
	glViewport(0, 0, horizontal, vertical);

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

/*************************************************************************************/
// 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)
	{
```

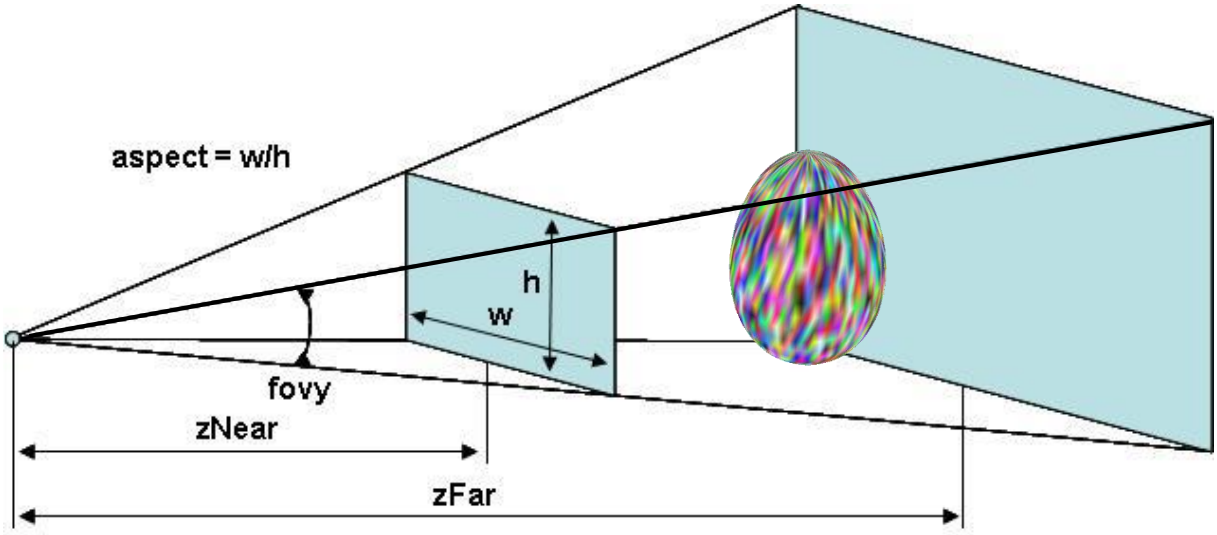
```
                x_pos_old = x;
                        // przypisanie aktualnie odczytanej pozycji kursora jako pozycji poprzedniej
                y_pos_old = y;
                status = 1;
                                // wcięnięty został lewy klawisz myszy
        }
        else if (btn == GLUT_RIGHT_BUTTON && state == GLUT_DOWN) {
                y_pos_old = y;
                        // przypisanie aktualnie odczytanej pozycji kursora jako pozycji poprzedniej
                status = 2;
                                //wciśnięty został prawy 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
        delta_y = y - y_pos_old;
        // obliczenie różnicy położenia kursora myszy
        y_pos_old = y;
                        // podstawienie bieżącego położenia jako poprzednie
        glutPostRedisplay();
                // przerysowanie obrazu sceny
}

/*************************************************************************************/
// Główny punkt wejścia programu. Program działa w trybie konsoli
void main(void)
{
        // zaalokuj tablice
        allocate();
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
        glutInitWindowSize(1000, 600);
        glutCreateWindow("Rzutowanie perspektywiczne - 209865");
        glutMouseFunc(Mouse);               // Ustala funkcję zwrotną odpowiedzialną za badanie stanu myszy
        glutMotionFunc(Motion);             // Ustala funkcję zwrotną odpowiedzialną za badanie ruchu myszy
        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);             // Funkcja zwrotna dla klawiatury
        glutMainLoop();                     // Funkcja uruchamia szkielet biblioteki GLUT
        relase();                           // zwolnij pamiec

}
```

OPIS DZIAŁANIA

Program badając różnicę położeń myszy z wciśniętym lewym przyciskiem obraca obiektem względem osi X (ruch w pionie) i Y (ruch w poziomie), wyliczając uprzednio kąt, o jaki obrócić obiekt. Dodatkowo przytrzymując prawy przycisk myszy i poruszając kursor w pionie, można zbliżyć się do jajka lub od niego oddalić. Nadal można zmieniać model (punkty, siatka, trójkąty) i jego rozdzielczość przyciskami „1” – „3” (i „p”, „s”, „t”) oraz „+” i „-”.

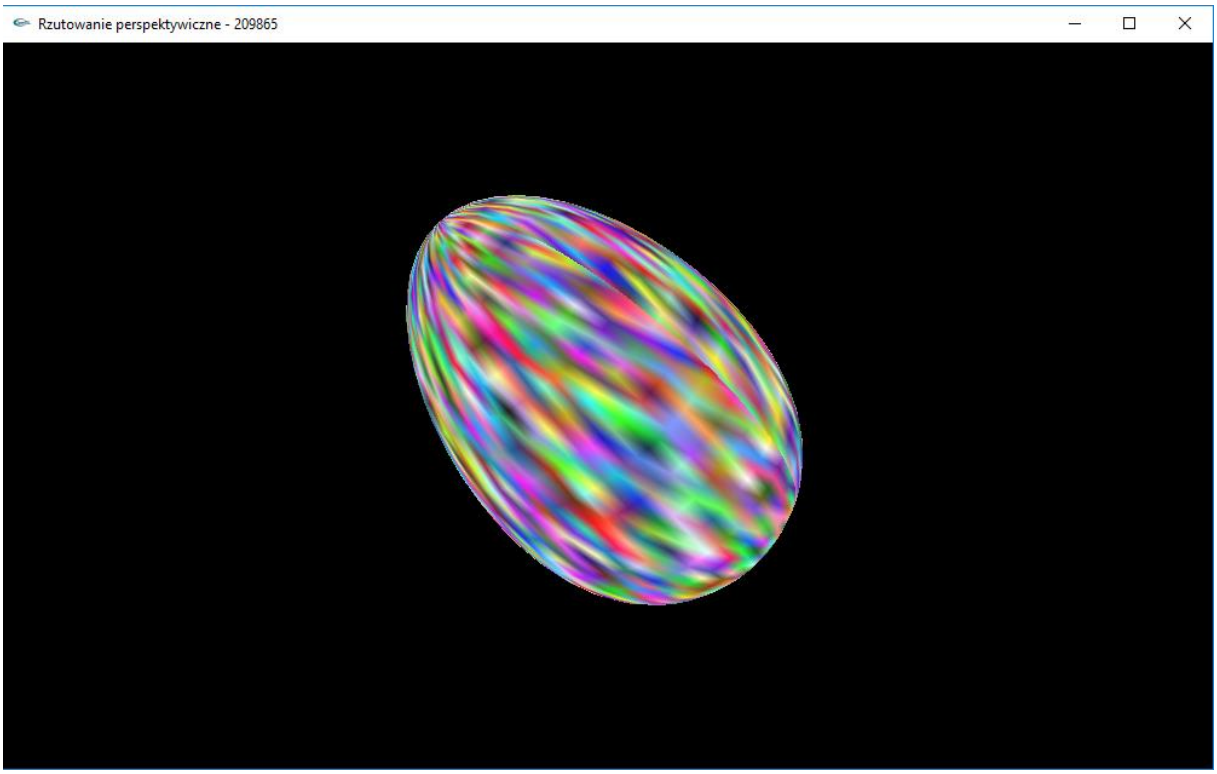
W funkcji ChangeSize wywołano funkcję glViewPort z podanymi w parametrach wartościami początkowymi i końcowymi wierzchołków okna, w którym wyświetla się jajko. Podając 0, 0, szerokość okna (windowsowego) i jego wysokość uzyskujemy okno przez które patrzymy wielkości okna windowsowego, co niweluje czarne ramki po bokach.

Sposób patrzenia na jajko w rzucie perspektywicznym najlepiej obrazuje poniższy rysunek:



Rys. 1 Rzut perspektywiczny

W przypadku obracania jajka względem osi, po obrocie o 90° względem Y, a następnie względem X nasze jajko „kładzie” się do poziomu, co widać poniżej:



Rys. 2 „Kładące się” jajko

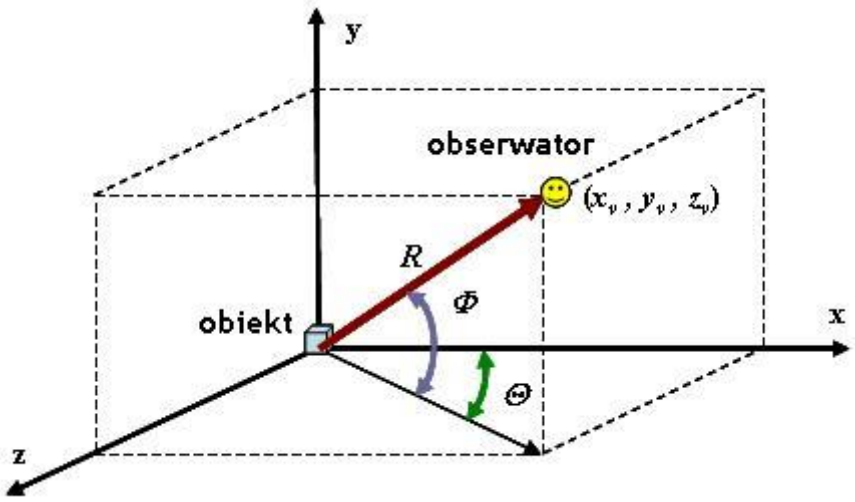
ZMODYFIKOWANE FRAGMENTY

```
/*************************************************************************************/
// zmienne globalne
typedef float point3[3];
static GLfloat R = 10;                                  // promień
static GLfloat viewer[] = { 0.0, 0.0, 10.0 };           // inicjalizacja położenia obserwatora
static GLfloat focus[] = { 0.0, 0.0, 0.0 };             // inicjalizacja punktu, na który patrzy obserwator
static GLfloat cam[] = { 0.0, 1.0, 0.0 };               // inicjalizacja pozycji kamery

const float pi = 3.14159265;                            // pi
point3 **pTab;                                          // tablica na punkty
point3 **pRGB;                                          // tablica na kolory
```

```
int n = 50;                                             // do posziału kwadratu jednostkowego
int model = 3;                                          // 1-punkty, 2-siatka, 3-kolorowe trojkaty
float squareLen = 1.0;                                  // długość boku kwadratu jednostkowego
float len = 2;                                          // długość osi
point3 v = { 0.05, 0.05, 0.05 };                        // szybkość obracania się

static GLfloat theta = 0.0;                             // kąt obrotu obiektu
static GLfloat phi = 0.0;                               // kąt obrotu obiektu
static GLfloat pix2angle_x;                             // przelicznik pikseli na stopnie
static GLfloat pix2angle_y;                             // przelicznik pikseli na stopnie

static GLint status = 0;                                // stan klawiszy myszy (o - brak, 1 - lewy)

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

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


/*************************************************************************************/
// 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 bieżącej

        gluLookAt(viewer[0], viewer[1], viewer[2], focus[0], focus[1], focus[2], cam[0], cam[1], cam[3]);
        // Zdefiniowanie położenia obserwatora

        if (status == 1)                                        // jeśli lewy klawisz myszy wcięnięty
        {
                theta += delta_x*pix2angle_x*pi / 180;          // modyfikacja kąta obrotu o kat proporcjonalny
                phi += delta_y*pix2angle_y*pi / 180;            // do różnicy położeń kursora myszy
                if (phi < -pi / 2)
                {
                        phi = -pi / 2;
                        theta *= -1;
                }
                else if (phi > pi / 2)
                {
                        phi = pi / 2;
                        theta *= -1;
                }
        }
        else if (status == 2)
                                                        // jeśli prawy klawisz myszy wciśnięty
        {
                viewer[2] += delta_y;                   // modyfikacja polozenia Z obserwatora (zoom)
                if (viewer[2] <= 7)                     // ograniczenie zblizenia
                        viewer[2] = 7;
                if (viewer[2] >= 200)                   // ograniczenie oddalenia
                        viewer[2] = 200;
        }

        viewer[0] = R * cos(theta) * cos(phi);
        viewer[1] = R * sin(phi);
        viewer[2] = R * sin(theta) * cos(phi);

        Axes(2);                                        // Narysowanie osi przy pomocy funkcji zdefiniowanej powyżej
        Egg();                                          // Narysowanie jajka
        glFlush();                                      // Przekazanie poleceń rysujących do wykonania
        glutSwapBuffers();
}
```

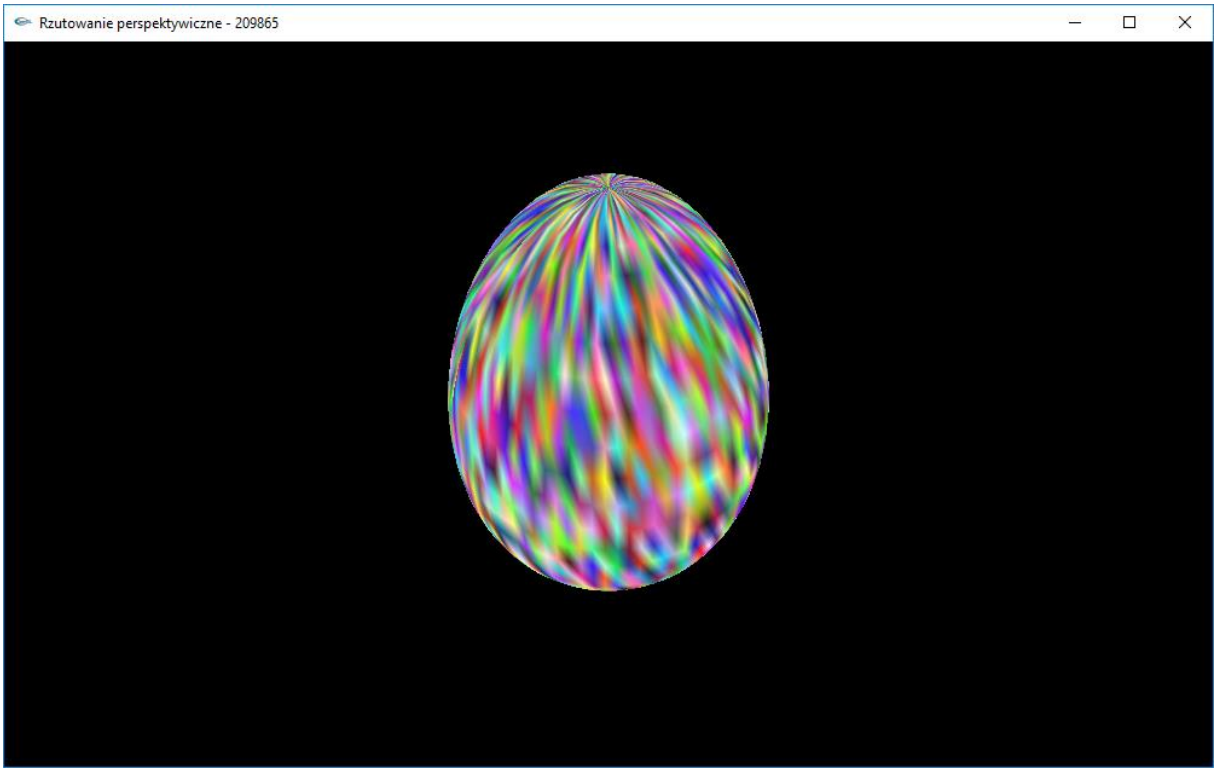
DZIAŁANIE PO MODYFIKACJI

Tym razem nie obraca się jajko, a kamera wokół niego. Najprościej obrazuje to poniższy rysunek:



Rys. 3 Obiekt, obserwator i kąty azymutu oraz elewacji

Kiedy zmienia się pozycja kamery, jajko pozostaje w tym samym miejscu. Dalej może sprawiać wrażenie „kładzenia się" do poziomu, ale to tak naprawdę kamera przesuwa się w górę lub w dół (kąt Φ) i jajko nie „położy się" na bok, a co najwyżej w przód lub tył.



Rys. 4 Jajko z obracającą się kamerą

Po tych modyfikacjach pojawił się drobny problem. Podczas ustawienia kamery nad lub pod jajkiem (Φ = ±90°) kamera nagle obraca się o 180°. Niestety problem ten nie został on rozwiązany.

PODSUMOWANIE

Ćwiczenie pokazało nowy typ rzutowania – rzutowanie perspektywiczne. Dodatkowo uwidocznione zostały różnice pomiędzy obrotem obiektu, a obrotem kamery wokół niego i trudności z nim związane.