



Politechnika Wrocławska

Sprawozdanie 4

Ćwiczenie 4. Oświetlenie scen

Krzysztof Zalewa

16.12.2024

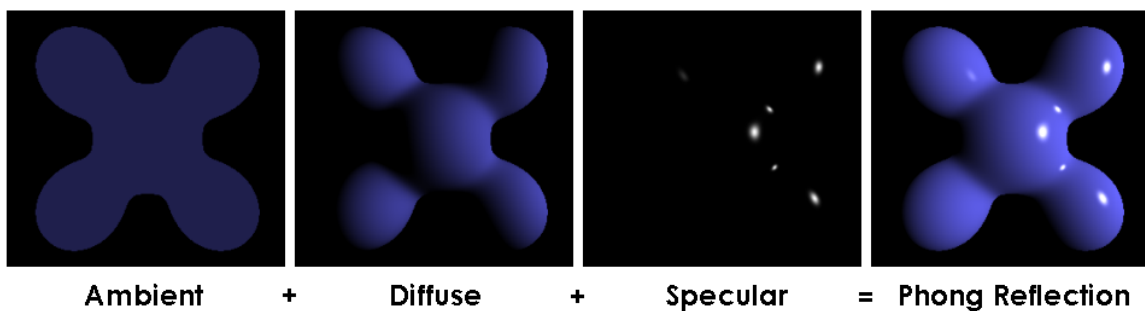
Spis treści

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

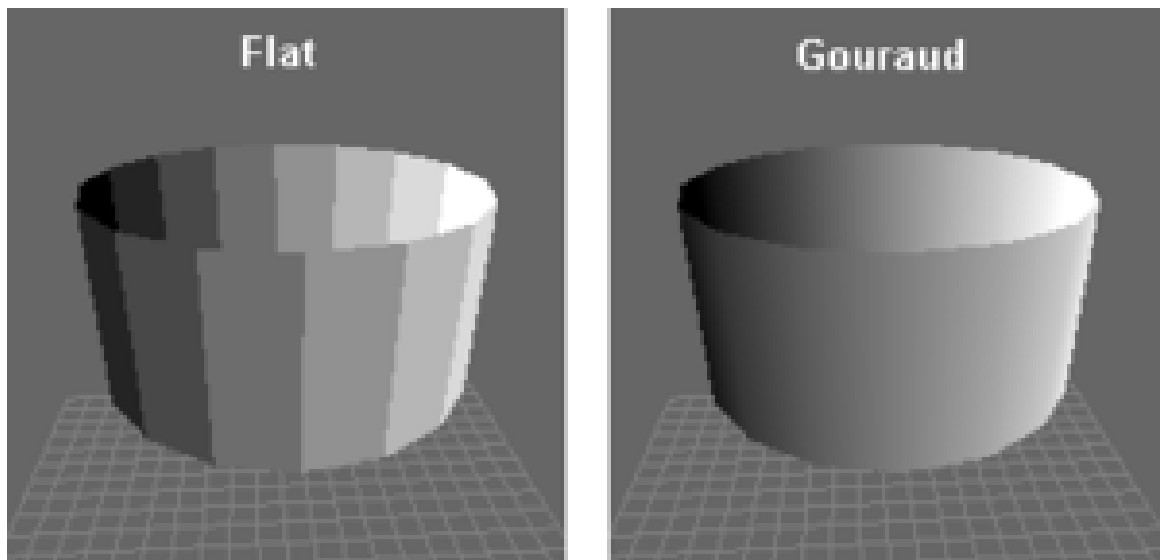


Rysunek 1: Model odbicia światła Phong

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

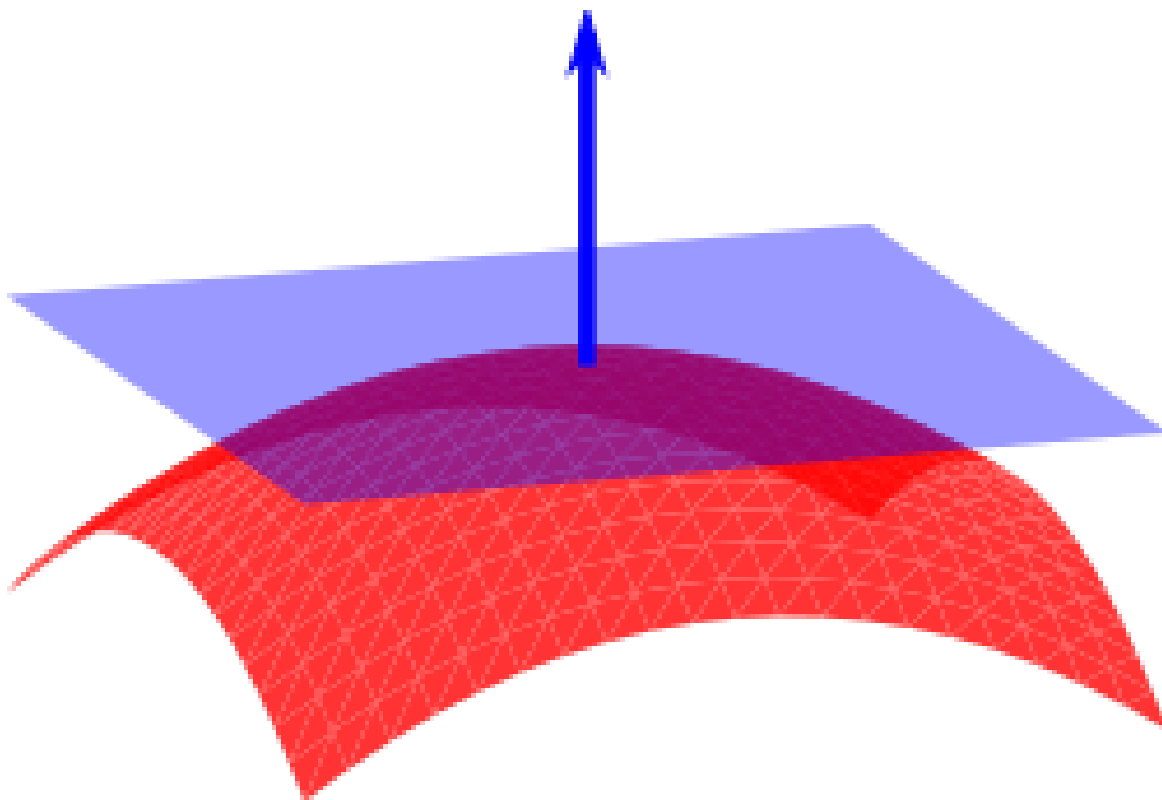
Cieniowanie Gourauda to metoda interpolacji polegająca na oświetlaniu wierzchołków w siatkach trójkątów i interpolacji wyników na cały trójkąt. Cieniowanie Gourauda jest uznawane za lepsze od cieniowania płaskiego i wymaga znacznie mniej obliczeń niż cieniowanie Phong, ale daje gorsze wyniki.



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\n";
77     cout<<"F2 - tryb obrotu kamery\n";
78     cout<<"F3 - tryb obrotu swiatlem 1 (Czerwone)\n";
79     cout<<"F4 - tryb obrotu swiatlem 2 (Zielone)\n";
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<<"Ruch myszy w osi X - Obrot osi X\n";
85     cout<<"Ruch myszy w osi Y - Obrot osi Y\n";
86     cout<<"Scroll up - Przyblizenie obiektu\n";
87     cout<<"Scroll down - Oddalenie obiektu\n";
88     cout<<"Nacisnij Enter zeby kontynuowac\n"<<flush;
89     cin.get();
90     cin.get();
91 }
92 void axis(){
93     glDisable(GL_LIGHTING);
94     glBegin(GL_LINES);
95
96     glColor3f(1.0, 0.0, 0.0);
97     glVertex3f(-5.0, 0.0, 0.0);
98     glVertex3f(5.0, 0.0, 0.0);
99
100    glColor3f(0.0, 1.0, 0.0);
101    glVertex3f(0.0, -5.0, 0.0);
102    glVertex3f(0.0, 5.0, 0.0);
103
104    glColor3f(0.0, 0.0, 1.0);
105    glVertex3f(0.0, 0.0, -5.0);
106    glVertex3f(0.0, 0.0, 5.0);

```

```

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

```

```

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

```



```

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

```

```

274         cameraRotationY = radius * sinf((phi*(M_PI/180.0f)));
275         cameraRotationZ = radius * sinf((theta*(M_PI/180.0f))) *
        ↪ cosf((phi*(M_PI/180.0f)));
276         break;
277     case 2:
278         light1RotationX = light1Radius * cosf((theta*(M_PI/180))) *
        ↪ cosf((phi*(M_PI/180)));
279         light1RotationY = light1Radius * sinf((phi*(M_PI/180)));
280         light1RotationZ = light1Radius * sinf((theta*(M_PI/180))) *
        ↪ cosf((phi*(M_PI/180)));
281         break;
282     case 3:
283         light2RotationX = light2Radius * cosf((theta*(M_PI/180))) *
        ↪ cosf((phi*(M_PI/180)));
284         light2RotationY = light2Radius * sinf((phi*(M_PI/180)));
285         light2RotationZ = light2Radius * sinf((theta*(M_PI/180))) *
        ↪ cosf((phi*(M_PI/180)));
286         break;
287     }
288     lastX = x;
289     lastY = y;
290     glutPostRedisplay();
291 }
292 void mouseWheel(int button, int dir, int x, int y){
293
294     if (dir > 0){
295         radius -= 1;
296     }else{
297         radius += 1;
298     }
299     if(radius>=10){
300         radius=10;
301     }
302     if(radius<=1){
303         radius=1;
304     }
305     glutPostRedisplay();
306 }
307 void display() {
308     GLfloat lPos1[] =
        ↪ {light1RotationX,light1RotationY,light1RotationZ,1};//x,y,z,czy światło
        ↪ jest odległość
309     GLfloat lPos2[] = {light2RotationX,light2RotationY,light2RotationZ,1};
310     GLfloat col[] = {1,0,0,1};
311     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
312     glLoadIdentity();
313     gluLookAt(cameraRotationX,cameraRotationY,cameraRotationZ,0,0,0,0,camOrientati
        ↪ on,0);//Ustawienie kamery
314     light1.setPosition(lPos1);
315     light2.setPosition(lPos2);
316     // glEnable(GL_COLOR_MATERIAL);
317     axis();
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     if(drawTeapot){

```

```

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

```

```

378     glutIdleFunc(nullptr);
379     glutKeyboardFunc(normalKey);
380     glutSpecialFunc(specialKey);
381     glutMotionFunc(mouse);
382     glutMouseWheelFunc(mouseWheel);
383     menu();
384
385     glutMainLoop();
386     system("pause");
387     return 0;
388 }

```

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++){

```

```

36     float _u = 0.5/((float)density-1);
37     _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]
      ↪ 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,
127             ↪ pointsMatrix[u+1][v].b);
128         glVertex3f(pointsMatrix[u+1][v].x, pointsMatrix[u+1][v].y,
129             ↪ pointsMatrix[u+1][v].z);
130
131         glColor3f(pointsMatrix[u][v].r, pointsMatrix[u][v].g, pointsMatrix[u]
132             ↪ [v].b);
133         glVertex3f(pointsMatrix[u][v].x, pointsMatrix[u][v].y, pointsMatrix[
134             ↪ u][v].z);
135         glColor3f(pointsMatrix[u][nextV].r, pointsMatrix[u][nextV].g,
136             ↪ pointsMatrix[u][nextV].b);
137         glVertex3f(pointsMatrix[u][nextV].x, pointsMatrix[u][nextV].y,
138             ↪ pointsMatrix[u][nextV].z);
139     }
140 }
141 glEnd();
142 break;
143 case 3:
144     glBegin(GL_TRIANGLES);
145     for(int u=0; u<density-1; u++){
146         //Obecnie trójkąty są CCW
147         if(u==0){
148             for(int v=0; v<density; v++){
149                 int nextV = (v + 1) % density;
150                 glColor3f(pointsMatrix[u][0].r, pointsMatrix[u][0].g, pointsMatr
151                     ↪ ix[u][0].b);
152                 glVertex3f(pointsMatrix[u][0].x, pointsMatrix[u][0].y, pointsMat
153                     ↪ rix[u][0].z);
154                 glColor3f(pointsMatrix[u+1][nextV].r, pointsMatrix[u+1][nextV].
155                     ↪ g, pointsMatrix[u+1][nextV].b);
156                 glVertex3f(pointsMatrix[u+1][nextV].x, pointsMatrix[u+1][nextV]
157                     ↪ .y, pointsMatrix[u+1][nextV].z);
158                 glColor3f(pointsMatrix[u+1][v].r, pointsMatrix[u+1][v].g, points
159                     ↪ Matrix[u+1][v].b);
160                 glVertex3f(pointsMatrix[u+1][v].x, pointsMatrix[u+1][v].y, point
161                     ↪ sMatrix[u+1][v].z);
162             }
163             continue;
164         }
165         if(u==density-2){
166             for(int v=0; v<density; v++){
167                 int nextV = (v + 1) % density;
168                 glColor3f(pointsMatrix[u+1][0].r, pointsMatrix[u+1][0].g, points
169                     ↪ Matrix[u+1][0].b);
170                 glVertex3f(pointsMatrix[u+1][0].x, pointsMatrix[u+1][0].y, point
171                     ↪ sMatrix[u+1][0].z);
172                 glColor3f(pointsMatrix[u][v].r, pointsMatrix[u][v].g, pointsMatr
173                     ↪ ix[u][v].b);
174                 glVertex3f(pointsMatrix[u][v].x, pointsMatrix[u][v].y, pointsMat
175                     ↪ rix[u][v].z);
176                 glColor3f(pointsMatrix[u][nextV].r, pointsMatrix[u][nextV].g, po
177                     ↪ intsMatrix[u][nextV].b);
178                 glVertex3f(pointsMatrix[u][nextV].x, pointsMatrix[u][nextV].y, p
179                     ↪ ointsMatrix[u][nextV].z);
180             }
181         }
182     }
183 }

```

```

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