



Politechnika Wrocławska

Sprawozdanie 2

Ćwiczenie 4. Oświetlenie scen

Krzysztof Zalewa

9.12.2024

Spis treści

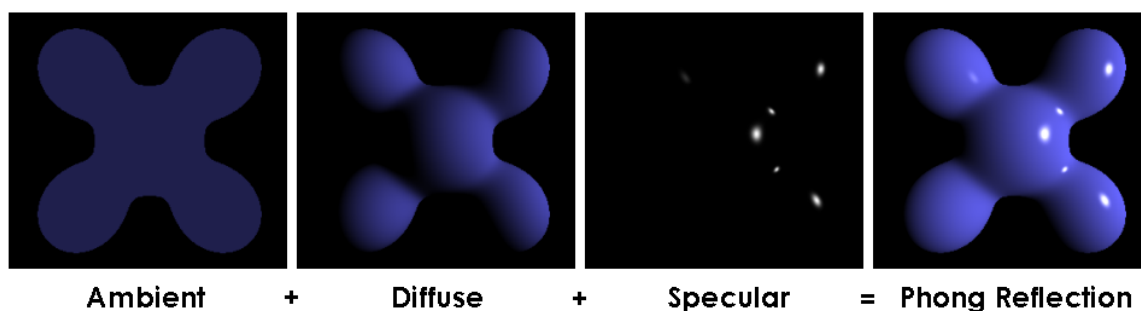
1	Wstęp teoretyczny	2
1.1	Model Phong	2
1.2	Model Gourauda	2
1.3	Wektor normalny	3
2	Zadanie laboratoryjne	3
2.1	Treść zadania	3
2.2	Opis działania programu	3
2.3	Kod programu	3
3	Wnioski	17
4	Źródła	17

1 Wstęp teoretyczny

1.1 Model Phong

Model Phong lub oświetlenie Phong to model lokalnego odbicia światła. By uzyskać najlepsze wyniki model ten uwzględnia trzy rodzaje światła (Rys1):

- Światło kierunkowe(ang. **Specular**) - refleksy odbite zgodnie z prawem Snella
- Światło rozproszone(ang. **Diffuse**) - wpływ bezpośredniego oświetlenia
- Światło otoczenia(ang. **Ambient**) - jednorodne światło oświetlające cały obiekt

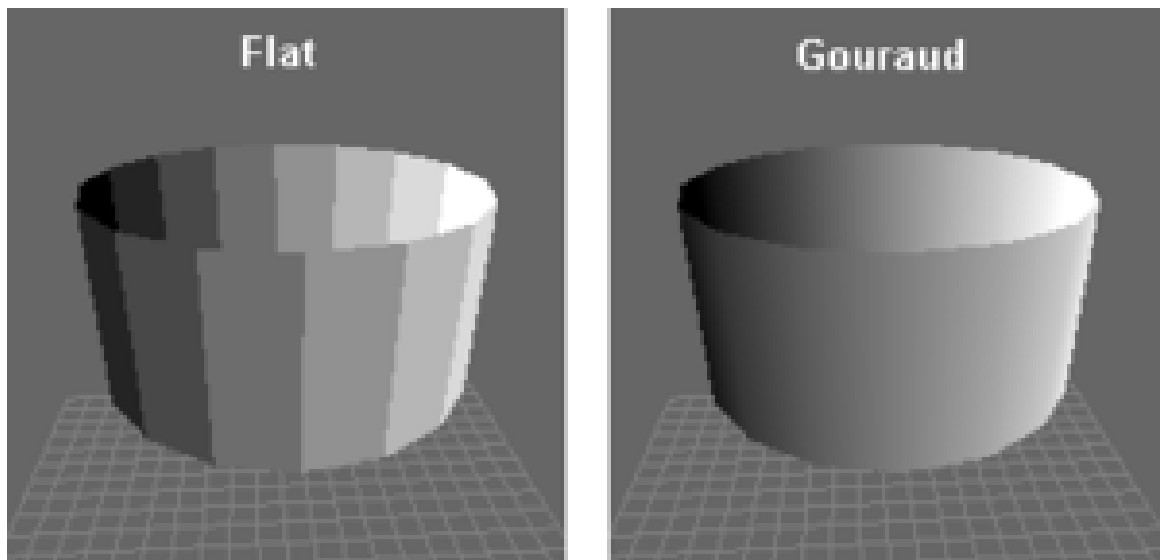


Rysunek 1: Model odbicia światła Phong

Każdy z materiałów na scenie ma zdefiniowane wartości K_s , K_d , K_a i alfa. Gdzie pierwsze trzy to stosunek odbicia światła (kolejno) kierunkowego, rozproszonego i otoczenia. Alfa to z kolei połysk

1.2 Model Gourauda

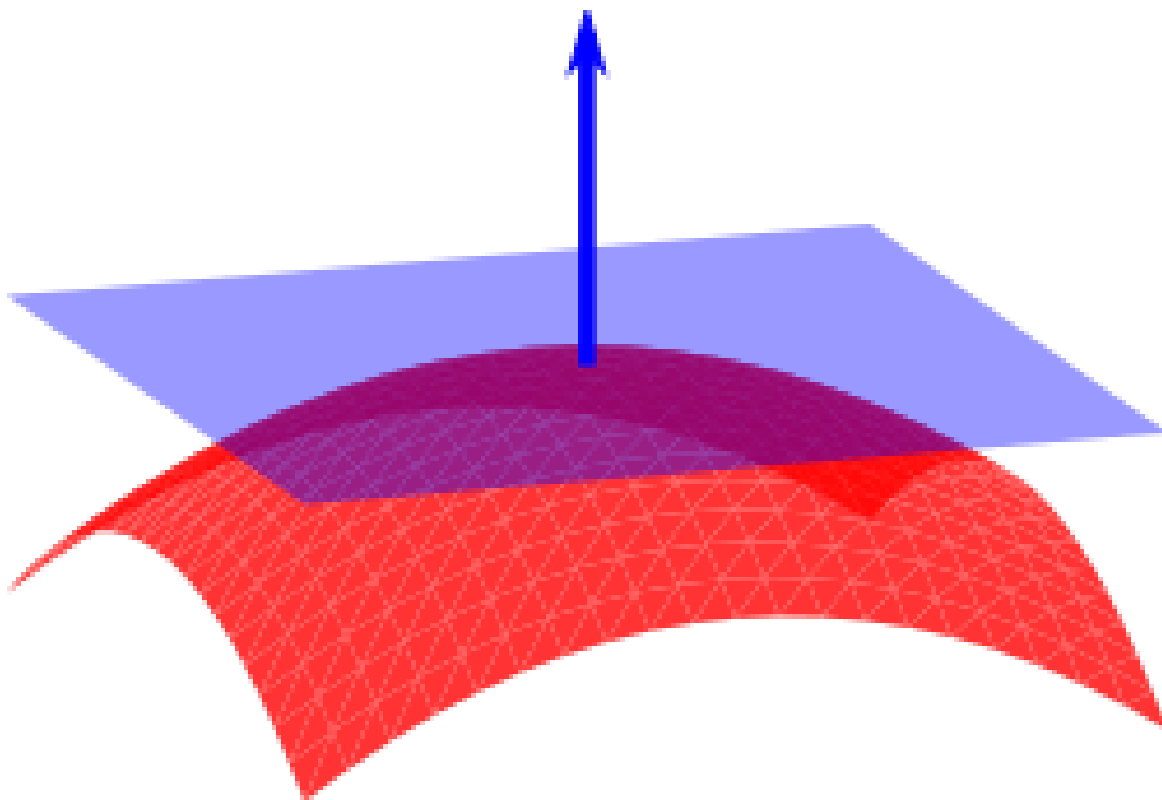
Cieniowanie Gourauda to metoda interpolacji polegająca na oświetlaniu wierzchołków w siatkach trójkątów i interpolacji wyników na cały trójkąt. Cieniowanie Gourauda jest uznawane za lepsze od cieniowania płaskiego i wymaga znacznie mniej obliczeń niż cieniowanie Phong ale daje gorsze wyniki.



Rysunek 2: Model cieniowania Gourauda

1.3 Wektor normalny

Wektor normalny to wektor prostopadły do płaszczyzny stycznej do danej powierzchni w danym punkcie. Pozwala to na rozróżnienie "Przodu" i "Tyłu" powierzchni co z kolei pozwala na ukrycie niewidocznych powierzchni (funkcja `glCullFace`)



Rysunek 3: Model cieniowania Gourauda

2 Zadanie laboratoryjne

2.1 Treść zadania

W ramach zadania należało do poprzednio stworzonego programu dodać dwa źródła światła. W kolorach przeciwnych (Czerwone i zielone). Światła te powinny świecić w stożku.

2.2 Opis działania programu

Zgodnie z treścią zadania program rysuje 4 obiekty. Domyślnie jajko i czajnik rysowane są w kolorze białym. Jednakże jest możliwość zmiany koloru na losowy. Wyświetlone obiekty można obracać za pomocą myszki (Przycisk musi być wciśnięty i przytrzymany). Program implementuje dwa światła niebieskie i czerwone.

Kontrola obrotu:

F1 - tryb obrotu obiektu

F2 - tryb obrotu kamery

F3 - tryb obrotu światłem 1 (Czerwone)

F4 - tryb obrotu światłem 2 (Zielone)

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

```

1  #include <windows.h>
2  #include <iostream>
3  #include <GL/glu.h>
4  #include <vector>
5  #include <math.h>
6  #define FREEGLUT_STATIC
7  #include <GL/freeglut.h>
8  #include "Egg.hpp"
9  #include "Light.hpp"
10 using namespace std;
11 HWND consoleWindow;
12 HWND glutWindow;
13
14 GLfloat deg = 0;
15 int sx = 0, sy = 0, sz = 0;
16 bool spin = false;
17 bool drawTeapot = true, smooth = true;
18 int eggMode = 0, moveMode = 0;
19 float sensitivity = 0.01f;
20 float totalRotationX = 0.0f, totalRotationY = 0.0f, totalRotationZ = 0.0f;
21 float pix2angle, theta = 0.0f, phi = 0.0f;
22 float dX, dY;
23 int radius = 6, lastX = 0, lastY = 0, camOrientation = 1;
24 float cameraRotationX = radius * cosf((theta*(M_PI/180))) * cosf((phi*(M_PI/180)));
25 float cameraRotationY = radius * sinf((phi*(M_PI/180)));
26 float cameraRotationZ = radius * sinf((theta*(M_PI/180))) * cosf((phi*(M_PI/180)));
27 Light light1(GL_LIGHT0);
28 int light1Radius = 10;
29 float light1RotationX = radius * cosf((theta*(M_PI/180))) * cosf((phi*(M_PI/180)));
30 float light1RotationY = radius * sinf((phi*(M_PI/180)));
31 float light1RotationZ = radius * sinf((theta*(M_PI/180))) * cosf((phi*(M_PI/180)));
32 Light light2(GL_LIGHT1);
33 int light2Radius = 10;
34 float light2RotationX = radius * cosf((theta*(M_PI/180))) * cosf((phi*(M_PI/180)));
35 float light2RotationY = radius * sinf((phi*(M_PI/180)));
36 float light2RotationZ = radius * sinf((theta*(M_PI/180))) * cosf((phi*(M_PI/180)));
37
38 Egg egg(200);
39 void toggleFocusToConsole() {
40     ShowWindow(glutWindow, SW_HIDE);
41     ShowWindow(consoleWindow, SW_SHOWNORMAL);
42     SetForegroundWindow(consoleWindow);
43 }
44
45 void toggleFocusToGLUT() {
46     ShowWindow(consoleWindow, SW_HIDE);
47     ShowWindow(glutWindow, SW_SHOWNORMAL);
48     SetForegroundWindow(glutWindow);
49 }
50 void reset_rotation(){
51     theta = 0.0f;
52     phi = 0.0f;
53     lastX = 0;
54     lastY = 0;
55     cameraRotationX = radius * cosf((theta*(M_PI/180.0f))) *
        ↪ cosf((phi*(M_PI/180.0f)));

```

```

56     cameraRotationY = radius * sinf((phi*(M_PI/180.0f)));
57     cameraRotationZ = radius * sinf((theta*(M_PI/180.0f))) *
    ↪ cosf((phi*(M_PI/180.0f)));
58
59     light1RotationX = light1Radius * cosf((theta*(M_PI/180))) *
    ↪ cosf((phi*(M_PI/180)));
60     light1RotationY = light1Radius * sinf((phi*(M_PI/180)));
61     light1RotationZ = light1Radius * sinf((theta*(M_PI/180))) *
    ↪ cosf((phi*(M_PI/180)));
62
63     light2RotationX = light2Radius * cosf((theta*(M_PI/180))) *
    ↪ cosf((phi*(M_PI/180)));
64     light2RotationY = light2Radius * sinf((phi*(M_PI/180)));
65     light2RotationZ = light2Radius * sinf((theta*(M_PI/180))) *
    ↪ cosf((phi*(M_PI/180)));
66 }
67 string bool_to_string(bool convert){
68     if(convert){
69         return "true";
70     }else{
71         return "false";
72     }
73 }
74 void printControls(){
75     cout<<"=====\n";
76     cout<<"F1 - tryb obrotu obiektu";
77     cout<<"F2 - tryb obrotu kamery";
78     cout<<"F3 - tryb obrotu swiatlem 1 (Czerwone)";
79     cout<<"F4 - tryb obrotu swiatlem 2 (Zielone)";
80     cout<<"ESC - Powrot do menu (okno konsolowe)\n";
81     cout<<"Nalezy nacisnac i przytrzymac PPM\n";
82     cout<<"Ruch myszy w osi X - Obrot osi X\n";
83     cout<<"Ruch myszy w osi Y - Obrot osi Y\n";
84     cout<<"Scroll up - Przyblizenie obiektu\n";
85     cout<<"Scroll down - Oddalenie obiektu\n";
86     cout<<"Nacisnij Enter zeby kontynuowac\n"<<flush;
87     cin.get();
88     cin.get();
89 }
90 void axis(){
91     glBegin(GL_LINES);
92
93     glColor3f(1.0, 0.0, 0.0);
94     glVertex3f(-5.0, 0.0, 0.0);
95     glVertex3f(5.0, 0.0, 0.0);
96
97     glColor3f(0.0, 1.0, 0.0);
98     glVertex3f(0.0, -5.0, 0.0);
99     glVertex3f(0.0, 5.0, 0.0);
100
101     glColor3f(0.0, 0.0, 1.0);
102     glVertex3f(0.0, 0.0, -5.0);
103     glVertex3f(0.0, 0.0, 5.0);
104
105     glEnd();
106 }

```

```

107 void printOptions();
108 void menu();
109 void printOptions(){
110     int density = egg.getDensity();
111     bool color = egg.getColor();
112     float scale = egg.getScale();
113     float pointSize = egg.getPointSize();
114     cout<<"=====\n";
115     cout<<"1.Skala obiektow: "<<scale<<"\n";
116     cout<<"2.Ilosc punktow: "<<density<<"\n";
117     cout<<"3.Rysowanie w kolorze: "<<bool_to_string(color)<<"\n";
118     cout<<"4.Promien kamery: "<<radius<<"\n";
119     cout<<"5.Czulosc myszki: "<<sensitivity<<"\n";
120     cout<<"6.Rozmiar punktow: "<<pointSize<<"\n";
121     cout<<"7.Wroc do menu"<<"\n";
122     cout<<"> ";
123     int x;
124     cin>>x;
125     switch (x){
126     case 1:
127         cout<<"Nowa skala\n";
128         cout<<"> ";
129         cin>>scale;
130         egg.setScale(scale);
131         printOptions();
132         break;
133     case 2:
134         cout<<"Nowa gestosc\n";
135         cout<<"> ";
136         cin>>density;
137         egg.setDensity(density);
138         printOptions();
139         break;
140     case 3:
141         color =! color;
142         egg.setColor(color);
143         egg.generateMatrix();
144         printOptions();
145         break;
146     case 4:
147         cout<<"Nowy promien kamery\n";
148         cout<<"> ";
149         cin>>radius;
150         printOptions();
151         break;
152     case 5:
153         cout<<"Nowa predkosc kamery\n";
154         cout<<"> ";
155         cin>>sensitivity;
156         printOptions();
157         break;
158     case 6:
159         cout<<"Nowy rozmiar punktow\n";
160         cout<<"> ";
161         cin>>pointSize;
162         egg.setPointSize(pointSize);

```

```

163         printOptions();
164         break;
165     case 7:
166         menu();
167         break;
168     }
169 }
170 void menu(){
171     toggleFocusToConsole();
172     reset_rotation();
173     cout<<"=====\n";
174     cout<<"1. Narysuj czajnik\n";
175     cout<<"2. Narysuj jajko (punkty)\n";
176     cout<<"3. Narysuj jajko (linie)\n";
177     cout<<"4. Narysuj jajko (trojkaty) \n";
178     cout<<"5. Opcje\n";
179     cout<<"6. Kontrola\n";
180     cout<<"7. Zakoncz program\n";
181     cout<<"> ";
182     int x;
183     cin>>x;
184     switch (x){
185     case 1:
186         drawTeapot = true;
187         break;
188     case 2:
189         drawTeapot = false;
190         eggMode = 1;
191         break;
192     case 3:
193         drawTeapot = false;
194         eggMode = 2;
195         break;
196     case 4:
197         drawTeapot = false;
198         eggMode = 3;
199         break;
200     case 5:
201         printOptions();
202         break;
203     case 6:
204         printControls();
205         menu();
206         break;
207     case 7:
208         exit(0);
209         break;
210     default:
211         cout<<"Podano nieporawny znak\n";
212         menu();
213         break;
214     }
215     toggleFocusToGLUT();
216     glutPostRedisplay();
217 }
218 void specialKey(int key,int x,int y){

```



```

219     switch (key){
220     //F1 - Ruch obiektu
221     case GLUT_KEY_F1:
222         moveMode = 0;
223         break;
224     //F2 - Ruch kamery
225     case GLUT_KEY_F2:
226         moveMode = 1;
227         break;
228     //F3 - Ruch światła 1
229     case GLUT_KEY_F3:
230         moveMode = 2;
231         break;
232     //F4 - Ruch światła 2
233     case GLUT_KEY_F4:
234         moveMode = 3;
235         break;
236     default:
237         break;
238     }
239 }
240 void normalKey(u_char key,int x,int y){
241     switch (key)
242     {
243     case 27:
244         menu();
245         break;
246     default:
247         break;
248     }
249     if (sx == 0 && sy == 0 && sz == 0) {
250         glutIdleFunc(nullptr);
251     }
252 }
253
254 void mouse(int x, int y){
255     dY = y - lastY;
256     lastY = y;
257     dX = x - lastX;
258     lastX = x;
259     theta += dX * pix2angle;
260     phi += dY * pix2angle;
261     if (phi > 89.0f) {phi = 89.0f;}
262     if (phi < -89.0f) {phi = -89.0f;}
263     switch(moveMode){
264     case 0:
265         totalRotationX += dY;
266         totalRotationY += dX;
267         totalRotationZ += atan2f(dY,dX);
268         break;
269     case 1:
270         cameraRotationX = radius * cosf((theta*(M_PI/180.0f))) *
271             ↪ cosf((phi*(M_PI/180.0f)));
272         cameraRotationY = radius * sinf((phi*(M_PI/180.0f)));
273         cameraRotationZ = radius * sinf((theta*(M_PI/180.0f))) *
274             ↪ cosf((phi*(M_PI/180.0f)));

```

```

273         break;
274     case 2:
275         light1RotationX = light1Radius * cosf((theta*(M_PI/180))) *
            ↪ cosf((phi*(M_PI/180)));
276         light1RotationY = light1Radius * sinf((phi*(M_PI/180)));
277         light1RotationZ = light1Radius * sinf((theta*(M_PI/180))) *
            ↪ cosf((phi*(M_PI/180)));
278         break;
279     case 3:
280         light2RotationX = light2Radius * cosf((theta*(M_PI/180))) *
            ↪ cosf((phi*(M_PI/180)));
281         light2RotationY = light2Radius * sinf((phi*(M_PI/180)));
282         light2RotationZ = light2Radius * sinf((theta*(M_PI/180))) *
            ↪ cosf((phi*(M_PI/180)));
283         break;
284     }
285     lastX = x;
286     lastY = y;
287     glutPostRedisplay();
288 }
289 void mouseWheel(int button, int dir, int x, int y){
290
291     if (dir > 0){
292         radius -= 1;
293     }else{
294         radius += 1;
295     }
296     if(radius>=10){
297         radius=10;
298     }
299     if(radius<=1){
300         radius=1;
301     }
302     glutPostRedisplay();
303 }
304 void display() {
305     GLfloat lPos1[] =
306         ↪ {light1RotationX,light1RotationY,light1RotationZ,1};//x,y,z,czy światło
307         ↪ jest odległość
308     GLfloat lPos2[] = {light2RotationX,light2RotationY,light2RotationZ,1};
309     GLfloat col[] = {1,0,0,1};
310     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
311     glLoadIdentity();
312     gluLookAt(cameraRotationX,cameraRotationY,cameraRotationZ,0,0,0,camOrientati
313         ↪ on,0);//Ustawienie kamery
314     light1.setPosition(lPos1);
315     light2.setPosition(lPos2);
316     glEnable(GL_COLOR_MATERIAL);
317
318     glRotatef(totalRotationX, 1.0f, 0.0f, 0.0f);
319     glRotatef(totalRotationY, 0.0f, 1.0f, 0.0f);
320     glRotatef(totalRotationZ, 0.0f, 0.0f, 1.0f);
321     axis();
322     if(drawTeapot){
323         glutSolidTeapot(1);
324     }else{

```

```

322     egg.initMaterial();
323     egg.draw(eggMode);
324 }
325
326 glutSwapBuffers();
327 }
328 void Init() {
329     pix2angle = 360.0/800;
330     egg.generateMatrix();
331     glEnable(GL_DEPTH_TEST); //bez tego frontalna sciana nadpisuje tylnia
332     glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
333     glMatrixMode(GL_PROJECTION);
334     glLoadIdentity();
335     glFrustum(-1,1,-1,1,2,20);
336     glMatrixMode(GL_MODELVIEW);
337     // Włącza culling, czyli pomijanie tylnych ścianek
338     glEnable(GL_CULL_FACE);
339     // Ustawia kierunek frontowych ścianek jako przeciwny do ruchu wskazówek zegara
340     glFrontFace(GL_CW);
341     // Ustawia pomijanie tylnych ścianek
342     glCullFace(GL_BACK);
343     // Kolor stały
344     light1.setColor(1.0,0.0,0.0);
345     light2.setColor(0.0,1.0,0.0);
346     light1.initLight();
347     light2.initLight();
348     //Drugie światło
349     glShadeModel(GL_SMOOTH);
350     glEnable(GL_LIGHTING); //Włączenie oświetlenia
351     glEnable(GL_LIGHT0); //Dodanie źródła światła
352     glEnable(GL_LIGHT1);
353
354 }
355 // Sprawko do 15 w pon
356 // W sprawku Phong,Gouraud i wektor normalny
357 // TODO - Kąty przestrzenne dla lamp radiany określają stożek świecenia światła
358 // ADS - (Nie odpowiada fizyce) światło nie jest jednorodne
359 // Ambient - ogólnie wszędzie bezkierunkowe
360 // Diffuse - kąt padania = kąt odbicia
361 // Specular - odbicia lustrzane
362 // TODO - Każdemu punktowi dodać ADS składowa to sposób w jaki obiekt odbija ads
363 // Tylko jednokrotne odbicie
364 // TODO - Światło z reflektora ma drogę reflektor/obiekt(Tłumienie) obiekt/kamera
365 // TODO - cieniowanie Phonga i Gourauda
366 // Różni się liczenie wektora normalnego
367 int main(int argc, char** argv){
368     consoleWindow = GetConsoleWindow();
369     glutInit(&argc, argv);
370     glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
371     glutInitWindowSize(800,800);
372     glutCreateWindow("Lab 3 - Czajnik i Jajko");
373     glutWindow = FindWindowW(NULL,L"Lab 3 - Czajnik i Jajko");
374     Init();
375     glutDisplayFunc(display);
376     glutIdleFunc(nullptr);
377     glutKeyboardFunc(normalKey);

```

```

378     glutSpecialFunc(specialKey);
379     glutMotionFunc(mouse);
380     glutMouseWheelFunc(mouseWheel);
381     menu();
382
383     glutMainLoop();
384     system("pause");
385     return 0;
386 }

```

Fragment kodu 1: Fragment kodu z programu

```

1  #include <math.h>
2  #include <GL/glu.h>
3  #define FREEGLUT_STATIC
4  #include <GL/freeglut.h>
5  #include "Egg.hpp"
6  using namespace std;
7
8  float Egg::randFloat(){
9      return (float)rand()/(float)(RAND_MAX);
10 }
11 Egg::Egg(int density) : density(density){
12     pointsMatrix.resize(density,vector<pointsRgb>(density));
13 }
14 vector<vector<pointsRgb>> Egg::getPointsMatrix(){
15     return pointsMatrix;
16 }
17 point Egg::generateNormalVect(int u,int v){
18     float x_u = (-450*pow(u,4) + 900*pow(u,3) - 810*pow(u,2) + 360*u - 45) *
19         ↪ 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) +
21         ↪ 45*u) * sin(M_PI*v);
22     float y_u = 640*pow(u,3) - 960*pow(u,2) + 320*u;
23     float y_v = 0;
24     float z_u = (-450*pow(u,4) + 900*pow(u,3) - 810*pow(u,2) + 360*u - 45) *
25         ↪ sin(M_PI*v);
26     float z_v = -M_PI * (90*pow(u,5) - 225*pow(u,4) + 270*pow(u,3) - 180*pow(u,2)
27         ↪ + 45*u) * cos(M_PI*v);
28     point newPoint;
29     newPoint.x = y_u * z_v - z_u * y_v;
30     newPoint.y = z_u * x_v - x_u * z_v;
31     newPoint.z = x_u * y_v - y_u * x_v;
32     float length = sqrt(newPoint.x*newPoint.x + newPoint.y*newPoint.y +
33         ↪ newPoint.z*newPoint.z);
34     newPoint.x /= length;
35     newPoint.y /= length;
36     newPoint.z /= length;
37     return newPoint;
38 }
39 void Egg::generateMatrix(){
40     for(int u=0;u<(density);u++){
41         float _u = 0.5/((float)density-1);
42         _u *= u;

```

```

38     if(u==density-1){
39         pointsMatrix[u][0].y = scale*((160*pow(_u,4)) - (320*pow(_u,3)) + (160
        ↪ * pow(_u,2)) - 5);
40         //Biały jajko
41         pointsMatrix[u][0].r = 1.0f;
42         pointsMatrix[u][0].g = 1.0f;
43         pointsMatrix[u][0].b = 1.0f;
44         point newPoint = generateNormalVect(u,0);
45         pointsMatrix[u][0].nx = newPoint.x;
46         pointsMatrix[u][0].ny = newPoint.y;
47         pointsMatrix[u][0].nz = newPoint.z;
48         break;
49     }
50     for(int v=0;v<density;v++){
51         float _v = v/((float)density);
52         _v *= 2.0f;
53         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));
54         pointsMatrix[u][v].y = scale*(160*pow(_u,4) - 320*pow(_u,3) + 160 *
        ↪ pow(_u,2) - 5);
55         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));
56         //Biały jajko
57         pointsMatrix[u][v].r = 1.0f;
58         pointsMatrix[u][v].g = 1.0f;
59         pointsMatrix[u][v].b = 1.0f;
60         point newPoint = generateNormalVect(u,v);
61         pointsMatrix[u][v].nx = newPoint.x;
62         pointsMatrix[u][v].ny = newPoint.y;
63         pointsMatrix[u][v].nz = newPoint.z;
64     }
65 }
66 }
67 void Egg::initMaterial(){
68     float mat_ambient[4] = {0.3f, 0.3f, 0.3f, 1.0f};
69     float mat_diffuse[4] = {0.6f, 0.3f, 0.3f, 1.0f};
70     float mat_specular[4] = {1.0f, 1.0f, 1.0f, 1.0f};
71     float mat_shininess = 10.0f;
72     glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, mat_ambient);
73     glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, mat_diffuse);
74     glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, mat_specular);
75     glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, mat_shininess);
76 }
77 void Egg::draw(int model){
78     switch (model)
79     {
80     case 1:
81         glPointSize(pointSize);
82         glBegin(GL_POINTS);
83         for(int u=0;u<density-1;u++){
84             if(u==0){
85                 glColor3f(pointsMatrix[u][0].r,pointsMatrix[u][0].g,pointsMatrix[u]
        ↪ [0].b);
86                 glVertex3f(pointsMatrix[u][0].x,pointsMatrix[u][0].y,pointsMatrix[
        ↪ u][0].z);
87                 continue;

```

```

88     }
89     if(u==density-2){
90         glColor3f(pointsMatrix[u+1][0].r,pointsMatrix[u+1][0].g,pointsMatr
           ↪ ix[u+1][0].b);
91         glVertex3f(pointsMatrix[u+1][0].x,pointsMatrix[u+1][0].y,pointsMat
           ↪ rix[u+1][0].z);
92         break;
93     }
94     for(int v=0;v<density;v++){
95         glColor3f(pointsMatrix[u][v].r,pointsMatrix[u][v].g,pointsMatrix[u]
           ↪ ][v].b);
96         glVertex3f(pointsMatrix[u][v].x,pointsMatrix[u][v].y,pointsMatrix[
           ↪ u][v].z);
97     }
98 }
99 glEnd();
100 break;
101 case 2:
102     glBegin(GL_LINES);
103     for(int u=0;u<density-1;u++){
104         if(u==0){
105             for(int v=0;v<density;v++){
106                 glColor3f(pointsMatrix[u][0].r,pointsMatrix[u][0].g,pointsMatr
                   ↪ ix[u][0].b);
107                 glVertex3f(pointsMatrix[u][0].x,pointsMatrix[u][0].y,pointsMat
                   ↪ rix[u][0].z);
108                 glColor3f(pointsMatrix[u+1][v].r, pointsMatrix[u+1][v].g,
                   ↪ pointsMatrix[u+1][v].b);
109                 glVertex3f(pointsMatrix[u+1][v].x, pointsMatrix[u+1][v].y,
                   ↪ pointsMatrix[u+1][v].z);
110             }
111             continue;
112         }
113         if(u==density-2){
114             for(int v=0;v<density;v++){
115                 glColor3f(pointsMatrix[u+1][0].r,pointsMatrix[u+1][0].g,points
                   ↪ Matrix[u+1][0].b);
116                 glVertex3f(pointsMatrix[u+1][0].x,pointsMatrix[u+1][0].y,point
                   ↪ sMatrix[u+1][0].z);
117                 glColor3f(pointsMatrix[u][v].r, pointsMatrix[u][v].g,
                   ↪ pointsMatrix[u][v].b);
118                 glVertex3f(pointsMatrix[u][v].x, pointsMatrix[u][v].y,
                   ↪ pointsMatrix[u][v].z);
119             }
120             break;
121         }
122         for(int v=0;v<density;v++){
123             int nextV = (v + 1) % density;
124             glColor3f(pointsMatrix[u][v].r,pointsMatrix[u][v].g,pointsMatrix[u]
                   ↪ ][v].b);
125             glVertex3f(pointsMatrix[u][v].x,pointsMatrix[u][v].y,pointsMatrix[
                   ↪ u][v].z);
126             glColor3f(pointsMatrix[u+1][v].r, pointsMatrix[u+1][v].g,
                   ↪ pointsMatrix[u+1][v].b);
127             glVertex3f(pointsMatrix[u+1][v].x, pointsMatrix[u+1][v].y,
                   ↪ pointsMatrix[u+1][v].z);

```

```

128
129         glColor3f(pointsMatrix[u][v].r,pointsMatrix[u][v].g,pointsMatrix[u]
    ↪   ][v].b);
130         glVertex3f(pointsMatrix[u][v].x,pointsMatrix[u][v].y,pointsMatrix[
    ↪   u][v].z);
131         glColor3f(pointsMatrix[u][nextV].r, pointsMatrix[u][nextV].g,
    ↪   pointsMatrix[u][nextV].b);
132         glVertex3f(pointsMatrix[u][nextV].x, pointsMatrix[u][nextV].y,
    ↪   pointsMatrix[u][nextV].z);
133
134     }
135 }
136 glEnd();
137 break;
138 case 3:
139     glBegin(GL_TRIANGLES);
140     for(int u=0;u<density-1;u++){
141         //Obecnie trójkąty są CCW
142         if(u==0){
143             for(int v=0;v<density;v++){
144                 int nextV = (v + 1) % density;
145                 glColor3f(pointsMatrix[u][0].r,pointsMatrix[u][0].g,pointsMatr
    ↪   ix[u][0].b);
146                 glVertex3f(pointsMatrix[u][0].x,pointsMatrix[u][0].y,pointsMat
    ↪   rix[u][0].z);
147                 glColor3f(pointsMatrix[u+1][nextV].r,pointsMatrix[u+1][nextV].
    ↪   g,pointsMatrix[u+1][nextV].b);
148                 glVertex3f(pointsMatrix[u+1][nextV].x,pointsMatrix[u+1][nextV]
    ↪   .y,pointsMatrix[u+1][nextV].z);
149                 glColor3f(pointsMatrix[u+1][v].r,pointsMatrix[u+1][v].g,points
    ↪   Matrix[u+1][v].b);
150                 glVertex3f(pointsMatrix[u+1][v].x,pointsMatrix[u+1][v].y,point
    ↪   sMatrix[u+1][v].z);
151             }
152             continue;
153         }
154         if(u==density-2){
155             for(int v=0;v<density;v++){
156                 int nextV = (v + 1) % density;
157                 glColor3f(pointsMatrix[u+1][0].r,pointsMatrix[u+1][0].g,points
    ↪   Matrix[u+1][0].b);
158                 glVertex3f(pointsMatrix[u+1][0].x,pointsMatrix[u+1][0].y,point
    ↪   sMatrix[u+1][0].z);
159                 glColor3f(pointsMatrix[u][v].r,pointsMatrix[u][v].g,pointsMatr
    ↪   ix[u][v].b);
160                 glVertex3f(pointsMatrix[u][v].x,pointsMatrix[u][v].y,pointsMat
    ↪   rix[u][v].z);
161                 glColor3f(pointsMatrix[u][nextV].r,pointsMatrix[u][nextV].g,po
    ↪   intsMatrix[u][nextV].b);
162                 glVertex3f(pointsMatrix[u][nextV].x,pointsMatrix[u][nextV].y,p
    ↪   ointsMatrix[u][nextV].z);
163             }
164             break;
165         }
166         for(int v=0;v<density;v++){
167             int nextV = (v + 1) % density;

```

```

168         glNormal3f(pointsMatrix[u][v].nx,pointsMatrix[u][v].ny,pointsMatrix[u][v].nz);
169         glColor3f(pointsMatrix[u][v].r,pointsMatrix[u][v].g,pointsMatrix[u][v].b);
170         glVertex3f(pointsMatrix[u][v].x,pointsMatrix[u][v].y,pointsMatrix[u][v].z);
171
172         glNormal3f(pointsMatrix[u+1][nextV].nx,pointsMatrix[u+1][nextV].ny,pointsMatrix[u+1][nextV].nz);
173         glColor3f(pointsMatrix[u+1][nextV].r,pointsMatrix[u+1][nextV].g,pointsMatrix[u+1][nextV].b);
174         glVertex3f(pointsMatrix[u+1][nextV].x, pointsMatrix[u+1][nextV].y, pointsMatrix[u+1][nextV].z);
175
176         glNormal3f(pointsMatrix[u+1][v].nx,pointsMatrix[u+1][v].ny,pointsMatrix[u+1][v].nz);
177         glColor3f(pointsMatrix[u+1][v].r,pointsMatrix[u+1][v].g,pointsMatrix[u+1][v].b);
178         glVertex3f(pointsMatrix[u+1][v].x, pointsMatrix[u+1][v].y, pointsMatrix[u+1][v].z);
179
180         glNormal3f(pointsMatrix[u+1][nextV].nx,pointsMatrix[u+1][nextV].ny,pointsMatrix[u+1][nextV].nz);
181         glColor3f(pointsMatrix[u+1][nextV].r,pointsMatrix[u+1][nextV].g,pointsMatrix[u+1][nextV].b);
182         glVertex3f(pointsMatrix[u+1][nextV].x, pointsMatrix[u+1][nextV].y, pointsMatrix[u+1][nextV].z);
183
184         glNormal3f(pointsMatrix[u][v].nx,pointsMatrix[u][v].ny,pointsMatrix[u][v].nz);
185         glColor3f(pointsMatrix[u][v].r,pointsMatrix[u][v].g,pointsMatrix[u][v].b);
186         glVertex3f(pointsMatrix[u][v].x,pointsMatrix[u][v].y,pointsMatrix[u][v].z);
187
188         glNormal3f(pointsMatrix[u][nextV].nx,pointsMatrix[u][nextV].ny,pointsMatrix[u][nextV].nz);
189         glColor3f(pointsMatrix[u][nextV].r,pointsMatrix[u][nextV].g,pointsMatrix[u][nextV].b);
190         glVertex3f(pointsMatrix[u][nextV].x, pointsMatrix[u][nextV].y, pointsMatrix[u][nextV].z);
191     }
192 }
193 glEnd();
194 break;
195 }
196 }
197 //Setters
198 void Egg::setDensity(int newDensity){
199     density = newDensity;
200     pointsMatrix.resize(density,vector<pointsRgb>(density));
201     generateMatrix();
202 }
203 void Egg::setColor(float newColor){color = newColor;}
204 void Egg::setScale(float newScale){scale = newScale;}
205 void Egg::setPointSize(float newPointSize){pointSize = newPointSize;}

```



```

206 //Getters
207 int Egg::getDensity(){return density;}
208 float Egg::getColor(){return color;}
209 float Egg::getScale(){return scale;}
210 float Egg::getPointSize(){return pointSize;}
211 Egg::~Egg(){
212
213 }

```

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::normalize(GLfloat* v) {
20     GLfloat length = sqrt(v[0] * v[0] + v[1] * v[1] + v[2] * v[2]);
21     if (length > 0.0f) {
22         v[0] /= length;
23         v[1] /= length;
24         v[2] /= length;
25     }
26 }
27 void Light::setPosition(GLfloat lPos[]){
28     GLfloat light_direction[3];
29     light_direction[0] = -lPos[0];
30     light_direction[1] = -lPos[1];
31     light_direction[2] = -lPos[3];
32     normalize(light_direction);
33     glLightfv(lightID, GL_POSITION, lPos);
34     glLightfv(lightID, GL_SPOT_DIRECTION, light_direction);
35     glLightf(lightID, GL_SPOT_CUTOFF, 25.0f);
36     glLightf(lightID, GL_SPOT_EXPONENT, 2.0f);
37 }
38 void Light::setColor(float r, float g, float b){
39     light_ambient[0] = r;
40     light_ambient[1] = g;
41     light_ambient[2] = b;
42     light_diffuse[0] = r;
43     light_diffuse[1] = g;

```

```
44     light_diffuse[2] = b;
45     light_specular[0] = r;
46     light_specular[1] = g;
47     light_specular[2] = b;
48 }
```

Fragment kodu 3: Kod Light.cpp

3 Wnioski

Na zajęciach nie udało się ukończyć programu. Po pracy w domu program działa poprawnie.

4 Źródła

- <https://gniewkowski.wroclaw.pl/gk/lab5.pdf>
- https://en.wikipedia.org/wiki/Phong_reflection_model
- https://en.wikipedia.org/wiki/Gouraud_shading
- https://pl.wikipedia.org/wiki/Wektor_normalny