Министерство образования и науки РФ

Санкт-Петербургский государственный электротехнический университет “ЛЭТИ”

Кафедра Вычислительной техники

Пояснительная записка

к работе по дисциплине

«Компьютерная графика»

Выполнил: Краснов Д.С.

Группа: 9811

Проверил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Санкт-Петербург

2013 г.

## Оглавление

[1. Описание программы 3](#_Toc377344186)

[Управление: 3](#_Toc377344187)

[2. Описание алгоритмов 4](#_Toc377344188)

[I. Видовые преобразования 4](#_Toc377344189)

[Перенос в мировое пространство (матрица модели) 4](#_Toc377344190)

[Пространство камеры 4](#_Toc377344191)

[Поворот камеры 5](#_Toc377344192)

[Проекция перспективы 7](#_Toc377344193)

[II. Освещение 9](#_Toc377344194)

[Фоновый свет 10](#_Toc377344195)

[Рассеянный свет 10](#_Toc377344196)

[Отраженный свет 11](#_Toc377344197)

[Приложение 1 – Код программы 14](#_Toc377344198)

## Описание программы

Программа загружает и рисует трёхмерную модель стены, сделанную в 3D-редакторе Bender, позволяет вращать её, перемещать вокруг неё камеру, а так же менять параметры освещения.



### Управление:

**←**, **→** - вращение модели вокруг оси Y

**↑**, **↓** - вращение модели вокруг оси X

**Мышь** – обзор

**W**, **S**, **A**, **D** – перемещение (камеры) вперёд, назад, влево и вправо соответственно

**Ctrl**, **Space** – перемещение (камеры) вверх и вниз соответственно

**+, -** – изменение интенсивности фонового освещения

**Shift** - **+**, **Shift -** **-** – изменение интенсивности рассеянного освещения

**Shift** - **←→↑** **↓** – вращение вектора направления рассеянного освещения

**[** , **]** – изменение интенсивности отражения материала

**;** , **‘** – изменение коэффициента блеска

**R** – сбросить состояние в первоначальное

**Esc** – выход

## Описание алгоритмов

### Видовые преобразования

Видовые преобразования, реализованные в программе, заключаются в переносе модели в мировое пространство, затем в пространство камеры и, наконец, проективном преобразовании. Технически, эти преобразования происходят путём умножения матрицы «модель-камера-проекция» на вектор-столбец координат каждой вершины модели. Матрица «модель-камера-проекция» в свою очередь получается перемножением матриц Mпроекция \* Mкамера \* Mмодель.

#### Перенос в мировое пространство (матрица модели)

Матрица модели получается путём перемножения матриц масштабирования, вращения и переноса. Матрица вращения получается перемножения матриц вращения вокруг каждой из осей:

Матрица масштабирования:

Матрица вращения вокруг оси X:

Матрица вращения вокруг оси Y:

Матрица вращения вокруг оси Z:

Матрица переноса:

#### Пространство камеры

Камера определяет свою собственную систему координат, которая может совпадать с мировой (когда камера находится в центре координат и направлена в сторону уменьшения оси Z) или отличаться, задавая таким образом свою систему координат.

В пространстве камеры сама камера расположена в начале координат и направлена в сторону, обратную оси Z. Поворот камеры на 45 градусов по часовой стрелке все равно, что и поворот объекта на 45 градусов против часовой. Движение объектов всегда противоположно движению камеры. Поэтому в целом перенос модели в пространство камеры состоит из 2 частей: передвижения объектов так, что бы расстояние от них до камеры было бы таким же, как если бы камера располагалась в начале координат, и поворота объектов в направлении, противоположном вращению камеры.

С передвижением всё просто: если координаты камеры (x,y,z), то преобразование позиции (-x, -y, -z).

Соответствующая матрица выглядит так:

Теперь рассмотрим поворот:

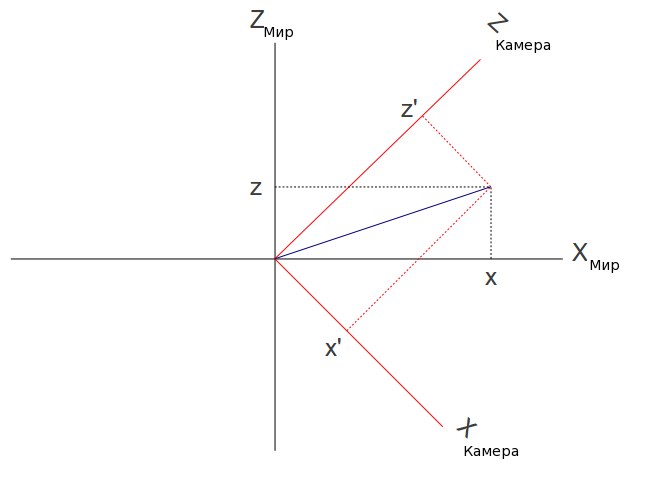


Рисунок показывает, как вектор указывается в 2 независимых системах координат (вектор y в обоих системах направлен на нас).

Для получения вектора (x',y',z') используется скалярная проекция вектора (x, y, z) на базисные единичные вектора системы камеры.

Решение называется «UVN-камера» и это одна из множества систем, характеризующих камеру. Камера определяется следующими векторами:

N - Вектор от камеры к ее цели (так же известен как вектор 'look at'). Этот вектор соответствует Z оси.

V - Вектор «вверх». Соответствует оси Y.

U - Выходит из камеры направо. Соответствует оси X.

Преобразование выглядит так:

Итоговая матрица преобразования камеры есть произведение соответствующих камере матриц переноса и поворота.

##### Поворот камеры

Вращение вектора направления камеры на горизонтальный угол довольно просто. Z координата - это синус горизонтального угла, а X - косинус.

Поворот вектора направления на вертикальный угол более сложный, так как вращается не только камера, но и ее горизонтальная ось. Горизонтальная ось может быть найдена векторным произведением вертикальной оси и вектором направления, после того, как он будет повернут на горизонтальный угол, но вращение вокруг произвольного вектора может быть довольно сложно.

Существует несколько типовых решений, позволяющих вращать камеру, таких, как, например, использование углов Эйлера, но есть более простое математическое решение – кватернионы.

Кватернион 'Q' определяется так:



Где i, j и k комплексные числа, и для них справедливо равенство:



Но математические подробности сложны и нас не обязательны для решения задачи. На практике мы указываем кватернион как 4-вектор (x, y, z, w).

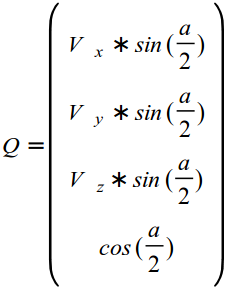
Сопряжение кватерниону 'Q' выглядит так:



В программе кватернионы используются для вращения вектора вокруг произвольного вектора. В целом функция для подсчета кватерниона 'W', который представляет повернутый вектор 'V' на угол 'a', такова:



Где Q - это кватернион вращения, определенный так:



, где V – вектор, вокруг которого будет производиться поворот, а α - угол, на который поворот будет производиться.

После подсчета 'W' повернутый вектор выглядит довольно просто: (W.x,W.y,W.z).

При движении мышки регулярно высчитываются горизонтальный и вертикальный углы поворота. В зависимости от них пересчитываются вектора N (к цели) и V (вверх):

const vec3 axisV = vec3(0.0f, 1.0f, 0.0f);

vec3 target = vec3(0.0f, 0.0f, 1.0f);

rotate(target, axisV, m\_angleH);

vec3 axisH = cross(target, axisV);

normalize(axisH);

rotate(target, axisH, m\_angleV);

m\_target = target;

normalize(m\_target);

m\_up = cross(axisH, m\_target);

normalize(m\_up);

Вектор «вправо» получается позже с помощью векторного произведения векторов «к цели» и «вверх».

#### Проекция перспективы

Проекция перспективы – это проекция из 3D мира на 2D плоскость с сохранением глубины и переводом координат в пространство экрана (от -1 до +1)

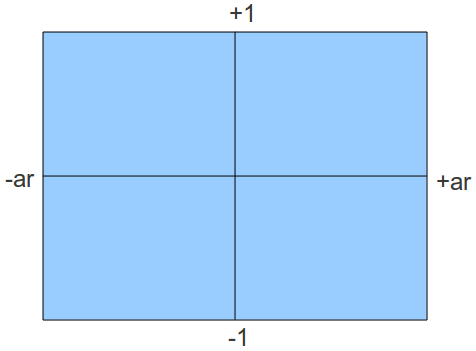
Для определения преобразования перспективы необходимы 4 параметра:

* Соотношение сторон - коэффициент между шириной и высотой прямоугольной области, на которую будет осуществляться проекция.
* Поле зрения по вертикали - угол обзора камеры.
* Позиция ближней Z плоскости. Позволяет обрезать объекты, находящиеся слишком близко к камере.
* Позиция дальней Z плоскости. Позволяет обрезать объекты, находящиеся слишком далеко от камеры.

Начнем с определения расстояния от плоскости проекции до камеры. Эта плоскость будет параллельна плоскости XY. Очевидно, видна будет не вся плоскость из-за ее огромных размеров. Мы сможем увидеть вещи из прямоугольной области (называемой окном проекции), которая имеет те же пропорции, что и наш экран. Соотношение сторон подсчитывается так:

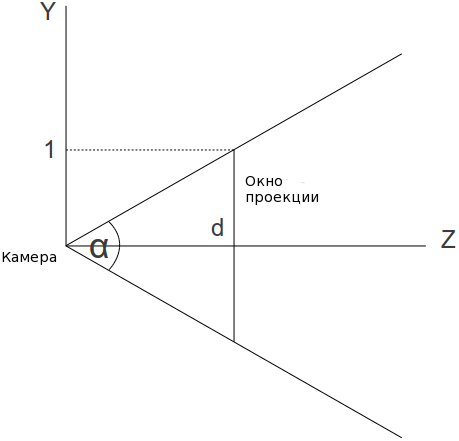
ar (aspect ratio) = ширина экрана / высота экрана

Если поместить камеру в начало координат и посмотрим на область с точки зрения камеры, можно увидеть:

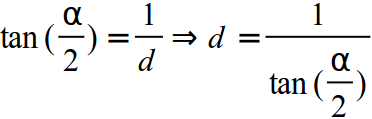


Все, что выходит за пределы этого прямоугольника, будет обрезано. Координаты внутри будут иметь Y компоненту в требуемом отрезке, но X координата пока что немного больше. Это будет исправлено позднее.

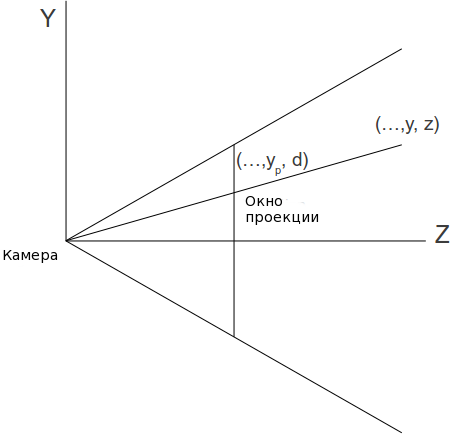
Вид сбоку:



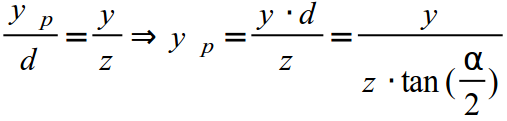
Расстояние от камеры до плоскости проекции находится с использованием высоты обзора по вертикали (обозначается углом альфа):



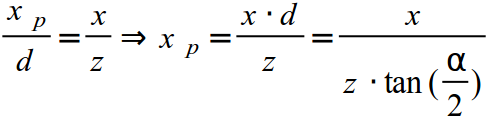
Следующий шаг - это подсчет проецированных координат X и Y. Рассмотрим следующее изображение:



У нас есть точка в 3D пространстве с координатами (x,y,z). Мы хотим найти (xp,yp), которые преобразуют проецированные координаты на плоскость проекции. Так как X компонента не видна на этой схеме, то начнем с Y. Согласно правилу подобных треугольников, мы получаем следующее:

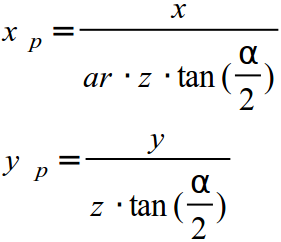


И то же самое для X координаты:



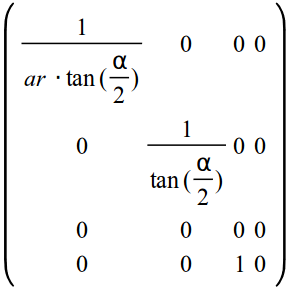
Так как размер окна проекции равен 2\*ar (ширина на 2 высоты), известно, что точка в 3D пространстве внутри окна, если она проецируется в точку, координата X которой в пределах от -ar и до +ar, и проецированная Y координата между -1 и +1. Поэтому Y компонента нормирована, но X еще нет. Мы можем нормировать Xp, просто поделив на соотношение сторон. Деление на соотношение сторон позволяет конденсировать точки по оси X.

Итак, для X и Y выражения следующие:



Проблема деления на z решается помещением значения z в компоненту w: таким образом при вычислении конечной позиции вершины x и y будут автоматически на неё разделены (об этом заботится OpenGL, операция так и называется: «деление перспективы»).

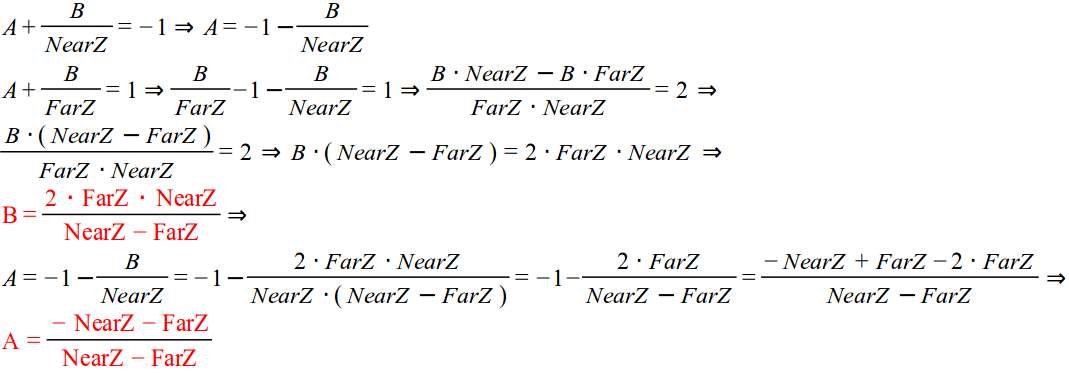
Промежуточная матрица, обобщающая написанное выше, выглядит так:



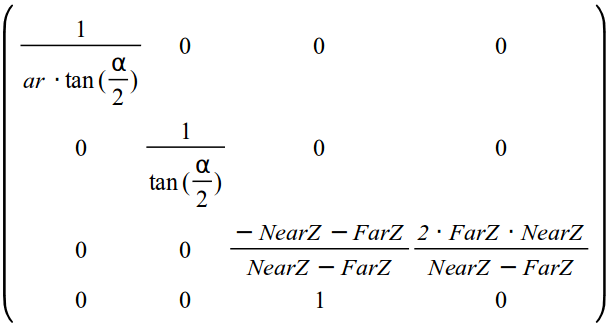
Осталось только добавить в неё нормирование значения Z, чтобы любое значение Z из промежутка (NearZ <= Z <= FarZ) было отображено в отрезок [-1,1]. Такое отображение состоит из 2 этапов. В первом уменьшается отрезок [NearZ, FarZ], пока его длина не будет равна 2. Затем он сдвигается так, что бы начало совпадало с -1. Уменьшение Z значения и его перемещение заменяется общей функцией:



Известно, что когда Z соответствует NearZ, итог будет -1, а когда Z соответствует FarZ, результат будет равен 1. Это можно записать так:



Конечная матрица преобразования такова:



### Освещение

В программе реализовано 3 вида освещения: фоновое (Ambient), рассеянное (Diffuse) и отражённое (Specular).

**Фоновое освещение** характерно тем, что не имеет направления и характеризуется цветом и интенсивностью.

**Рассеянный свет** представляет собой модель освещения бесконечно удалённым источником света постоянной интенсивности (солнцем) и подчеркивает тот факт, что угол, под которым свет падает на поверхность, влияет на яркость освещаемого объекта. Когда свет падает на объект, одна сторона будет ярче по сравнению с другой стороной. Главный параметр рассеянного света - это его направление.

**Отраженный свет** - это больше свойство самого объекта, чем света. Это блики, когда свет падает под определенным углом и зритель расположен в определенной позиции. При подсчете отраженного света требуется внимание и к направлению света, под которым он падает (и отражается), и позиции зрителя.

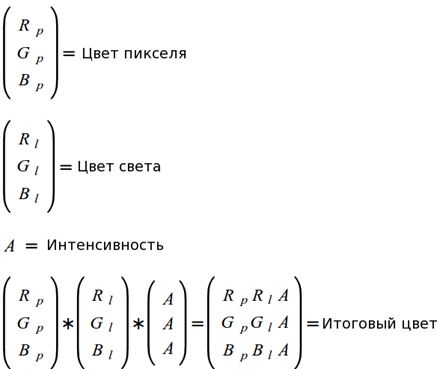
Формула итогового цвета пикселя следующая:

**Ситог = Стекст \* (Сфон + Сдифф + Сотр)**

#### Фоновый свет

Когда белый свет падает на поверхность, то отражаемый цвет - это просто цвет поверхности. Он может стать ярче или темнее в зависимости от силы источника света, но останется прежним. Если источник света чистый красный (255,0,0), то отраженный цвет будет только одним из оттенков красного. Это происходит из-за того, что свет не имеет зеленого и голубого каналов, которые могли бы отразится от поверхности. Если поверхность синяя, то в итоге получим абсолютно черный цвет. Суть заключается в том, что свет может только повлиять на фактический цвет объекта, но не может "нарисовать" его.

Цвет источника света указывается, как тройка вещественных чисел в отрезке [0-1]. Отраженный цвет получается умножением цвета объекта на цвет света. Фоновая интенсивность указана в виде единственного вещественного числа в отрезке [0-1], которое будет умножаться на все каналы отраженного цвета, сразу после его вычисления. Это и будет итоговый цвет.



В программе компонента фонового цвета вычисляется по формуле:

#### Рассеянный свет

Главное отличие между фоновым и рассеянным освещением в том, что диффузный свет полагается на направление лучей света, в то время как фоновый игнорирует его полностью. Если представлен только фоновый свет, то все объекты освещены в равной степени. Диффузный же делает часть объекта, на которую падает свет, ярче по сравнению с остальными.

Рассеянный свет также добавляет условие, в котором угол, под которым свет падает на поверхность, определяет яркость этой поверхности: чем угол ближе к 90°, тем поверхность ярче.

Модель рассеянного света основывается на законе косинуса Ламберта, который гласит, что интенсивность света, отраженного от поверхности, прямо пропорциональна косинусу угла между линией зрения наблюдателя и нормалью к поверхности, только вместо линии обзора используется направление света.

Для подсчета интенсивности света в рассеянной модели используется косинус угла между направлением света и нормалью к поверхности (в то время как закон Ламберта относится к более общему понятию "прямо пропорциональности"). Если этот угол = 0°, косинус его равен 1 и, т.о., влияние рассеянного света максимально. Если же угол ≥ 90°, значение косинуса будет от 0 до -1 и в этом случае влияние рассеянного света не будет учитываться.

Косинус считается, как скалярное произведение вектора направления света на нормаль.

Формула подсчёта диффузной компоненты цвета такова:

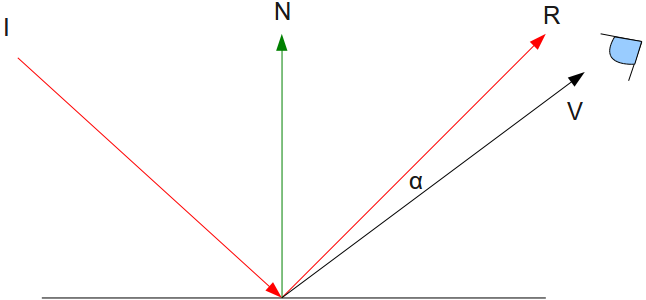
Вычисление косинуса происходит во фрагментарном шейдере. Нормали передаются туда из вершинного, что означает, что OpenGL будет интерполировать их. Поэтому для более реалистичного изображения и отсутствия цветовых границ между соседними похоже ориентированными полигонами вместо обычных в модели могут использоваться вершинные нормали.

#### Отраженный свет

Отраженный свет включает в себя новый элемент - позицию зрителя. Идея в том, что свет падает на поверхность под неким углом, и отражается под таким же (по другую сторону нормали). Если зритель расположен где-то вдоль пути отраженного луча, то он увидит гораздо больше отраженного света.

В конечном итоге объект будет казаться блестящим вдоль луча и яркость будет убывать, если зритель будет отходить в сторону.

Коэффициент отражения зависит больше от объекта, чем от самого света: многие предметы (например, дерево) не отражают свет вообще, поглощая его почти целиком, и даже если стоять на луче отражения эффекта почти не будет.



В модели есть 5 объектов, на которые нужно обратить внимание:

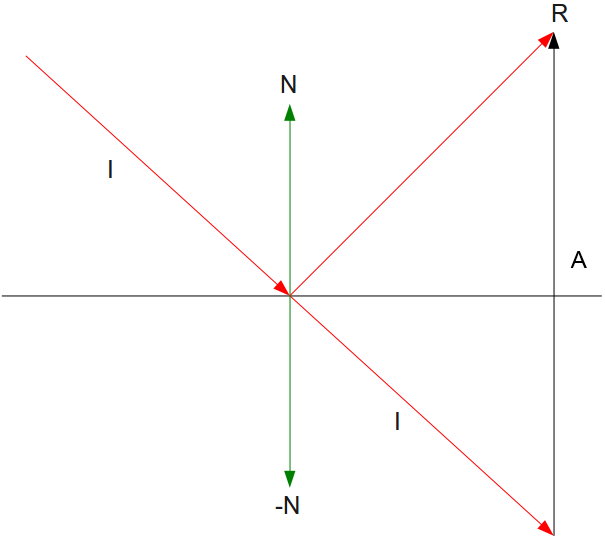
* '**I**' - это случайный луч, падающий на поверхность (и генерирующий рассеянный свет).
* '**N**' - нормаль к поверхности.
* '**R**' - луч света, отраженный от поверхности. Он симметричен 'I' относительно нормали, но направлен противоположно.
* '**V**' - это вектор из точки, в которую падает луч до "глаза" (обозначает зрителя).
* '**α**' - это угол между 'R' и 'V'.

Сила луча будет наибольшей на векторе 'R'. В этом случае 'V' идентичен 'R' и угол между ними равен 0. Как только зритель начинает уходить в сторону, угол растет, а яркость падает. Косинус угла α (скалярное произведение) будет служить коэффициентом отражения в формуле. Как и в случае рассеянного света, если этот угол = 0°, косинус его равен 1 и, т.о., коэффициент максимален. Если же угол ≥ 90°, значение косинуса будет от 0 до -1 и в этом случае влияние отражённого света на итоговый не будет учитываться.

Для подсчета cos(α) требуются 'R' и 'V'.

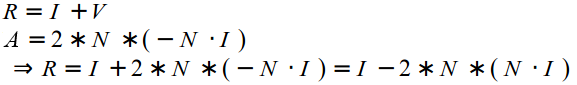
'V' может быть найдет через вычитание позиции точки, в которую падает свет в мировом пространстве, из позиции зрителя (то же в мировом пространстве). Треугольник освещен целиком, поэтому вычислять отраженный свет нужно для каждого отдельно. Для этого требуется позиция пикселя в мировом пространстве. Решение – передавать позицию вершины в мировом пространстве (без учёта перспективы и камеры) из вершинного шейдера, что позволит растеризатору интерполировать позицию пикселя в мировом пространстве и предоставить результат в фрагментный шейдер.

Отраженный луч 'R' будет найден с использованием вектора 'I':

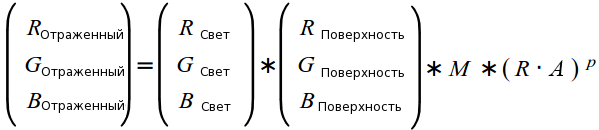


Используя скалярное произведение между 'I' и '-N', можно найти значение вектора, получаемого при проецировании 'I' на '-N'. Это значение равно половине 'А'. Так как 'А' имеет то же направление, что и 'N', можно найти 'А' через умножение 'N' (чья длина 1.0) на удвоенное значение вектора.

Итого:



Итоговая формула компоненты отражённого света выглядит так:



где:

**М** - интенсивность отражения материала (материал, который не имеет отражающих свойств (например дерево) будет иметь интенсивность равной 0, что превратит результат вычислений в 0, а блестящие объекты (например из металла) могут иметь повышенную интенсивность отражения)

**P -**  specular power, сила отражения (иногда называется коэффициентом блеска - shininess factor). От этого параметра зависит величина блика. Сила отражения зависит от материала, поэтому разные объекты будут иметь различные коэффициенты блеска.

## Приложение 1 – Код программы