Grafika komputerowa i komunikacja człowiek-komputer

Patryk Jurkiewicz 263896

Ćwiczenie 6 - teksturowanie powierzchni obiektów

1. Cel ćwiczenia

   Głównym celem ćwiczenia jest demonstracja technik teksturowania powierzchni obiektów, z wykorzystaniem mechanizmów udostępnianych przez bibliotekę OpenGL wraz z rozszerzeniem GLUT. Sprawozdanie skupi się na przestawieniu kroków niezbędnych do efektywnego zastosowania tekstur w trójwymiarowym modelowaniu, rozpoczynając od procesu wczytywania obrazu tekstury, a kończąc na nałożeniu odpowiednich fragmentów na poszczególne elementy modelu obiektu. Przedstawione zostaną praktyczne przykłady teksturowania form geometrycznych, takich jak wielościan oraz bardziej złożony model w postaci jajka siatki trójkątów.

2. Oświetlanie scen 3D

   Teksturowanie modeli 3D stanowi kluczowy aspekt w grafice komputerowej, umożliwiając realistyczne odwzorowanie powierzchni obiektów poprzez nanoszenie na nie tekstur w postaci map bitowych. Proces ten można podzielić na trzy główne etapy. Po pierwsze, konieczne jest wczytanie danych obrazu tekstury z pliku i przechowanie ich w pamięci. Następnie definiuje się teksturę, czyli określa się sposób interpretacji danych odczytanych z pliku. Ostatecznym krokiem jest nałożenie tekstury na odpowiednie elementy trójwymiarowego modelu obiektu. Biblioteka OpenGL ułatwia realizację dwóch ostatnich etapów poprzez dostarczenie funkcji do definiowania i nakładania tekstur. W dalszej części sprawozdania przedstawiony zostanie przykład ilustrujący zagadnienie teksturowania powierzchni ostrosłupa. W ramach ćwiczenia zostanie zaprezentowany program, który pierwotnie wyświetlał obraz obracającego się jajka. W celu lepszego zrozumienia procesu teksturowania, program ten zostanie zaadaptowany do wyświetlania ostrosłupa. Dodatkowo, korzystając z informacji zdobytych w ćwiczeniu 5 dotyczącym oświetlenia, trójkąt zostanie odpowiednio oświetlony poprzez zdefiniowanie własności materiału i wprowadzenie punktowego źródła światła.

3. Kod

```cpp
// Definicje umożliwiające użycie wartości matematycznych z biblioteki M_PI itp.
#define _USE_MATH_DEFINES

// Wyłącza ostrzeżenia związane z używaniem niebezpiecznych funkcji CRT w systemie Windows
#define _CRT_SECURE_NO_WARNINGS

// Nagłówki bibliotek do obsługi grafiki OpenGL oraz standardowe biblioteki C++
#include <windows.h>
#include <gl/gl.h>
#include <gl/glut.h>
#include <cmath>
#include <iostream>
#include <stdlib.h>
#include <time.h>

// Przestrzeń nazw standardowa
using namespace std;

// Definicja typu tablicy trójwymiarowej punktów
typedef float point3[3];

// Domyślna liczba punktów do generowania dla powierzchni jajka
int N = 100;

// Zmienna przechowująca aktualny model do wyświetlenia
int model;

// Flagi włączające/wyłączające światła
bool light0 = false;
bool light1 = false;

// Domyślny promień obiektu
static GLfloat R = 10.0;

// Współrzędne widza
static GLfloat viewer[] = { 0.0, 0.0, 10.0 };

// Punkt obserwacji (punkt, na który patrzy widz)
static GLfloat punkt_obserwacji[] = { 0.0, 0.0, 0.0 };

// Kąty obrotu obiektu wokół osi x i y
static GLfloat theta_x = 0.0;
static GLfloat theta_y = 0.0;

// Współczynniki obracania obiektu wokół osi x i y
static GLfloat theta_x1 = 0.0;
static GLfloat theta_y1 = 0.0;

// Rozdzielczość jednostkowa kąta (piksele na stopień)
static GLfloat pix2angle;

// Flagi kontrolujące widoczność poszczególnych ścian obiektu
bool sciana1_show = true;
bool sciana2_show = true;
bool sciana3_show = true;
bool sciana4_show = true;
bool sciana5_show = true;

// Stan interakcji myszą (0 - brak interakcji, 1 - obrót, 2 - przesunięcie w pionie)
static GLint status = 0;

// Zmienne przechowujące poprzednie położenia myszy i ich zmiany
static int x_pos_old = 0;
static int delta_x = 0;
static int y_pos_old = 0;
static int delta_y = 0;
static int z_pos_old = 0;
static int delta_z = 0;

```

Kod rozpoczyna się od definicji umożliwiających korzystanie z wartości matematycznych z biblioteki, takich jak M_PI. Następnie wyłącza ostrzeżenia związane z używaniem niebezpiecznych funkcji CRT w systemie Windows. Nagłówki bibliotek do obsługi grafiki OpenGL oraz standardowe biblioteki C++ są inkludowane, co umożliwia późniejsze korzystanie z funkcji z tych bibliotek. W kolejnej części kodu definiowany jest typ point3 jako tablica trójwymiarowa punktów, co zapewne będzie używane do przechowywania współrzędnych punktów w trójwymiarowej przestrzeni. Następnie zadeklarowana jest zmienna N, określająca domyślną liczbę punktów do generowania dla powierzchni jajka. Następnie mamy flagi light0 i light1, które pozwalają na włączanie i wyłączanie świateł. Domyślny promień obiektu R oraz współrzędne widza i punktu obserwacji są zdefiniowane jako stałe. Kod zawiera również zmienne związane z obrotem obiektu wokół osi x i y, a także ich współczynniki. Zmienna pix2angle określa rozdzielczość jednostkową kąta (piksele na stopień). Flagi sciana1_show, sciana2_show, ..., sciana5_show kontrolują widoczność poszczególnych ścian obiektu. Stan interakcji myszą jest przechowywany w zmiennej status, a zmienne x_pos_old, y_pos_old, z_pos_old przechowują poprzednie położenia myszy.

```
// Funkcja do wczytywania obrazów w formacie TGA
GLbyte* LoadTGAImage(const char* FileName, GLint* ImWidth, GLint* ImHeight, GLint* ImComponents, GLenum* ImFormat)
{
    // Struktura 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)

    // Deklaracje zmiennych i wskaźników
    FILE* pFile;
    TGAHEADER tgaHeader;
    unsigned long lImageSize;
    short sDepth;
    GLbyte* pbitsperpixel = NULL;

    // Inicjalizacja wartości zmiennych wyjściowych
    *ImWidth = 0;
    *ImHeight = 0;
    *ImFormat = GL_BGR_EXT;
    *ImComponents = GL_RGB8;

    // Otwarcie pliku do odczytu binarnego
    pFile = fopen(FileName, "rb");
    if (pFile == NULL) {
        cout << "Błąd odczytu pliku" << endl;
        return NULL;
    }

    // Odczyt nagłówka pliku TGA
    fread(&tgaHeader, sizeof(TGAHEADER), 1, pFile);
```

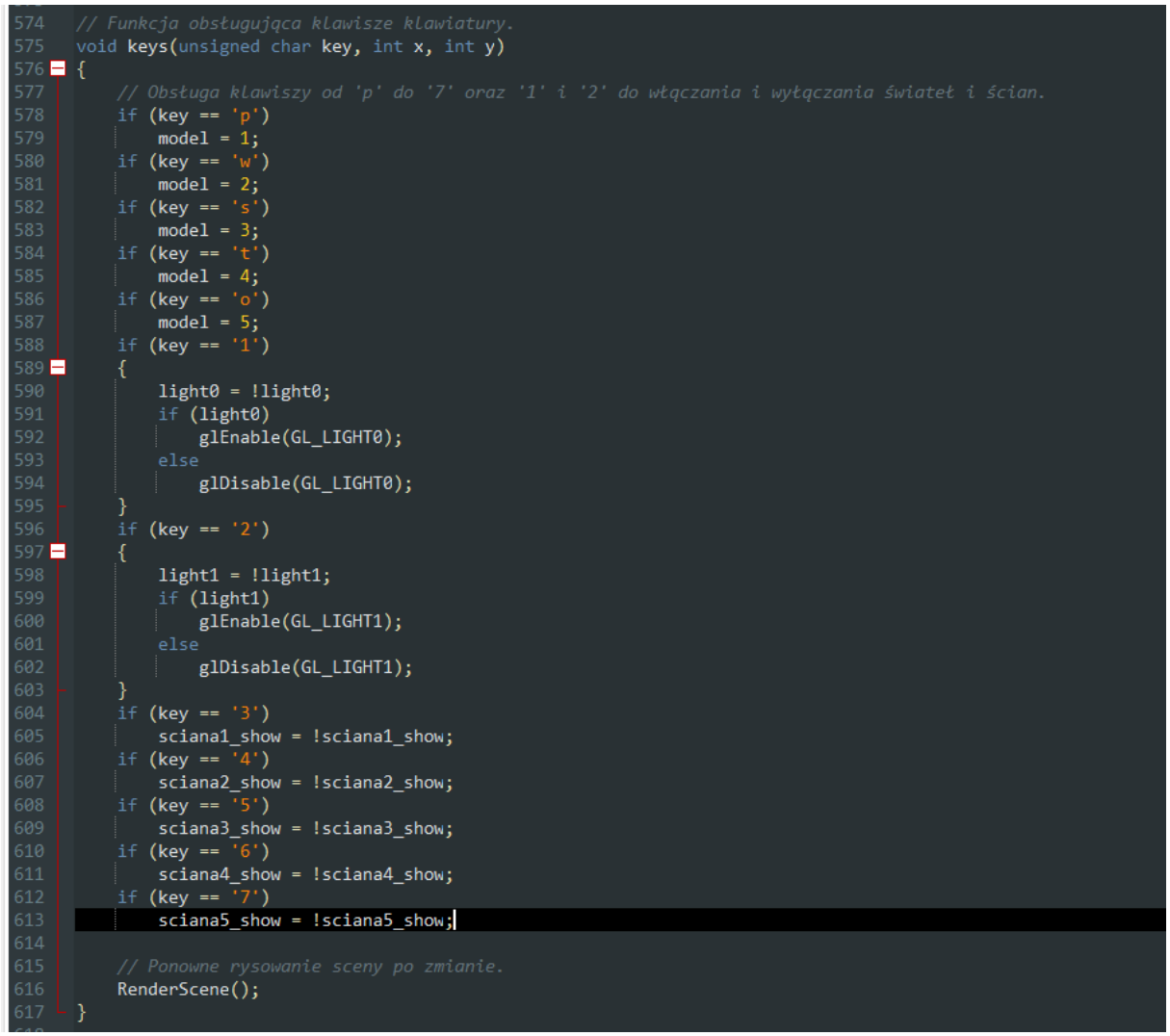
```
115
116     // Przypisanie szerokości, wysokości i głębokości pikseli obrazu
117     *ImWidth = tgaHeader.width;
118     *ImHeight = tgaHeader.height;
119     sDepth = tgaHeader.bitsperpixel / 8;
120
121     // Sprawdzenie poprawności głębokości pikseli
122     if (tgaHeader.bitsperpixel != 8 && tgaHeader.bitsperpixel != 24 && tgaHeader.bitsperpixel != 32)
123         return NULL;
124
125     // Obliczenie rozmiaru obrazu
126     lImageSize = tgaHeader.width * tgaHeader.height * sDepth;
127
128     // Alokacja pamięci dla danych pikseli
129     pbitsperpixel = (GLbyte*)malloc(lImageSize * sizeof(GLbyte));
130
131     // Sprawdzenie poprawności alokacji pamięci
132     if (pbitsperpixel == NULL)
133         return NULL;
134
135     // Odczyt danych pikseli z pliku
136     if (fread(pbitsperpixel, lImageSize, 1, pFile) != 1)
137     {
138         free(pbitsperpixel);
139         return NULL;
140     }
141
142     // Ustalenie formatu i składowych obrazu w zależności od głębokości pikseli
143     switch (sDepth)
144     {
145     case 3:
146         *ImFormat = GL_BGR_EXT;
147         *ImComponents = GL_RGB8;
148         break;
149
150     case 4:
151         *ImFormat = GL_BGRA_EXT;
152         *ImComponents = GL_RGBA8;
153         break;
154
155     case 1:
156         *ImFormat = GL_LUMINANCE;
157         *ImComponents = GL_LUMINANCE8;
158         break;
159     };
160
161     // Zamknięcie pliku
162     fclose(pFile);
163
164     // Zwrócenie wskaźnika na dane pikseli obrazu
165     return pbitsperpixel;
166 }
167
```

Funkcja LoadTGAImage została stworzona w celu wczytywania obrazów w formacie TGA, który jest szeroko stosowany w grafice komputerowej. Kod inicjalizuje strukturę TGAHEADER, reprezentującą nagłówek pliku TGA, a następnie otwiera plik binarny do odczytu. W przypadku niepowodzenia otwarcia pliku, funkcja zwraca wartość NULL, sygnalizując błąd. Następnie odczytuje nagłówek pliku TGA, uzyskując informacje takie jak szerokość, wysokość i głębokość pikseli obrazu. Po sprawdzeniu poprawności głębokości pikseli, funkcja alokuje pamięć na dane pikseli i odczytuje te dane z pliku. Format i składowe obrazu są ustawiane zależnie od głębokości pikseli. Ostatecznie funkcja zamyka plik i zwraca wskaźnik na zaalokowaną pamięć zawierającą dane pikseli obrazu. Warto zauważyć, że funkcja obsługuje obrazy o głębokości pikseli 8, 24 i 32 bitów na piksel. W przypadku błędu odczytu pliku lub alokacji pamięci, zwracany jest wskaźnik NULL, co może być wykorzystane do obsługi błędów w programie wykorzystującym tę funkcję.

```
// Funkcja rysująca ostrzałup składający się z pięciu ścian.
void rysujOstroslup()
{
    // Rysowanie pierwszej ściany (QUADS) jeśli flaga sciana1_show jest ustawiona na true.
    if (sciana1_show)
    {
        glBegin(GL_QUADS);
        glNormal3f(-5.0, -5.0, 0.0);
        glTexCoord2f(0.0f, 0.0f);
        glVertex3f(-5.0, -5.0, 0.0);
        glNormal3f(5.0, -5.0, 0.0);
        glTexCoord2f(1.0f, 0.0f);
        glVertex3f(5.0, -5.0, 0.0);
        glNormal3f(5.0, 5.0, 0.0);
        glTexCoord2f(1.0f, 1.0f);
        glVertex3f(5.0, 5.0, 0.0);
        glNormal3f(-5.0, 5.0, 0.0);
        glTexCoord2f(0.0f, 1.0f);
        glVertex3f(-5.0, 5.0, 0.0);
        glEnd();
    }

    // Rysowanie drugiej ściany (TRIANGLES) jeśli flaga sciana2_show jest ustawiona na true.
    if (sciana2_show)
    {
```

Funkcja rysujOstroslup została zaimplementowana w celu rysowania ostrzałupu składającego się z pięciu ścian. Każda ściana jest rysowana w zależności od ustawionych flag (sciana1_show, sciana2_show, sciana3_show, sciana4_show, sciana5_show). Kod używa funkcji do rysowania trójkątów i czworokątów, a także określa normalne i współrzędne tekstur dla każdego wierzchołka. Przykładowo, pierwsza ściana (sciana1_show) jest rysowana jako kwadrat (GL_QUADS) zdefiniowany przez cztery wierzchołki. Kolejne ściany (sciana2_show do sciana5_show) są rysowane jako trójkąty (GL_TRIANGLES) i korzystają z różnych kombinacji wierzchołków, co tworzy efekt trójwymiarowej bryły. Warto zauważyć, że dla każdej ściany określana jest normalna, co wpływa na sposób oświetlenia powierzchni. Ponadto, dla każdej ściany można włączyć/wyłączyć rysowanie poprzez odpowiednie ustawienie flagi, co umożliwia dynamiczną modyfikację wyglądu ostrzałupu w trakcie działania programu.



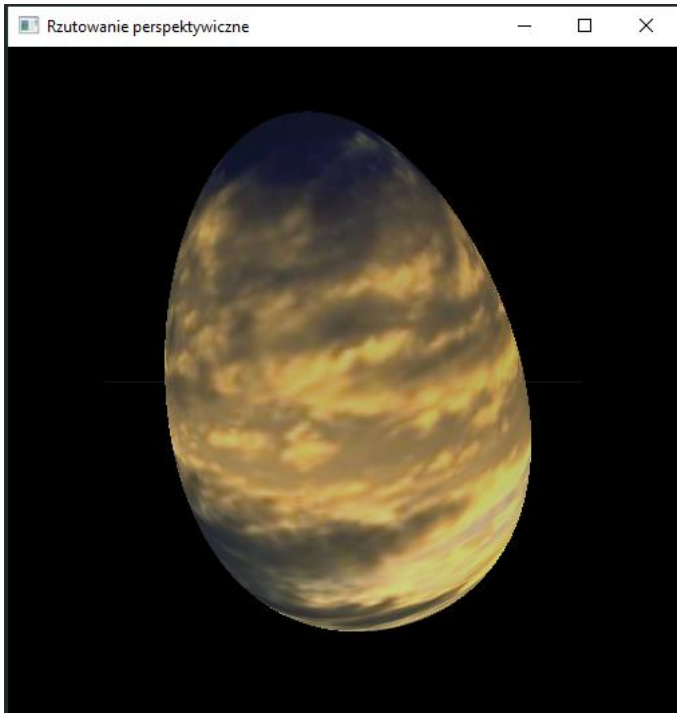
```
// Funkcja obsługująca klawisze klawiatury.
void keys(unsigned char key, int x, int y)
{
    // Obsługa klawiszy od 'p' do '7' oraz '1' i '2' do włączania i wyłączania świateł i ścian.
    if (key == 'p')
        model = 1;
    if (key == 'w')
        model = 2;
    if (key == 's')
        model = 3;
    if (key == 't')
        model = 4;
    if (key == 'o')
        model = 5;
    if (key == '1')
    {
        light0 = !light0;
        if (light0)
            glEnable(GL_LIGHT0);
        else
            glDisable(GL_LIGHT0);
    }
    if (key == '2')
    {
        light1 = !light1;
        if (light1)
            glEnable(GL_LIGHT1);
        else
            glDisable(GL_LIGHT1);
    }
    if (key == '3')
        sciana1_show = !sciana1_show;
    if (key == '4')
        sciana2_show = !sciana2_show;
    if (key == '5')
        sciana3_show = !sciana3_show;
    if (key == '6')
        sciana4_show = !sciana4_show;
    if (key == '7')
        sciana5_show = !sciana5_show;

    // Ponowne rysowanie sceny po zmianie.
    RenderScene();
}
```

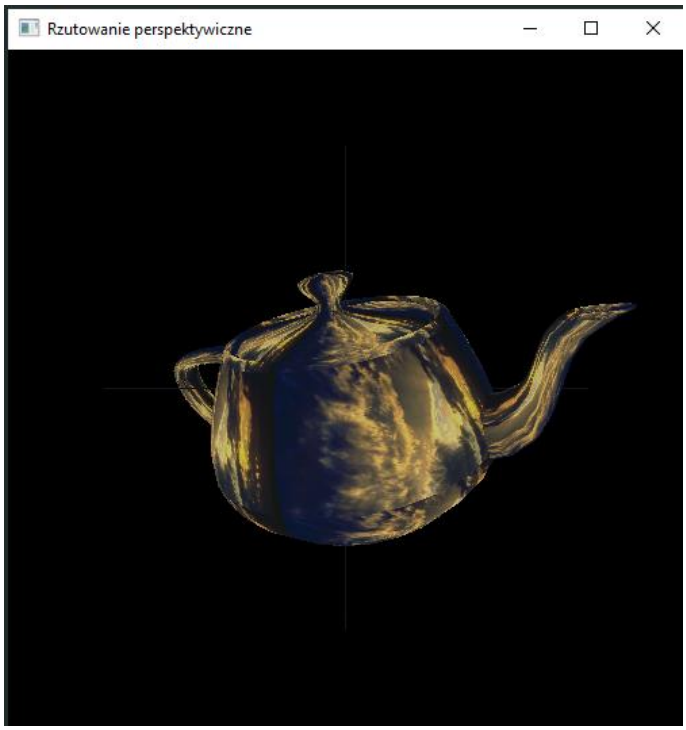
Funkcja "keys" stanowi kluczowy element obsługi klawiatury w kontekście pewnej aplikacji graficznej. Jej głównym zadaniem jest reagowanie na konkretne klawisze poprzez wprowadzanie zmian w programie. Na przykład, po naciśnięciu klawisza 'p', zmienna "model" zostaje ustawiona na 1, co wpływa na wybór określonego modelu w scenie. Analogicznie, inne klawisze, takie jak 'w', 's', 't', 'o', czy '1' i '2', odpowiadają za różne aspekty wizualne, takie jak modele, światła czy ściany. Dodatkowo, funkcja umożliwia dynamiczne przełączanie stanów, na przykład włączanie lub wyłączanie świateł (GL_LIGHT0, GL_LIGHT1) oraz kontrolę widoczności poszczególnych ścian. Po każdej zmianie, funkcja inicjuje ponowne renderowanie sceny za pomocą wywołania funkcji "RenderScene()". Dzięki temu interakcja z klawiaturą umożliwia użytkownikowi elastyczną modyfikację wyglądu graficznego sceny, zapewniając dynamiczną kontrolę nad jej elementami.

4. Efekty i wnioski

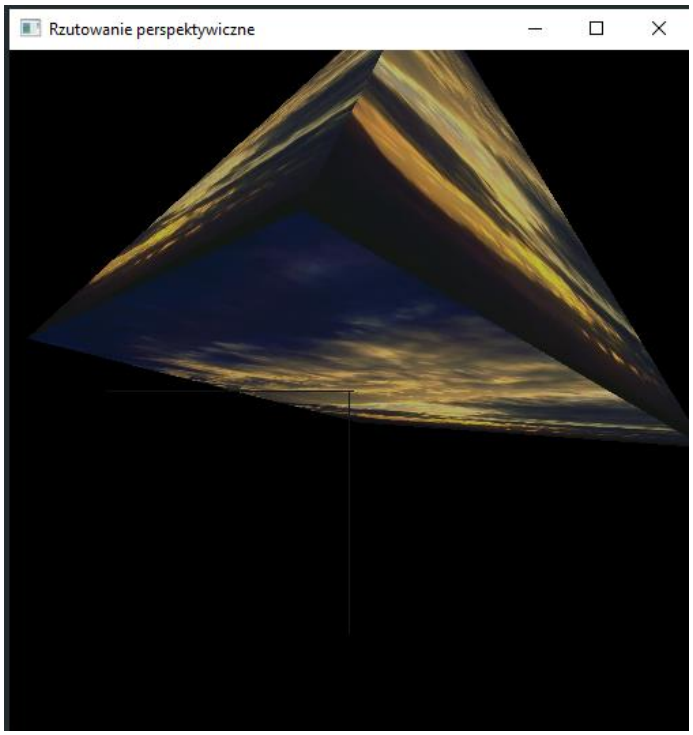
Podczas wykonywania ćwiczenia dotyczącego teksturowania powierzchni obiektów w grafice komputerowej, zdobyłem głębsze zrozumienie procesów związanych z nanoszeniem tekstur na trójwymiarowe modele. Ćwiczenie skoncentrowało się na praktycznych aspektach implementacji, począwszy od wczytywania obrazu tekstury po jej nałożenie na poszczególne elementy obiektu. Poprzez eksperymenty z różnymi typami tekstur oraz różnymi obiektami, takimi jak wielościan czy jajko siatki trójkątów, udało mi się uzyskać wgląd w wpływ teksturowania na realizm i estetykę renderowanej sceny. W ramach rozszerzenia ćwiczenia o elementy związane z oświetleniem scen 3D, zaimplementowałem funkcję rysującą ostrosłup składający się z pięciu ścian. To połączenie teksturowania i oświetlenia pozwoliło mi lepiej zrozumieć, jak te dwa aspekty mogą współgrać, tworząc bardziej realistyczne wrażenie trójwymiarowych obiektów. Dodatkowo, dynamiczna obsługa klawiatury umożliwiła mi interaktywną kontrolę nad elementami sceny, co jest istotnym aspektem w projektowaniu grafiki komputerowej. Podsumowując, ćwiczenie pozwoliło mi rozwijać umiejętności programistyczne z zakresu grafiki komputerowej oraz zwiększyć moją wiedzę na temat zastosowań tekstur i oświetlenia w trójwymiarowym modelowaniu. Eksperymentalne podejście do implementacji, eksploracja różnych ustawień i dynamiczna kontrola nad sceną dostarczyły mi cennego doświadczenia w obszarze grafiki komputerowej i komunikacji człowiek-komputer.



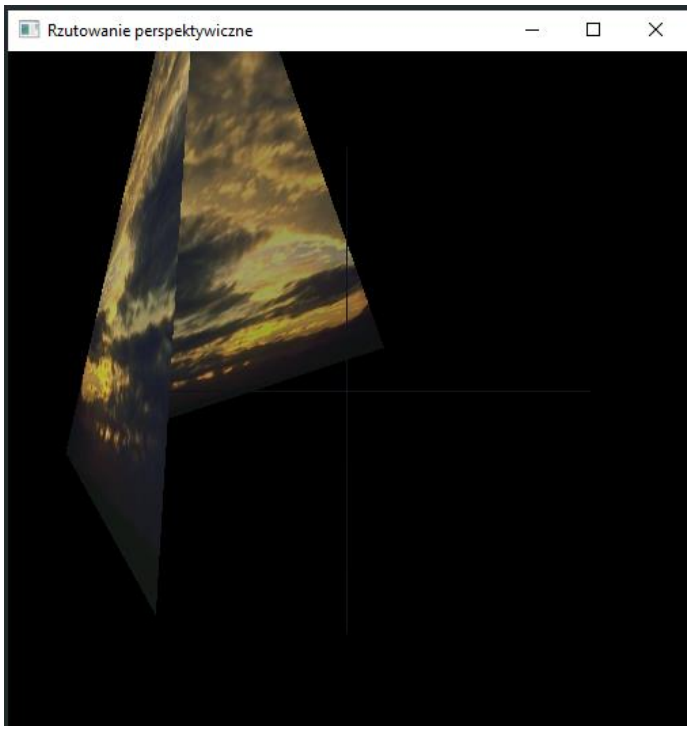
Rys1. Jajko z nałożoną teksturą



Rys 2. Czajniczek z teksturą



Rys 3. Pełny ostrosłup z teksturą



Rys 4. Ostrosłup z niektórymi wyłączonymi ścianami