**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Компьютерная графика»

**Лабораторная работа № 3**

**по курсу «Компьютерная графика»**

Студент: Кондратьев Егор Алексеевич

Группа: 80-306

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2021

1. Постановка задачи

Используя результаты Л.Р.No2, аппроксимировать заданное тело выпуклым многогранником. Точность аппроксимации задается пользователем. Обеспечить возможность вращения и масштабирования многогранника и удаление невидимых линий и поверхностей. Реализовать простую модель закраски для случая одного источника света.

Параметры освещения и отражающие свойства материала задаются пользователем в диалоговом режиме.

Вариант Эллипсоид.

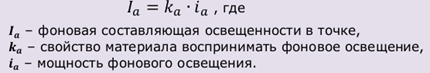
1. Описание программы

Для решения задачи было решено использовать C++ и фреймворк Qt, в котором использовалась библиотека QPainter.

Создан класс Polygon для хранения полигонов, класс Ellipsoid, представляющий фигуру эллипсоид. Такая фигура состоит из множества полигонов. Пользователь может задавать количество полигонов. Все преобразования для фигуры выполняются для каждого полигона, и в каждом полигоне преобразования выполняются для каждой точки. Так выполняются пространственные повороты фигуры и масштабирование фигуры.

Из нового была использована модель освещения, построенная как сумма трех световых составляющих: фоновая, рассеянная, зеркальная.

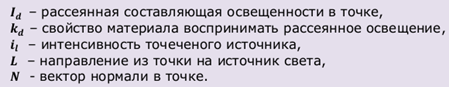
Фоновая составляющая:

****

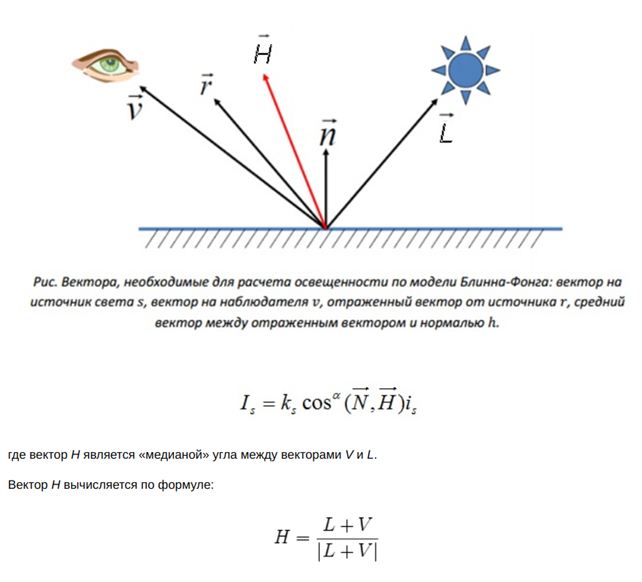
Фоновая составляющая освещенности не зависит от пространственных координат освещаемой точки и источника.

Рассеянное отражение света происходит, когда свет как бы проникает под поверхность объекта, поглощается, а затем вновь испускается. При этом положение наблюдателя не имеет значения, так как диффузно отраженный свет рассеивается равномерно по всем направлениями. Интенсивность света обратно пропорциональна квадрату расстояния от источника, следовательно, объект, лежащий дальше от него, должен быть темнее.

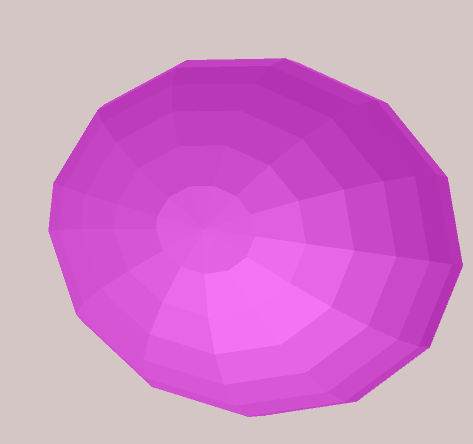
****



Зеркальная составляющая влияет на появление блика на объекте. Местонахождение блика на объекте определяется из закона равенства углов падения и отражения. Если наблюдатель находится вблизи углов отражения, яркость соответствующей точки повышается.



1. Результаты



1. Листинг программы

#include "polygon.h"

#include <QVector3D>

#include <QPen>

#include <QColor>

#include <QPolygonF>

#include <QPointF>

#include <cmath>

Polygon::Polygon(const std::vector<QVector4D> &v) {

vertices = v;

}

QVector3D Polygon::get\_normal() {

QVector3D first{

vertices[1][0] - vertices[0][0],

vertices[1][1] - vertices[0][1],

vertices[1][2] - vertices[0][2]

};

QVector3D second{

vertices[2][0] - vertices[0][0],

vertices[2][1] - vertices[0][1],

vertices[2][2] - vertices[0][2]

};

QVector3D normal{

first[1] \* second[2] - first[2] \* second[1],

first[2] \* second[0] - first[0] \* second[2],

first[0] \* second[1] - first[1] \* second[0]

};

return normal;

}

void Polygon::add\_vertex(const QVector4D &vertex) {

vertices.push\_back(vertex);

}

void Polygon::add\_vertex(double x, double y, double z, double d) {

QVector4D v{static\_cast<float>(x), static\_cast<float>(y), static\_cast<float>(z), static\_cast<float>(d)};

vertices.push\_back(v);

}

void Polygon::clear\_verticies() {

vertices.clear();

}

void Polygon::change\_vertices(const QMatrix4x4 &mtrx) {

for (auto &vertex: vertices) {

vertex = mtrx \* vertex;

}

}

int Polygon::calc\_ambient\_component(Lamp \*lamp) {

return static\_cast<int>(ambient\_coef \* lamp->power);

}

int Polygon::calc\_diffuse\_component(int dx, int dy, Lamp \*lamp) {

QVector3D toLamp = QVector3D{

lamp->position.x() - (vertices[0].x() + dx),

lamp->position.y() - (vertices[0].y() + dy),

lamp->position.z() - vertices[0].z()

};

QVector3D toLampNormalized = toLamp.normalized();

QVector3D normal = this->get\_normal().normalized();

double scalarProduct = static\_cast<double>(toLampNormalized.x() \* normal.x() +

toLampNormalized.y() \* normal.y() +

toLampNormalized.z() \* normal.z());

int res = static\_cast<int>(diffuse\_coef \* scalarProduct \* lamp->power \* 100 / pow((static\_cast<double>(toLamp.length())), 1.2));

if (res < 0) {

res = 0;

}

return res;

}

int Polygon::calc\_specular\_component(int dx, int dy, Lamp \*lamp) {

QVector3D toLamp = QVector3D{

lamp->position.x() - (vertices[0].x() + dx),

lamp->position.y() - (vertices[0].y() + dy),

lamp->position.z() - vertices[0].z()

};

QVector3D toLampNormalized = toLamp.normalized();

QVector3D toObserver = QVector3D{0 - (vertices[0].x()), 0 - (vertices[0].y()), vertices[0].z()}.normalized();

QVector3D median = (toLampNormalized + toObserver) / (toLampNormalized + toObserver).length();

QVector3D normal = this->get\_normal().normalized();

float scalarProduct = median.x() \* normal.x() + median.y() \* normal.y() + median.z() \* normal.z();

int res = static\_cast<int>(specular\_coef \* pow(static\_cast<double>(scalarProduct), gloss\_coef) \*

lamp->power \* 100 / pow((static\_cast<double>(toLamp.length())), 1.2));

if (res < 0) {

res = 0;

}

return res;

}

void Polygon::draw(QPainter \*ptr, int center\_x, int center\_y, double step\_pixels,

int window\_center\_x, int window\_center\_y, Lamp \*lamp,

bool displayCarcass) {

QPen oldPen = ptr->pen();

int resCalcAmbientComponent = calc\_ambient\_component(lamp);

int resCalcDiffuseComponent = calc\_diffuse\_component(center\_x - window\_center\_x,

center\_y - window\_center\_y, lamp);

int resCalcSpecularComponent = calc\_specular\_component(center\_x - window\_center\_x,

center\_y - window\_center\_y, lamp);

int r = rgb['r'] + resCalcAmbientComponent + resCalcDiffuseComponent + resCalcSpecularComponent;

int g = rgb['g'] + resCalcAmbientComponent + resCalcDiffuseComponent + resCalcSpecularComponent;

int b = rgb['b'] + resCalcAmbientComponent + resCalcDiffuseComponent + resCalcSpecularComponent;

if (r > 255) {

r = 255;

}

if (g > 255) {

g = 255;

}

if (b > 255) {

b = 255;

}

QPen newPen(QColor(r, g, b), 0.5, Qt::SolidLine, Qt::FlatCap, Qt::RoundJoin);

ptr->setPen(newPen);

ptr->setBrush(QColor(r, g, b));

QPolygonF pol;

for (size\_t i = 0; i < 3; i++) {

pol << QPointF(

static\_cast<double>(vertices[i][0]) \* step\_pixels + center\_x,

static\_cast<double>(vertices[i][1]) \* step\_pixels + center\_y

);

}

ptr->drawPolygon(pol);

if (displayCarcass) {

ptr->setPen(oldPen);

for (size\_t i = 0; i < 3; i++) {

ptr->drawLine(

static\_cast<int>(static\_cast<double>(vertices[i][0]) \* step\_pixels + center\_x),

static\_cast<int>(static\_cast<double>(vertices[i][1]) \* step\_pixels + center\_y),

static\_cast<int>(static\_cast<double>(vertices[(i + 1) % 3][0]) \* step\_pixels + center\_x),

static\_cast<int>(static\_cast<double>(vertices[(i + 1) % 3][1]) \* step\_pixels + center\_y)

);

}

}

}

1. ЛИТЕРАТУРА
2. QT documentation [Электронный ресурс]. URL: <https://doc.qt.io/>

(дата обращения: 13.10.2021).