осковский Авиационный Институт
(Национальный исследовательский Университет)

Факультет: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»

Лабораторная работа №3
по курсу «Компьютерная графика»

Студент:	Марков А.Н.
Группа:	М80-308Б-18
Преподаватель:	Филиппов Г.С.
Оценка:	
Дата:	

Москва
2020

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.

2. Решение задачи.

Для решения задачи я решил использовать C++ и фреймворк Qt, в котором использовал библиотеку QPainter.

Я создал 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.

Я использовал модель освещения, построенную как сумма трех световых составляющих: фоновая, рассеянная, зеркальная.

Фоновая составляющая:

$$I_a = k_a \cdot i_a \text{ , где}$$

I_a – фоновая составляющая освещенности в точке,
k_a – свойство материала воспринимать фоновое освещение,
i_a – мощность фонового освещения.

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.

$$I_d = \frac{k_d \cdot i_l}{d+K} \cdot cos(\vec{L}, \vec{N}) = \frac{k_d \cdot i_l}{d+K} \cdot (\vec{L} \cdot \vec{N}),$$ где K — произвольная постоянная, а d - расстояние от центра проекции до объекта.

I_d – рассеянная составляющая освещенности в точке,
k_d – свойство материала воспринимать рассеянное освещение,
i_l – интенсивность точеченого источника,
L – направление из точки на источник света,
N - вектор нормали в точке.

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.

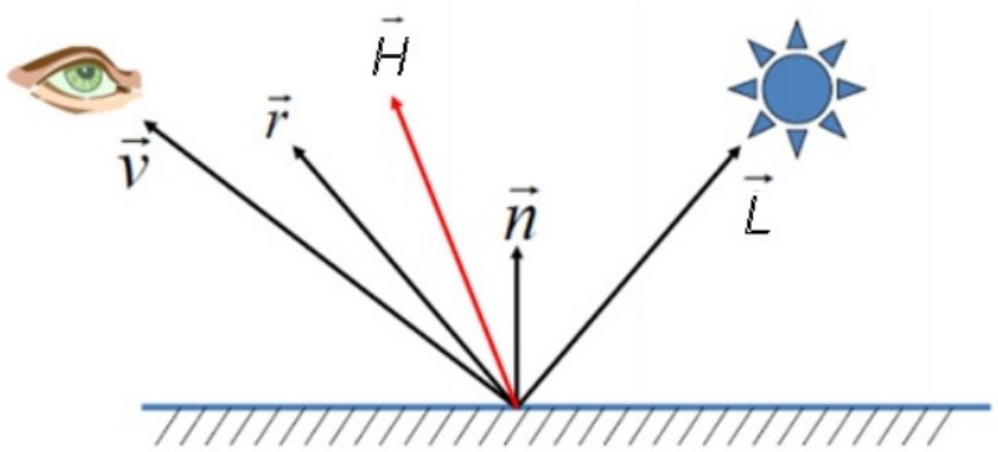


Рис. Вектора, необходимые для расчета освещенности по модели Блинна-Фонга: вектор на источник света s, вектор на наблюдателя v, отраженный вектор от источника r, средний вектор между отраженным вектором и нормалью h.

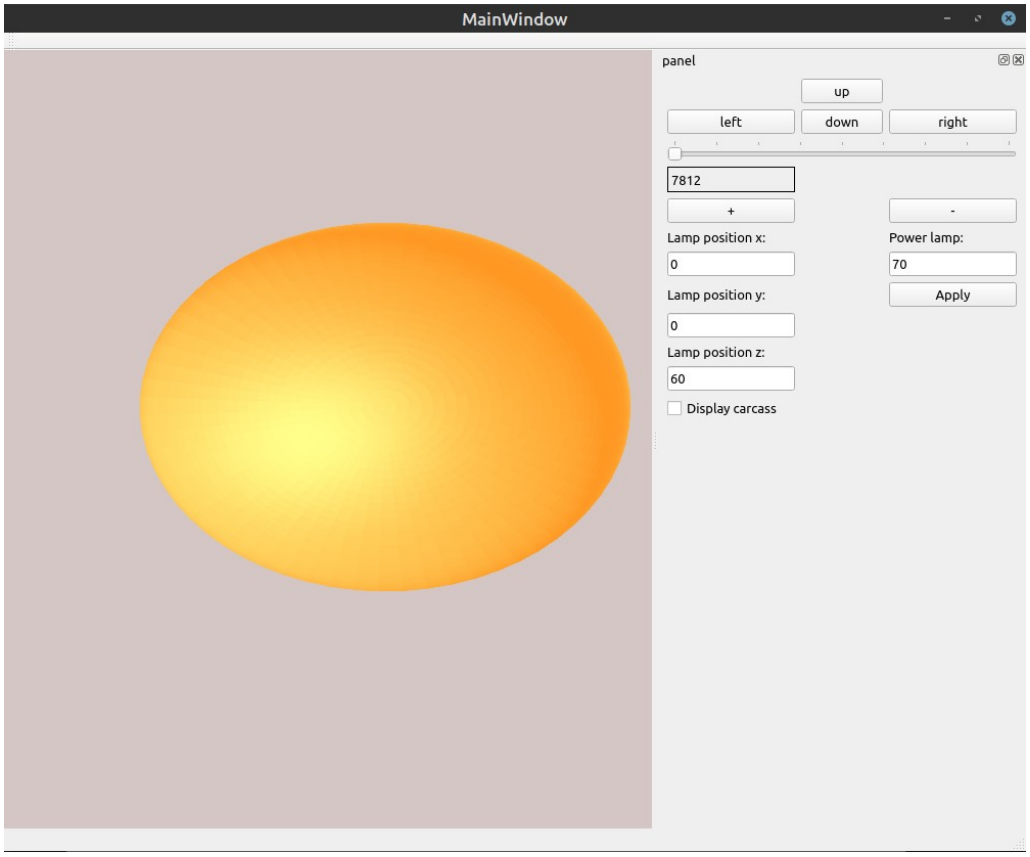
$$I_s = k_s \cos^{\alpha}(\vec{N}, \vec{H}) i_s$$

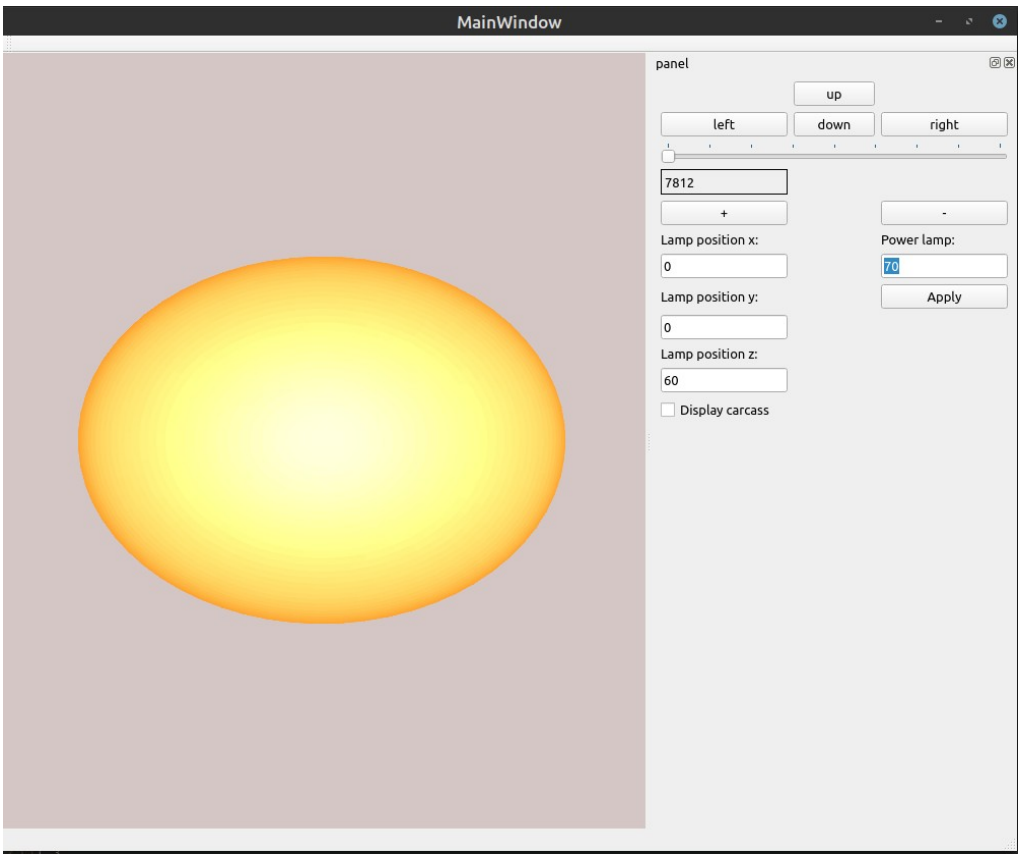
где вектор *H* является «медианой» угла между векторами *V* и *L*.

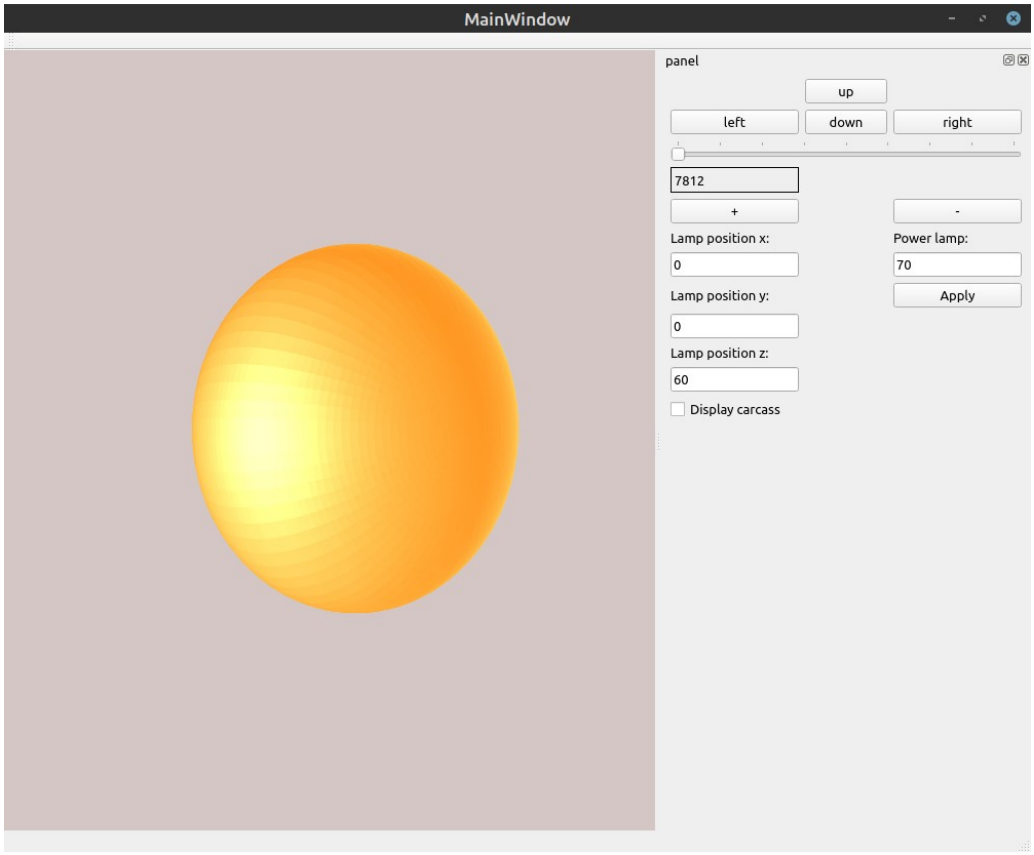
Вектор *H* вычисляется по формуле:

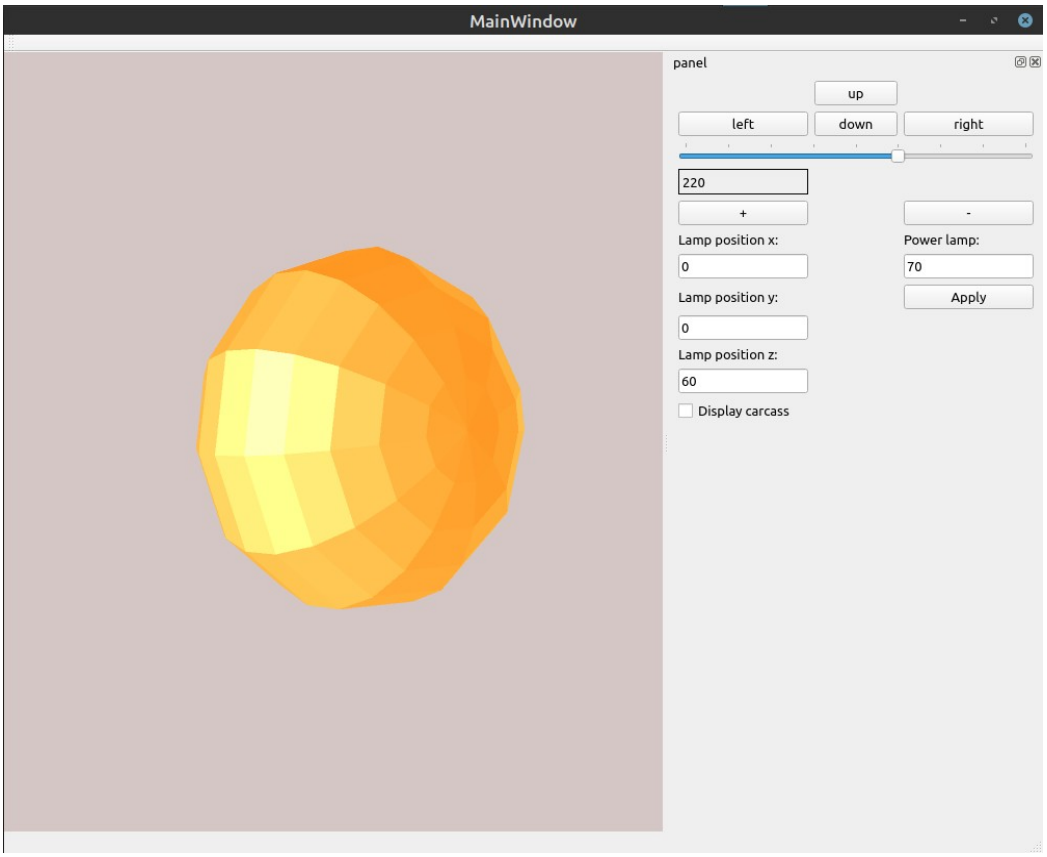
$$H = \frac{L + V}{|L + V|}$$

3. Демонстрация работы программы






MainWindow
panel
up
left
down
right
7812
+
-
Lamp position x:
Power lamp:
0
70
Lamp position y:
Apply
0
Lamp position z:
60
Display carcass


MainWindow
panel
up
left
down
right
220
+
-
Lamp position x:
Power lamp:
0
70
Lamp position y:
Apply
0
Lamp position z:
60
Display carcass

4. Листинг программы.
polygon.cpp:

```
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)
            );
        }
    }
}
```

5. Вывод.

В ходе выполнения данной лабораторной работы я научился аппроксимировать фигуру, освежил знания по линейной алгебре, узнал о разных моделях освещения и реализовал одну из них программно.