Politechnika Wrocławska

Sprawozdanie 5

Ćwiczenie 5.Teksturowanie

Krzysztof Zalewa

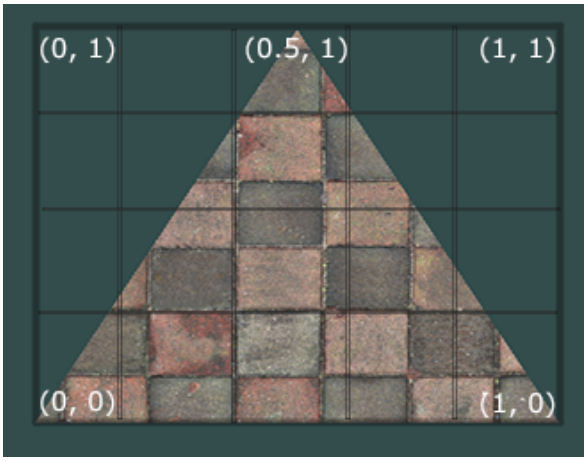
16.12.2024

Spis treści

1 Wstęp teoretyczny

1.1 Tekstury

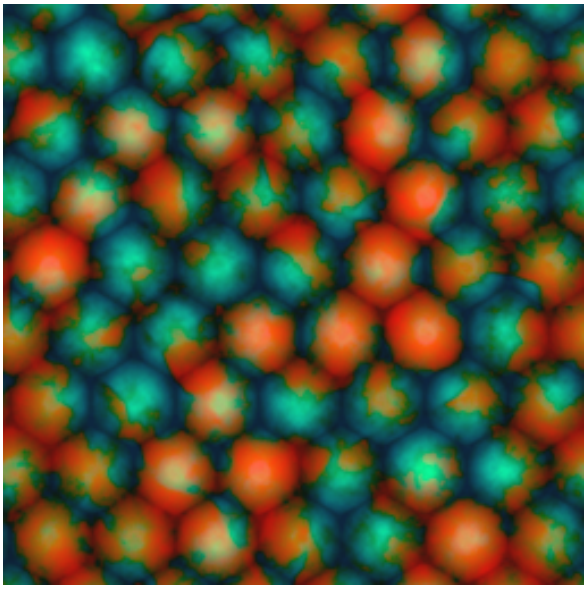
Tekstura w OpenGL to obiekt który zawiera jeden lub więcej obrazów. Obiekt ten może zostać wykożystany w shaderach lub do renderowania obiektu. W OpenGL niezależnie od wielkości teksutry jej kordynaty zawsze są w zakresie 0 - 1.



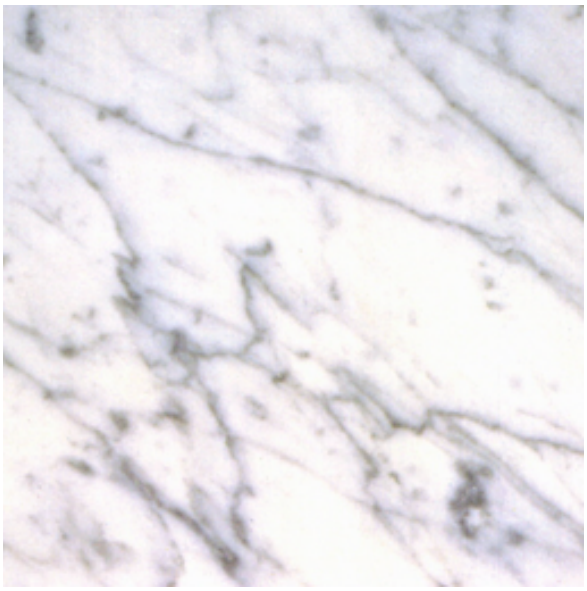
Rysunek 1: Przykład kordynatów w OpenGL

1.2 Użyte tekstury

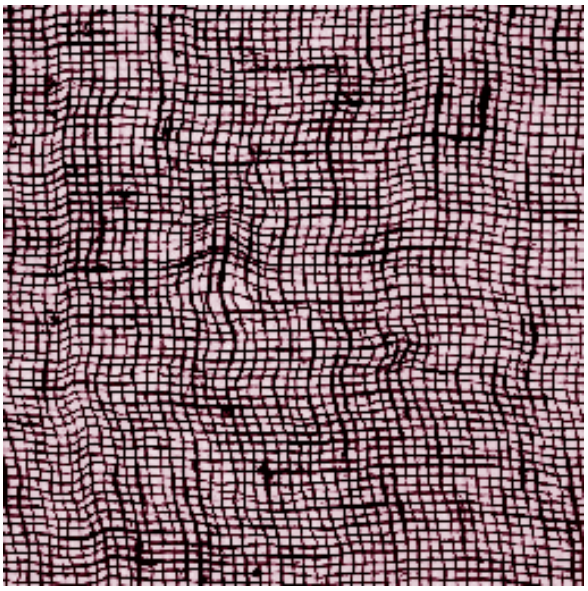
Użyte tekstury pochodzą ze strony Dra. Gniewkowskiego(Źródła 2.)
Tekstura 2 - To przykładowa tekstura
Tekstura 3 - To plik **M1_t.tga** z archwium tekstur
Tekstura 4 - To plik **D8_t.tga** z archwium tekstur



Rysunek 2: Tekstura nr 1



Rysunek 3: Tekstura nr 2



Rysunek 4: Tekstura nr 2

1.3 Funkcje w OpenGL

`glGenTextures(1,&textureIDs[texID])` - Generuje tablice identyfikatorów tekstur
`glBindTexture(GL_TEXTURE_2D,textureIDs[texID])` - Wybiera teksturę o nr texID i przypisuje ją jako obecnie wybraną. Pozwala to na wykonywanie działań na tej teksturze (Przypisywanie konkretnej grafiki, rysowanie na obiekcie itd.)
`glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR)` - Definuje zachowanie tekstury gdy zostanie zrenderowana w skali mniejszej niż jej orginalny rozmiar. W tym przypadku **`GL_LINEAR`** oznacza że tekstura jest interpolowana liniowo
`glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR)` - Definuje zachowanie tekstury gdy zostanie zrenderowana w skali większej niż jej orginalny rozmiar. Podobnie jak poprzednio **`GL_LINEAR`** oznacza że tekstura jest interpolowana liniowo
`glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,GL_REPEAT)` - Ustawia tryb zapętlania tekstury w osi S (**`GL_TEXTURE_WRAP_S`**) oś S to oś pozioma tekstury która odpowiada osi X obiektu. Opcja **`GL_REPEAT`** oznacza że tekstura się powtórzy. Np. Jeżeli trzeba wyświetlić teksturę w w punkcie **`S = 1.5`** to wyświetlona zostanie tekstura w punkcie **`S = 0.5`**
`glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,GL_REPEAT)` - Ustawia tryb zapętlania tekstury w osi S (**`GL_TEXTURE_WRAP_T`**) oś T to oś pionowa tekstury która odpowiada osi X obiektu. Opcja **`GL_REPEAT`** oznacza że tekstura się powtórzy. Np. Jeżeli trzeba wyświetlić teksturę w w punkcie **`T = 1.5`** to wyświetlona zostanie tekstura w punkcie $\mathbf{T = 0.5}$
`glTexImage2D(GL_TEXTURE_2D,0,GL_RGB,width,height,0,GL_RGB,GL_UNSIGNED_BYTE,data)` - Do wybranej tekstury przypisuje tablicę typu **`GL_UNSIGNED_BYTE`** (Wczytany obraz). Obraz ten ma rozmiary width*height i jego kolory zapisane są w formacie RGB

2 Zadanie laboratoryjne

2.1 Treść zadania

W ramach zadania należało do poprzednio stworzonego programu dodać możliwość teksturowania obiektów. Powinna być możliwość wyświetlenia trzech tekstur głęboko strukturalnej, pośredniej i bez ustrukturyzowania. Program nie musi rysować obiektów za pomocą punktów i linii.

2.2 Opis działania programu

Zgodnie z treścią zadania program rysuje 2 obiekty. Domyślnie jajko i czajnik rysowane są w kolorze białym. Po wciśnięciu klawisza F3 lub F4 zmieniana jest tekstura. Wyświetlone obiekty można obracać za pomocą myszki (Przycisk musi być wciśnięty i zytrzymany). Program implementuje dwa światła niebieskie i czerwone.
Kontrola obrotu:
F1 - Tryb obrotu obiektu
F2 - Tryb obrotu kamery
F3 - Następna tekstura
F4 - Poprzednia tekstura
ESC - Powrót do menu (okno konsolowe)
Ruch myszy w osi X - Obrót kamery w osi X
Ruch myszy w osi Y - Obrót kamery w osi Y
Scroll up - Przybliżenie obiektu
Scroll down - Oddalenie obiektu

2.3 Kod programu

.

```
#include <windows.h>
#include <iostream>
#include <GL/glu.h>
#include <vector>
#include <math.h>
#define FREEGLUT_STATIC
#include <GL/freeglut.h>

#define STB_IMAGE_IMPLEMENTATION
#include "stb_image.h"

#include "Egg.hpp"
#include "Light.hpp"
using namespace std;
HWND consoleWindow;
HWND glutWindow;

u_int textureIDs[4];
int currentTex = 0;
```

```
20 GLfloat deg = 0;
21 int sx =0,sy = 0,sz = 0;
22 bool drawTeapot = true;
23 int moveMode = 0;
24 float totalRotationX = 0.0f,totalRotationY = 0.0f,totalRotationZ =
   ↪ 0.0f;
25 float pix2angle,theta = 0.0f,phi = 0.0f;
26 int radius = 6,lastX = 0,lastY = 0;
27 float cameraRotationX = radius * cosf((theta*(M_PI/180))) *
   ↪ cosf((phi*(M_PI/180)));
28 float cameraRotationY = radius * sinf((phi*(M_PI/180)));
29 float cameraRotationZ = radius * sinf((theta*(M_PI/180))) *
   ↪ cosf((phi*(M_PI/180)));
30 Light light1(GL_LIGHT0);
31 GLfloat light1Radius = 10;
32 unsigned char *texture1;
33 unsigned char *texture2;
34 unsigned char *texture3;
35
36 Egg egg(100);
37 void toggleFocusToConsole() {
38     ShowWindow(glutWindow, SW_HIDE);
39     ShowWindow(consoleWindow, SW_SHOWNORMAL);
40     SetForegroundWindow(consoleWindow);
41 }
42
43 void toggleFocusToGLUT() {
44     ShowWindow(consoleWindow, SW_HIDE);
45     ShowWindow(glutWindow, SW_SHOWNORMAL);
46     SetForegroundWindow(glutWindow);
47 }
48 void reset_rotation(){
49     theta = 0.0f;
50     phi = 0.0f;
51     lastX = 0;
52     lastY = 0;
53     cameraRotationX = radius * cosf((theta*(M_PI/180.0f))) *
       ↪ cosf((phi*(M_PI/180.0f)));
54     cameraRotationY = radius * sinf((phi*(M_PI/180.0f)));
55     cameraRotationZ = radius * sinf((theta*(M_PI/180.0f))) *
       ↪ cosf((phi*(M_PI/180.0f)));
56 }
57 void renderString(const unsigned char* string){
58
59     glDisable(GL_LIGHTING);
60     glColor3f(1.0f, 1.0f, 1.0f);
61      glRasterPos2f(0, 0);
62     while (*string) {
```

```
63          glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_10, *string);
64          string++;
65      }
66      glEnable(GL_LIGHTING);
67  }
68  void printControls(){
69      cout<<"============================\n";
70      cout<<"F1 - Tryb obrotu obiektu\n";
71      cout<<"F2 - Tryb obrotu kamery\n";
72      cout<<"F3 - Nastepna tekstura\n";
73      cout<<"F4 - Poprzednia tekstura\n";
74      cout<<"ESC - Powrot do menu (okno konsolowe)\n";
75      cout<<"Nalezy nacisnac i przytrzymac PPM\n";
76      cout<<"Ruch myszy w osi X - Obrot osi X\n";
77      cout<<"Ruch myszy w osi Y - Obrot osi Y\n";
78      cout<<"Scroll up - Przybilizenie obiektu\n";
79      cout<<"Scroll down - Oddalenie obiektu\n";
80      cout<<"Nacisnij Enter zeby kontynuowac\n"<<flush;
81      cin.get();
82      cin.get();
83  }
84  void axis(){
85      glDisable(GL_LIGHTING);
86      glBegin(GL_LINES);
87
88      glColor3f(1.0, 0.0, 0.0);
89      glVertex3f(-5.0, 0.0, 0.0);
90      glVertex3f(5.0, 0.0, 0.0);
91
92      glColor3f(0.0, 1.0, 0.0);
93      glVertex3f(0.0, -5.0, 0.0);
94      glVertex3f(0.0, 5.0, 0.0);
95
96      glColor3f(0.0, 0.0, 1.0);
97      glVertex3f(0.0, 0.0, -5.0);
98      glVertex3f(0.0, 0.0, 5.0);
99
100     glEnd();
101     glEnable(GL_LIGHTING);
102 }
103 void printOptions();
104 void menu();
105 void printOptions(){
106     int density = egg.getDensity();
107     bool color = egg.getColor();
108     float scale = egg.getScale();
109     float pointSize = egg.getPointSize();
110     cout<<"============================\n";
```

```
111     cout<<"1.Skala obiektow: "<<scale<<"\n";
112     cout<<"2.Ilosc punktow: "<<density<<"\n";
113     cout<<"3.Promien kamery: "<<radius<<"\n";
114     cout<<"4.Wroc do menu"<<"\n";
115     cout<<"> ";
116     int x;
117     cin>>x;
118     switch (x){
119     case 1:
120         cout<<"Nowa skala\n";
121         cout<<"> ";
122         cin>>scale;
123         egg.setScale(scale);
124         printOptions();
125         break;
126     case 2:
127         cout<<"Nowa gestosc\n";
128         cout<<"> ";
129         cin>>density;
130         egg.setDensity(density);
131         printOptions();
132         break;
133     case 3:
134         cout<<"Nowy promien kamery\n";
135         cout<<"> ";
136         cin>>radius;
137         printOptions();
138         break;
139     case 4:
140         menu();
141         break;
142     }
143 }
144 void menu(){
145     toggleFocusToConsole();
146     reset_rotation();
147     cout<<"============================\n";
148     cout<<"1. Narysuj czajnik\n";
149     cout<<"2. Narysuj jajko (trojkaty) \n";
150     cout<<"3. Opcje\n";
151     cout<<"4. Kontrola\n";
152     cout<<"5. Zakoncz program\n";
153     cout<<"> ";
154     int x;
155     cin>>x;
156     switch (x){
157     case 1:
158         drawTeapot = true;
```

```
159             break;
160         case 2:
161             drawTeapot = false;
162             break;
163         case 3:
164             printOptions();
165             break;
166         case 4:
167             printControls();
168             menu();
169             break;
170         case 5:
171             exit(0);
172             break;
173         default:
174             cout<<"Podano nieporawny znak\n";
175             menu();
176             break;
177         }
178         toggleFocusToGLUT();
179         glutPostRedisplay();
180     }
181     void specialKey(int key,int x,int y){
182         switch (key){
183         //F1 - Ruch obiektu
184         case GLUT_KEY_F1:
185             moveMode = 0;
186             break;
187         //F2 - Ruch kamery
188         case GLUT_KEY_F2:
189             moveMode = 1;
190             break;
191         //F3 - Następna tekstura
192         case GLUT_KEY_F3:
193             currentTex++;
194             if(currentTex>3){
195                 currentTex = 0;
196             }
197             break;
198         //F3 - Poprzednia tekstura
199         case GLUT_KEY_F4:
200             currentTex--;
201             if(currentTex<0){
202                 currentTex = 3;
203             }
204             break;
205         default:
206             break;
```

```
207     }
208 }
209 void normalKey(u_char key,int x,int y){
210     switch (key)
211     {
212     case 27:
213         menu();
214         break;
215     default:
216         break;
217     }
218     if (sx == 0 && sy == 0 && sz == 0) {
219         glutIdleFunc(nullptr);
220     }
221 }
222 
223 void mouse(int x, int y){
224     float dY = y - lastY;
225     lastY = y;
226     float dX = x - lastX;
227     lastX = x;
228     theta += dX * pix2angle;
229     phi += dY * pix2angle;
230     if (phi > 89.0f) {phi = 89.0f;}
231     if (phi < -89.0f) {phi = -89.0f;}
232     switch(moveMode){
233         case 0:
234             totalRotationX += dY;
235             totalRotationY += dX;
236             totalRotationZ += atan2f(dY,dX);
237             break;
238         case 1:
239             cameraRotationX = radius * cosf((theta*(M_PI/180.0f))) *
                ↪  cosf((phi*(M_PI/180.0f)));
240             cameraRotationY = radius * sinf((phi*(M_PI/180.0f)));
241             cameraRotationZ = radius * sinf((theta*(M_PI/180.0f))) *
                ↪  cosf((phi*(M_PI/180.0f)));
242             break;
243     }
244     lastX = x;
245     lastY = y;
246     glutPostRedisplay();
247 }
248 void mouseWheel(int button, int dir, int x, int y){
249     if (dir > 0){
250         radius -= 1;
251     }else{
252         radius += 1;
```

```
253         }
254         if(radius<=1){
255             radius=1;
256         }
257         glutPostRedisplay();
258     }
259     void display() {
260         GLfloat lPos1[] = {0,0,10,1};//x,y,z,czy światło jest odległe
261         GLfloat col[] = {1,0,0,1};
262         renderString((const unsigned char*)"Test");
263         glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
264 
265         glLoadIdentity();
266         gluLookAt(cameraRotationX,cameraRotationY,cameraRotationZ,0,0,0,0
            ↪  ,1,0);//Ustawienie kamery
267         light1.setPosition(lPos1);
268         glEnable(GL_COLOR_MATERIAL);
269         axis();
270         glRotatef(totalRotationX, 1.0f, 0.0f, 0.0f);
271         glRotatef(totalRotationY, 0.0f, 1.0f, 0.0f);
272         glRotatef(totalRotationZ, 0.0f, 0.0f, 1.0f);
273 
274         glBindTexture(GL_TEXTURE_2D, textureIDs[currentTex]);
275         if(drawTeapot){
276             glColor3f(1.0, 1.0, 1.0);
277             glutSolidTeapot(1);
278         }else{
279             egg.draw();
280         }
281         glutSwapBuffers();
282     }
283     void loadTexture(const char* fileName,int texID){
284         glGenTextures(1, &textureIDs[texID]);
285         glBindTexture(GL_TEXTURE_2D, textureIDs[texID]);
286         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
287         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
288         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
289         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
290         int width,height,nrChannels;
291         unsigned char *data = stbi_load(fileName, &width, &height,
            ↪  &nrChannels, 0);
292         if (data){
293             glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0,
                ↪  GL_RGB, GL_UNSIGNED_BYTE, data);
294         }
295         else{
296             cout << "Failed to load texture!" <<stbi_failure_reason()<<
                ↪  endl;
```

```
297             system("pause");
298             exit(1);
299         }
300         stbi_image_free(data);
301         egg.setTextureSize(height,width);
302     }
303     void Init() {
304         pix2angle = 360.0/800;
305         glEnable(GL_DEPTH_TEST); //bez tego frontalna sciana nadpisuje
            ↪  tylnią
306         glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
307         glMatrixMode(GL_PROJECTION);
308         // glLoadIdentity();
309         glFrustum(-1,1,-1,1,2,20);
310         glMatrixMode(GL_MODELVIEW);
311         // Ustawia kierunek frontowych ścianek jako przeciwny do ruchu
            ↪  wskazówek zegara
312         glFrontFace(GL_CW);
313         // Włącza culling, czyli pomijanie tylnych ścianek
314         glEnable(GL_CULL_FACE);
315         // Ustawia pomijanie tylnych ścianek
316         glCullFace(GL_BACK);
317         // Kolor stały
318         light1.setColor(1.0,1.0,1.0);
319         light1.initLight();
320         //Drugie światło
321         glShadeModel(GL_SMOOTH);
322         glEnable(GL_LIGHTING); //Włączenie oświetlenia
323         glEnable(GL_LIGHT0); //Dodanie źródła światła
324 
325         glEnable(GL_TEXTURE_2D); //Włącza teksturowanie
326         glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
327 
328         loadTexture("../tekstura1.tga",1);
329         loadTexture("../tekstura2.tga",2);
330         loadTexture("../tekstura3.tga",3);
331     }
332     int main(int argc, char** argv){
333         consoleWindow = GetConsoleWindow();
334         glutInit(&argc, argv);
335         glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
336         glutInitWindowSize(800,800);
337         glutCreateWindow("Lab 5 - Tekstury");
338         glutWindow = FindWindowW(NULL,L"Lab 5 - Tekstury");
339         Init();
340         glutDisplayFunc(display);
341         glutIdleFunc(nullptr);
342         glutKeyboardFunc(normalKey);
```

```
343     glutSpecialFunc(specialKey);
344     glutMotionFunc(mouse);
345     glutMouseWheelFunc(mouseWheel);
346     menu();
347     glutMainLoop();
348     return 0;
349 }
```

Fragment kodu 1: Fragment kodu z programu

.

```
1  #include <math.h>
2  #include <GL/glu.h>
3  #include <iostream>
4  #define FREEGLUT_STATIC
5  #include <GL/freeglut.h>
6  #include "Egg.hpp"
7  using namespace std;
8
9  float Egg::randFloat(){
10     return (float)rand()/(float)(RAND_MAX);
11 }
12 Egg::Egg(int density ) : density(density){
13     pointsMatrix.resize(density,vector<pointsRgb>(density));
14 }
15 vector<vector<pointsRgb>> Egg::getPointsMatrix(){
16     return pointsMatrix;
17 }
18 point Egg::generateNormalVect(int u,int v){
19     float x_u = (-450*pow(u,4) + 900*pow(u,3) - 810*pow(u,2) + 360*u
       ↪  - 45) * cos(M_PI*v);
20     float 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);
21     float y_u = 640*pow(u,3) - 960*pow(u,2) + 320*u;
22     float y_v = 0;
23     float z_u = (-450*pow(u,4) + 900*pow(u,3) - 810*pow(u,2) + 360*u
       ↪  - 45) * sin(M_PI*v);
24     float 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);
25     point newPoint;
26     newPoint.x = y_u * z_v - z_u * y_v;
27     newPoint.y = z_u * x_v - x_u * z_v;
28     newPoint.z = x_u * y_v - y_u * x_v;
29     float length = sqrt(newPoint.x*newPoint.x + newPoint.y*newPoint.y
       ↪  + newPoint.z*newPoint.z);
30     newPoint.x /= length;
31     newPoint.y /= length;
32     newPoint.z /= length;
```

```
33      return newPoint;
34  }
35  void Egg::setTextureSize(int newHeight,int newWidth){
36      height = newHeight;
37      width = newWidth;
38      generateMatrix();
39  }
40  void Egg::generateMatrix(){
41      for(int u=0;u<(density);u++){
42          float _u = 0.5/((float)density-1);
43          _u *= u;
44          for(int v=0;v<density;v++){
45              float _v = v/((float)density);
46              _v *= 2.0f;
47              pointsMatrix[u][v].x = scale*((-90*pow(_u,5) +
                ↪  225*pow(_u,4) - 270*pow(_u,3) + 180*pow(_u,2) -
                ↪  45*_u) * cos(M_PI*_v));
48              pointsMatrix[u][v].y = scale*(160*pow(_u,4) -
                ↪  320*pow(_u,3) + 160 * pow(_u,2) - 5);
49              pointsMatrix[u][v].z = scale*((-90*pow(_u,5) +
                ↪  225*pow(_u,4) - 270*pow(_u,3) + 180*pow(_u,2) -
                ↪  45*_u) * sin(M_PI*_v));
50              //Białe jajko
51              pointsMatrix[u][v].r = 1.0f;
52              pointsMatrix[u][v].g = 1.0f;
53              pointsMatrix[u][v].b = 1.0f;
54              point newPoint = generateNormalVect(u,v);
55              pointsMatrix[u][v].nx = newPoint.x;
56              pointsMatrix[u][v].ny = newPoint.y;
57              pointsMatrix[u][v].nz = newPoint.z;
58              pointsMatrix[u][v].u = ((float)u/(density));
59              pointsMatrix[u][v].v = 2.0f * abs(((float)v / density) -
                ↪  0.5f);
60          }
61      }
62  }
63  void Egg::initMaterial(){
64      float mat_ambient[4] = {0.3f, 0.3f, 0.3f, 1.0f};
65      float mat_diffuse[4] = {0.6f, 0.3f, 0.3f, 1.0f};
66      float mat_specular[4] = {1.0f, 1.0f, 1.0f, 1.0f};
67      float mat_shininess = 10.0f;
68      glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, mat_ambient);
69      glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, mat_diffuse);
70      glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, mat_specular);
71      glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, mat_shininess);
72  }
73  void Egg::draw(){
74      glBegin(GL_TRIANGLES);
```

```
75      for(int u=0;u<density-1;u++){
76          //Obecnie trójkąty są CCW
77          if(u==0){
78              for(int v=0;v<density;v++){
79                  int nextV = (v + 1) % density;
80                  glTexCoord2f(pointsMatrix[u+1][nextV].u,pointsMatrix[
                        u+1][nextV].v);
81                  glColor3f(pointsMatrix[u+1][nextV].r,pointsMatrix[u+1
                        ][nextV].g,pointsMatrix[u+1][nextV].b);
82                  glVertex3f(pointsMatrix[u+1][nextV].x,pointsMatrix[u+
                        1][nextV].y,pointsMatrix[u+1][nextV].z);
83
84                  glTexCoord2f(pointsMatrix[u+1][v].u,pointsMatrix[u+1]
                        [v].v);
85                  glColor3f(pointsMatrix[u+1][v].r,pointsMatrix[u+1][v]
                        .g,pointsMatrix[u+1][v].b);
86                  glVertex3f(pointsMatrix[u+1][v].x,pointsMatrix[u+1][v
                        ].y,pointsMatrix[u+1][v].z);
87
88                  glTexCoord2f(pointsMatrix[u][nextV].u,pointsMatrix[u]
                        [nextV].v);
89                  glColor3f(pointsMatrix[u][0].r,pointsMatrix[u][0].g,p
                        ointsMatrix[u][0].b);
90                  glVertex3f(pointsMatrix[u][0].x,pointsMatrix[u][0].y,
                        pointsMatrix[u][0].z);
91
92              }
93              continue;
94          }
95          if(u==density-2){
96              for(int v=0;v<density;v++){
97                  int nextV = (v + 1) % density;
98                  glTexCoord2f(pointsMatrix[u+1][nextV].u,pointsMatrix[
                        u+1][nextV].v);
99                  glColor3f(pointsMatrix[u+1][0].r,pointsMatrix[u+1][0]
                        .g,pointsMatrix[u+1][0].b);
100                 glVertex3f(pointsMatrix[u+1][0].x,pointsMatrix[u+1][0
                        ].y,pointsMatrix[u+1][0].z);
101
102                 glTexCoord2f(pointsMatrix[u][v].u,pointsMatrix[u][v].
                        v);
103                 glColor3f(pointsMatrix[u][v].r,pointsMatrix[u][v].g,p
                        ointsMatrix[u][v].b);
104                 glVertex3f(pointsMatrix[u][v].x,pointsMatrix[u][v].y,
                        pointsMatrix[u][v].z);
105
106                 glTexCoord2f(pointsMatrix[u][nextV].u,pointsMatrix[u]
                        [nextV].v);
```

```
107                 glColor3f(pointsMatrix[u][nextV].r,pointsMatrix[u][ne
                    ↪  xtV].g,pointsMatrix[u][nextV].b);
108                 glVertex3f(pointsMatrix[u][nextV].x,pointsMatrix[u][n
                    ↪  extV].y,pointsMatrix[u][nextV].z);
109             }
110             break;
111         }
112         for(int v=0;v<density;v++){
113             int nextV = (v + 1) % density;
114             //Pierwszy trójkąt
115             glTexCoord2f(pointsMatrix[u][v].u,pointsMatrix[u][v].v);
116             glColor3f(pointsMatrix[u][v].r,pointsMatrix[u][v].g,point
                ↪  sMatrix[u][v].b);
117             glVertex3f(pointsMatrix[u][v].x,pointsMatrix[u][v].y,poin
                ↪  tsMatrix[u][v].z);
118             if(nextV!=0){
119                 glTexCoord2f(pointsMatrix[u+1][nextV].u,pointsMatrix[
                    ↪  u+1][nextV].v);
120             }else{
121                 glTexCoord2f(pointsMatrix[u+1][nextV].u,1);
122             }
123             glColor3f(pointsMatrix[u+1][nextV].r,pointsMatrix[u+1][ne
                ↪  xtV].g,pointsMatrix[u+1][nextV].b);
124             glVertex3f(pointsMatrix[u+1][nextV].x,
                ↪  pointsMatrix[u+1][nextV].y,
                ↪  pointsMatrix[u+1][nextV].z);
125             glTexCoord2f(pointsMatrix[u+1][v].u,pointsMatrix[u+1][v].
                ↪  v);
126             glColor3f(pointsMatrix[u+1][v].r,pointsMatrix[u+1][v].g,p
                ↪  ointsMatrix[u+1][v].b);
127             glVertex3f(pointsMatrix[u+1][v].x,
                ↪  pointsMatrix[u+1][v].y, pointsMatrix[u+1][v].z);
128             //Drugi trójkąt
129             if(nextV!=0){
130                 glTexCoord2f(pointsMatrix[u+1][nextV].u,pointsMatrix[
                    ↪  u+1][nextV].v);
131             }else{
132                 glTexCoord2f(pointsMatrix[u+1][nextV].u,1);
133             }
134             glColor3f(pointsMatrix[u+1][nextV].r,pointsMatrix[u+1][ne
                ↪  xtV].g,pointsMatrix[u+1][nextV].b);
135             glVertex3f(pointsMatrix[u+1][nextV].x,
                ↪  pointsMatrix[u+1][nextV].y,
                ↪  pointsMatrix[u+1][nextV].z);
136             glTexCoord2f(pointsMatrix[u][v].u,pointsMatrix[u][v].v);
137             glColor3f(pointsMatrix[u][v].r,pointsMatrix[u][v].g,point
                ↪  sMatrix[u][v].b);
```

```
138                 glVertex3f(pointsMatrix[u][v].x,pointsMatrix[u][v].y,poin
                    ↪  tsMatrix[u][v].z);
139                 if(nextV!=0){
140                     glTexCoord2f(pointsMatrix[u][nextV].u,pointsMatrix[u]
                        ↪  [nextV].v);
141                 }else{
142                     glTexCoord2f(pointsMatrix[u][nextV].u,1);
143                 }
144                 glColor3f(pointsMatrix[u][nextV].r,pointsMatrix[u][nextV]
                    ↪  .g,pointsMatrix[u][nextV].b);
145                 glVertex3f(pointsMatrix[u][nextV].x,
                    ↪  pointsMatrix[u][nextV].y, pointsMatrix[u][nextV].z);
                    ↪
146             }
147         }
148         glEnd();
149     }
150     //Setters
151     void Egg::setDensity(int newDensity){
152         density = newDensity;
153         pointsMatrix.resize(density,vector<pointsRgb>(density));
154         generateMatrix();
155     }
156     void Egg::setColor(float newColor){color = newColor;}
157     void Egg::setScale(float newScale){scale = newScale;}
158     void Egg::setPointSize(float newPointSize){pointSize = newPointSize;}
159     //Getters
160     int Egg::getDensity(){return density;}
161     float Egg::getColor(){return color;}
162     float Egg::getScale(){return scale;}
163     float Egg::getPointSize(){return pointSize;}
164     Egg::~Egg(){
165
166     }
```

Fragment kodu 2: Kod Egg.cpp

.

```
1   #include <GL/glu.h>
2   #include <math.h>
3   #define FREEGLUT_STATIC
4   #include <GL/freeglut.h>
5   #include "Light.hpp"
6   using namespace std;
7
8   void Light::initLight(){
9       glLightfv(lightID, GL_AMBIENT, light_ambient);
10      glLightfv(lightID, GL_DIFFUSE, light_diffuse);
```

```
11      glLightfv(lightID, GL_SPECULAR, light_specular);
12      glLightf(lightID, GL_CONSTANT_ATTENUATION, att_constant);
13      glLightf(lightID, GL_LINEAR_ATTENUATION, att_linear);
14      glLightf(lightID, GL_QUADRATIC_ATTENUATION, att_quadratic);
15  }
16  Light::Light(GLenum newLightID){
17      lightID = newLightID;
18  }
19  void Light::setPosition(GLfloat lPos[]){
20      glLightfv(lightID,GL_POSITION,lPos);
21  }
22  void Light::setColor(float r,float g,float b){
23      light_ambient[0] = r;
24      light_ambient[1] = g;
25      light_ambient[2] = b;
26      light_diffuse[0] = r;
27      light_diffuse[1] = g;
28      light_diffuse[2] = b;
29      light_specular[0] = r;
30      light_specular[1] = g;
31      light_specular[2] = b;
32  }
```

Fragment kodu 3: Kod Light.cpp

3 Wnioski

Na zajęciach nie udało się dokończyć zadania. Po pracy w domu program działa poprawnie.

4 Źródła

1. https://learnopengl.com/Getting-started/Textures
2. https://gniewkowski.wroclaw.pl/gk/
3. https://www.khronos.org/opengl/wiki/texture