Wydział Elektroniki

Kierunek Informatyka

Politechnika Wrocławska

LABORATORIUM GRAFIKI KOMPUTEROWEJ I KOMUNIKACJI CZŁOWIEK KOMPUTER

Ćwiczenie nr 6

Temat:

Teksturowanie

Autor sprawozdania: Dominik Kurowski (248840)

Grupa laboratoryjna: Środa TP 11:45 – 14:15

Prowadzący: Dr inż. Jan Nikodem

1. Cel ćwiczenia

Celem ćwiczenia jest pokazanie podstawowych technik teksturowania powierzchni obiektów z wykorzystaniem mechanizmów biblioteki OpenGL z rozszerzeniem GLUT. Na przykładach zilustrowana będzie droga od przeczytania obrazu tekstury, do nałożenia jaj fragmentów na poszczególne fragmenty modelu obiektu trójwymiarowego. Pokazane zostaną przykłady teksturowania wielościanu i bardziej skomplikowanego modelu w postaci siatki trójkątów (jajka).

2. Zrealizowane zadania

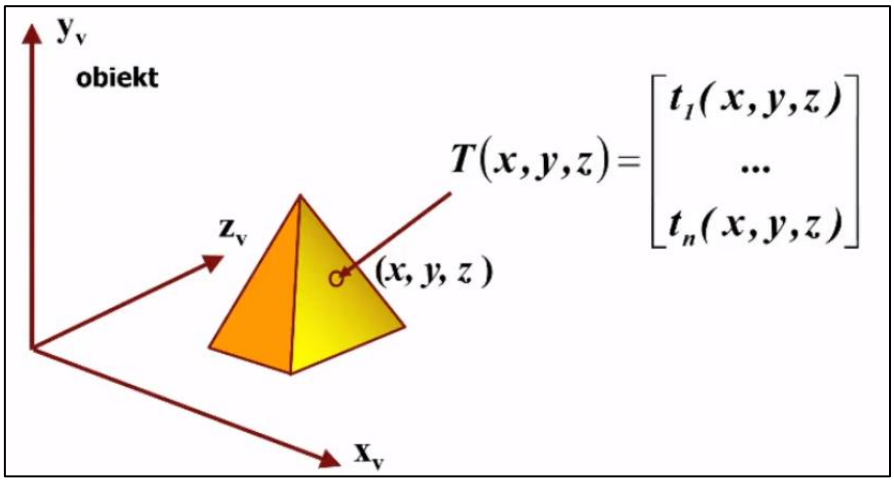
W ramach laboratorium mieliśmy wykonać zadania opisane w instrukcji „OpenGL - teksturowanie powierzchni obiektów" znajdującej się na stronie Zespołu Systemów Komputerowych. Były to kolejno:

A. Teksturowanie wielościanu (piramidy) z możliwością sterowania widocznością poszczególnych ścian przy pomocy klawiatury.
B. Teksturowanie bardziej skomplikowanego modelu w postaci siatki trójkątów (jajka).

3. Zagadnienia teoretyczne

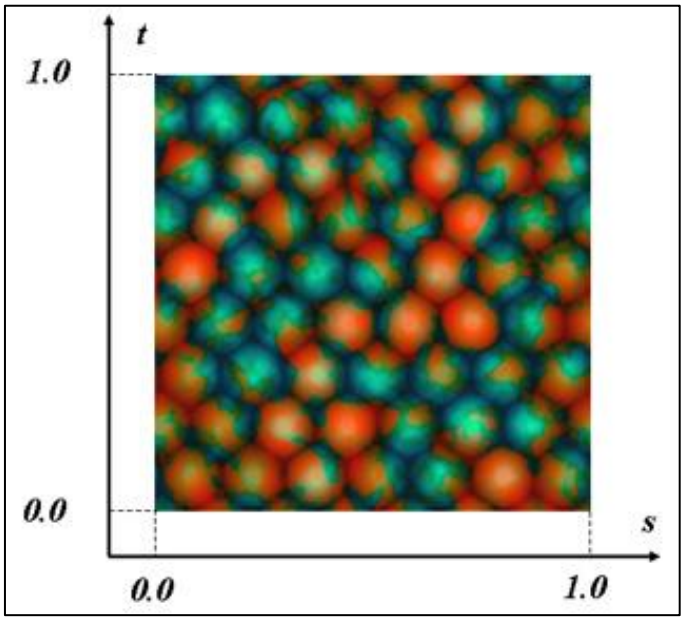
- **Tekstura oraz teksturowanie**

Tekstura jest funkcją wektorową określoną dla punktów należących do powierzchni obiektu. W przypadku tego ćwiczenia tekstura będzie opisywana na podstawie zewnętrznego pliku graficznego.

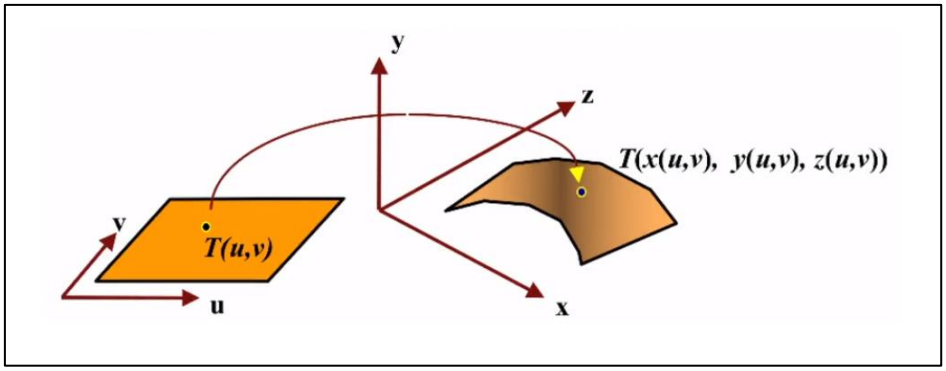


Rysunek 1. Intuicja stojąca za teksturą

Odwzorowanie współrzędnych dwuwymiarowej tekstury na współrzędne obiektu trójwymiarowego nosi nazwę – mapowanie tekstury:



Rysunek 3. Wzorzec w układzie współrzędnych tekstury



Rysunek 2. Rzutowanie parametryczne

1. Odwzorowanie może zostać utworzone automatycznie, w procesie generacji obiektu przez oprogramowanie.
2. Jeśli obiekt jest reprezentowany przez siatkę wielokątów projektant może określić współrzędne tekstury dla każdego wierzchołka – jest to tak zwany UV mapping.
3. Tekstura mapowana jest najpierw na jakąś prostą figurę przestrzenną, a następnie konkretne punkty są rzutowane na teksturowaną figurę. Używane figury:
    - płaszczyzna (mapowanie płaskie)
    - sfera lub elipsoida (mapowanie sferyczne)
    - powierzchnia walca (mapowanie cylindryczne)
    - sześcian (mapowanie sześcienne, ang. box mapping).

Odwzorowanie jest wykonywane w lokalnym układzie współrzędnych obiektu, co sprawia, że tekstura jest „przyklejona” na stałe do obiektu, nadając jego powierzchni pożądany charakter. Rzadziej tekstura jest ruchoma względem obiektu; w grafice czasu rzeczywistego (np. grach komputerowych), w ten sposób symuluje się metaliczne obiekty, które odbijają otoczenie (jest to tzw. mapowanie środowiska, ang. environment mapping).

- **Współrzędne jednorodne**

Niech x_p, y_p, z_p, opisują położenie punktu w trójwymiarowym kartezjańskim układzie współrzędnych. W grafice komputerowej do opisu położenia oraz opisu operacji (transformacji geometrycznych) jest używany ***układ współrzędnych jednorodnych znormalizowanych***. Dzięki temu wszystkie stosowane transformacje geometryczne mogą być opisane w identyczny sposób za pomocą mnożenia macierzowego. Jeśli współrzędne x_p, y_p, z_p opisują położenie punktu, to odpowiada temu wektor:

$$P = \begin{bmatrix} x_p & y_p & z_p & 1 \end{bmatrix}^T$$

Zastosowanie w przypadku przesunięcia na płaszczyźnie współrzędnych jednorodnych można sobie wyobrazić jako umieszczenie płaszczyzny, na której pracujemy, w trójwymiarowym układzie współrzędnych, w taki sposób, aby nie przechodziła ona przez początek układu (tzn. dla $z = h_z \neq 0$). Wtedy analogiczne opisanie operacji translacji na płaszczyźnie (ale już jako macierz 3x3) da poprawne rozwiązanie, gdyż punkt stały – początek układu współrzędnych jest poza płaszczyzną, na której jest wykonywana operacja. Jednocześnie, aby wynik operacji znajdował się na tej samej płaszczyźnie, najprościej operować na współrzędnych znormalizowanych, czyli pracować na płaszczyźnie $z = 1$.

Analogicznie dla przekształceń trójwymiarowych można wyobrazić sobie umieszczenie przestrzeni 3D i trójwymiarowego układu współrzędnych wewnątrz układu czterowymiarowego, tak aby nie zawierał on początku układu współrzędnych.

- **Wektor normalny – wyznaczenie wektora dla modelu jajka**

W ćwiczeniu „Modelowanie 3D” budowaliśmy model jajka na postawie równań parametrycznych powierzchni w postaci:

$$x(u,v) = (-90u^5 + 225u^4 - 270u^3 + 180u^2 - 45u)\cos(\pi v)$$
$$y(u,v) = 160u^4 - 320u^3 + 160u^2$$
$$z(u,v) = (-90u^5 + 225u^4 - 270u^3 + 180u^2 - 45u)\sin(\pi v)$$
$$0 \le u \le 1$$
$$0 \le v \le 1$$

Wektor normalny do punktu leżącego na tak opisanej powierzchni można znaleźć posługując się wzorami:

$$N(u,v) = \left[\begin{vmatrix} y_u & z_u \\ y_v & z_v \end{vmatrix}, \begin{vmatrix} z_u & x_u \\ z_v & x_v \end{vmatrix}, \begin{vmatrix} x_u & y_u \\ x_v & y_v \end{vmatrix}\right] =$$
$$= \left[y_u \cdot z_v - z_u y_v, \quad z_u \cdot x_v - x_u z_v, \quad x_u \cdot y_v - y_u x_v\right] \neq 0$$

przy czym

$$x_u = \frac{\partial x(u,v)}{\partial u}, \quad x_v = \frac{\partial x(u,v)}{\partial v}$$
$$y_u = \frac{\partial y(u,v)}{\partial u}, \quad y_v = \frac{\partial y(u,v)}{\partial v}$$
$$z_u = \frac{\partial z(u,v)}{\partial u}, \quad z_v = \frac{\partial z(u,v)}{\partial v}$$

Po wykonaniu operacji różniczkowania otrzymuje się wzory pozwalające na łatwe obliczenie wektora normalnego:

$$x_u = \frac{\partial x(u,v)}{\partial u} = \left(-450u^4 + 900u^3 - 810u^2 + 360u - 45\right) \cdot \cos(\pi v)$$

$$x_v = \frac{\partial x(u,v)}{\partial v} = \pi \cdot \left(90u^5 - 225u^4 + 270u^3 - 180u^2 + 45u\right) \cdot \sin(\pi v)$$

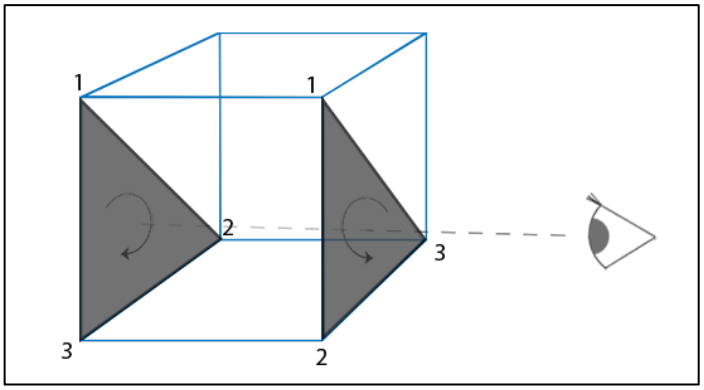
$$y_u = \frac{\partial y(u,v)}{\partial u} = 640u^3 - 960u^2 + 320u$$

$$y_v = \frac{\partial y(u,v)}{\partial v} = 0$$

$$z_u = \frac{\partial z(u,v)}{\partial u} = \left(-450u^4 + 900u^3 - 810u^2 + 360u - 45\right) \cdot \sin(\pi v)$$

$$z_v = \frac{\partial z(u,v)}{\partial v} = -\pi \cdot \left(90u^5 - 225u^4 + 270u^3 - 180u^2 + 45u\right) \cdot \cos(\pi v)$$

Wektor normalny jest kluczowy w przypadku ustawienia teksturowania wyłącznie jednej ze stron modelu (`glEnable(GL_CULL_FACE)`). Na jego podstawie możemy ustalić gdzie jest przód danej ściany.



Rysunek 4. Zjawisko face cullingu w OpenGL

4. Fragmenty kodu

- **Inicjalizacja parametrów teksturowania**

```
223  void MyInit(void) {
224
225      GLbyte* pBytes;
226      GLint ImWidth, ImHeight, ImComponents;
227      GLenum ImFormat;
228
229      pBytes = LoadTGAImage("textury/D1_t.tga", &ImWidth, &ImHeight, &ImComponents, &ImFormat);
230
231      glTexImage2D(GL_TEXTURE_2D, 0, ImComponents, ImWidth, ImHeight, 0, ImFormat, GL_UNSIGNED_BYTE, pBytes);
232      free(pBytes);
233      glEnable(GL_TEXTURE_2D);
234      glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
235      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
236      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
```

Kod 1. Inicjalizacja parametrów teksturowania

[wiersze 225-227] : zmienne przechowujące informacje o teksturze. **pBytes* jest wskaźnikiem na strumień bajtów. *ImWidth* oraz *ImHeight* to odpowiednio szerokość oraz wysokość używanej tekstury. *ImComponents* informuje o ilość używanych składowych koloru natomiast *ImFormat* o formacie danych obrazu tekstury

[wiersz 229] : wczytanie interesującej nas tekstury do strumienia bajtów **pBytes*

[wiersz 231] : zdefiniowanie tekstury dwuwymiarowej

```
void glTexImage2D(GLenum target, GLint level, GLint components, GLsizei
width, GLsizei height, GLint border, GLenum format, GLenum type, const
GLvoid *pixels)
```

Argumenty funkcji mają następujące znaczenie:

Argument	Opis
target	Rodzaj definiowanej tekstury
level	Poziom szczegółowości. Jest ustawiany na 0 gdy ***mipmapy*** nie są stosowane, gdy ***mipmapy*** są używane ***level*** jest liczbą określająca poziom redukcji obrazu tekstury.
components	Ilość używanych składowych koloru, liczba od 1 do 4.
width	Szerokość obrazu tekstury, zawsze jest potegą liczby 2, może być powiększona o szerokość tak zwanej ramki tekstury
heigth	Wysokość obrazu tekstury, zawsze jest potęgą liczby 2, może być powiększona o szerokość ramki tekstury.
border	Szerokość ramki tekstury, może wynosić 0, 1 lub 2
format	Format danych obrazu tekstury, określa co opisują dane, czy są indeksami kolorów czy ich składowymi (np. R, G, B) i w jakiej są podane kolejności.
type	Typ danych dla punktów obrazu tekstury, określa jak kodowane są dane, istnieje możliwość używania różnych formatów liczb, od 8 bitowych liczb stałoprzecinkowych, do 32 bitowych liczb zmiennoprzecinkowych.
piksels	Tablica o rozmiarze ***width*** x ***height*** x ***components***, w ktorej znajdują się dane z obrazem tekstury.

[wiersz 232] : zwolnienie pamięci

[wiersz 233] : włączenie mechanizmu teksturowania

[wiersz 234] : ustalenie trybu teksturowania. Możliwe parametry to *GL_MODULATE* (kolor piksela ekranu jest mnożony przez kolor piksela tekstury, następuje w ten sposób mieszanie barw tekstury i tego co jest teksturowane), *GL_DECAL* (piksele tekstury zastępują piksele na ekranie), *GL_BLEND* (kolor piksela ekranu jest mnożony przez kolor piksela tekstury i łączony ze stałym kolorem) oraz *GL_REPLACE* (działa podobnie jak *GL_DECAL* różnica występuje wtedy, gdy wprowadzona zostaje przezroczystość)

[wiersz 235-236] : określenie sposobu nakładania tekstury

```
void glTexParameteri(GLenum target, GLenum pname, GLint param)
```

Argumenty funkcji mogą przyjmować następujące wartości:

pname	*param*
GL_TEXTURE_MIN_FILTER	Opisuje w jaki sposób następuje pominiejszanie tekstury (usuwanie pikseli) **GL_NEAREST** - usuwa się najbliższy (w sensie metryki Manhattan) punkt sąsiedni **GL_LINEAR** - filtracja liniowa z odpowiednimi wagami
GL_TEXTURE_MAG_FILTER	Opisuje w jaki sposób następuje powiększanie tekstury (dodawanie pikseli) **GL_NEAREST** - dodaje się najbliższy (w sensie metryki Manhattan) punkt sąsiedni **GL_LINEAR** - interpolacja liniowa wykorzystująca wartości sąsiednich pikseli
GL_TEXTURE_WRAP_S	Opisuje sposób traktowania współrzędnej ***s*** tekstury, gdy wykracza ona poza zakres [0, 1] **GL_CLAMP** - poza zakresem stoswany jest kolor ramki tekstury lub stały kolor **GL_REPEAT** - tekstura jest powtarzana na całej powierzchni wielokąta
GL_TEXTURE_WRAP_T	Opisuje sposób traktowania współrzędnej ***t*** tekstury, gdy wykracza ona poza zakres [0, 1] **GL_CLAMP** - poza zakresem stosowany jest kolor ramki tekstury lub stały kolor **GL_REPEAT** - tekstura jest powtarzana na całej powierzchni wielokąta

- **Rysowanie jajka**

```
119 void drawEgg() {
120     for (int i = 0; i < N; ++i) {
121         for (int j = 0; j < N; ++j) {
122             glBegin(GL_TRIANGLES);
123                 glNormal3fv(normal_vectors[i][j]);
124                 glTexCoord2fv(texture_points[i][j]);
125                 glVertex3fv(points[i][j]);
126 
127                 glNormal3fv(normal_vectors[i + 1][j]);
128                 glTexCoord2fv(texture_points[i + 1][j]);
129                 glVertex3fv(points[i + 1][j]);
130 
131                 glNormal3fv(normal_vectors[i + 1][j + 1]);
132                 glTexCoord2fv(texture_points[i + 1][j + 1]);
133                 glVertex3fv(points[i + 1][j + 1]);
134             glEnd();
135 
136             glBegin(GL_TRIANGLES);
137                 glNormal3fv(normal_vectors[i][j]);
138                 glTexCoord2fv(texture_points[i][j]);
139                 glVertex3fv(points[i][j]);
140 
141                 glNormal3fv(normal_vectors[i][j + 1]);
142                 glTexCoord2fv(texture_points[i][j + 1]);
143                 glVertex3fv(points[i][j + 1]);
144 
145                 glNormal3fv(normal_vectors[i + 1][j + 1]);
146                 glTexCoord2fv(texture_points[i + 1][j + 1]);
147                 glVertex3fv(points[i + 1][j + 1]);
148             glEnd();
149         }
150     }
151 }
```

Kod 2. Rysowanie jajka – rysowanie modelu, określanie wektorów normalnych oraz nakładanie tekstury

- **Główny fragment programu (generowanie danych)**

```
239     const float offset = 1.0f / N;
240
241     float v = 0.0f;
242     float u = 0.0f;
243
244     float x_u, x_v, y_u, y_v, z_u, z_v;
245
246     for (int i = 0; i <= N; ++i) {
247         u = 0.0f;
248
249         for (int j = 0; j <= N; ++j) {
250             // Generowanie punktow w przestrzeni
251             points[i][j][0] =
252                 (-90 * pow(u, 5) + 225 * pow(u, 4) - 270 * pow(u, 3) + 180 * pow(u, 2) - 45 * u) * cos(M_PI * v);
253             points[i][j][1] =
254                 (160 * pow(u, 4) - 320 * pow(u, 3) + 160 * pow(u, 2)) - 5;
255             points[i][j][2] =
256                 (-90 * pow(u, 5) + 225 * pow(u, 4) - 270 * pow(u, 3) + 180 * pow(u, 2) - 45 * u) * sin(M_PI * v);
257
258             // Generowanie punktow dla tekstury
259             texture_points[i][j][0] = u;
260             texture_points[i][j][1] = v;
261
262             // Generowanie wektora normalnego do zadanego punktu
263             x_u = (-450 * pow(u, 4) + 900 * pow(u, 3) - 810 * pow(u, 2) + 360 * u - 45) * cos(M_PI * v);
264             x_v = M_PI * (90 * pow(u, 5) - 225 * pow(u, 4) + 270 * pow(u, 3) - 180 * pow(u, 2) + 45 * u) * sin(M_PI * v);
265             y_u = 640 * pow(u, 3) - 960 * pow(u, 2) + 320 * u;
266             y_v = 0;
267             z_u = (-450 * pow(u, 4) + 900 * pow(u, 3) - 810 * pow(u, 2) + 360 * u - 45) * sin(M_PI * v);
268             z_v = -M_PI * (90 * pow(u, 5) - 225 * pow(u, 4) + 270 * pow(u, 3) - 180 * pow(u, 2) + 45 * u) * cos(M_PI * v);
269
270             normal_vectors[i][j][0] = y_u * z_v - z_u * y_v;
271             normal_vectors[i][j][1] = z_u * x_v - x_u * z_v;
272             normal_vectors[i][j][2] = x_u * y_v - y_u * x_v;
273
274             if (i == 0 || i == N) {
275                 normal_vectors[i][j][0] =  0;
276                 normal_vectors[i][j][1] = -1;
277                 normal_vectors[i][j][2] =  0;
278             }
279             else if (i == (N + 1) / 2) {
280                 normal_vectors[i][j][0] = 0;
281                 normal_vectors[i][j][1] = 1;
282                 normal_vectors[i][j][2] = 0;
283             }
284             else if (i > (N + 1) / 2) {
285                 normal_vectors[i][j][0] *= -1;
286                 normal_vectors[i][j][1] *= -1;
287                 normal_vectors[i][j][2] *= -1;
288             }
289
290             // Normalizacja wektora
291             float vector_lenght = sqrt(pow(normal_vectors[i][j][0], 2) + pow(normal_vectors[i][j][1], 2) + pow(normal_vectors[i][j][2], 2));
292             normal_vectors[i][j][0] /= vector_lenght;
293             normal_vectors[i][j][1] /= vector_lenght;
294             normal_vectors[i][j][2] /= vector_lenght;
295
296             u += offset;
297         }
298         v += offset;
299     }
300 }
```

Kod 3. Generowanie danych

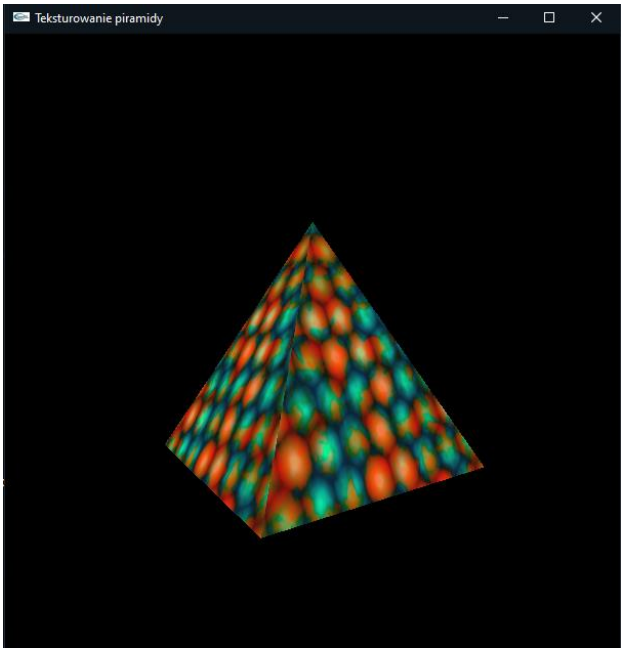
[wiersze 250-255] : generowanie punktów na podstawie równania parametrycznego jajka

[wiersze 259-260] : wyznaczenie tekstury dla danego fragmentu modelu jajka

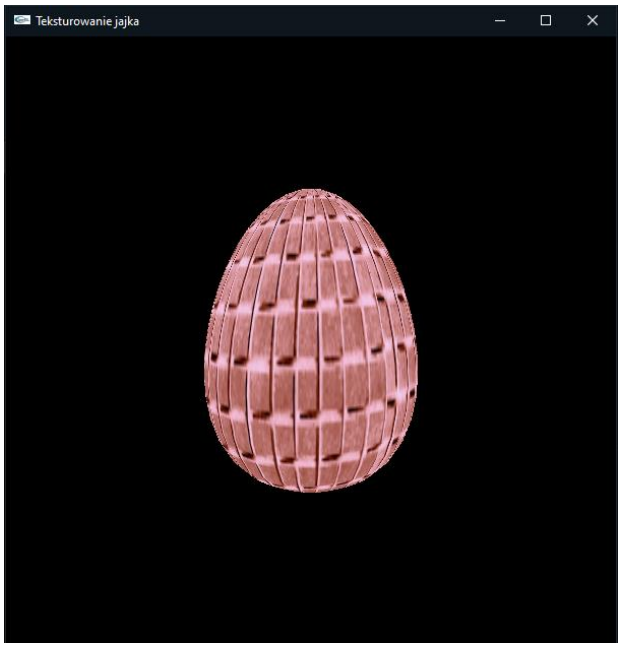
[wiersze 262-288] : wyznaczenie wektorów normalnych na podstawie równań [strona (5)]

[wiersze 291-294] : normalizacja wektorów

5. Wyniki i wnioski



Rysunek 6. zadanie 1 – teksturowanie piramidy



Rysunek 5. zadanie 2 – teksturowanie jajka

Stworzony program poprawnie realizuje wszystkie zadania opisane w instrukcji laboratoryjnej. Podczas jego projektowania nie natrafiliśmy na żadne problemy. Pierwszy program (Rysunek 6) pokazuje przykład teksturowania wielościanu (w naszym przypadku piramidy) oraz oferuje możliwość wyłączenia/włączenia rysowania poszczególnych ścian poprzez aktywację klawiszy 1-5. Drugi program (Rysunek 5) pokazuje przykład teksturowania bardziej złożonego modelu w postaci znanego nam już z poprzednich ćwiczeń jajka. Uważamy, że uzyskana podczas tych ćwiczeń wiedza pozwoliła na lepsze zrozumienie mechanizmów stojących za procesem teksturowania w grafice komputerowej.