Politechnika Wrocławska

Sprawozdanie 2

Ćwiczenie 4.Oświetlenie scen

Krzysztof Zalewa

9.12.2024

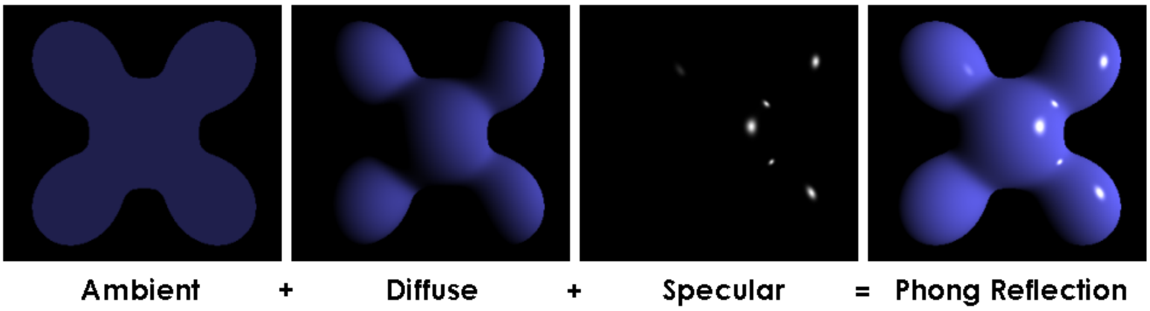
Spis treści

1 Wstęp teoretyczny

1.1 Model Phonga

Model Phonga lub oświetlenie Phonga 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 bezpośerednego oświetlenia
- **Światło otoczenia(ang. Ambient)** - jednorodne światło oświetlające cały obiekt

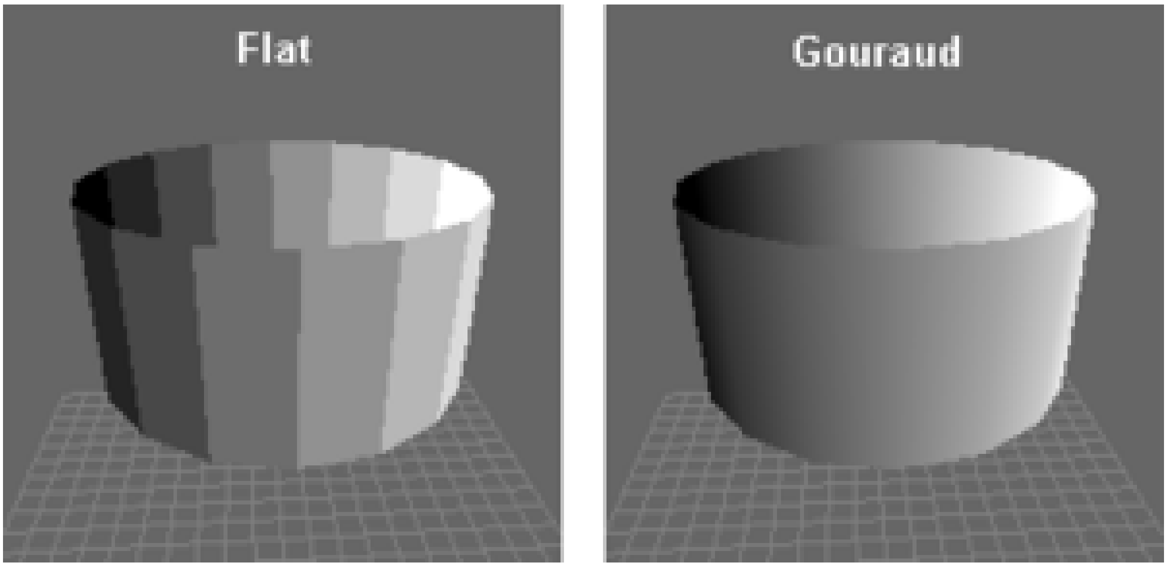


Rysunek 1: Model odbicia światła Phonga

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

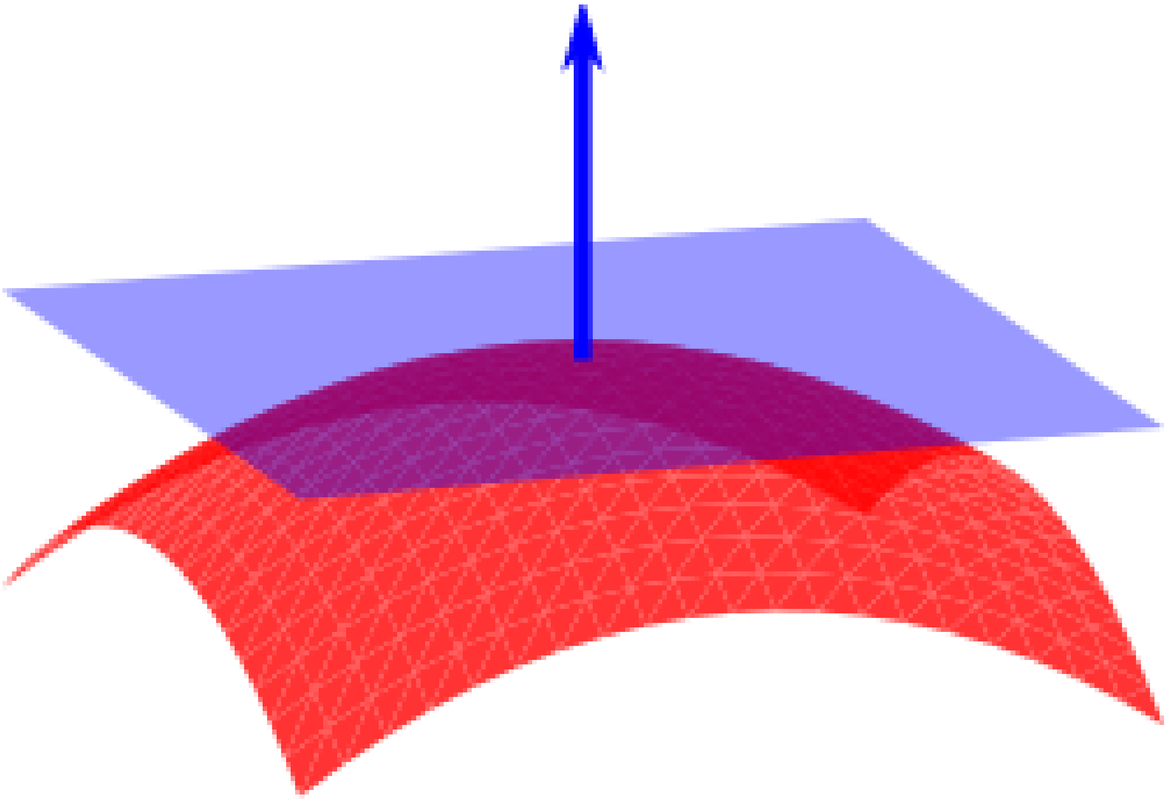
Cieniowanie Gourauda to metoda interpolacj polegająca na oświetlaniu wierzchołków w siatkach trójkątów i interpolacji wyników na cały trójkąt. Cieniowanie Gourauda jest uznawane za lepsze od cieniowania płaskiego i wymaga znacznie mniej obliczneń niż cieniowanie Phonga ale daje gorsze wyniki.



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" powirzchni 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 przeciwstawnych (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 swiatlem 1 (Czerwone)
F4 - tryb obrotu swiatlem 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 - Przybiliżenie 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>
#include "Egg.hpp"
#include "Light.hpp"
using namespace std;
HWND consoleWindow;
HWND glutWindow;

GLfloat deg = 0;
int sx =0,sy = 0,sz = 0;
bool spin = false;
bool drawTeapot = true,smooth = true;
int eggMode = 0,moveMode = 0;
float sensitivity = 0.01f;
float totalRotationX = 0.0f,totalRotationY = 0.0f,totalRotationZ = 0.0f;
float pix2angle,theta = 0.0f,phi = 0.0f;
float dX , dY;
int radius = 6,lastX = 0,lastY = 0, camOrientation = 1;
float cameraRotationX = radius * cosf((theta*(M_PI/180))) * cosf((phi*(M_PI/180)));
float cameraRotationY = radius * sinf((phi*(M_PI/180)));
float cameraRotationZ = radius * sinf((theta*(M_PI/180))) * cosf((phi*(M_PI/180)));
Light light1(GL_LIGHT0);
int light1Radius = 10;
float light1RotationX = radius * cosf((theta*(M_PI/180))) * cosf((phi*(M_PI/180)));
float light1RotationY = radius * sinf((phi*(M_PI/180)));
float light1RotationZ = radius * sinf((theta*(M_PI/180))) * cosf((phi*(M_PI/180)));
Light light2(GL_LIGHT1);
int light2Radius = 10;
float light2RotationX = radius * cosf((theta*(M_PI/180))) * cosf((phi*(M_PI/180)));
float light2RotationY = radius * sinf((phi*(M_PI/180)));
float light2RotationZ = radius * sinf((theta*(M_PI/180))) * cosf((phi*(M_PI/180)));

Egg egg(200);
void toggleFocusToConsole() {
    ShowWindow(glutWindow, SW_HIDE);
    ShowWindow(consoleWindow, SW_SHOWNORMAL);
    SetForegroundWindow(consoleWindow);
}

void toggleFocusToGLUT() {
    ShowWindow(consoleWindow, SW_HIDE);
    ShowWindow(glutWindow, SW_SHOWNORMAL);
    SetForegroundWindow(glutWindow);
}
void reset_rotation(){
    theta = 0.0f;
    phi = 0.0f;
    lastX = 0;
    lastY = 0;
    cameraRotationX = radius * cosf((theta*(M_PI/180.0f))) *
     ↪  cosf((phi*(M_PI/180.0f)));
```

```cpp
    cameraRotationY = radius * sinf((phi*(M_PI/180.0f)));
    cameraRotationZ = radius * sinf((theta*(M_PI/180.0f))) * cosf((phi*(M_PI/180.0f)));

    light1RotationX = light1Radius * cosf((theta*(M_PI/180))) * cosf((phi*(M_PI/180)));
    light1RotationY = light1Radius * sinf((phi*(M_PI/180)));
    light1RotationZ = light1Radius * sinf((theta*(M_PI/180))) * cosf((phi*(M_PI/180)));

    light2RotationX = light2Radius * cosf((theta*(M_PI/180))) * cosf((phi*(M_PI/180)));
    light2RotationY = light2Radius * sinf((phi*(M_PI/180)));
    light2RotationZ = light2Radius * sinf((theta*(M_PI/180))) * cosf((phi*(M_PI/180)));
}
string bool_to_string(bool convert){
    if(convert){
        return "true";
    }else{
        return "false";
    }
}
void printControls(){
    cout<<"==============================\n";
    cout<<"F1 - tryb obrotu obiektu";
    cout<<"F2 - tryb obrotu kamery";
    cout<<"F3 - tryb obrotu swiatlem 1 (Czerwone)";
    cout<<"F4 - tryb obrotu swiatlem 2 (Zielone)";
    cout<<"ESC - Powrot do menu (okno konsolowe)\n";
    cout<<"Nalezy nacisnac i przytrzymac PPM\n";
    cout<<"Ruch myszy w osi X - Obrot osi X\n";
    cout<<"Ruch myszy w osi Y - Obrot osi Y\n";
    cout<<"Scroll up - Przybilizenie obiektu\n";
    cout<<"Scroll down - Oddalenie obiektu\n";
    cout<<"Nacisnij Enter zeby kontynuowac\n"<<flush;
    cin.get();
    cin.get();
}
void axis(){
    glBegin(GL_LINES);

    glColor3f(1.0, 0.0, 0.0);
    glVertex3f(-5.0, 0.0, 0.0);
    glVertex3f(5.0, 0.0, 0.0);

    glColor3f(0.0, 1.0, 0.0);
    glVertex3f(0.0, -5.0, 0.0);
    glVertex3f(0.0, 5.0, 0.0);

    glColor3f(0.0, 0.0, 1.0);
    glVertex3f(0.0, 0.0, -5.0);
    glVertex3f(0.0, 0.0, 5.0);

    glEnd();
}
```

```
void printOptions();
void menu();
void printOptions(){
    int density = egg.getDensity();
    bool color = egg.getColor();
    float scale = egg.getScale();
    float pointSize = egg.getPointSize();
    cout<<"==============================\n";
    cout<<"1.Skala obiektow: "<<scale<<"\n";
    cout<<"2.Ilosc punktow: "<<density<<"\n";
    cout<<"3.Rysowanie w kolorze: "<<bool_to_string(color)<<"\n";
    cout<<"4.Promien kamery: "<<radius<<"\n";
    cout<<"5.Czulosc myszki: "<<sensitivity<<"\n";
    cout<<"6.Rozmiar punktow: "<<pointSize<<"\n";
    cout<<"7.Wroc do menu"<<"\n";
    cout<<"> ";
    int x;
    cin>>x;
    switch (x){
    case 1:
        cout<<"Nowa skala\n";
        cout<<"> ";
        cin>>scale;
        egg.setScale(scale);
        printOptions();
        break;
    case 2:
        cout<<"Nowa gestosc\n";
        cout<<"> ";
        cin>>density;
        egg.setDensity(density);
        printOptions();
        break;
    case 3:
        color =! color;
        egg.setColor(color);
        egg.generateMatrix();
        printOptions();
        break;
    case 4:
        cout<<"Nowy promien kamery\n";
        cout<<"> ";
        cin>>radius;
        printOptions();
        break;
    case 5:
        cout<<"Nowa predkosc kamery\n";
        cout<<"> ";
        cin>>sensitivity;
        printOptions();
        break;
    case 6:
        cout<<"Nowy rozmiar punktow\n";
        cout<<"> ";
        cin>>pointSize;
        egg.setPointSize(pointSize);
        printOptions();
        break;
    case 7:
        menu();
        break;
    }
}
void menu(){
    toggleFocusToConsole();
    reset_rotation();
    cout<<"==============================\n";
    cout<<"1. Narysuj czajnik\n";
    cout<<"2. Narysuj jajko (punkty)\n";
    cout<<"3. Narysuj jajko (linie)\n";
    cout<<"4. Narysuj jajko (trojkaty) \n";
    cout<<"5. Opcje\n";
    cout<<"6. Kontrola\n";
    cout<<"7. Zakoncz program\n";
    cout<<"> ";
    int x;
    cin>>x;
    switch (x){
    case 1:
        drawTeapot = true;
        break;
    case 2:
        drawTeapot = false;
        eggMode = 1;
        break;
    case 3:
        drawTeapot = false;
        eggMode = 2;
        break;
    case 4:
        drawTeapot = false;
        eggMode = 3;
        break;
    case 5:
        printOptions();
        break;
    case 6:
        printControls();
        menu();
        break;
    case 7:
        exit(0);
        break;
    default:
        cout<<"Podano nieporawny znak\n";
        menu();
        break;
    }
    toggleFocusToGLUT();
    glutPostRedisplay();
}
void specialKey(int key,int x,int y){
    switch (key){
    //F1 - Ruch obiektu
    case GLUT_KEY_F1:
        moveMode = 0;
        break;
    //F2 - Ruch kamery
    case GLUT_KEY_F2:
        moveMode = 1;
        break;
    //F3 - Ruch światła 1
    case GLUT_KEY_F3:
        moveMode = 2;
        break;
    //F4 - Ruch światła 2
    case GLUT_KEY_F4:
        moveMode = 3;
        break;
    default:
        break;
    }
}
void normalKey(u_char key,int x,int y){
    switch (key)
    {
    case 27:
        menu();
        break;
    default:
        break;
    }
    if (sx == 0 && sy == 0 && sz == 0) {
        glutIdleFunc(nullptr);
    }
}

void mouse(int x, int y){
    dY = y - lastY;
    lastY = y;
    dX = x - lastX;
    lastX = x;
    theta += dX * pix2angle;
    phi += dY * pix2angle;
    if (phi > 89.0f) {phi = 89.0f;}
    if (phi < -89.0f) {phi = -89.0f;}
    switch(moveMode){
        case 0:
            totalRotationX += dY;
            totalRotationY += dX;
            totalRotationZ += atan2f(dY,dX);
            break;
        case 1:
            cameraRotationX = radius * cosf((theta*(M_PI/180.0f)) *
            ↪  cosf((phi*(M_PI/180.0f)));
            cameraRotationY = radius * sinf((phi*(M_PI/180.0f)));
            cameraRotationZ = radius * sinf((theta*(M_PI/180.0f)) *
            ↪  cosf((phi*(M_PI/180.0f)));
            break;
        case 2:
            light1RotationX = light1Radius * cosf((theta*(M_PI/180))) * cosf((phi*(M_PI/180)));
            light1RotationY = light1Radius * sinf((phi*(M_PI/180)));
            light1RotationZ = light1Radius * sinf((theta*(M_PI/180))) * cosf((phi*(M_PI/180)));
            break;
        case 3:
            light2RotationX = light2Radius * cosf((theta*(M_PI/180))) * cosf((phi*(M_PI/180)));
            light2RotationY = light2Radius * sinf((phi*(M_PI/180)));
            light2RotationZ = light2Radius * sinf((theta*(M_PI/180))) * cosf((phi*(M_PI/180)));
            break;
    }
    lastX = x;
    lastY = y;
    glutPostRedisplay();
}
void mouseWheel(int button, int dir, int x, int y){

    if (dir > 0){
        radius -= 1;
    }else{
        radius += 1;
    }
    if(radius>=10){
        radius=10;
    }
    if(radius<=1){
        radius=1;
    }
    glutPostRedisplay();
}
void display() {
    GLfloat lPos1[] = {light1RotationX,light1RotationY,light1RotationZ,1};//x,y,z,czy światło jest odległe
    GLfloat lPos2[] = {light2RotationX,light2RotationY,light2RotationZ,1};
    GLfloat col[] = {1,0,0,1};
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    gluLookAt(cameraRotationX,cameraRotationY,cameraRotationZ,0,0,0,0,camOrientation,0);//Ustawienie kamery
    light1.setPosition(lPos1);
    light2.setPosition(lPos2);
    glEnable(GL_COLOR_MATERIAL);

    glRotatef(totalRotationX, 1.0f, 0.0f, 0.0f);
    glRotatef(totalRotationY, 0.0f, 1.0f, 0.0f);
    glRotatef(totalRotationZ, 0.0f, 0.0f, 1.0f);
    axis();
    if(drawTeapot){
        glutSolidTeapot(1);
    }else{
        egg.initMaterial();
        egg.draw(eggMode);
    }

    glutSwapBuffers();
}
void Init() {
    pix2angle = 360.0/800;
    egg.generateMatrix();
    glEnable(GL_DEPTH_TEST); //bez tego frontalna sciana nadpisuje tylnią
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glFrustum(-1,1,-1,1,2,20);
    glMatrixMode(GL_MODELVIEW);
    // Włącza culling, czyli pomijanie tylnych ścianek
    glEnable(GL_CULL_FACE);
    // Ustawia kierunek frontowych ścianek jako przeciwny do ruchu wskazówek zegara
    glFrontFace(GL_CW);
    // Ustawia pomijanie tylnych ścianek
    glCullFace(GL_BACK);
    // Kolor stały
    light1.setColor(1.0,0.0,0.0);
    light2.setColor(0.0,1.0,0.0);
    light1.initLight();
    light2.initLight();
    //Drugie światło
    glShadeModel(GL_SMOOTH);
    glEnable(GL_LIGHTING); //Włączenie oświetlenia
    glEnable(GL_LIGHT0); //Dodanie źródła światła
    glEnable(GL_LIGHT1);

}
// Sprawko do 15 w pon
// W sprawku Phong,Gouraud i wektor normalny
// TODO - Kąty przestzenne dla lamp radiany określają stożek świecenia światła
// ADS - (Nie odpowiada fizyce) światło nie jest jednorodne
// Ambient - ogólnie wszędzie bezkierunkowe
// Diffuse - kąt padania = kąt odbicia
// Specular - odbicia lustrzane
// TODO - Każdemu punktowi dodać ADS składowa to sposób w jaki obiekt odbija ads
// Tylko jednokrotne odbicie
// TODO - Światło z reflektora ma drogę reflektor/obiekt(Tłumienie) obiekt/kamera
// TODO - cieniowanie Phonga i Gourauda
// Różnią się liczenie wektora normalnego
int main(int argc, char** argv){
    consoleWindow = GetConsoleWindow();
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(800,800);
    glutCreateWindow("Lab 3 - Czajnik i Jajko");
    glutWindow = FindWindowW(NULL,L"Lab 3 - Czajnik i Jajko");
    Init();
    glutDisplayFunc(display);
    glutIdleFunc(nullptr);
    glutKeyboardFunc(normalKey);
    glutSpecialFunc(specialKey);
    glutMotionFunc(mouse);
    glutMouseWheelFunc(mouseWheel);
    menu();

    glutMainLoop();
    system("pause");
    return 0;
}
```

Fragment kodu 1: Fragment kodu z programu

.

```
#include <math.h>
#include <GL/glu.h>
#define FREEGLUT_STATIC
#include <GL/freeglut.h>
#include "Egg.hpp"
using namespace std;

float Egg::randFloat(){
    return (float)rand()/(float)(RAND_MAX);
}
Egg::Egg(int density ) : density(density){
    pointsMatrix.resize(density,vector<pointsRgb>(density));
}
vector<vector<pointsRgb>> Egg::getPointsMatrix(){
    return pointsMatrix;
}
point Egg::generateNormalVect(int u,int v){
    float x_u = (-450*pow(u,4) + 900*pow(u,3) - 810*pow(u,2) + 360*u - 45) *
    ↪   cos(M_PI*v);
    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);
    float y_u = 640*pow(u,3) - 960*pow(u,2) + 320*u;
    float y_v = 0;
    float z_u = (-450*pow(u,4) + 900*pow(u,3) - 810*pow(u,2) + 360*u - 45) *
    ↪   sin(M_PI*v);
    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);
    point newPoint;
    newPoint.x = y_u * z_v - z_u * y_v;
    newPoint.y = z_u * x_v - x_u * z_v;
    newPoint.z = x_u * y_v - y_u * x_v;
    float length = sqrt(newPoint.x*newPoint.x + newPoint.y*newPoint.y +
    ↪   newPoint.z*newPoint.z);
    newPoint.x /= length;
    newPoint.y /= length;
    newPoint.z /= length;
    return newPoint;
}
void Egg::generateMatrix(){
    for(int u=0;u<(density);u++){
        float _u = 0.5/((float)density-1);
        _u *= u;
        if(u==density-1){
            pointsMatrix[u][0].y = scale*((160*pow(_u,4)) - (320*pow(_u,3)) + (160 * pow(_u,2)) - 5);
            //Białe jajko
            pointsMatrix[u][0].r = 1.0f;
            pointsMatrix[u][0].g = 1.0f;
            pointsMatrix[u][0].b = 1.0f;
            point newPoint = generateNormalVect(u,0);
            pointsMatrix[u][0].nx = newPoint.x;
            pointsMatrix[u][0].ny = newPoint.y;
            pointsMatrix[u][0].nz = newPoint.z;
            break;
        }
        for(int v=0;v<density;v++){
            float _v = v/((float)density);
            _v *= 2.0f;
            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));
            pointsMatrix[u][v].y = scale*(160*pow(_u,4) - 320*pow(_u,3) + 160 * pow(_u,2) - 5);
            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));
            //Białe jajko
            pointsMatrix[u][v].r = 1.0f;
            pointsMatrix[u][v].g = 1.0f;
            pointsMatrix[u][v].b = 1.0f;
            point newPoint = generateNormalVect(u,v);
            pointsMatrix[u][v].nx = newPoint.x;
            pointsMatrix[u][v].ny = newPoint.y;
            pointsMatrix[u][v].nz = newPoint.z;
        }
    }
}
void Egg::initMaterial(){
    float mat_ambient[4] = {0.3f, 0.3f, 0.3f, 1.0f};
    float mat_diffuse[4] = {0.6f, 0.3f, 0.3f, 1.0f};
    float mat_specular[4] = {1.0f, 1.0f, 1.0f, 1.0f};
    float mat_shininess = 10.0f;
    glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, mat_ambient);
    glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, mat_diffuse);
    glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, mat_specular);
    glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, mat_shininess);
}
void Egg::draw(int model){
    switch (model)
    {
    case 1:
        glPointSize(pointSize);
        glBegin(GL_POINTS);
        for(int u=0;u<density-1;u++){
            if(u==0){
                glColor3f(pointsMatrix[u][0].r,pointsMatrix[u][0].g,pointsMatrix[u][0].b);
                glVertex3f(pointsMatrix[u][0].x,pointsMatrix[u][0].y,pointsMatrix[u][0].z);
                continue;
            }
            if(u==density-2){
                glColor3f(pointsMatrix[u+1][0].r,pointsMatrix[u+1][0].g,pointsMatrix[u+1][0].b);
                glVertex3f(pointsMatrix[u+1][0].x,pointsMatrix[u+1][0].y,pointsMatrix[u+1][0].z);
                break;
            }
            for(int v=0;v<density;v++){
                glColor3f(pointsMatrix[u][v].r,pointsMatrix[u][v].g,pointsMatrix[u][v].b);
                glVertex3f(pointsMatrix[u][v].x,pointsMatrix[u][v].y,pointsMatrix[u][v].z);
            }
        }
        glEnd();
        break;
    case 2:
        glBegin(GL_LINES);
        for(int u=0;u<density-1;u++){
            if(u==0){
                for(int v=0;v<density;v++){
                    glColor3f(pointsMatrix[u][0].r,pointsMatrix[u][0].g,pointsMatrix[u][0].b);
                    glVertex3f(pointsMatrix[u][0].x,pointsMatrix[u][0].y,pointsMatrix[u][0].z);
                    glColor3f(pointsMatrix[u+1][v].r, pointsMatrix[u+1][v].g, pointsMatrix[u+1][v].b);
                    glVertex3f(pointsMatrix[u+1][v].x, pointsMatrix[u+1][v].y, pointsMatrix[u+1][v].z);
                }
                continue;
            }
            if(u==density-2){
                for(int v=0;v<density;v++){
                    glColor3f(pointsMatrix[u+1][0].r,pointsMatrix[u+1][0].g,pointsMatrix[u+1][0].b);
                    glVertex3f(pointsMatrix[u+1][0].x,pointsMatrix[u+1][0].y,pointsMatrix[u+1][0].z);
                    glColor3f(pointsMatrix[u][v].r, pointsMatrix[u][v].g, pointsMatrix[u][v].b);
                    glVertex3f(pointsMatrix[u][v].x, pointsMatrix[u][v].y, pointsMatrix[u][v].z);
                }
                break;
            }
            for(int v=0;v<density;v++){
                int nextV = (v + 1) % density;
                glColor3f(pointsMatrix[u][v].r,pointsMatrix[u][v].g,pointsMatrix[u][v].b);
                glVertex3f(pointsMatrix[u][v].x,pointsMatrix[u][v].y,pointsMatrix[u][v].z);
                glColor3f(pointsMatrix[u+1][v].r, pointsMatrix[u+1][v].g, pointsMatrix[u+1][v].b);
                glVertex3f(pointsMatrix[u+1][v].x, pointsMatrix[u+1][v].y, pointsMatrix[u+1][v].z);

                glColor3f(pointsMatrix[u][v].r,pointsMatrix[u][v].g,pointsMatrix[u][v].b);
                glVertex3f(pointsMatrix[u][v].x,pointsMatrix[u][v].y,pointsMatrix[u][v].z);
                glColor3f(pointsMatrix[u][nextV].r, pointsMatrix[u][nextV].g, pointsMatrix[u][nextV].b);
                glVertex3f(pointsMatrix[u][nextV].x, pointsMatrix[u][nextV].y, pointsMatrix[u][nextV].z);

            }
        }
        glEnd();
        break;
    case 3:
        glBegin(GL_TRIANGLES);
        for(int u=0;u<density-1;u++){
            //Obecnie trójkąty są CCW
            if(u==0){
                for(int v=0;v<density;v++){
                    int nextV = (v + 1) % density;
                    glColor3f(pointsMatrix[u][0].r,pointsMatrix[u][0].g,pointsMatrix[u][0].b);
                    glVertex3f(pointsMatrix[u][0].x,pointsMatrix[u][0].y,pointsMatrix[u][0].z);
                    glColor3f(pointsMatrix[u+1][nextV].r,pointsMatrix[u+1][nextV].g,pointsMatrix[u+1][nextV].b);
                    glVertex3f(pointsMatrix[u+1][nextV].x,pointsMatrix[u+1][nextV].y,pointsMatrix[u+1][nextV].z);
                    glColor3f(pointsMatrix[u+1][v].r,pointsMatrix[u+1][v].g,pointsMatrix[u+1][v].b);
                    glVertex3f(pointsMatrix[u+1][v].x,pointsMatrix[u+1][v].y,pointsMatrix[u+1][v].z);
                }
                continue;
            }
            if(u==density-2){
                for(int v=0;v<density;v++){
                    int nextV = (v + 1) % density;
                    glColor3f(pointsMatrix[u+1][0].r,pointsMatrix[u+1][0].g,pointsMatrix[u+1][0].b);
                    glVertex3f(pointsMatrix[u+1][0].x,pointsMatrix[u+1][0].y,pointsMatrix[u+1][0].z);
                    glColor3f(pointsMatrix[u][v].r,pointsMatrix[u][v].g,pointsMatrix[u][v].b);
                    glVertex3f(pointsMatrix[u][v].x,pointsMatrix[u][v].y,pointsMatrix[u][v].z);
                    glColor3f(pointsMatrix[u][nextV].r,pointsMatrix[u][nextV].g,pointsMatrix[u][nextV].b);
                    glVertex3f(pointsMatrix[u][nextV].x,pointsMatrix[u][nextV].y,pointsMatrix[u][nextV].z);
                }
                break;
            }
            for(int v=0;v<density;v++){
                int nextV = (v + 1) % density;
                glNormal3f(pointsMatrix[u][v].nx,pointsMatrix[u][v].ny,pointsMatrix[u][v].nz);
                glColor3f(pointsMatrix[u][v].r,pointsMatrix[u][v].g,pointsMatrix[u][v].b);
                glVertex3f(pointsMatrix[u][v].x,pointsMatrix[u][v].y,pointsMatrix[u][v].z);

                glNormal3f(pointsMatrix[u+1][nextV].nx,pointsMatrix[u+1][nextV].ny,pointsMatrix[u+1][nextV].nz);
                glColor3f(pointsMatrix[u+1][nextV].r,pointsMatrix[u+1][nextV].g,pointsMatrix[u+1][nextV].b);
                glVertex3f(pointsMatrix[u+1][nextV].x, pointsMatrix[u+1][nextV].y, pointsMatrix[u+1][nextV].z);

                glNormal3f(pointsMatrix[u+1][v].nx,pointsMatrix[u+1][v].ny,pointsMatrix[u+1][v].nz);
                glColor3f(pointsMatrix[u+1][v].r,pointsMatrix[u+1][v].g,pointsMatrix[u+1][v].b);
                glVertex3f(pointsMatrix[u+1][v].x, pointsMatrix[u+1][v].y, pointsMatrix[u+1][v].z);

                glNormal3f(pointsMatrix[u+1][nextV].nx,pointsMatrix[u+1][nextV].ny,pointsMatrix[u+1][nextV].nz);
                glColor3f(pointsMatrix[u+1][nextV].r,pointsMatrix[u+1][nextV].g,pointsMatrix[u+1][nextV].b);
                glVertex3f(pointsMatrix[u+1][nextV].x, pointsMatrix[u+1][nextV].y, pointsMatrix[u+1][nextV].z);

                glNormal3f(pointsMatrix[u][v].nx,pointsMatrix[u][v].ny,pointsMatrix[u][v].nz);
                glColor3f(pointsMatrix[u][v].r,pointsMatrix[u][v].g,pointsMatrix[u][v].b);
                glVertex3f(pointsMatrix[u][v].x,pointsMatrix[u][v].y,pointsMatrix[u][v].z);

                glNormal3f(pointsMatrix[u][nextV].nx,pointsMatrix[u][nextV].ny,pointsMatrix[u][nextV].nz);
                glColor3f(pointsMatrix[u][nextV].r,pointsMatrix[u][nextV].g,pointsMatrix[u][nextV].b);
                glVertex3f(pointsMatrix[u][nextV].x, pointsMatrix[u][nextV].y, pointsMatrix[u][nextV].z);
            }
        }
        glEnd();
        break;
    }
}
//Setters
void Egg::setDensity(int newDensity){
    density = newDensity;
    pointsMatrix.resize(density,vector<pointsRgb>(density));
    generateMatrix();
}
void Egg::setColor(float newColor){color = newColor;}
void Egg::setScale(float newScale){scale = newScale;}
void Egg::setPointSize(float newPointSize){pointSize = newPointSize;}
```

```
//Getters
int Egg::getDensity(){return density;}
float Egg::getColor(){return color;}
float Egg::getScale(){return scale;}
float Egg::getPointSize(){return pointSize;}
Egg::~Egg(){

}
```

Fragment kodu 2: Kod Egg.cpp

.

```
#include <GL/glu.h>
#include <math.h>
#define FREEGLUT_STATIC
#include <GL/freeglut.h>
#include "Light.hpp"
using namespace std;

void Light::initLight(){
    glLightfv(lightID, GL_AMBIENT, light_ambient);
    glLightfv(lightID, GL_DIFFUSE, light_diffuse);
    glLightfv(lightID, GL_SPECULAR, light_specular);
    glLightf(lightID, GL_CONSTANT_ATTENUATION, att_constant);
    glLightf(lightID, GL_LINEAR_ATTENUATION, att_linear);
    glLightf(lightID, GL_QUADRATIC_ATTENUATION, att_quadratic);
}
Light::Light(GLenum newLightID){
    lightID = newLightID;
}
void Light::normalize(GLfloat* v) {
    GLfloat length = sqrt(v[0] * v[0] + v[1] * v[1] + v[2] * v[2]);
    if (length > 0.0f) {
        v[0] /= length;
        v[1] /= length;
        v[2] /= length;
    }
}
void Light::setPosition(GLfloat lPos[]){
    GLfloat light_direction[3];
    light_direction[0] = -lPos[0];
    light_direction[1] = -lPos[1];
    light_direction[2] = -lPos[3];
    normalize(light_direction);
    glLightfv(lightID,GL_POSITION,lPos);;
    glLightfv(lightID,GL_SPOT_DIRECTION,light_direction);
    glLightf(lightID, GL_SPOT_CUTOFF, 25.0f);
    glLightf(lightID, GL_SPOT_EXPONENT, 2.0f);
}
void Light::setColor(float r,float g,float b){
    light_ambient[0] = r;
    light_ambient[1] = g;
    light_ambient[2] = b;
    light_diffuse[0] = r;
    light_diffuse[1] = g;
    light_diffuse[2] = b;
    light_specular[0] = r;
    light_specular[1] = g;
    light_specular[2] = b;
}
```

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