|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кафедра |  | O7 |  | Информационные системы и программная инженерия |
|  |  | шифр |  | наименование кафедры, по которой выполняется работа |
| Дисциплина |  | Компьютерная геометрия и графика | | |
|  |  | наименование дисциплины | | |

|  |  |  |
| --- | --- | --- |
| ЛАБОРАТОРНАЯ РАБОТА | 7 |  |
|  | номер (при наличии) |  |
| Освещение. | | |
|  | | |
|  | | |

при наличии указать тему лабораторной работы и (или) номер варианта

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **ОБУЧАЮЩИЙСЯ** | | | | | | |  |
| группы | | | |  | | И913Б |  |
|  |  | Кульга Н.И | | | | |  |
| подпись |  | фамилия и инициалы | | | | |  |
|  | | | | |  |  |  |
| дата сдачи | | | | |  |  |  |
| **ПРОВЕРИЛ** | | | | | | |  |
|  | | | | | | | |
| ученая степень, ученое звание, должность | | | | | | | |
|  |  | Мажайцев Е.А | | | | | |
| подпись |  | фамилия и инициалы | | | | | |
| Оценка / балльная оценка | | |  | | | |  |
|  | | | | |  |  |  |
| дата проверки | | | | |  |  |  |

Задание:

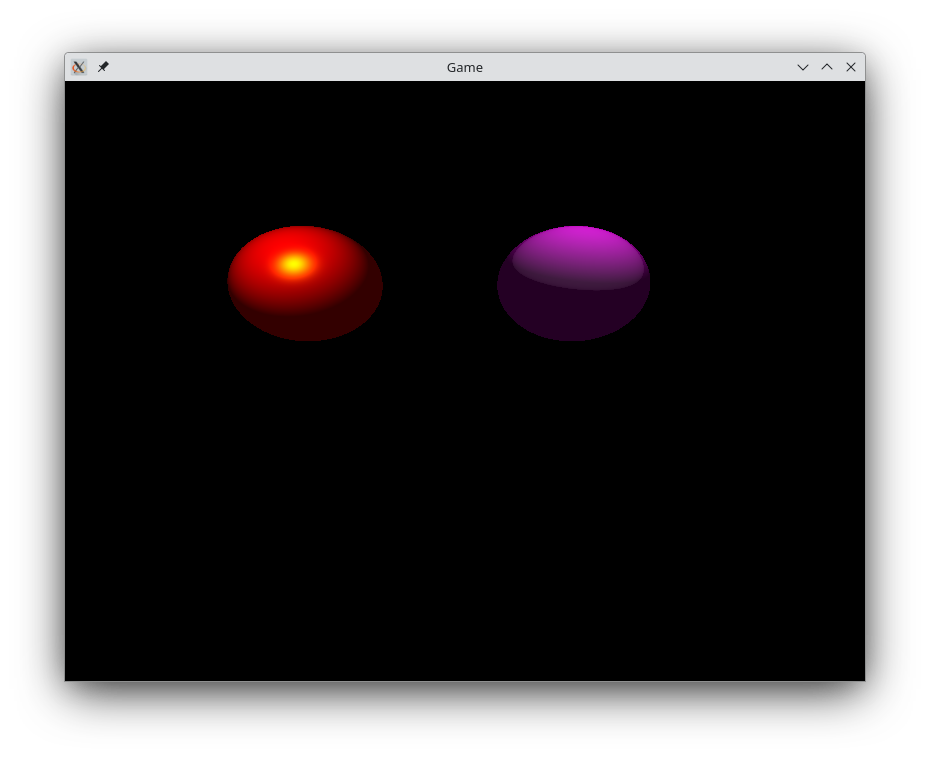
Задание 1. Построить 2 сферы с координатами и радиусом, заданными в таблице 1 для каждого варианта.

Задание 2. Задать фоновое освещение с параметрами, заданными в таблице 1 для каждого варианта.

Задание 3. Задать параметры материала для сфер. Одна из сфер должна быть матовой, другая - глянцевой. Цвет сфер задан в таблице 1.

Задание 4. Создать источники света, количество которых различно для каждого варианта. Параметры источников света задаются произвольно, однако они должны быть различны для разных ламп. Хотя бы один из источников должен быть направленным.

Результат работы программы:

Рисунок 1 – Результат работы.

Код программы:

package lr6.primitives;  
  
import org.joml.Vector3f;  
  
public class Sphere {  
 private float radius;  
 private int slices;  
 private int stacks;  
 private int vertexCount;  
 private float[] vertices;  
 private float[] textureCoords;  
 private float[] normalsArray;  
 private int[] indices;  
  
  
  
 public Sphere(float radius, int slices, int stacks) {  
 this.radius = radius;  
 this.slices = slices;  
 this.stacks = stacks;  
 createSphere(radius,slices,stacks);  
 }  
  
 private void createSphere(float radius, int slices, int stacks) {  
 vertexCount = (slices + 1) \* (stacks + 1);  
 vertices = new float[vertexCount \* 3];  
 textureCoords = new float[vertexCount \* 2];  
 normalsArray = new float[vertexCount \* 3]; // Нормали  
 indices = new int[slices \* stacks \* 6];  
  
 int vertexIndex = 0;  
 int textureIndex = 0;  
 int normalIndex = 0;  
 int index = 0;  
  
 for (int stack = 0; stack <= stacks; stack++) {  
 float stackAngle = (float) Math.*PI* / 2 - (float) stack \* (float) Math.*PI* / stacks;  
 float y = radius \* (float) Math.*sin*(stackAngle);  
 float r = radius \* (float) Math.*cos*(stackAngle);  
  
 for (int slice = 0; slice <= slices; slice++) {  
 float sliceAngle = (float) slice \* 2 \* (float) Math.*PI* / slices;  
 float x = r \* (float) Math.*cos*(sliceAngle);  
 float z = r \* (float) Math.*sin*(sliceAngle);  
  
 vertices[vertexIndex++] = x;  
 vertices[vertexIndex++] = y;  
 vertices[vertexIndex++] = z;  
  
 textureCoords[textureIndex++] = (float) slice / slices;  
 textureCoords[textureIndex++] = (float) stack / stacks;  
  
 Vector3f vertex = new Vector3f(x, y, z).normalize();  
 normalsArray[normalIndex++] = vertex.x;  
 normalsArray[normalIndex++] = vertex.y;  
 normalsArray[normalIndex++] = vertex.z;  
  
 if (stack < stacks && slice < slices) {  
 int next = slice + 1;  
 int nextStack = stack + 1;  
  
 indices[index++] = slice + stack \* (slices + 1);  
 indices[index++] = next + stack \* (slices + 1);  
 indices[index++] = slice + nextStack \* (slices + 1);  
  
 indices[index++] = next + stack \* (slices + 1);  
 indices[index++] = next + nextStack \* (slices + 1);  
 indices[index++] = slice + nextStack \* (slices + 1);  
 }  
 }  
 }  
  
 // У вас есть массивы vertices, textureCoords, normalsArray и indices для рисования сферы.  
 }  
  
 public float getRadius() {  
 return radius;  
 }  
  
 public int getSlices() {  
 return slices;  
 }  
  
 public int getStacks() {  
 return stacks;  
 }  
  
 public int getVertexCount() {  
 return vertexCount;  
 }  
  
 public float[] getVertices() {  
 return vertices;  
 }  
  
 public float[] getTextureCoords() {  
 return textureCoords;  
 }  
  
 public int[] getIndices() {  
 return indices;  
 }  
  
 public void setRadius(float radius) {  
 this.radius = radius;  
 }  
  
 public void setSlices(int slices) {  
 this.slices = slices;  
 }  
  
 public void setStacks(int stacks) {  
 this.stacks = stacks;  
 }  
  
 public void setVertexCount(int vertexCount) {  
 this.vertexCount = vertexCount;  
 }  
  
 public void setVertices(float[] vertices) {  
 this.vertices = vertices;  
 }  
  
 public void setTextureCoords(float[] textureCoords) {  
 this.textureCoords = textureCoords;  
 }  
  
 public void setIndices(int[] indices) {  
 this.indices = indices;  
 }  
  
 public float[] getNormalsArray() {  
 return normalsArray;  
 }  
  
 public void setNormalsArray(float[] normalsArray) {  
 this.normalsArray = normalsArray;  
 }  
}

package lr6.entities;  
  
import org.joml.Vector3f;  
  
public class Light {  
 */\*\* Позиция источника света \*/* private Vector3f position;  
 */\*\* Цвет свечения \*/* private Vector3f colour;  
  
 public Light(Vector3f position, Vector3f colour) {  
 this.position = position;  
 this.colour = colour;  
 }  
  
 public Vector3f getPosition() {  
 return position;  
 }  
  
 public void setPosition(Vector3f position) {  
 this.position = position;  
 }  
  
 public Vector3f getColour() {  
 return colour;  
 }  
  
 public void setColour(Vector3f colour) {  
 this.colour = colour;  
 }  
}

package lr6.entities;  
  
import lr6.io.Keyboard;  
import org.joml.Vector3f;  
import org.lwjgl.glfw.GLFW;  
  
*/\*\*  
 \* Класс симуляции камеры  
 \*/*public class Camera {  
  
 */\*\* позиция камеры \*/* private Vector3f position;  
 */\*\* тангаж - поворот по оси X(Поперечная ось) \*/* private float pitch;  
 */\*\* рыскание - поворот по оси Y(Вертикальная ось, лежащая в плоскости) \*/* private float yaw;  
 */\*\* крен - поворот по оси Z(Продольная ось) \*/* private float roll;  
  
 public Camera() {  
 this.position = new Vector3f(0, 0, 0);  
 }  
  
 public Camera(float x,float y,float z) {  
  
 this.position = new Vector3f(x, y, z);  
 }  
   
 */\*\*  
 \* Смещение камеры  
 \*/* public void move() {  
 if (Keyboard.*isKeyDown*(GLFW.*GLFW\_KEY\_W*)) {  
 position.z -= 0.02f;  
 } else if (Keyboard.*isKeyDown*(GLFW.*GLFW\_KEY\_S*)) {  
 position.z += 0.02f;  
 }  
 if (Keyboard.*isKeyDown*(GLFW.*GLFW\_KEY\_A*)) {  
 position.x += 0.02f;  
 } else if (Keyboard.*isKeyDown*(GLFW.*GLFW\_KEY\_D*)) {  
 position.x -= 0.02f;  
 }  
 }  
  
 public Vector3f getPosition() {  
 return position;  
 }  
   
 */\*\* тангаж - поворот по оси X(Поперечная ось) \*/* public float getPitch() {  
 return pitch;  
 }  
  
 */\*\* рыскание - поворот по оси Y(Вертикальная ось, лежащая в плоскости) \*/* public float getYaw() {  
 return yaw;  
 }  
  
 */\*\* крен - поворот по оси Z(Продольная ось) \*/* public float getRoll() {  
 return roll;  
 }  
}

package lr6.entities;  
  
import org.joml.Vector4f;  
  
public class Background {  
  
 Vector4f background;  
  
 public void setBackground(Vector4f background) {  
 this.background = background;  
 }  
  
 public Vector4f getBackground() {  
 return background;  
 }  
  
 public Background(Vector4f background) {  
 this.background = background;  
 }  
}

#version 400 core  
  
// получаем позиции вершин из VAO  
in vec3 position;  
// получаем текстурные координаты из VAO  
in vec2 textureCoords;  
// получаем вектор нормали из VAO  
in vec3 normal;  
  
// отправляем в фрагментный шейдер текстурные координаты  
out vec2 pass\_textureCoords;  
out vec3 surfaceNormal; // нормаль поверхности  
out vec3 toLightVector; // вектор направленный на источник света  
out vec3 toCameraVector; // вектор положения камеры  
  
// матрица трансформации  
uniform mat4 transformationMatrix;  
// матрица проекции  
uniform mat4 projectionMatrix;  
// матрица вида  
uniform mat4 viewMatrix;  
uniform vec3 lightPosition; // позиция источника света  
  
void main()  
{  
 vec4 worldPosition = transformationMatrix \* vec4(position, 1.0);  
 // высчитываем новое положение вершины, в зависимости от трансформации и проекции  
 gl\_Position = projectionMatrix \* viewMatrix \* worldPosition;  
  
 pass\_textureCoords = textureCoords;  
 // применяем трансформацию к нормали  
 surfaceNormal = (transformationMatrix \* vec4(normal, 0.0)).xyz;  
 toLightVector = lightPosition - worldPosition.xyz;  
 // положение камеры относительно вершины  
 toCameraVector = (inverse(viewMatrix) \* vec4(0.0,0.0,0.0,1.0)).xyz - worldPosition.xyz;  
}

#version 400 core  
  
in vec2 pass\_textureCoords; // текстурные координаты  
in vec3 surfaceNormal; // нормаль поверхности  
in vec3 toLightVector; // вектор направленный на источник света  
in vec3 toCameraVector; // вектор положения камеры  
  
out vec4 fragColor;  
  
uniform sampler2D textureSampler;  
uniform vec3 lightColour; // цвет источника света  
uniform vec4 background; // фон  
  
  
uniform float shineDamper; // коэффициент блеска материала  
uniform float reflectivity; // отражательная способность материала  
  
void main()  
{  
 // нормализируем данные  
 vec3 unitNormal = normalize(surfaceNormal);  
 vec3 unitLight = normalize(toLightVector);  
 vec3 unitVectorCamera = normalize(toCameraVector);  
  
 // вычисляем скалярную разницу между векторами  
 float nDotl = dot(unitNormal,unitLight);  
 float brightness = max(nDotl, 0.2); // яркость пикселя от 0.2 до 1  
 vec3 diffuse = brightness \* lightColour; // фоновый свет  
  
 vec3 lightDirection = -unitLight; // противоположность вектору света  
 // расчет вектора отражения  
 vec3 reflectedLightDirection = reflect(lightDirection,unitNormal);  
  
 // расчет яркости отражения, вектор отражения к вектору камеры  
 float specularFactor = dot(reflectedLightDirection, unitVectorCamera);  
 specularFactor = max(specularFactor, 0.0); // отсекаем отрицательные значения  
 // рассчитываем зеркальный коэффициент  
 float dampedFactor = pow(specularFactor,shineDamper);  
 // рассчитываем зеркальный эффект  
 vec3 finalSpecular = dampedFactor \* reflectivity \* lightColour;  
  
 // метод texture координату точки на модели сопоставляет с координатой точи на текстуре  
 // и возвращает цвет пикселя  
 fragColor = vec4(diffuse, 1.0) \* background  
 + vec4(finalSpecular, 1.0);  
}