Ten moduł cieniujący implementuje Model Fonga do rozjaśniania obiektów na podstawie tekstury i wejściowych parametrów oświetlenia

W F\_glass ustawiamy stały kolor szkła (niebieski), a V\_glass przedstawia końcową pozycję vezholka.

F\_water liczy kolor piksela i oswietlanie, a V\_water liczy wierzcholki i liczy wektory światła i widoku.

В F\_phong shader wykonuje model oświetlenia Phonga, oblicza składniki oświetlenia (ambient, dyfuzyjne i kierunkowe) na podstawie wektorów światła, normalnej, widoku i współczynników odbicia, а V\_phong liczy wierzcholki I wiektory swiatla I widoku.

**F\_glass.glsl – fragment shader**

#version 330

out vec4 pixelColor; //deklaracja zmiennej wyjsciowej pixelColor, oznancza kolor

void main()

{

pixelColor = vec4(0.2,0.2,1,0.2);//robimy 4-skladnikowy wektor – kolor niebieski

}

ten shader przypisuje stały kolor (niebieski) do każdego piksela renderowanego obrazu.

**V\_glass.glsl – vertex shader**

#version 330

in vec4 vertex; //deklaracja zmiennej, reprezentuje wspolrzędne wierzcholka

uniform mat4 M; //zmienna globalna transformacji modelu

uniform mat4 V; // zmienna globalna widoku

uniform mat4 P; // zmienna globalna projekcji

void main()

{

gl\_Position = P\*V\*M\*vertex; //pozycja wierzcholka po transformacji. aby określić, gdzie wierzchołek powinien być wyświetlany na ekranie

}

ten shader wykonuje transformację wierzchołka, mnożąc go przez macierze M, V i P, a następnie przypisuje wynik do zmiennej "gl\_Position", która reprezentuje końcową pozycję wierzchołka.

**f\_phong.glsl**

#version 330

uniform sampler2D textureMap; // tekstura używana w shaderze do próbkowania kolorów

in vec2 iTexCoord; // reprezentuje współrzędne tekstury

in vec4 l1; // wektor światła

in vec4 l2; // wektor światła

in vec4 v; // wektor widoku

in vec4 n; // normalna

out vec4 pixelColor; // przechowuje końcowy kolor piksela, który zostanie zwrócony przez shader

void main(void) {

vec4 texColor=texture(textureMap,iTexCoord);// funkcja "texture()" pobiera kolor z tekstury na podstawie współrzędnych "iTexCoord"

vec4 n\_l1 = normalize(l1); //znormalizowany wektor swiatla

vec4 n\_l2 = normalize(l2); // znormalizowany wektor swiatla

vec4 n\_v = normalize(v); // znormalizowany wektor widoku

vec4 n\_n = normalize(n); // znormalizowana normalna

vec4 n\_r1 = reflect(n\_l1, n\_n); // vektory odbite- odbicie swiatla od normalnej

vec4 n\_r2 = reflect(n\_l2, n\_n); // odbicie swiatla od normalnej

float nl1 = max(0, dot(n\_n,n\_l1)); // iloczyny skalarne n i l miedzy norm i swiatlem

float nl2 = max(0, dot(n\_n,n\_l2));

float rv1 = pow(max(0, dot(n\_r1,n\_v)), 50); // iloczyny skalarne r i v miedzy wektorami odbitymi a wektorem widoku

float rv2 = pow(max(0, dot(n\_r2,n\_v)), 50);

vec4 kd = vec4(1,1,1,1); // reprezentuja wspolczynniki odbicia defuzyjnego I kierunkowego

vec4 ks = vec4(1,1,1,1);//

vec4 La = texColor \* vec4(0.5,0.5,0.5,1); //kolor dla skladnikow ambient

vec4 Ld = texColor \* vec4(1,1,1,1); //kolor dla dyfuz. skladnikow

vec4 Ls = texColor \* vec4(1,1,1,1); // kolor dla kierunkowych skladnikow

pixelColor = La + Ld\*vec4(kd.rgb \* nl1, kd.a) + Ld\*vec4(kd.rgb \* nl2, kd.a) + Ls\*vec4(ks.rgb \* rv1, 0) + Ls\*vec4(ks.rgb \* rv2, 0); //laczymy kolory–reprezentuje kolor koncowy

}

ten shader wykonuje model oświetlenia Phonga, obliczając różne składniki oświetlenia (ambient, dyfuzyjne i kierunkowe) na podstawie wektorów światła, normalnej, widoku i współczynników odbicia

V\_phong.glsl

#version 330

uniform mat4 P; //macierz projekcji

uniform mat4 V; //macierz widoku

uniform mat4 M; //macierz modelu

uniform vec4 light1; //reprezentuje pozycje swiatla

uniform vec4 light2; //reprezentuje pozycje swiatla

in vec4 vertex; // zmienna reprezentuje wierzcholek

in vec4 normal; // zmienna normalna

in vec2 texCoord; // wspolrzedne tekstury dla wierzchoka

out vec4 l1; // wektor swiatla

out vec4 l2; // wektor swiatla

out vec4 v; // wektor widoku

out vec4 n; // normalna

out vec2 iTexCoord; // wspolrzedne tekstury ktore zostana przekazane do f\_shader

void main(void) {

iTexCoord = texCoord; // przypis texCoord do iTexCoord I tekstury beda dostepne w f\_shader

l1 = normalize(V\*light1 - V\*M\*vertex); // wektor od pozycji swiatla do oswietlanego wierzcholka

l2 = normalize(V\*light2 - V\*M\*vertex); // wektor od pozycji swiatla do oswietlanego wierzcholka

v = normalize(V\*vec4(0, 0, 0, 1) - V\*M\*vertex); // wektor kierunku od obserwatora do oswietlanego wierzcholka

n = normalize(V\*M\*normal); // wektor normalny do wspolrzednych widoku

gl\_Position=P\*V\*M\*vertex; //pozycja wierzcholka w przestrzeni projekcyjnej

}

**F\_water.glsl**

#version 330

uniform sampler2D textureMap; //zmienna reprezentuje teksturę, na którą zostaną nałożone obliczone kolory pikseli

in vec2 iTexCoord; //wspolrzedne tekstury piksela

in vec4 l1; //wektor kierunku swiatla przekazane z v\_shader

in vec4 l2; // wektor kierunku swiatla

in vec4 v; // wektor widoku

in vec4 n; // normalna

out vec4 pixelColor; // zmienna reprezentuje końcowy kolor piksela, który zostanie przekazany do bufora kolorów

void main(void) {

vec4 texColor=texture(textureMap,iTexCoord); //pobieramy kolor tekstury

vec4 n\_l1 = normalize(l1); //obliczamy znormalizowane tekstury

vec4 n\_l2 = normalize(l2);

vec4 n\_v = normalize(v);

vec4 n\_n = normalize(n);

vec4 n\_r1 = reflect(n\_l1, n\_n); //obliczenie wektorow odbitych za pomocą funkcji "reflect", która oblicza wektor odbity dla danego wektora i normalnej

vec4 n\_r2 = reflect(n\_l2, n\_n);

float nl1 = max(0, dot(n\_n,n\_l1));//obliczamy iloczyny skalarne miedzy wektorami normalnymi I kierunku swiatla

float nl2 = max(0, dot(n\_n,n\_l2));

float rv1 = pow(max(0, dot(n\_r1,n\_v)), 50); //obliczamy iloczyny skalarne miedzy wektorami normalnymi I kierunku widoku

float rv2 = pow(max(0, dot(n\_r2,n\_v)), 50);

vec4 kd = vec4(0.7, 0.85, 1, 1); //reprezentują współczynniki rozproszenia światła

vec4 ks = vec4(0.7, 0.85, 1, 1); //reprezentują współczynniki odbicia światła

//reprezentujące składowe oświetlenia ambient, rozproszonego i odbitego na podstawie koloru tekstury

vec4 La = texColor \* vec4(0.5,0.7,1,1);

vec4 Ld = texColor \* vec4(1,1,1,1);

vec4 Ls = texColor \* vec4(1,1,1,1);

//sumujemy te składowe w odpowiednich proporcjach i przypisujemy wynik do zmiennej "pixelColor"

pixelColor = La + Ld\*vec4(kd.rgb \* nl1, kd.a) + Ld\*vec4(kd.rgb \* nl2, kd.a) + Ls\*vec4(ks.rgb \* rv1, 0) + Ls\*vec4(ks.rgb \* rv2, 0);

}

ten shader oblicza kolor piksela na podstawie tekstury i uwzględnia oświetlenie ambient, rozproszone i odbite.

**V\_water.glsl**

#version 330

uniform mat4 P; //macierz projekcji

uniform mat4 V; //macierz widoku

uniform mat4 M; //macierz modelu

uniform vec4 light1; //reprezentuje pozycje swiatla

uniform vec4 light2; //reprezentuje pozycje swiatla

in vec4 vertex; //reprezentuje współrzędne wierzchołka

in vec4 normal; //wektor normalny

in vec2 texCoord; //reprezentuje współrzędne tekstury dla wierzchołka

out vec4 l1; //wektor kierunku swiatla

out vec4 l2; //wektor kierunku swiatla

out vec4 v; // wektor od obserwatora do wierzcholka

out vec4 n; // wektor normalny wierzcholka

out vec2 iTexCoord;

void main(void) {

iTexCoord = texCoord; //przypisujemy wartości zmiennych "texCoord" do "iTexCoord", aby przekazać je do f\_shadera

//obliczamy znormalizowane wektory poprzez transformację ich do przestrzeni widoku i modelu, a następnie normalizację:

l1 = normalize(V\*light1 - V\*M\*vertex);

l2 = normalize(V\*light2 - V\*M\*vertex);

v = normalize(V\*M\*vec4(0, 0, 0, 1) - V\*M\*vertex);

n = normalize(V\*M\*normal);

gl\_Position=P\*V\*M\*vertex;// obliczamy pozycje wierzcholka w przestrzeni projekcji

}

Ten shader przekształca wierzchołki sceny 3D na przestrzeń ekranu, oblicza wektory kierunku światła i widoku, oraz przekazuje je do fragment shadera

Main

void drawGlass(glm::mat4 P, glm::mat4 V, glm::mat4 M) { //wykorzystany do renderinga szkla z wykorzystaniem shaderow I danych wierzcholkow I normalnych

glassShader->use(); //wykorzystanie objekta shader do renderinga objekta

glUniformMatrix4fv(glassShader->u("P"), 1, false, glm::value\_ptr(P)); //przekazuje macierz projekcji do shadera do zmiennej P

glUniformMatrix4fv(glassShader->u("V"), 1, false, glm::value\_ptr(V));

glUniformMatrix4fv(glassShader->u("M"), 1, false, glm::value\_ptr(M));

glEnableVertexAttribArray(glassShader->a("vertex")); //wlacza wykorzyst. Atrybuta vertex w shaderze

glVertexAttribPointer(glassShader->a("vertex"), 4, GL\_FLOAT, false, 0, myCubeVertices); //wskaznik na dane wierzcholkow

glEnableVertexAttribArray(glassShader->a("normal")); //wlacza wykorzyst. Atrybuta normal w shaderze

glVertexAttribPointer(glassShader->a("normal"), 4, GL\_FLOAT, false, 0, myCubeVertexNormals); //wskaznik na dane normalnych wierzcholkow

glDrawArrays(GL\_TRIANGLES, 0, myCubeVertexCount); //rysuje trojkat na podstawie danych wierzcholkow i normalnych

glDisableVertexAttribArray(glassShader->a("vertex")); // wylacza atrybut vertex

glDisableVertexAttribArray(glassShader->a("normal")); //normalna

}

void drawModel(glm::mat4 P, glm::mat4 V, glm::mat4 M, int model\_i, int texture) { //rysowanie modeli przy wykorzystaniu water shader

waterShader->use(); //wlacza wykorzystanie water shader

glUniformMatrix4fv(waterShader->u("P"), 1, false, glm::value\_ptr(P)); //przekaza macierz projekcji P do water shader

glUniformMatrix4fv(waterShader->u("V"), 1, false, glm::value\_ptr(V));

glUniformMatrix4fv(waterShader->u("M"), 1, false, glm::value\_ptr(M));

glUniform4fv(waterShader->u("light1"), 1, glm::value\_ptr(light1)); //przekaza dane wektora swiatla do shadera

glUniform4fv(waterShader->u("light2"), 1, glm::value\_ptr(light2));

glEnableVertexAttribArray(waterShader->a("vertex")); //wlacza atrybut wierzcholkow dla shadera z nazwa vertex

glVertexAttribPointer(waterShader->a("vertex"), 4, GL\_FLOAT, false, 0, models[model\_i].Vertices.data()); //przekaza dane wierzchokow do shadera

glEnableVertexAttribArray(waterShader->a("texCoord")); //wlacza atrybut wspolrzednych tekstur do shadera

glVertexAttribPointer(waterShader->a("texCoord"), 2, GL\_FLOAT, false, 0, models[model\_i].TexCoords.data()); //przekaza dane

glActiveTexture(GL\_TEXTURE0); //aktywuje tekstury

glBindTexture(GL\_TEXTURE\_2D, texs[texture]); //powiazuje tekstury z jednostka teksturujaca

glUniform1i(waterShader->u("tex"), 0); //przekaza indeks jednostki teksturujacej do shadera

glDrawElements(GL\_TRIANGLES, models[model\_i].Indices.size(), GL\_UNSIGNED\_INT, models[model\_i].Indices.data()); //rysuje elementy modeli na podstawie indeksow

glDisableVertexAttribArray(waterShader->a("vertex")); //wylacza atrybut wierzcholkow

glDisableVertexAttribArray(waterShader->a("normal"));

glDisableVertexAttribArray(waterShader->a("color"));

}

void drawLight(glm::mat4 P, glm::mat4 V, glm::mat4 M) { //odpowiada za rysowanie swiatla na scenie

Models::Sphere light(0.25, 36, 36); //robi objekt sphere o okreslonym promieniu I rozdzielczosci

spConstant->use(); //wykorzystanie shadera spconstant

glUniformMatrix4fv(spConstant->u("P"), 1, false, glm::value\_ptr(P)); //przekaza macierz projekcji P do shadera

glUniformMatrix4fv(spConstant->u("V"), 1, false, glm::value\_ptr(V));

glUniformMatrix4fv(spConstant->u("M"), 1, false, glm::value\_ptr(M));

glUniform4f(spConstant->u("color"), 1, 1, 0.8, 1.8); //przekaza kolor swiatla do shadera

light.drawSolid(); //rysuje sphere

}

void drawTank(glm::mat4 P, glm::mat4 V, glm::mat4 M) { //odpowiedzialna za akwarium

phongShader->use(); //wykorzystanie phongShadera, odpowiedzialna za efekt oswietlenia phonga

glUniformMatrix4fv(phongShader->u("P"), 1, false, glm::value\_ptr(P)); //przekazuje macierz projekcji P do shadera

glUniformMatrix4fv(phongShader->u("V"), 1, false, glm::value\_ptr(V));

glUniformMatrix4fv(phongShader->u("M"), 1, false, glm::value\_ptr(M));

glUniform4fv(phongShader->u("light1"), 1, glm::value\_ptr(light1)); //wylaczyc swiatlo, przekazuje znaczenie wektora swiatla do shadea

glUniform4fv(phongShader->u("light2"), 1, glm::value\_ptr(light2));

glEnableVertexAttribArray(phongShader->a("vertex")); //wlacza atrybut wierzcholkow dla shadera z nazwa vertex

glVertexAttribPointer(phongShader->a("vertex"), 4, GL\_FLOAT, false, 0, models[TANK].Vertices.data()); //przekazuje dane

glEnableVertexAttribArray(phongShader->a("texCoord")); //wlacza atrybut tekstur dla shadera o nazwie texCoord

glVertexAttribPointer(phongShader->a("texCoord"), 2, GL\_FLOAT, false, 0, models[TANK].TexCoords.data()); //przekazuje dane

glActiveTexture(GL\_TEXTURE0);

glBindTexture(GL\_TEXTURE\_2D, texs[TANK]); //wiazy teksture tank z jednastka teksturujaca

glUniform1i(phongShader->u("tex"), 0); //przekazuje indeks jednostki tekstur do shadera

glDrawElements(GL\_TRIANGLES, models[TANK].Indices.size(), GL\_UNSIGNED\_INT, models[TANK].Indices.data()); //rysuje elementy na podstawie indeksow w modeltank

glDisableVertexAttribArray(phongShader->a("vertex")); //wylacza atrybut wierzcholkow dla shadera o nazwie vertex

glDisableVertexAttribArray(phongShader->a("normal"));

glDisableVertexAttribArray(phongShader->a("color"));

}