Grafika komputerowa - Teksturowanie

Kacper Małkowski 252724

Prowadzący - dr. inż. Jan Nikodem
Zajęcia: Wtorek P 7.30-10.30

1 Wstęp

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 trójkąta, wielościanu i bardziej skomplikowanego modelu w postaci siatki trójkątów.

2 Plan działania

Program ma wyswielać 3 obiekty: trójkąt, piramidę i jajko. Piramida i trójkąt oteksturowane są teksturą cegły. Jajko oteksturowane jest teskturą z szesciokątami - tekstura ta jest niedopasowana do kształtu jajka, więc dodałem możliwość dopasowania tej tesktury. Zadanie rozpocząłem od wykorzystania kodu dostepnego w instrukcji labolatoryjnej. Do projektu została skopiowana funkcja wczytujaća plik. Dodatkowo stworzyłem funkcję tekstura(plik) (kod także z instrukcji), która wczytuje teksturę na początku programu i przy każdej zmianie.

```
1  void tesktura(const char plik[]) {//plik to nazwa wczytywanego pliku
2    GLbyte* pBytes;
3    GLint ImWidth, ImHeight, ImComponents;
4    GLenum ImFormat;
5
6    pBytes = LoadTGAImage(plik, &ImWidth, &ImHeight, &ImComponents,
         &ImFormat);//wczytanie tekstury
7    glTexImage2D(GL_TEXTURE_2D, 0, ImComponents, ImWidth, ImHeight, 0, ImFormat,
         GL_UNSIGNED_BYTE, pBytes);//zdefiniowanie tekstury
8    free(pBytes);
9
10   glEnable(GL_CULL_FACE);//uruchomienie teksturowania jednostronnego
11   glCullFace(GL_FRONT);//uruchomienie teksturowania frontu
12   glEnable(GL_TEXTURE_2D);//uruchomienie tekstur
13   glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);//tryb teksturowania
14   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);//sposob
         nakladania tekstur
15 }
```

3 Program

Jako szkielet programu użyłem kodu z poprzedniego laboratorium. Dodałem wyżej opisaną funkcję do uruchomienia teksturowania i wczytania tekstur. Następne zmiany opiszę po kolei.

3.1 Zmienne globalne

```
//zmienne ogolne
typedef float point3[3];
typedef float point9[9];
int testedObject = 4; //rysowany obiekt
static int x_pos_old = 0;       // pozycje kursora myszy
static int delta_x = 0;
static int y_pos_old = 0;
static int delta_y = 0;
static GLfloat pix2angle;     // przelicznik pikseli na stopnie
const char piramida[] = "D1_t.tga";//tekstury
const char jajko[] = "P3_t.tga";
int dopasowanie = 0;//tryb dopasowanie tektury dla jajka


//zmienne obiektu
int n = 50;//ilosc punktow jajka
float scale = 3.0;//wielkosc obiektu
point3** points;//siatka punktow
point3** vectors;//wektory punktow powierzchni jajka

//zmienne obrotu obiektu
static GLint statusMiddle = 0;//stan srdkowego klawisza
float rViewer = 10;//R wokol punktu obserwowanego
static GLfloat viewer[] = { 0.0, 0.0, 10.0 };//pozycja punktu widzenia
static GLfloat azymuth = 0;    //katy obserwacji punktu obserwowanego
static GLfloat elevation = 0;
static GLint statusLeft = 0;       // stan klawisza lewego
static GLint statusRight = 0;    // stan klawisza prawego
```

Najpierw zmieniłem zmienne globalne używane w poprzednim zadaniu. Usunąłem niepotrzebne zmienne, oraz dodałem zmienne do obsługi wczytywania tekstur, oraz zmienną obsługi dopasowania tekstury. Każda zmienna jest opisana w kodzie.

3.2 Main()

```
void main(void)
{
  srand(time(NULL));
  points = new  point3 * [n + 1];//tablica punktow
  vectors = new point3 * [n + 1];//tablice
  for (int i = 0; i <= n; i++) {
    points[i] = new point3[n + 1];
    vectors[i] = new point3[n + 1];
  }

  cout << "Obsluga programu:\np - piramida\t - trojkat\nj - jajko\nlewy przycisk
      myszy - obrot\nsrodkowy przycisk myszy - zoom\nd - wlacz/wylacz dopasowanie
      tekstury" << endl;
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
  glutInitWindowSize(1000, 1000);
  glutCreateWindow("Teksturowanie");
  glutDisplayFunc(RenderScene);
  glutReshapeFunc(ChangeSize);
  MyInit();
  glutMouseFunc(Mouse);//"lapanie" akcji na przyciskach myszy
  glutMotionFunc(Motion);//"lapanie" ruchu myszki
  glutKeyboardFunc(keys);//"lapanie" akcji na klawiaturze
  glEnable(GL_DEPTH_TEST);
  glutMainLoop();
}
```

Najpierw alokowana jest tablica punktów jajka, oraz jego wektorów normalnych. Następnie wyświetlana w terminalu jest instrukcja obsługi programu. Na końcu jest blok funkcji z biblioteki GLUT

obsługujące funkcje rysowania, myszy, klawiatury i pozostałych implementowanych na zajęciach funkcjonalności. Wszystkie te funkcje były wcześniej użyte.

3.3 keys()

```
void keys(unsigned char key, int x, int y)
{
  if (key == 'c') {
    testedObject = 1;
  }
  if (key == 'd') {
    if (dopasowanie == 0) {
      dopasowanie = 1;
    }
    else {
      dopasowanie = 0;
    }
  }
  if (key == 'j') {
    tesktura(jajko);
    testedObject = 2;
  }
  if (key == 't') {
    tesktura(piramida);
    testedObject = 3;
  }
  if (key == 'p') {
    tesktura(piramida);
    testedObject = 4;
  }
  RenderScene();
}
```

Zmianom uległa funckja keys(), która zyskała obsługę, nowych przycisków dla trójkąta, piramidy i zmiany dopasowania.

3.4 renderScene()

```
void RenderScene(void)
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);//deklaracja buforow
  glLoadIdentity();
  gluLookAt(viewer[0], viewer[1], viewer[2], 0.0, 0.0, 0.0, 0.0, 1.0,
      0.0);//ustawienie pozycji punktu widzenia i pozycji punktu obserwowanego

  Axes();
  zadanie();

  glutSwapBuffers();//zmiana buforow i wyswietlanie
}
```

Funkcja renderująca scenę posiada wyświetlanie wygenerowanych obiektów, definicję punktu widzenia. Uruchamia takze funkcję zadanie, która generuje wyświetlane obiekty.

3.5 zadanie.txt

```
void zadanie() {
  if (statusMiddle == 1) {//obracanie obiektu
    elevation += 0.01 * delta_y * pix2angle;
    azymuth += 0.01 * delta_x * pix2angle;
  }

  if (statusRight == 1) {//zoom
    if (delta_y > 0) {
      rViewer += 0.2;
    }
    else {
      rViewer -= 0.2;
    }
  }

  viewer[0] = rViewer * cos(azymuth) * cos(elevation);//obliczenie pozycji punktu
      widzenia
  viewer[1] = rViewer * sin(elevation);
  viewer[2] = rViewer * sin(azymuth) * cos(elevation);


  switch (testedObject) {//wybranie obiektu do wyswietlania
  case 1:
    glutSolidTeapot(scale);
    break;
  case 2:
    egg();
    break;
  case 3:
    triangle();
    break;
  case 4:
    piramid();
    break;
  }
}
```

Funkcja zadanie obsługuje obrót kamery wokół obiektu, oraz przybliżanie. Uruchamia także funkcję generującą wybrany obiekt.

3.6 trojkat()

```
void triangle() {
  glBegin(GL_TRIANGLES);

  glNormal3f(0, 0, 1);
  glTexCoord2f(0.0f, 1.0f);
  glVertex3f(4, 0, 0);

  glNormal3f(0, 0, 1);
  glTexCoord2f(0.5f, 0.75f);
  glVertex3f(0, 6, 0);

  glNormal3f(0, 0, 1);
  glTexCoord2f(0.0f, 0.0f);
  glVertex3f(-4, 0, 0);

  glEnd();
}
```

Funkcja jest pierwszą próbą użycia tekstur. Generuje ona trojkąt i nakłada teksturę cegły. W celu naniesienia tekstury na kształ używam funkcji GLUT glTesCoord2f(), która łączy punkt na powierzchni tekstury z punktem w przestrzeni.

3.7 piramida()

```
void piramid() {

  float triangleHeight = 2 * sqrt(3);
  point3 vector = { 0, 0.5, 0.5 * sqrt(3) };
  float piramidHeight = triangleHeight / 2 * sqrt(3);

  glCullFace(GL_FRONT);
  glBegin(GL_POLYGON);//podstawa
  glNormal3f(0, -1, 0);
  glTexCoord2f(0.0f, 0.0f);
  glVertex3f(2, -1, 2);

  glNormal3f(0, -1, 0);
  glTexCoord2f(0.0f, 1.0f);
  glVertex3f(2, -1, -2);

  glNormal3f(0, -1, 0);
  glTexCoord2f(1.0f, 1.0f);
  glVertex3f(-2, -1, -2);

  glNormal3f(0, -1, 0);
  glTexCoord2f(1.0f, 0.0f);
  glVertex3f(-2, -1, 2);
  glEnd();


  glBegin(GL_TRIANGLES);//trojkaty prawy
  glNormal3f(vector[0], vector[1], -1 * vector[2]);
  glTexCoord2f(1.0f, 0.0f);
  glVertex3f(2, -1, -2);

  glNormal3f(vector[0], vector[1], -1 * vector[2]);
  glTexCoord2f(0.5f, triangleHeight);
  glVertex3f(0, piramidHeight, 0);

  glNormal3f(vector[0], vector[1], -1 * vector[2]);
  glTexCoord2f(0.0f, 0.0f);
  glVertex3f(-2, -1, -2);
  glEnd();


  glBegin(GL_TRIANGLES);//trojkaty lewy
  glNormal3f(vector[0], vector[1], vector[2]);
  glTexCoord2f(1.0f, 0.0f);
  glVertex3f(-2, -1, 2);

  glNormal3f(vector[0], vector[1], vector[2]);
  glTexCoord2f(0.5f, triangleHeight);
  glVertex3f(0, piramidHeight, 0);

  glNormal3f(vector[0], vector[1], vector[2]);
  glTexCoord2f(0.0f, 0.0f);
  glVertex3f(2, -1, 2);
  glEnd();


  glBegin(GL_TRIANGLES);//trojkaty przod
  glNormal3f(vector[2], vector[1], vector[0]);
  glTexCoord2f(1.0f, 0.0f);
  glVertex3f(2, -1, 2);

  glNormal3f(vector[2], vector[1], vector[0]);
  glTexCoord2f(0.5f, triangleHeight);
  glVertex3f(0, piramidHeight, 0);

  glNormal3f(vector[2], vector[1], vector[0]);
```

```
67    glTexCoord2f(0.0f, 0.0f);
68    glVertex3f(2, -1, -2);
69    glEnd();
70
71
72    glCullFace(GL_BACK);
73    glBegin(GL_TRIANGLES);//trojkaty tyl
74    glNormal3f(-1 * vector[2], vector[1], vector[0]);
75    glTexCoord2f(0.0f, 0.0f);
76    glVertex3f(-2, -1, 2);
77
78    glNormal3f(-1 * vector[2], vector[1], vector[0]);
79    glTexCoord2f(0.5f, triangleHeight);
80    glVertex3f(0, piramidHeight, 0);
81
82    glNormal3f(-1 * vector[2], vector[1], vector[0]);
83    glTexCoord2f(1.0f, 0.0f);
84    glVertex3f(-2, -1, -2);
85    glEnd();
86  }
```

Funkcja generuje piramidę. W tym celu najpierw obliczam potrzebne długości wektorów normlanych i potrzebne oległości w przestrzeni, aby piramida była jednolita. Następnie generuję podstawę, a potem każdą ścianę.

3.8 egg()

```
1  void egg() {
2    float u = 0, v = 0;
3    float diff = 1.0 / n; //n - liczba punktow na powierzchni jajka
4    glTranslated(0, (-(160 * pow(0.5, 4) - 320 * pow(0.5, 3) + 160 * pow(0.5, 2)) / 2)
         * (scale + 7) / 10, 0);//obnizenie  rodka  figury do centrum ukladu
         wspolrzednych
5
6    for (int i = 0; i <= n; i++) {//punkty powierzchni
7      v = 0;//obliczenie poteg w celu ulatwienia kodu
8      float u2 = pow(u, 2);
9      float u3 = pow(u, 3);
10     float u4 = pow(u, 4);
11     float u5 = pow(u, 5);
12
13     for (int ii = 0; ii <= n; ii++) {//obliczenie wspolrzednych punktow
14       points[i][ii][0] = ((-90 * u5 + 225 * u4 - 270 * u3 + 180 * u2 - 45 * u) *
             cos(M_PI * v)) * (scale + 7) / 10;
15       points[i][ii][1] = (160 * u4 - 320 * u3 + 160 * u2) * (scale + 7) / 10;
16       points[i][ii][2] = ((-90 * u5 + 225 * u4 - 270 * u3 + 180 * u2 - 45 * u) *
             sin(M_PI * v)) * (scale + 7) / 10;
17       v = v + diff;
18     }
19     u = u + diff;
20   }
21
22   u = 0;
23   for (int i = 0; i <= n; i++) {//wektory normalne
24     v = 0;//obliczenie poteg w celu ulatwienia kodu
25     float u2 = pow(u, 2);
26     float u3 = pow(u, 3);
27     float u4 = pow(u, 4);
28     float u5 = pow(u, 5);
29
30     for (int ii = 0; ii <= n; ii++) {
31       point9 vector;
32       vector[0] = (-450 * u4 + 900 * u3 - 810 * u2 + 360 * u - 45) * cos(M_PI * v);
33       vector[1] = M_PI * (90 * u5 - 225 * u4 + 270 * u3 - 180 * u2 + 45) * sin(M_PI
             * v);
34       vector[2] = 640 * u3 - 960 * u2 + 320 * u;
```

```
35        vector[3] = 0;
36        vector[4] = (-450 * u4 + 900 * u3 - 810 * u2 + 360 * u - 45) * sin(M_PI * v);
37        vector[5] = -1 * M_PI * (90 * u5 - 225 * u4 + 270 * u3 - 180 * u2 + 45) *
              cos(M_PI * v);
38
39        vector[6] = vector[2] * vector[5] - vector[4] * vector[3];//wektory
40        vector[7] = vector[4] * vector[1] - vector[0] * vector[5];
41        vector[8] = vector[0] * vector[3] - vector[2] * vector[1];
42        float vectorSize = sqrt(pow(vector[6], 2) + pow(vector[7], 2) + pow(vector[8],
              2));//normalizacja wektora
43        if (vectorSize == 0) {
44          vectorSize = 1;
45        }
46
47        vectors[i][ii][0] = vector[6] / vectorSize;
48        vectors[i][ii][1] = vector[7] / vectorSize;
49        vectors[i][ii][2] = vector[8] / vectorSize;
50
51        v = v + diff;
52      }
53      u = u + diff;
54    }
55
56    u = 0;
57    for (int i = 0; i < n; i++) {//rysowanie
58      v = 0;
59      for (int ii = 0; ii < n; ii++) {
60        float pozycja[4];
61        if (dopasowanie == 0) {
62          pozycja[0] = u;
63          pozycja[1] = u + diff;
64          pozycja[2] = v;
65          pozycja[3] = v + diff;
66        }
67        else {
68          pozycja[0] = sin(M_PI * u);
69          pozycja[1] = sin(M_PI * (u + diff));
70          pozycja[2] = v;
71          pozycja[3] = v + diff;
72        }
73
74        if (i < n / 2) {
75          glCullFace(GL_BACK);
76        }
77        else {
78          glCullFace(GL_FRONT);
79        }
80        glBegin(GL_TRIANGLES);//rysowanie pierwszego trojkata
81        glNormal3fv(vectors[i][ii]);
82        glTexCoord2f(pozycja[0], pozycja[2]);
83        glVertex3f(points[i][ii][0], points[i][ii][1], points[i][ii][2]);
84        glNormal3fv(vectors[i + 1][ii]);
85        glTexCoord2f(pozycja[1], pozycja[2]);
86        glVertex3f(points[i + 1][ii][0], points[i + 1][ii][1], points[i + 1][ii][2]);
87        glNormal3fv(vectors[i + 1][ii + 1]);
88        glTexCoord2f(pozycja[1], pozycja[3]);
89        glVertex3f(points[i + 1][ii + 1][0], points[i + 1][ii + 1][1], points[i +
              1][ii + 1][2]);
90        glEnd();
91
92
93        if (i < n / 2) {
94          glCullFace(GL_FRONT);
95        }
96        else {
97          glCullFace(GL_BACK);
98        }
99        glBegin(GL_TRIANGLES);//rysowanie drugiego trojkata
```

```
100         glNormal3fv(vectors[i][ii]);
101         glTexCoord2f(pozycja[0], pozycja[2]);
102         glVertex3f(points[i][ii][0], points[i][ii][1], points[i][ii][2]);
103         glNormal3fv(vectors[i][ii + 1]);
104         glTexCoord2f(pozycja[0], pozycja[3]);
105         glVertex3f(points[i][ii + 1][0], points[i][ii + 1][1], points[i][ii + 1][2]);
106         glNormal3fv(vectors[i + 1][ii + 1]);
107         glTexCoord2f(pozycja[1], pozycja[3]);
108         glVertex3f(points[i + 1][ii + 1][0], points[i + 1][ii + 1][1], points[i +
                1][ii + 1][2]);
109         glEnd();
110         v = v + diff;
111       }
112       u = u + diff;
113     }
114 }
```

Funkcja generuje jajko. Większość kodu została objaśniona w poprzednich sprawozdaniach. Zmieniłem zawartość petli odpowiedzialnej za rysowanie jajka(linijki 57-113). W odpowiednich miejscach dodałem funkcję glTExCoord, a także tablicę zawierającą współrzędne dla tekstury. dodałem także dwa warunki if, które zapewniają prawidłowe rysowanie tekstur an powierzchni jajka.

Warunek w 61 linijce uruchamia dopasowanie tekstury do jajka. Wybrana tekstura na obiekcie jest rozciągnięta, tak też to się obrazuje po uruchomieniu programu. W celu dopasowania tej tekstury należy przycisnąć przycisk d. W takim przypadku tesktura zostanie dopasowana do powierzchni jajka, dzięki czemu bedzie widać kształty na jej powierzchni.

4 Wnioski

Używanie tekstur nie sprawia problemów. W implementacji najważniejsze jest odpowiednie przypisanie punktów tekstury do punktów przestrzeni, oraz odpowiednie sterowanie wyświetlaniem na ścianie. Problemy sprawiły także drobne niedociągnięcia w liczeniu wektorów normalnych, ale i to nie sprawiło większych kłopotów.