МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №7

по дисциплине «Компьютерная графика»

Тема: Реализация трехмерного объекта с использованием библиотеки OpenGL

Студент гр. 0382 Корсунов А.А.

Преподаватель Герасимова Т.В.

Санкт-Петербург

2023

Цель работы.

Разработать программу, реализующую представление разработанной в предыдущей лабораторной работе трехмерной сцены с добавлением возможности формирования различного типа проекций теней, используя предложенные функции OpenGL.

Теоретические положения.

Проекции

В OpenGL существуют стандартные команды для задания ортографической (параллельной) и перспективной проекций. Первый тип проекции может быть задан командами

void glOrtho (GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far) и void gluOrtho2D (GLdouble left, GLdouble right, GLdouble bottom, GLdouble top);

Первая команда создает матрицу проекции в усеченный объем видимости (параллелепипед видимости) в левосторонней системе координат. Параметры команды задают точки (left, bottom, znear) и (right, top, zfar), которые отвечают левому нижнему и правому верхнему углам окна вывода. Параметры near и far задают расстояние до ближней и дальней плоскостей отсечения по удалению от точки (0,0,0) и могут быть отрицательными.

Во второй команде, в отличие от первой, значения near и far устанавливаются равными –1 и 1 соответственно. Это удобно, если OpenGL используется для рисования двумерных объектов. В этом случае положение вершин можно задавать, используя команды glVertex2*().

Перспективная проекция определяется командой void gluPerspective (GLdouble angley, GLdouble aspect, GLdouble znear, GLdouble zfar) , которая задает усеченный конус видимости в левосторонней системе координат. Параметр angley определяет угол видимости в градусах по оси y и должен

находиться в диапазоне от 0 до 180. Угол видимости вдоль оси x задается параметром aspect, который обычно задается как отношение сторон области вывода (как правило, размеров окна) Параметры zfar и znear задают расстояние от наблюдателя до плоскостей отсечения по глубине и должны быть положительными. Чем больше отношение zfar/znear, тем хуже в буфере глубины будут различаться расположенные рядом поверхности, так как по умолчанию в него будет записываться ‘сжатая’ глубина в диапазоне от 0 до 1.

Создание источника света

Добавить в сцену источник света можно с помощью команд

void glLight[i f](GLenum light, GLenum pname, GLfloat param);

void glLight[i f](GLenum light, GLenum pname, GLfloat *params).

Параметр light однозначно определяет источник, и выбирается из набора специальных символических имен вида GL_LIGHTi, где i должно лежать в диапазоне от 0 до GL_MAX_LIGHT, которое не превосходит восьми.

Рассмотрим назначение остальных двух параметров (вначале описываются параметры для первой команды, затем для второй) pname:

GL_SPOT_EXPONENT параметр param должен содержать целое или вещественное число от 0 до 128, задающее распределение интенсивности света. Этот параметр описывает уровень сфокусированности источника света или задает экспоненциальное распределение интенсивности светового пучка прожектора. Значение по умолчанию: 0 (рассеянный свет);

GL_SPOT_CUTOFF параметр param должен содержать целое или вещественное число между 0 и 90 или равное 180, которое определяет максимальный угол разброса света. Значение этого параметра есть половина угла в вершине конусовидного светового потока, создаваемого источником. Подробнее будет рассмотрено ниже. Значение по умолчанию: 180 (рассеянный свет);

GL_SPOT_DIRECTION параметр params должен содержать четыре целых или вещественных числа, которые определяют направление света. Значение по

умолчанию: (0.0,0.0,-1.0,1.0);

GL_AMBIENT параметр params должен содержать четыре целых или вещественных значения цветов RGBA, которые определяют цвет фонового освещения. Значение по умолчанию: (0.0,0.0,0.0,1.0);

GL_DIFFUSE параметр params должен содержать четыре целых или вещественных значения цветов RGBA, которые определяют цвет диффузного освещения. Значение по умолчанию: (1.0,1.0,1.0,1.0) для LIGHT0 и (0.0,0.0,0.0,1.0) для остальных;

GL_SPECULAR параметр params должен содержать четыре целых или вещественных значения цветов RGBA, которые определяют цвет зеркального отражения. Значение по умолчанию: (1.0,1.0,1.0,1.0) для LIGHT0 и (0.0,0.0,0.0,1.0) для остальных;

GL_POSITION параметр params должен содержать четыре целых или вещественных, которые определяют положение источника света. Если значение компоненты w равно 0.0, то источник считается бесконечно удаленным и при расчете освещенности учитывается только направление на точку (x, y, z), в противном случае считается, что источник расположен в точке (x, y, z, w). Значение по умолчанию: (0.0,0.0,1.0,0.0);

GL_CONSTANT_ATTENUATION_EXPONENT параметр param должен содержать целое или вещественное число от 0 до 1.0, задающее постоянный коэффициент затухания - kc. Значение по умолчанию: 1.0;

GL_LINEAR_ATTENUATION_EXPONENT параметр param должен содержать целое или вещественное число от 0 до 1.0, задающее линейный коэффициент затухания – kl. Значение по умолчанию: 0.0;

GL_QUADRATIC_ATTENUATION_EXPONENT параметр param должен содержать целое или вещественное число от 0 до 1.0, задающее квадратичный коэффициент затухания – kq. Значение по умолчанию: 0.0.

При изменении положения источника света следует учитывать следующие факты: если положение задается командой glLight..() перед определением ориентации взгляда (командой glLookAt()), то будет считаться, что источник

находится в точке наблюдения. Если положение устанавливается между заданием ориентации и преобразованиями видовой матрицы, то оно фиксируется и не зависит от видовых преобразований. В последнем случае, когда положение задано после ориентации и видовой матрицы, его положение можно менять, устанавливая как новую ориентацию наблюдателя, так и меняя видовую матрицу.

Для использования освещения сначала надо установить соответствующий режим вызовом команды glEnable(GL_LIGHTNING), а затем включить нужный источник командой glEnable(GL_LIGHTn).

Например, для создания источника, расположенного в точке (3,6,5) в мировых координатах, следует выполнить следующий код:

```
Glfloat myLightPosition[]={3.0,6.0,5.0,1.0};
GlLightfv(GL_LIGHT0,GL_POSITION,myLightPosition);
GlEnable(GL_LIGHTING);//ВКЛЮЧЕНИЕ СВЕТА
GlEnable(GL_LIGHT0);//ВКЛЮЧЕНИЕ КОНКРЕТНОГО ИСТОЧНИКА СВЕТА
```

Массив myLightPosition[] определяет положение источника света и передается в функцию glLightfv() вместе с именем GL_LIGHT0, для того чтобы связать его с конкретным источником, обозначенным именем GL_LIGHT0.

Некоторые источники света, такие как настольная лампа, находятся "внутри" сцены, в то время как другие, например солнце, бесконечно удалены от сцены. OpenGL позволяет создавать источники света обоих типов посредством задания положения источника в однородных координатах.

Тогда получается (x, y, z, 1) - локальный источник света в положении (x, y, z), и (x, y, z, 0) - вектор к бесконечно удаленному источнику света в направлении (x, y, z).

Можно также разложить источник света на различные цвета. OpenGL позволяет присвоить различный цвет каждому из трех типов света, испускаемого источником: фоновому, диффузному и зеркальному. Преимущество привязки фонового света к источнику заключается в том, что

его можно включать и выключать во время работы приложения. С помощью приведенного ниже кода определяются массивы для хранения цветов, испускаемых источниками света. Эти массивы затем передаются в функцию glLightfv():

```
//определяем некоторые цвета
GLfloat amb0[]={0.2,0.4,0.6,1.0};
GLfloat diff0[]={0.8,0.9,0.5,1.0};
GLfloat spec0[]={1.0,0.8,1.0,1.0};
//привязываем их к LIGHT0
glLightfv(GL_LIGHT0,GL_AMBIENT,amb0);
glLightfv(GL_LIGHT0,GL_DIFFUSE,diff0);
glLightfv(GL_LIGHT0,GL_SPECULAR,spec0);
```

Цвета задаются в так называемом формате RGBA, что означает: Red - красный, Green - зеленый, Blue - синий и Alpha - альфа. Величина альфа иногда используется для смешения двух цветов на экране. Источники света имеют различные значения по умолчанию.

Для всех источников: фоновая компонента по умолчанию - (0,0,0,1), наименьшая яркость - черный. Для источника GL_LIGHT0: диффузная компонента по умолчанию - (1,1,1,1), наибольшая яркость – белый; зеркальная компонента по умолчанию - (1,1,1,1), наибольшая яркость - белый. Для других источников света значения диффузного и зеркального компонентов по умолчанию устанавливаются в черный цвет.

Прожекторы

Как упоминалось ранее, можно заставить позиционный источник света работать в качестве прожектора. Источники света по умолчанию являются точечными источниками. Это означает, что они излучают свет равномерно по всем направлениям. Однако OpenGL позволяет превратить их в прожекторы, так чтобы они излучали свет в ограниченном числе направлений. На рис. 4.3 показан прожектор, нацеленный в направлении d с углом пропускания ("cutoff

angle") - a.

Для создания прожектора необходимо определить границы конуса света. В точках, лежащих вне конуса пропускания, свет не виден вообще. Для таких вершин, как P, которые лежат внутри этого конуса, количество света, достигающего точки P, пропорционально множителю cosE (b), где b - угол между вектором d и прямой, соединяющий источник с точкой P. Также необходимо задать направление света прожектора, которое определяет ось конуса света. По умолчанию это направление равно (0,0,-1), то есть свет испускается вдоль отрицательной оси z. В дополнение к направлению и углу прямого выхода излучения для прожектора существуют два способа управления интенсивностью распространения света внутри конуса. Во-первых, можно задать коэффициент затухания, который умножается на интенсивность света. Так же можно установить параметр GL_SPOT_EXPONENT для управления концентрацией света. Интенсивность имеет наивысшее значение в центре конуса. И она затухает по направлению к краям конуса – по косинуса угла, возведенному в степень точечной экспоненты. Показатель степени E выбирается пользователем так, чтобы обеспечить нужное уменьшение интенсивности света в зависимости от угла.

Параметры прожектора устанавливаются следующим образом: с помощью функции glLightf() задается его единственный параметр, а с помощью функции glLightfv() - вектор:

```
glLightf(GL_LIGHT0,GL_SPOT_CUTOFF,45.0);// угол пропускания - 45
glLightf(GL_LIGHT0,GL_SPOT_EXPONENT,4.0);// значение E
GLfloat dir[]={2.0,1.0,-4.0};// направление прожектора
glLightfv(Gl_LIGHT0,GL_SPOT_DIRECTION,dir);
```

По умолчанию установлены следующие значения этих параметров: d=(0,0,-1), a=180, E=0, что соответствует действующему во всех направлениях точечному источнику света.

Модель освещения в OpenGL

В OpenGL используется модель освещения Фонга, в соответствии, с которой цвет точки определяется несколькими факторами: свойствами материала и текстуры, величиной нормали в этой точке, а также положением источника света и наблюдателя. Для корректного расчета освещенности в точке надо использовать единичные нормали, однако, команды типа glScale..(), могут изменять длину нормалей. Чтобы это учитывать, используется уже упоминавшийся режим нормализации нормалей, который включается вызовом команды glEnable(GL_NORMALIZE).

OpenGL предусматривает задание трех параметров, определяющих общие законы применения модели освещения. Эти параметры передаются в функцию glLightModel() и некоторые ее модификации. Для задания глобальных параметров освещения используются команды:

void glLightModel[i f](GLenum pname, GLenum param)

void glLightModel[i f]v(GLenum pname, const GLtype *params)

Аргумент pname определяет, какой параметр модели освещения будет настраиваться и может принимать следующие значения:

GL_LIGHT_MODEL_LOCAL_VIEWER - является ли точка наблюдения локальной или удаленной. OpenGL вычисляет зеркальные отражения с помощью "промежуточного вектора" h=s+v, описанного ранее. Истинные направления s и v, различаются для каждой вершины сетки. Если источник света является направленным, то вектор s-величина посто-янная, а v все же изменяется от вершины к вершине. Скорость визуализации возрастает, если сделать и вектор v постоянным для всех вершин. По умолчанию OpenGL использует значение v=(0,0,1), при этом вектор указывает в сторону положительной оси z в координатах камеры. В то же время можно принудительно заставить графический конвейер вычислять истинное значение вектора v для каждой вершины с помощью выполнения оператора:

GlLightModel(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE);

Параметр param должен быть булевским и задает положение наблюдателя. Если он равен FALSE, то направление обзора считается параллельным оси z, вне зависимости от положения в видовых координатах. Если же он равен TRUE, то наблюдатель находится в начале видовой системы координат. Это может улучшить качество освещения, но усложняет его расчет. Значение по умолчанию: FALSE.;

GL_LIGHT_MODEL_TWO_SIDE - правильно ли происходит закрашивание обеих сто-рон полигона. Параметр param должен быть булевским

и управляет режимом расчета освещенности, как для лицевых, так и для обратных граней. Если он равен FALSE, то осве-щенность рассчитывается только для лицевых граней. Если же он равен TRUE, расчет про-водится и для обратных граней. Значение по умолчанию: FALSE. Каждая полигональная грань модели имеет две стороны. При моделировании можно рассматривать внутреннюю сторону и внешнюю. Принято заносить эти вершины в список против часовой стрелки, если смотреть с внешней стороны объекта. Большинство каркасных объектов представляют сплошные тела, ограничивающие некоторое пространство, так что четко определены понятия внешней и внутренней стороны. Для таких объектов камера может наблюдать только внешнюю поверхность каждой грани (если конечно камера не находится внутри объекта). При правильном удалении невидимых поверхностей внутренняя поверхность каждой грани скрыта от глаза какой-нибудь более близкой гранью.

В OpenGL нет понятия "внутри" и "снаружи", он может различать только "лицевые грани" и "нелицевые грани". Грань является лицевой (front face), если ее вершины расположены в списке против часовой стрелки, в том порядке, каком их видит глаз. Можно заменить этот порядок на обратный с помощью функции glFrontFace(GL_CW),в которой обусловлено, что грань является лицевой только в том случае, если ее вершины занесены в список в порядке по часовой стрелке. Для объекта, ограничивающего некоторое пространство, все грани, которые видит глаз, являются лицевыми, и OpenGL правильно рисует и закрашивает их. Нелицевые грани также рисуются в OpenGL, но, в конце концов, они скрываются за более близкими лицевыми гранями. Можно ускорить работу процессора, если запретить OpenGL визуализацию нелицевых граней. При этом используется следующий код:

glCullFace(GL_BACK);

glEnable(GL_CULL_FACE);.

Для правильного закрашивания нелицевых граней нужно проинструктировать OpenGL с помощью оператора glLightModel (GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE). При вы-полнении этой команды OpenGL изменяет направления нормалей всех нелицевых граней таким образом, чтобы они указывали на наблюдателя, после чего закрашивание осуществляется корректно. Замена величины GL_TRUE на GL_FALSE отключает эту опцию. Грани, нарисованные с помощью OpenGL, не отбрасывает теней, поэтому все нелицевые грани получают от источника такой же свет, даже если между ними и источником света находится какая-нибудь

другая грань;

GL_LIGHT_MODEL_AMBIENT - цвет глобального фонового света. Параметр params должен содержать четыре целых или вещественных числа, которые определяют цвет фоно-вого освещения даже в случае отсутствия определенных источников света. Для любой задан-ной сцены можно установить глобальный фоновый свет, не зависимый ни от какого опреде-ленного источника. Для создания такого освещения следует задать его цвет с помощью следующих команд:

```
GLfloat   amb[] = {0.2, 0.3, 0.1, 1.0};
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, amb);
```

Этот код придает источнику фонового света цвет (0.2, 0.3, 0.1). Значение по умолчанию составляет (0.2, 0.2, 0.2, 0.1), так что фоновый свет присутствует всегда, если только умышленно не изменить его. Задание фоновому источнику ненулевого значения обеспечивает видимость объектов сцены, даже если вы не активизировали ни одной функции освещения;

GL_LIGHT_MODEL_COLOR_CONTROL отделение зеркальной составляющей цвета. Для обычных расчетов освещенности фоновая, диффузная, зеркальная и эмиссионная составляющие вычисляются и просто складываются друг с другом. По умолчанию отображение текстур применяется после освещения, так что зеркальные блики могут появиться приглушенными, или текстурирование будет выглядеть по-другому. При следующем вызове – glLightModel(GL_LIGHT_MODEL_COLOR_CONTROL,GL_ SEPARATE_SPECULAR_COLOR) OpenGL отделяет вычисление зеркального цвета из приложения. После этого освещение генерирует два цвета для каждой вершины: первоначальный цвет, состоящий из неотраженных составляющих освещенности, и второй цвет, являющийся суммой зеркальных составляющих освещенности. При отображении текстур только первый цвет комбинируется с цветами текстуры. После выполнения операции текстурирования второй цвет добавляется к итоговой комбинации первого и текстурного компонентов цвета. Объекты, для которых выполнено освещение и текстурирование с отделением зеркального цвета, обычно более видимы и имеют более заметные зеркальные блики. Для возвращения к установкам по умолчанию необходимо сделать вызов glLightModeli(GL_LIGHT_MODEL_COLOR_CONTROL,GL_SINGLE_ COLOR). После этого опять первоначальный цвет будет состоять из всех составляющих цвета: рассеянной, диффузной, эмиссионной и зеркальной. Составляющие освещения не прибавляются после текстурирования.

Работа со свойствами материалов в OpenGL

Влияние источника света можно увидеть только при отражении света от поверхности объекта. В OpenGL предусмотрены возможности задания различных коэффициентов отраже-ния, фигурирующих в уравнении интенсивности отраженного света. Эти коэффициенты устанавливаются с помощью различных версий функции glMaterial(), причем коэффициенты можно устанавливать отдельно для лицевых и нелицевых граней.

Для задания параметров текущего материала используются команды:

void glMaterial[i f](GLenum face, GLenum pname, GLtype param);

void glMaterial[i f]v(GLenum face, GLenum pname, GLtype *params);

С их помощью можно определить рассеянный, диффузный и зеркальный цвета материала, а также цвет, степень зеркального отражения и интенсивность излучения света, если объект должен светиться. Какой именно параметр будет определяться значением param, зависит от значения pname:

GL_AMBIENT параметр params должен содержать четыре целых или вещественных значения цветов RGBA, которые определяют рассеянный цвет материала (цвет материала в тени). Значение по умолчанию: (0.2,0.2,0.2,1.0);

GL_DIFFUSE параметр params должен содержать четыре целых или вещественных значения цветов RGBA, которые определяют цвет диффузного отражения материала. Значение по умолчанию: (0.8,0.8,0.8,1.0);

GL_SPECULAR параметр params должен содержать четыре целых или вещественных значения цветов RGBA, которые определяют цвет зеркального отражения материала. Значение по умолчанию: (0.0,0.0,0.0,1.0);

GL_SHININESS параметр params должен содержать одно целое или вещественное значение в диапазоне от 0 до 128, которое определяет степень зеркального отражения материала. Значение по умолчанию: 0;

GL_EMISSION параметр params должен содержать четыре целых или вещественных значения цветов RGBA, которые определяют интенсивность излучаемого света материала. Здесь устанавливается эмиссионный цвет для грани. В дополнение к рассеянному, диффузному и зеркальному компоненту

цвета материалы могут иметь эмиссионный цвет, который имитирует свет, исходящий от объекта. В модели освещения OpenGL, эмиссионный цвет поверхности добавляет объекту яркость, но на него не влияют никакие источники света. Значение по умолчанию: (0.0,0.0,0.0,1.0);

GL_AMBIENT_AND_DIFFUSE эквивалентно двум вызовам команды glMaterial..() со значением pname GL_AMBIENT и GL_DIFFUSE и одинаковыми значениями params. Задаваемые коэффициенты фонового и диффузного отражения устанавливаются равным одному и тому же значению. Такая установка сделана для удобства, поскольку коэффициенты фонового и диффузного отражения часто выбираются одинаковыми.

Из этого следует, что вызов команды glMaterial[i f]() возможен только для установки степени зеркального отражения материала. В большинстве моделей учитывается диффузный и зеркальный отраженный свет; первый определяет естественный цвет объекта, а второй – размер и форму бликов на его поверхности. Параметр face определяет тип граней, для которых задается этот материал и может принимать следующие значения:

GL_FRONT - задается коэффициент отражения для лицевых граней;

GL_BACK - задается коэффициент отражения для нелицевых граней;

GL_FRONT_AND_BACK - задается коэффициент отражения и для лицевых граней, и для нелицевых граней.

Если в сцене материалы объектов различаются лишь одним параметром, рекомендуется сначала установить нужный режим, вызвав glEnable() с параметром GL_COLOR_MATERIAL, а затем использовать команду:

void glColorMaterial(GLenum face, GLenum pname);

где параметр face имеет аналогичный смысл, а параметр pname может принимать все перечисленные значения. После этого, значения, выбранного с помощью pname, свойства материала для конкретного объекта (или вершины) устанавливается вызовом команды glColor(), что позволяет избежать вызовов более ресурсоемкой команды glMaterial() и повышает эффективность программы. Например, следующий код:

GLfloat myDiffuse[]= {0.8, 0.2, 0.0, 1.0};

glMaterialfv(GL_FRONT, GL_DIFFUSE, myDiffuse);

присваивает коэффициенту диффузного отражения (pdr,pdg,pdb) значение (0.8,0.2,0,0) для всех последовательно заданных лицевых граней. Коэффициенты отражения задаются в формате RGBA в виде четверки, аналогично заданию цвета.

Задание.

Разработать программу, реализующую представление трехмерной сцены с добавлением возможности формирования различного типа проекций, отражений, используя предложенные функции OpenGL(модель освещения, типы источников света, свойства материалов(текстура)).

Разработанная программа должна быть пополнена возможностями остановки интерактивно различных атрибутов через вызов соответствующих элементов интерфейса пользователя:

- замена типа источника света,
- управление положением камеры,
- изменение свойств материала модели, как с помощью мыши, так и с помощью диалоговых элементов)

В отчете должны быть подробно описаны все реализованные в программе методы с комментариями.

Ход работы.

Лабораторная работа выполнялась на языке программирования «Python» с использованием модулей «tkinter» и «OpenGL» на базе программы предыдущей лабораторной работы.

1. Ортографическая и перспективная проекции

В 6-ой лабораторной работе использовалась перспективная проекция

для отображения сцена из трехмерных фигур. Была добавлена возможность использования ортографической проекции и переключение между ней и перспективной проекцией.

Следующий код отвечает за установку и выбор между проекциями:

```
glMatrixMode(GL_PROJECTION)
glLoadIdentity()
if not select_matrix_projection:
    glOrtho(-4, 4, -4, 4, -150, 150)  # left right bottom top near far # ортографическая
    # left - координата левой границы видимой области вдоль оси X;
    # right - координата правой границы видимой области вдоль оси X;
    # bottom - координата нижней границы видимой области вдоль оси Y;
    # top - координата верхней границы видимой области вдоль оси Y;
    # near - координата ближней плоскости отсечения;
    # far - координата дальней плоскости отсечения.
else:
    gluPerspective(40.0, float(SCREEN_SIZE[0] / SCREEN_SIZE[1]), 0.1, 50.0)  # перспективная
glMatrixMode(GL_MODELVIEW)
```

через функцию glOrtho устанавливается ортографическая проекция, через функцию gluPerspective – перспективная, переключение реализовано через кнопку интерфейса на экране:

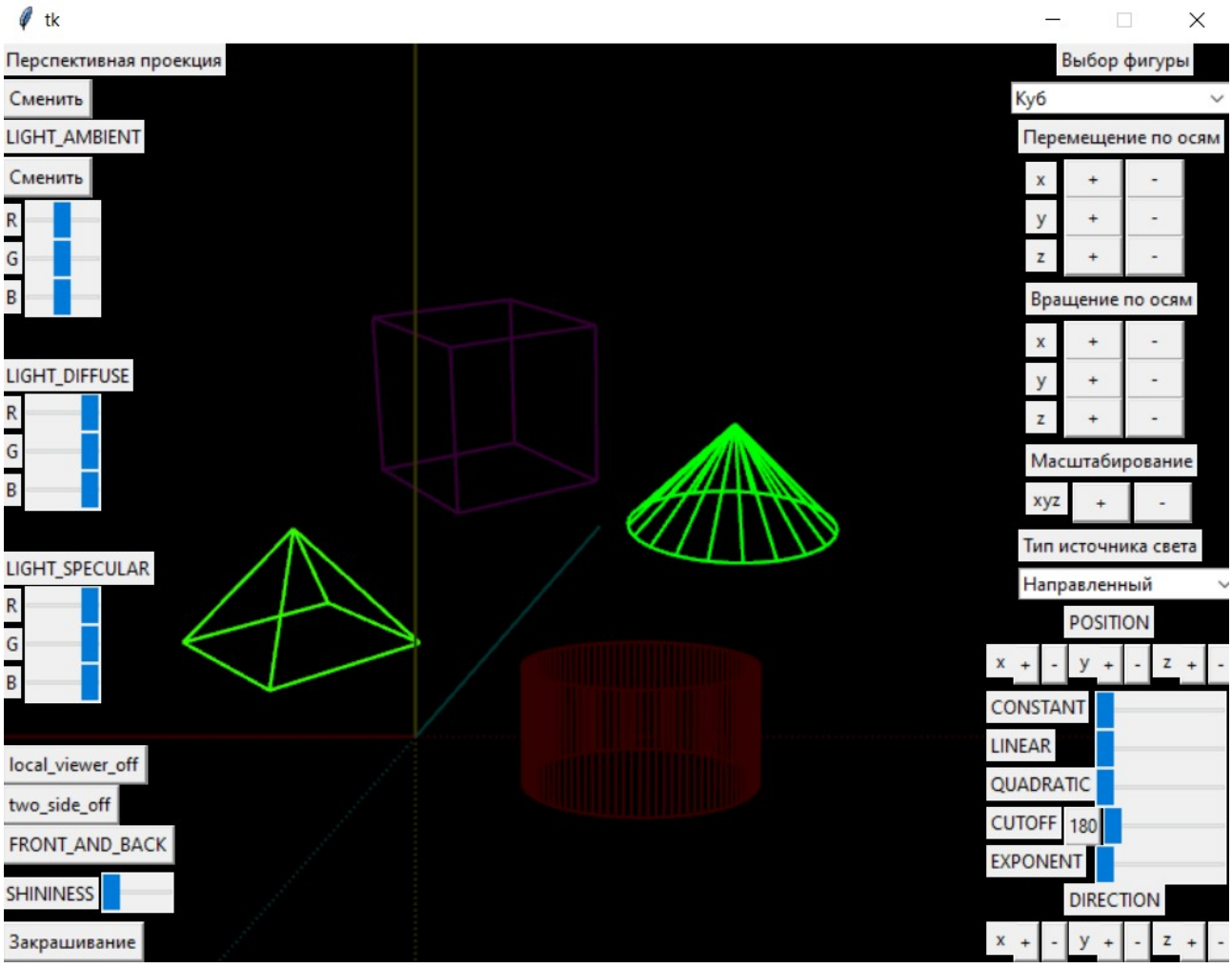

Рисунок 1 — Перспективная проекция

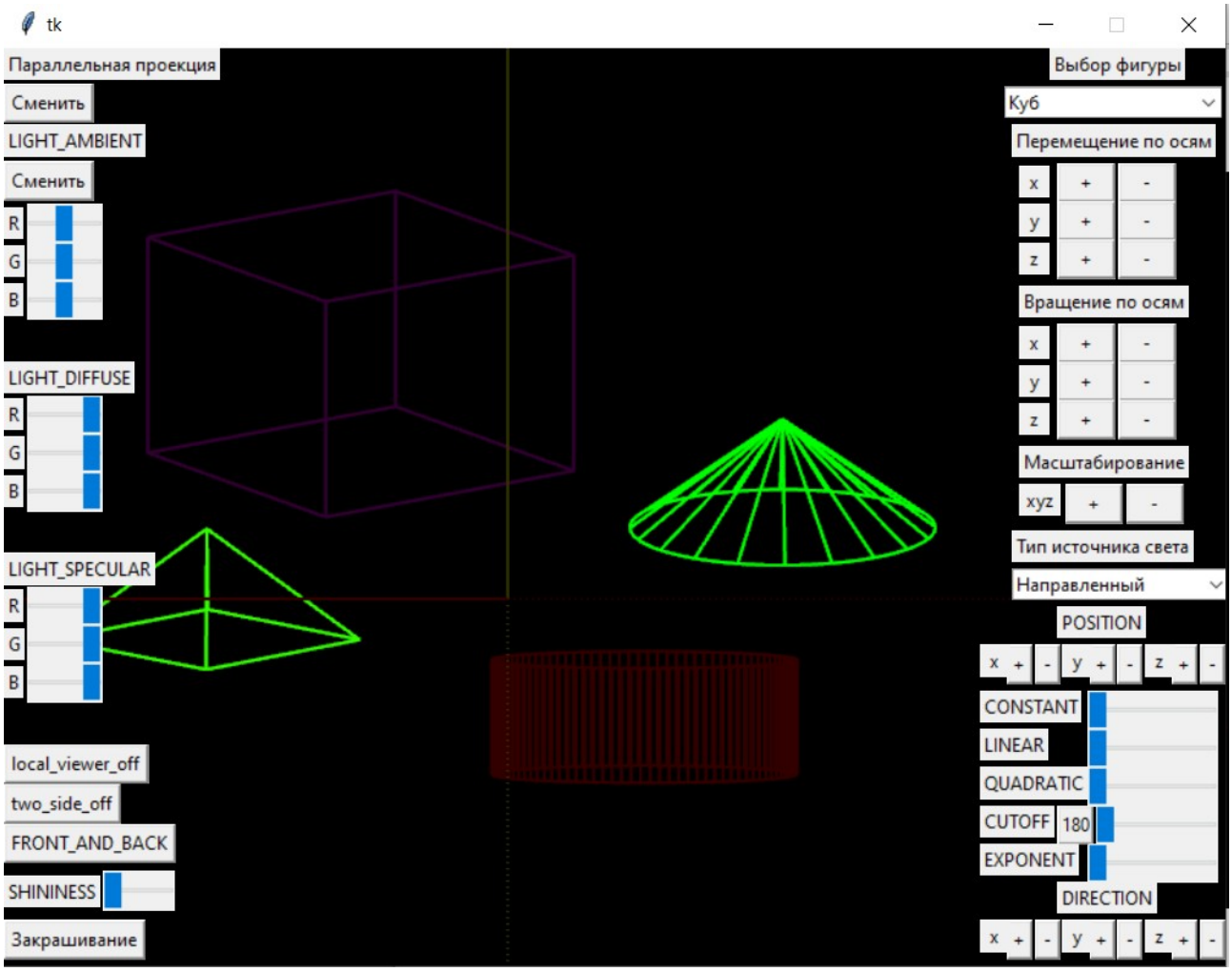

Рисунок 2 — Ортографическая проекция

Как видно из скриншотов (рис.1 и рис.2) переключение между проекциями меняет отображения фигур в сцене, кнопка переключения («Сменить») между проекциями находится слева-сверху под меткой названия проекции (метки проекций переключаются в зависимости от текущей проекции).

Перемещение камеры было реализовано и продемонстрировано в предыдущей лабораторной работе, более наглядную демонстрацию можно будет увидеть к соответствующей видеозаписи демонстрации работы программы

2. Добавление источника света и задание его типа

Освещение и источник света были добавлены следующими командами:

```
glEnable(GL_LIGHTING)
glEnable(GL_LIGHT0)
```

Далее был реализован класс Lighter, хранящий в полях параметры освещения и материала, а также рисующий отображения направления (куда) и исток (откуда) света (в случае, если таковой имеется)

В OpenGL источник света может быть трех типов, переключение между типами было реализовано через виджет combobox справа экрана под меткой «Тип источника света» (рис.2 — выбран направленный тип), параметры для соответствующих типов расположены снизу от этого виджета.

Далее будут описаны реализация и продемонстрирована информация по каждому из трех типов:

а) Направленный тип источника света

При задании направленного источника света в OpenGL имеется только параметр GL_POSITION, который отвечает вектор направления света:

```
if self.light_sample[0] == 0:

    glColor4f(0, 0, 255, 1)

    self.position[3] = 0

    glLightfv(GL_LIGHT0, GL_POSITION, self.position)

    glLineWidth(4)
    glBegin(GL_LINES)
    glVertex3f(0, 0, 0)
    glVertex3f(x, y, z)
    glEnd()
```

В этом коде объявляется, что тип источника — направленный («self.position[3] = 0» - в параметр GL_POSITION в последний элемент массива устанавливаем

1), далее через примитив GL_LINES отображается вектор, который задает направление освещения.

Демонстрация:

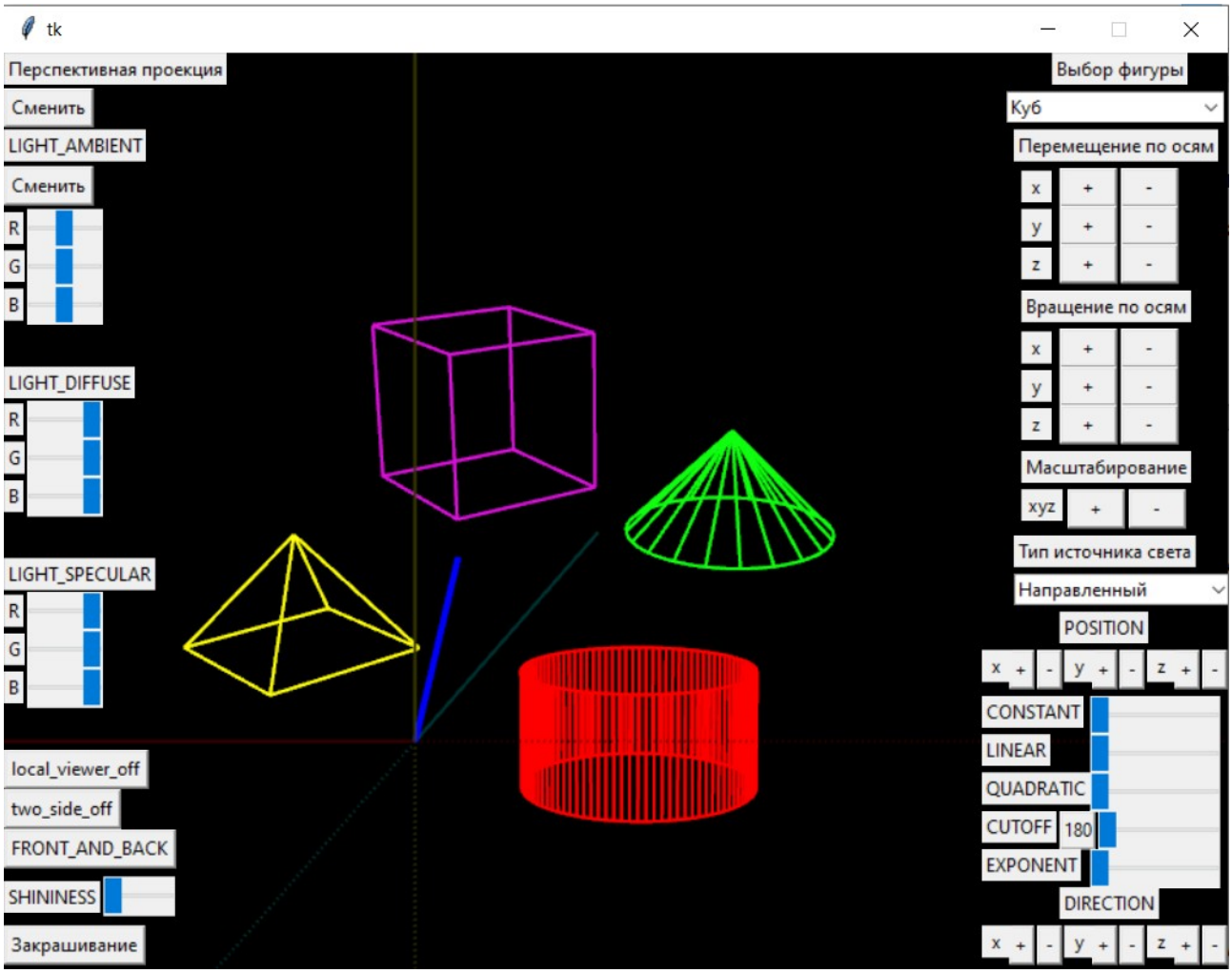

Рисунок 3 — Направленный тип источника света

Координаты направления можно изменять через кнопки под меткой «POSITION», например чтобы передвинуть вектор по оси x в положительную сторону (влево на экране), надо будет нажать на «+» справа от метки «x»:

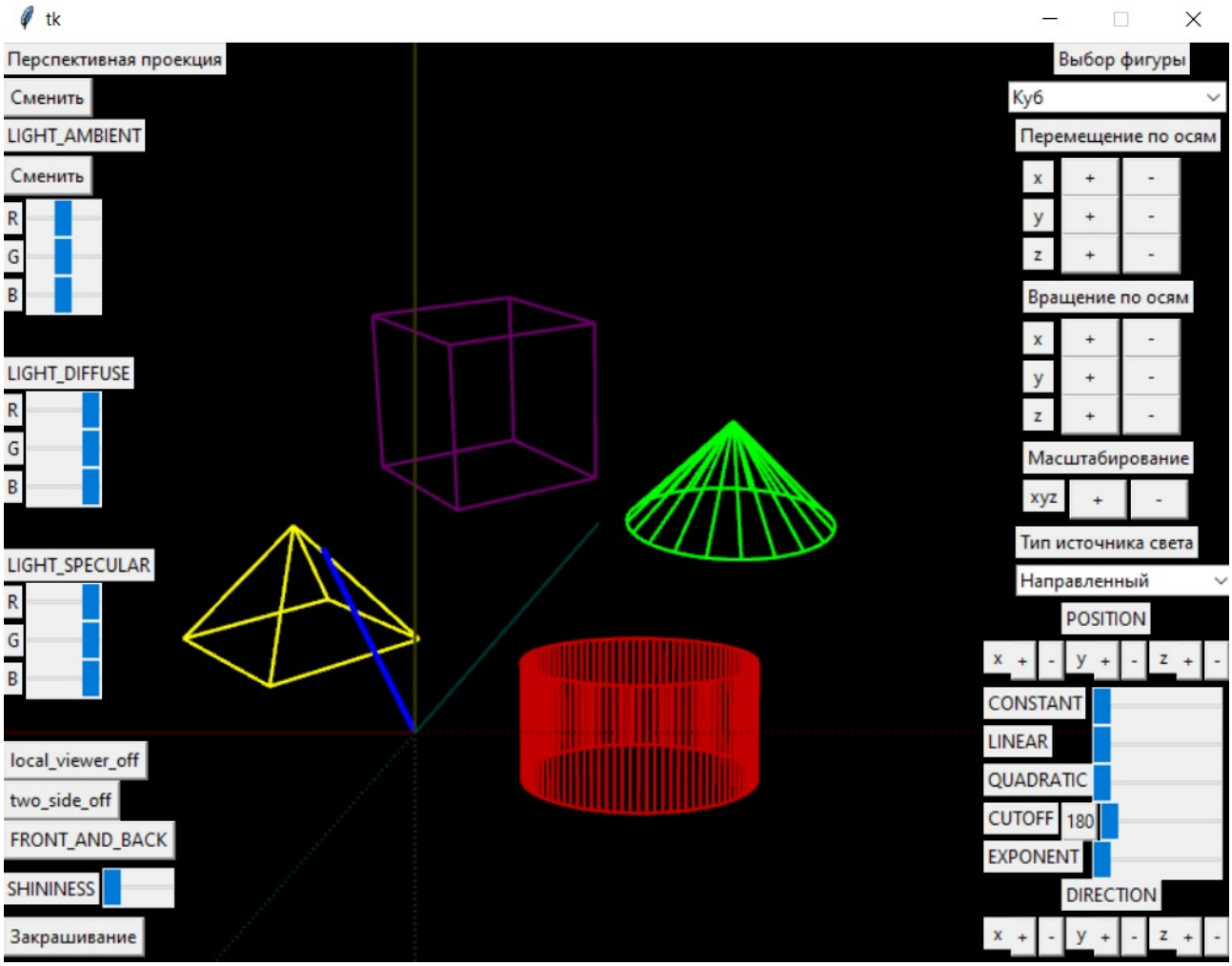

Рисунок 4 — Изменение положения векторам

Как видно из рис. 4, при изменении положения освещение сцены изменилось (все фигуры, кроме пирамиды, стали темнее)

б) Точечный тип источника

Точечный источник светит во все стороны от истока равномерно

При задании точечного источника света в OpenGL имеются параметры:

- GL_POSITION – исток источника света (откуда он светит);
- GL_CONSTANT – постоянный коэффициент затухания (от 0 до 1);
- GL_LINEAR – линейный коэффициент затухания (от 0 до 1);
- GL_QUADRATIC – квадратичный коэффициент затухания (от 0 до 1):

```
elif self.light_sample[0] == 1:
    glColor4f(255, 255, 255, 1)
```

```
    self.position[3] = 1

    glLightfv(GL_LIGHT0, GL_POSITION, self.position)  # задает положение источника света в пространстве.
    # Чем дальше источник света от поверхности, тем больше затухание света.
    glLight(GL_LIGHT0, GL_CONSTANT_ATTENUATION, self.constant)  # задает постоянное затухание света в
    # зависимости от расстояния между источником света и поверхностью. Чем больше это значение,
    # тем меньше затухание.
    glLightfv(GL_LIGHT0, GL_LINEAR_ATTENUATION, self.linear)  # задает линейное затухание света. Оно
    # увеличивает затухание света линейно в зависимости от расстояния между источником света и поверхностью
    glLightfv(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, self.quadratic)  # задает квадратичное затухание света.
    # Оно увеличивает затухание света квадратично в зависимости от расстояния между источником света
    # и поверхностью.

    glPointSize(50)
    glBegin(GL_POINTS)
    glVertex3f(x, y, z)
    glEnd()
```

В этом коде объявляется, что тип источника точечный, задаются все параметры для точечного типа, после чего рисуется большая точка, которая соответствует истоку точечного источника света.

Демонстрация:

по умолчанию источник света появляется в точке (0, 0, 0) и плохо демонстрирует работу освещения, позицию можно изменяться через уже описанные виджеты по перемещению (под меткой POSITION, рис. 4), но чтобы не тратить время была написана функция, которая по нажатию клавиши «r» на клавиатуре, перемещает источник в удобную для демонстрации позицию:

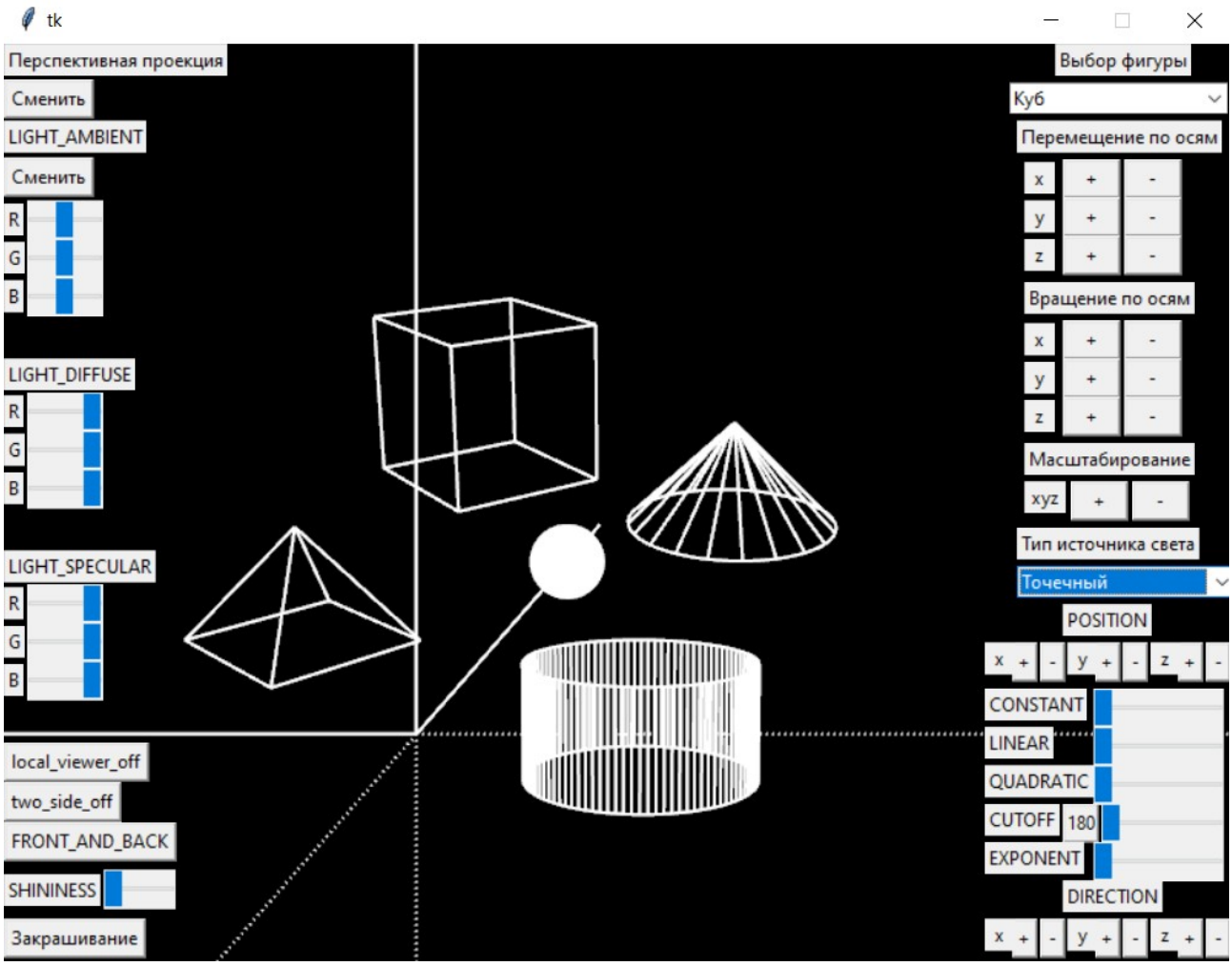

Рисунок 5 — Точечный источник света

Сцена белая, потому что все коэффициенты затухания установлены в 0. Чтобы изменить значения коэффициентов можно воспользоваться виджетом Scale (ползунок) для каждого из коэффициентов (рис. 5):

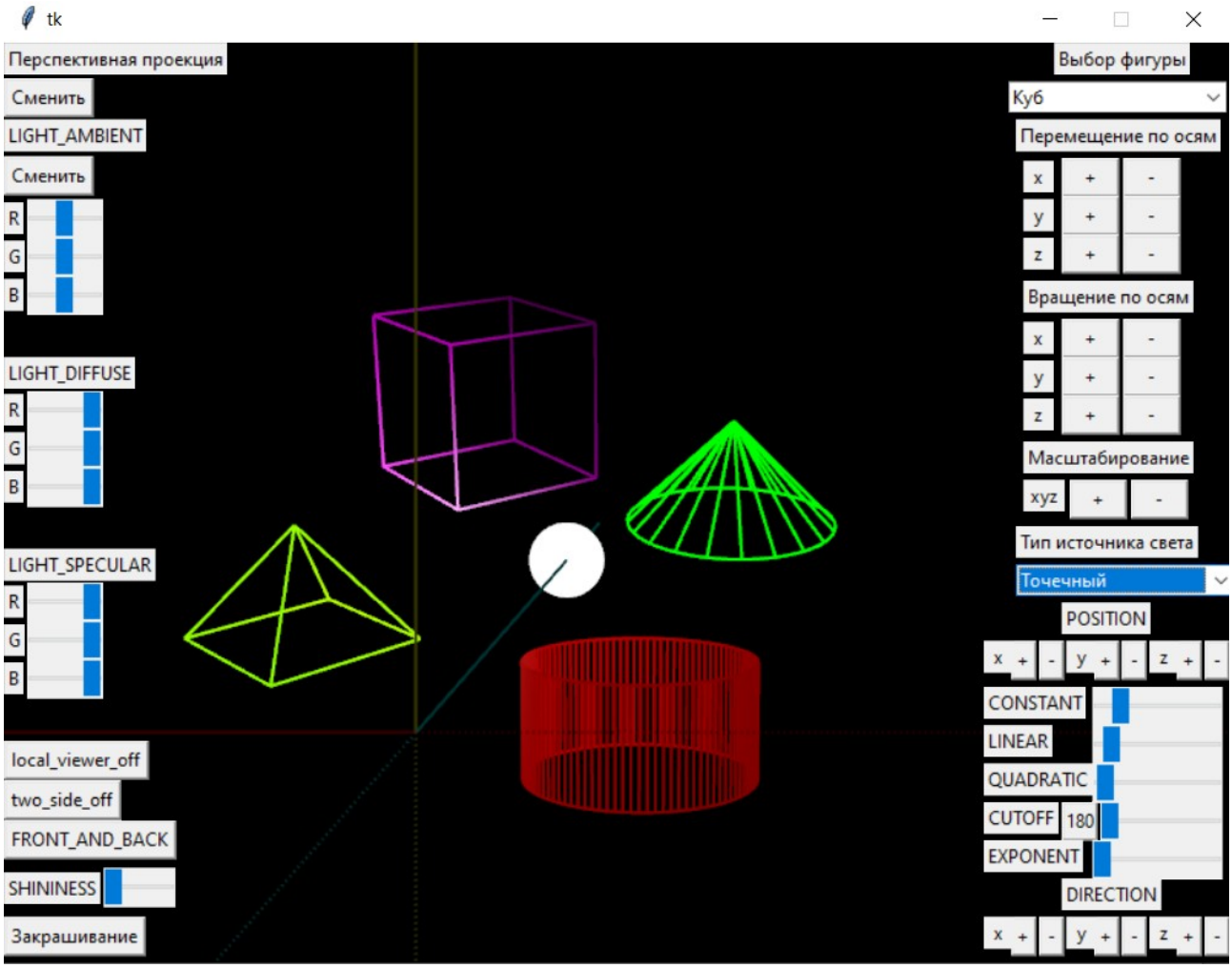

Рисунок 6 — Значения коэффициентов были изменены

в) Прожектор

Прожектор имеет исток и светит в указанном направлении. У прожектора имеются следующие параметры:

- GL_POSITION — исток источника света (откуда он светит);
- GL_DIRECTION – направление источника света (куда он светит);
- GL_CONSTANT – постоянный коэффициент затухания (от 0 до 1);
- GL_LINEAR – линейный коэффициент затухания (от 0 до 1);
- GL_QUADRATIC – квадратичный коэффициент затухания (от 0 до 1):
- GL_CUTOFF – угловая ширина светового луча (от 0 до 90 или 180);
- GL_EXPONENT – концентрация светового луча (от 0 до 128):

```
elif self.light_sample[0] == 2:

    self.position[3] = 1

    direction = self.direction
    position = self.position

    glLightfv(GL_LIGHT0, GL_POSITION, position)
    glLightfv(GL_LIGHT0, GL_CONSTANT_ATTENUATION, self.constant)
    glLightfv(GL_LIGHT0, GL_LINEAR_ATTENUATION, self.linear)
    glLightfv(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, self.quadratic)

    glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, direction)  # направление света прожектора.
    glLightfv(GL_LIGHT0, GL_SPOT_CUTOFF, self.cutoff)  # угловая ширина светового луча.
    glLightfv(GL_LIGHT0, GL_SPOT_EXPONENT, self.exponent)  # концентрация светового луча.

    glPointSize(50)
    glBegin(GL_LINES)
    glColor4f(0, 255, 255, 1)
    glVertex3f(x, y, z)
    glColor4f(255, 255, 0, 1)
    glVertex3f(self.direction[0], self.direction[1], self.direction[2])
    glEnd()
```

В этом коде объявляется, что тип источника — точечный (прожектор наследует точечный тип и расширяет его), далее устанавливаются все параметры источника, после чего рисуется линия (через примитив GL_LINES), по точка истока и направления источника:

Демонстрация:

По умолчанию обе точки прожектора находятся в (0, 0, 0), что не демонстрирует освещение прожектора, поэтому с помощью той же клавиши «r» можно сразу переместить точки в более удобное для демонстрации место (можно так же перемещать через соответствующие виджеты, как это было с меткой POSITION – рис. 6)

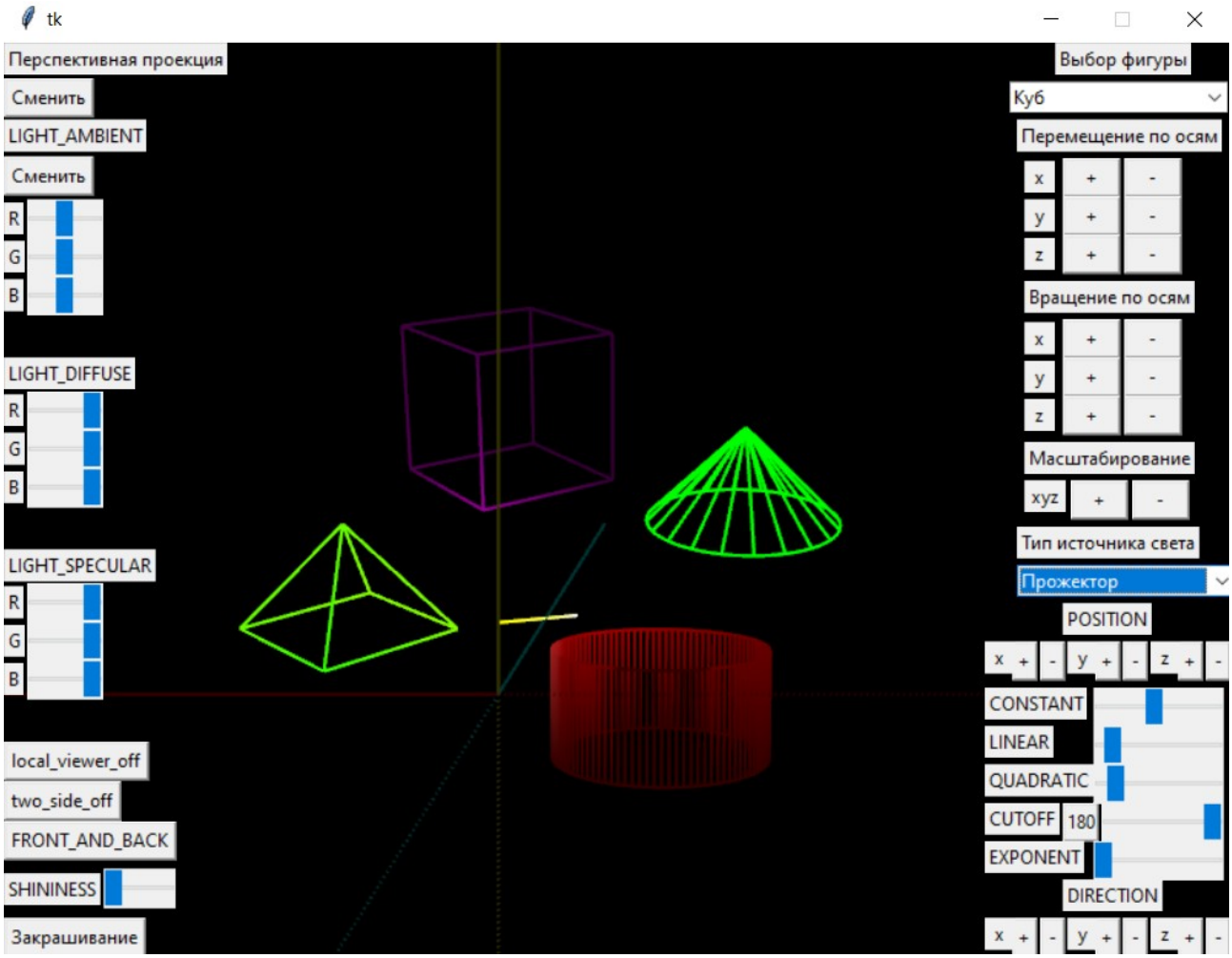

Рисунок 7 — Освещение прожектора

Параметры освещения прожектора так же настраиваются через соответствующие ползунки. Для большей наглядности следует переместить камеру к в направления от истока и закрасить фигуры:

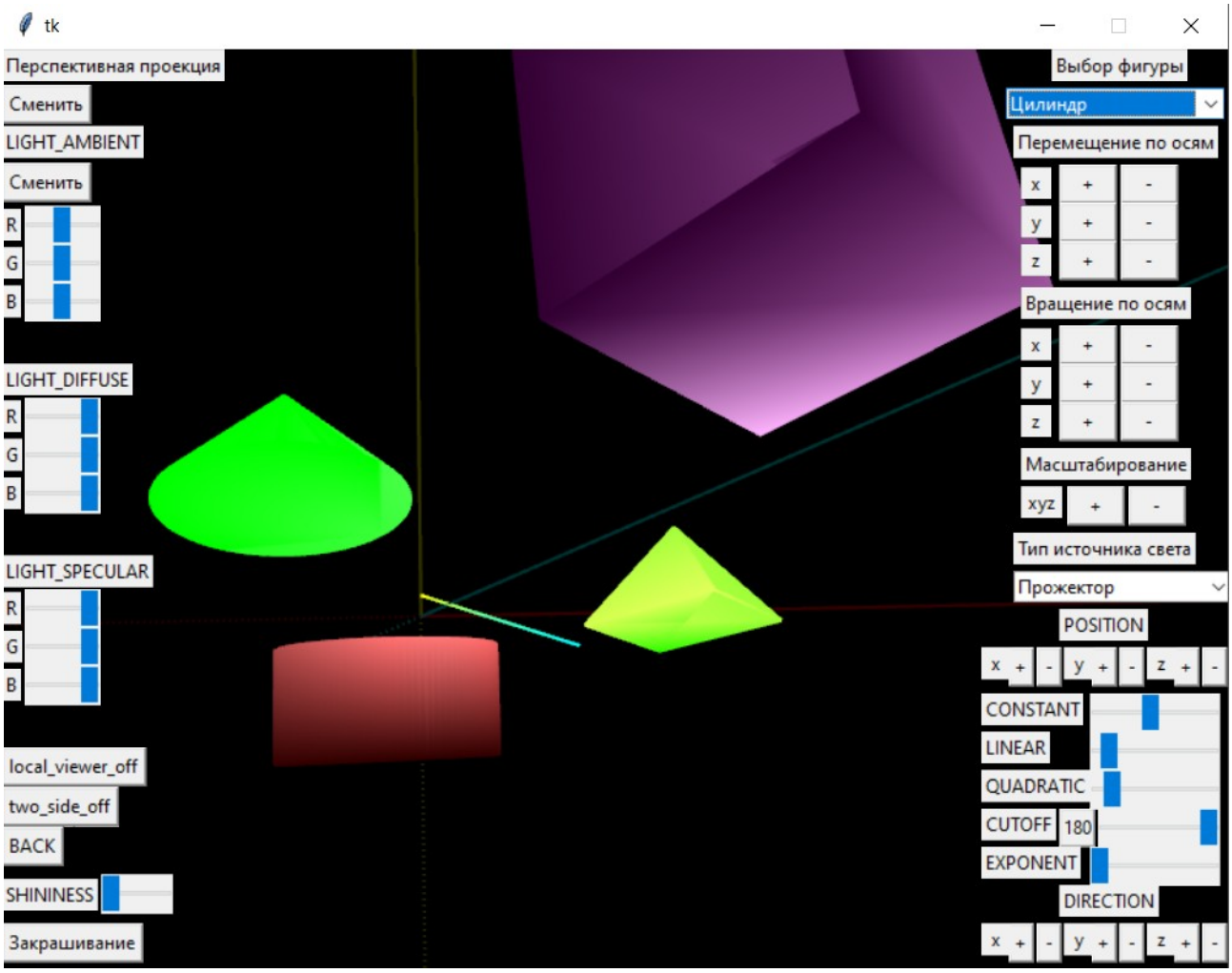

Рисунок 8 — Перемещение камеры для наглядности освещения прожектора

г) Общие параметры освещения для всех типов источника света

Общими параметрами выступают:

- GL_LIGHT_AMBIENT – цвет фонового освещения (RGB);
- GL_LIGHT_DIFFUSE – цвет диффузного освещения (RGB);
- GL_LIGHT_SPECULAR – цвет зеракального отражения (RGB);
- Параметр, отвечающий за лицо граней (face) (1 из 3):

```
glLightfv(GL_LIGHT0, GL_AMBIENT, self.ambient)  # цвет фонового освещения
glLightfv(GL_LIGHT0, GL_DIFFUSE, self.diffuse)  # цвет диффузного освещения
glLightfv(GL_LIGHT0, GL_SPECULAR, self.specular)  # цвет зеркального отражения
```

Демонстрация:

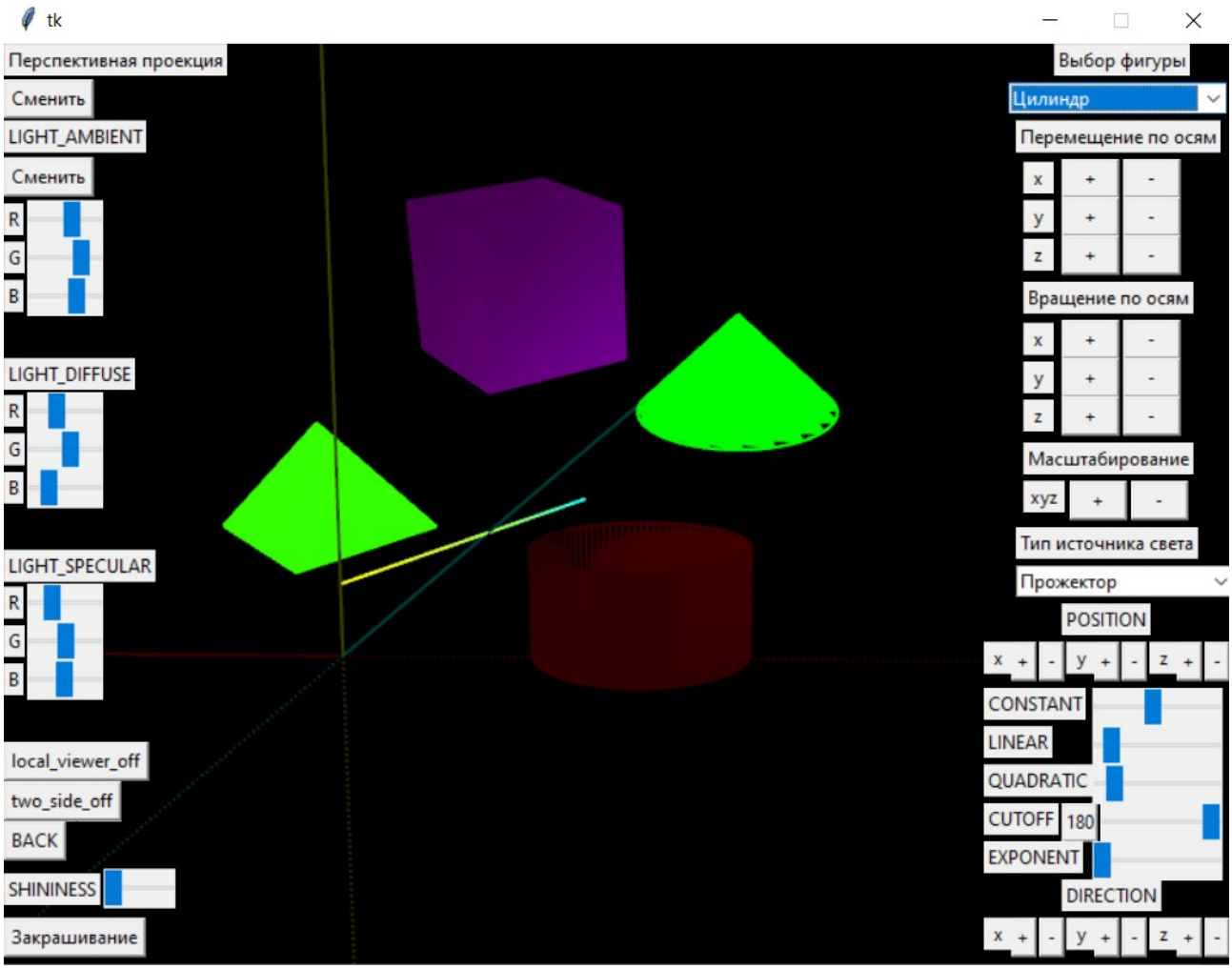

Рисунок 9 — Изменение параметров освещения

Как можно заметить из рис.9 настройка параметров по RGB производится с помощью ползунков слева экрана

3. Модель освещения

Для модели освещения реализованы изменения параметров:

- GL_LIGHT_MODEL_AMBIENT – цвет глобального фонового цвета (RGB);
- GL_LIGHT_MODEL_LOCAL_VIEWER – является ли точка наблюдения локальной или удаленной (GL_TRUE или GL_FALSE);
- GL_LIGHT_MODEL_TWO_SIDE – правильно ли происходит закрашивание обеих сторон полигона (GL_TRUE или GL_FALSE):

```
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, self.model_ambient)  # цвет глобального фонового света
glLightModelfv(GL_LIGHT_MODEL_LOCAL_VIEWER, self.model_local_viewer)  # является ли
```

точка наблюдения локальной # или удаленной
glLightModelfv(GL_LIGHT_MODEL_TWO_SIDE, self.model_two_side) # правильно ли происходит закрашивание обеих # сто-рон полигона

Демонстрация:

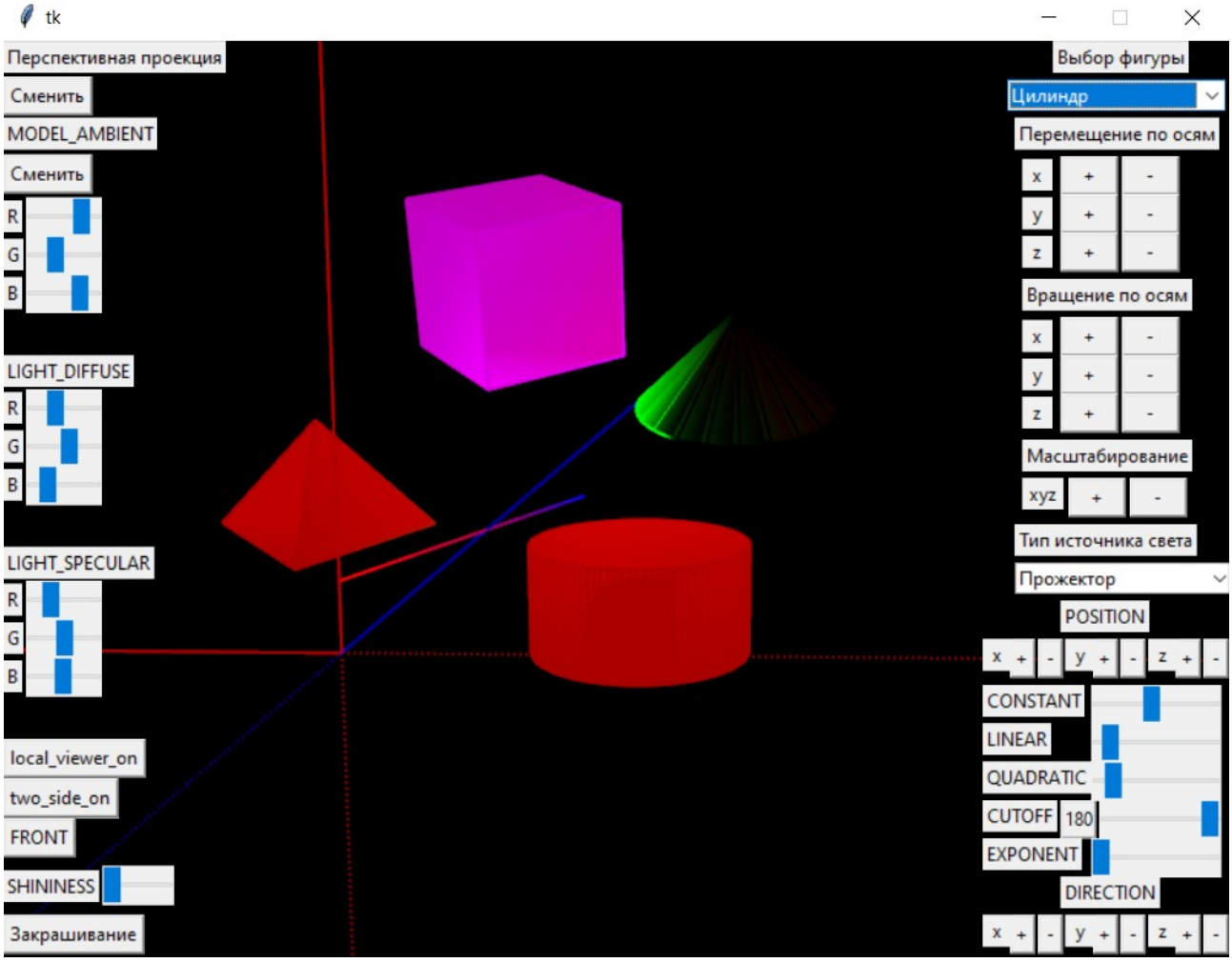

Рисунок 10 — Изменение модели освещения

GL_LIGHT_MODEL_AMBIENT меняется через тот же ползунок, что и GL_AMBIENT (предварительно нужно будет переключиться через второй «Сменить» на MODEL_AMBIENT), GL_LIGHT_MODEL_LOCAL_VIEWER и GL_LIGHT_MODEL_TWO_SIDE меняются через кнопки-переключатели слева-снизу (рис. 10)

4. Материал объектов

Для изменения материалов объектов реализованы следующие параметры:

- GL_AMBIENT – определяет рассеянный цвет материала (RGB);
- GL_DIFFUSE – определяет цвет диффузного отражения материала (RGB);
- GL_SPECULAR – определяет цвет зеркального отражения материала (RGB);
- GL_SHININESS – определяет степень зеркального отражения материала (от 0 до 128):

```
glMaterialfv(self.face[0], GL_AMBIENT, self.material_ambient)  # определяют рассеянный цвет материала
glMaterialfv(self.face[0], GL_DIFFUSE, self.material_diffuse)  # определяют цвет диффузного отражения материала
glMaterialfv(self.face[0], GL_SPECULAR, self.material_specular)  # определяют цвет зеркального отражения
# материала
glMaterialfv(self.face[0], GL_SHININESS, self.shininess[0])  # определяет степень зеркального отражения
# материала
```

Демонстрация:

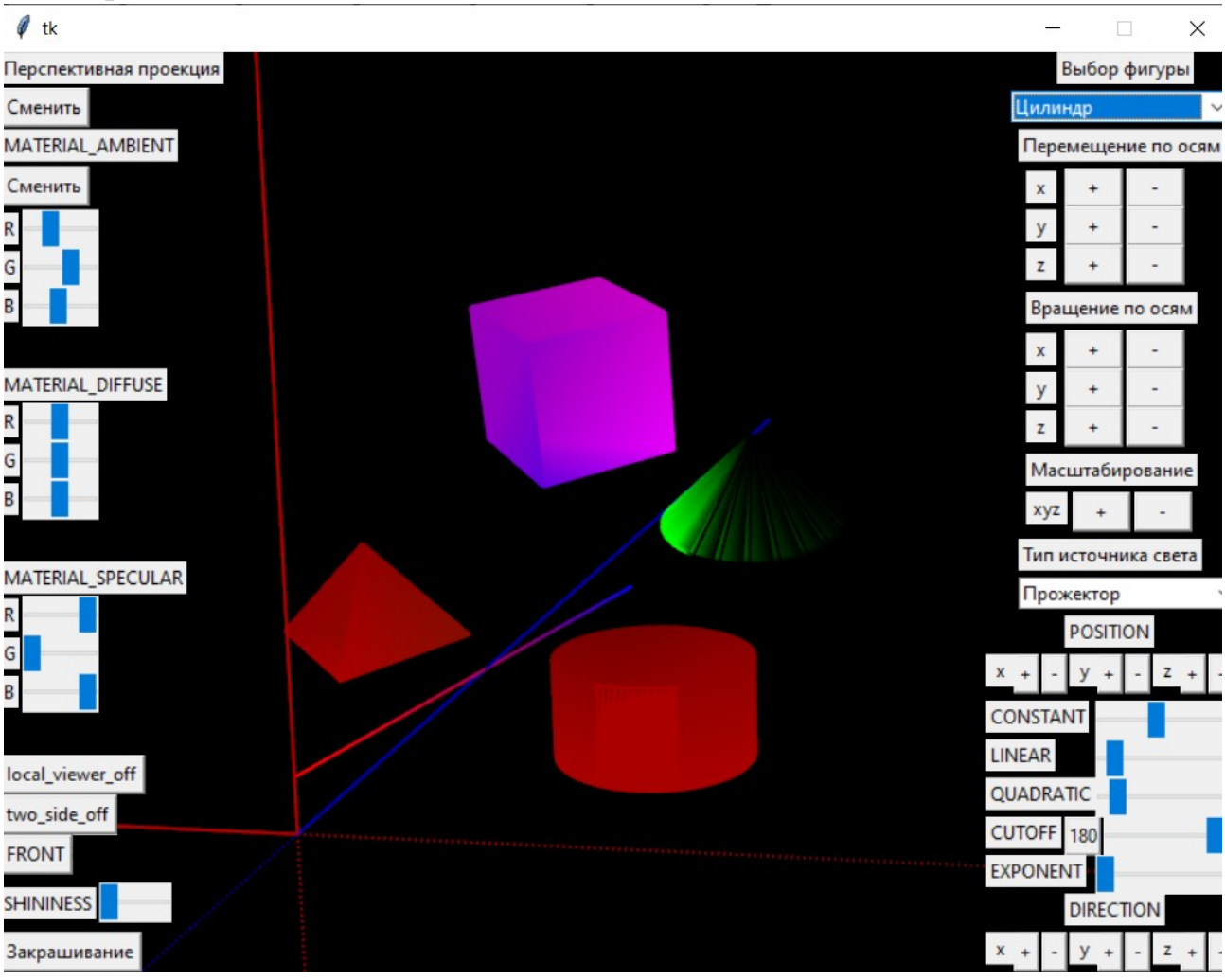

Рисунок 11 - Изменение параметров материала объектов

Для того чтобы получить доступ к изменению параметров материала объектов необходимо нажать на уже описанную кнопку «Сменить» до тех пор, пока не появятся соответствующие надписи (например, MATERIAL_SPECULAR)

Вывод.

Была разработана программа, реализующую представление трехмерной сцены с добавлением возможности формирования различного типа проекций, отражений, используя предложенные функции OpenGL(модель освещения, типы источников света, свойства материалов(текстура)).

Приложение А. Исходный код.

Файл model.axis.py

```
from model.edge import Edge
from OpenGL.GL import glPushMatrix, glMatrixMode, glColor4f, glPopMatrix, GL_MODELVIEW
from model.identity_mat import identity_mat44
from model.vertex import Vertex


class Axis:
    def __init__(self):  # класс под зарисовку осей
        self._identity_mat = identity_mat44()  # единичная матрица 4x4
        self._vertices = [Vertex(0, 0, 0),  # вершины под оси
                          Vertex(-100, 0, 0),
                          Vertex(0, 100, 0),
                          Vertex(0, 0, -100),
                          Vertex(100, 0, 0),
                          Vertex(0, -100, 0),
                          Vertex(0, 0, 100),
                          ]
        self._edges = [(0, 1),  # для отрисовки ребер
                       (0, 2),
                       (0, 3),
                       (0, 4),
                       (0, 5),
                       (0, 6),
                       ]

    def draw(self):

        glMatrixMode(GL_MODELVIEW)  # Эта функция сообщает OpenGL, что следующие операции над матрицами будут
        # применяться к матрице моделирования-вида, которая определяет положение и ориентацию камеры, а также положение
        # и ориентацию всех объектов, отображаемых на экране.
        glPushMatrix()  # Вызов glPushMatrix() позволяет сохранить текущее состояние матрицы в стеке матриц OpenGL,
        # чтобы его можно было восстановить позже с помощью функции glPopMatrix()
        #gluLookAt(-2, 2, -6, 0, 0, 0, 0, 1, 0)  # Таким образом, данная функция устанавливает камеру в точке (-2, 2, -6)
        # с направлением взгляда на точку (0, 0, 0) и считает, что направление "вверх" камеры определяется вектором
        # (0, 1, 0). Это позволяет создать 3D-вид, который можно использовать для рендеринга сцены с заданной камерой.

        # glMultMatrixf(self._identity_mat)
        color = 0
        colors = [(1, 0, 0),  # цвета осей
```

```
            (1, 1, 0),
            (0, 1, 1),
            (1, 0, 0),
            (1, 1, 0),
            (0, 1, 1)
            ]

        for edge in self._edges:
            glColor4f(colors[color][0], colors[color][1], colors[color][2], 1)
            if color > 2:  # отрицательные полуоси красятся точечными линиями
                Edge.draw_dotted_edge(Edge(self._vertices[edge[0]], self._vertices[edge[1]]))
            else:  # положительные полуоси красятся сплошными линиями
                Edge.draw_edge(Edge(self._vertices[edge[0]], self._vertices[edge[1]]))
            color += 1

        glPopMatrix()

    def render(self):
        self.draw()
```

файл model.camera.py:

```
from OpenGL.GL import *

from model.identity_mat import identity_mat44


class Camera:
    def __init__(self):
        self._identity_mat = identity_mat44()  # матрица преобразований (изначально единичная матрица 4x4)
        self.tx = 0  # координаты по перемещению
        self.ty = 0
        self.tz = 0
        self.ry = 0  # координаты по вращению
        self.rx = 0
        self.rz = 0

    def rotate_x(self):  # поворот фигуры по оси x
        glLoadIdentity()  # загрузка единичной матрицы
        glRotatef(self.rx, 1, 0, 0)  # поворот вокруг оси X на угол rx
        glMultMatrixf(self._identity_mat)  # перемножает текущую матрицу с единичной матрицей,
        # чтобы сохранить текущее положение объекта.
        self._identity_mat = glGetFloatv(GL_MODELVIEW_MATRIX)  # чтобы получить новую матрицу моделирования и просмотра,
        # которая включает в себя поворот вокруг оси X, и сохраняет ее в переменную
```

```
self._identity_mat

    def rotate_y(self):  # то же самое, но по оси y
        glLoadIdentity()
        glRotatef(self.ry, 0, 1, 0)
        glMultMatrixf(self._identity_mat)
        self._identity_mat = glGetFloatv(GL_MODELVIEW_MATRIX)

    def rotate_z(self):  # то же самое, но по оси z
        glLoadIdentity()
        glRotatef(self.rz, 0, 0, 1)
        glMultMatrixf(self._identity_mat)
        self._identity_mat = glGetFloatv(GL_MODELVIEW_MATRIX)

    def translate(self):  # перемещение фигуры
        glLoadIdentity()  # загрузка единичной матрицы
        glTranslatef(self.tx, self.ty, self.tz)  # перемещение на tx, ty, tz
        glMultMatrixf(self._identity_mat)  # перемножает текущую матрицу с единичной
матрицей,
        # чтобы сохранить текущее положение объекта.
        self._identity_mat = glGetFloatv(GL_MODELVIEW_MATRIX)  # чтобы получить новую
матрицу моделирования и просмотра,
        # которая включает в себя поворот вокруг оси X, и сохраняет ее в переменную
self._identity_mat

    def render(self):  # рендер измененного объекта
        self.translate()
        self.rotate_y()
        self.rotate_x()
        self.rotate_z()
```

файл model.edge.py:

```
from OpenGL.GL import *


class Edge:  # класс под зарисовку ребер
    def __init__(self, vertex1, vertex2):  # конструктор принимает 2 вершины для ребра
        self._vertex1 = vertex1
        self._vertex2 = vertex2

    def draw_edge(self):  # рисует сплошнуб линию через класс vertex
        glLineWidth(2)
        glBegin(GL_LINES)
        self._vertex1.draw()
        self._vertex2.draw()
        glEnd()

    def draw_dotted_edge(self):  # рисует точечную линию через класс vertex
```

```
        glPushAttrib(GL_ENABLE_BIT)
        glLineStipple(1, 0x1111)
        glEnable(GL_LINE_STIPPLE)
        self.draw_edge()
        glPopAttrib()

    def get_vertexes(self):  # возвращает вершины ребра
        return [self._vertex1, self._vertex2]
```

файл model.figure.py:

```
from OpenGL.GL import *
from OpenGL.GLU import gluLookAt
from model.identity_mat import identity_mat44


class Figure:
    def __init__(self, edges, color, coordinates):  # конструктору нужно передать ребра куба
        self._trans_mat = identity_mat44()  # матрица перемещения
        self._rotation_mat = identity_mat44()  # матрица поворота
        self._scale_mat = identity_mat44()  # матрица масштабирования
        self._edges = edges
        self.tx = 0  # координаты перемещения
        self.ty = 0
        self.tz = 0
        self.rx = 0  # углы поворота
        self.ry = 0
        self.rz = 0
        self.sxyz = 1  # масштабирование
        self.fill = False
        self.color = color
        self.coordinates = coordinates

    def change_fill(self):  # для закрашывания фигуры
        self.fill = not self.fill

    def draw(self):
        glPushMatrix()

        glColor4f(self.color[0], self.color[1], self.color[2], 1)
        temp_vertexes = []

        for edge in self._edges:  # отрисовка ребер
            edge.draw_edge()
            temp_vertexes.append(edge.get_vertexes()[0])
            temp_vertexes.append(edge.get_vertexes()[1])

        if self.fill:  # закрашивание граней
```

```
            glBegin(GL_POLYGON)
            for i in range(len(temp_vertexes)):
                temp_vertexes[i].draw()
            glEnd()

        glPopMatrix()

    def render(self):
        glMatrixMode(GL_MODELVIEW)
        glPushMatrix()
        gluLookAt(self.coordinates[0], self.coordinates[1], self.coordinates[2], self.coordinates[3],
                  self.coordinates[4], self.coordinates[5], self.coordinates[6], self.coordinates[7],
                  self.coordinates[8])  # перемещение куба
        # в начало координат (его центра)
        glMultMatrixf(self._trans_mat)  # перемножение матриц, чтобы сохранить
преобразования
        glMultMatrixf(self._rotation_mat)
        glMultMatrixf(self._scale_mat)

        glPushMatrix()
        self.move_x()  # перемещение
        self.move_y()
        self.move_z()
        glPopMatrix()

        glPushMatrix()
        glLoadIdentity()
        self.rotate_global()  # поворот
        self._rotation_mat = glGetFloatv(GL_MODELVIEW_MATRIX)
        glPopMatrix()

        glPushMatrix()
        glLoadIdentity()
        self.scale()  # масштабирование
        self._scale_mat = glGetFloatv(GL_MODELVIEW_MATRIX)
        glPopMatrix()

        self.draw()
        glPopMatrix()

    def move_x(self):  # перемещение по x
        glLoadMatrixf(self._trans_mat)  # загрузка единичной матрицы
        glTranslatef(self.tx, 0, 0)  # перемещение на tx, 0, 0
        self._trans_mat = glGetFloatv(GL_MODELVIEW_MATRIX)  # чтобы получить новую
матрицу моделирования и просмотра,
        # которая включает в себя поворот вокруг оси X, и сохраняет ее в переменную
self._trans_mat

    def move_y(self):  # то же самое по y
```

```
        glLoadMatrixf(self._trans_mat)
        glTranslatef(0, self.ty, 0)
        self._trans_mat = glGetFloatv(GL_MODELVIEW_MATRIX)

    def move_z(self):  # то же самое по z
        glLoadMatrixf(self._trans_mat)
        glTranslatef(0, 0, self.tz)
        self._trans_mat = glGetFloatv(GL_MODELVIEW_MATRIX)

    def rotate_global(self):  # сначала поворачивает объект вокруг глобальной системы координат, а затем применяет
        # локальную матрицу преобразования. Это означает, что объект вращается вокруг глобальной оси, а затем его
        # новая ориентация преобразуется обратно в локальную систему координат.
        glRotatef(self.rx, 1, 0, 0)
        glRotatef(self.ry, 0, 1, 0)
        glRotatef(self.rz, 0, 0, 1)
        glMultMatrixf(self._rotation_mat)

    def scale(self):
        glScale(self.sxyz, self.sxyz, self.sxyz)
        glMultMatrixf(self._scale_mat)

    def stop(self):  # остановить все преобразования
        self.tx = 0
        self.ty = 0
        self.tz = 0
        self.ry = 0
        self.rx = 0
        self.rz = 0
        self.sxyz = 1
```

файл model.identity_mat.py:

```
import numpy


def identity_mat44():  # возвращает единичную матрицу 4x4
    return numpy.matrix(numpy.i
```

файл model.vertex.py:

```
from OpenGL.GL import glVertex3fv, glVertex2fv


class Vertex:  # класс под зарисовки вершин (объект класса принимает 3 координаты и через метод draw через glVertex
    # красит), метод draw необходимо вызывать в конструкции glBegin - glEnd
    def __init__(self, x, y, z):
```

```
        self._x = x
        self._y = y
        self._z = z

    def draw(self):
        if self._z is None:
            glVertex2fv((self._x, self._y))
        else:
            glVertex3fv((self._x, self._y, self._z))
```

файл model.lighter.py

```
from OpenGL.GL import *
from OpenGL.GLU import *
from model.identity_mat import identity_mat44


class Lighter:
    def __init__(self):
        self._rotation_mat = identity_mat44()  # матрица поворота
        self.position = [0.0, 0.0, 0.0, 0]
        self.tx = 0
        self.ty = 0
        self.tz = 0
        self.ry = 0
        self.rx = 0
        self.rz = 0
        self.light_sample = [0.0]
        self.constant = [0.0]
        self.linear = [0.0]
        self.quadratic = [0.0]
        self.cutoff = [0.0]
        self.exponent = [0.0]
        self.direction = [0.0, 0.0, 0.0, 1]
        self.ambient = [0.0, 0.0, 0.0, 1]
        self.diffuse = [1.0, 1.0, 1.0, 1.0]
        self.specular = [1.0, 1.0, 1.0, 1.0]
        self.model_ambient = [0.2, 0.2, 0.2, 1]
        self.model_local_viewer = [0]
        self.model_two_side = [0]
        self.face = [GL_FRONT_AND_BACK]
        self.shininess = [1.0]
        self.material_ambient = [0.2, 0.2, 0.2, 1]
        self.material_diffuse = [0.8, 0.8, 0.8, 1]
        self.material_specular = [0.0, 0.0, 0.0, 1]

    def draw(self):
        glMatrixMode(GL_MODELVIEW)
        glPushMatrix()
```

```
        x = self.position[0]
        y = self.position[1]
        z = self.position[2]

        glCullFace(self.face[0])
        glMaterialfv(self.face[0], GL_AMBIENT, self.material_ambient)  # определяют рассеянный цвет материала
        glMaterialfv(self.face[0], GL_DIFFUSE, self.material_diffuse)  # определяют цвет диффузного отражения материала
        glMaterialfv(self.face[0], GL_SPECULAR, self.material_specular)  # определяют цвет зеркального отражения
        # материала
        glMaterialfv(self.face[0], GL_SHININESS, self.shininess[0])  # определяет степень зеркального отражения
        # материала

        #glMaterialfv(GL_FRONT, GL_SPECULAR, (1, 1, 1, 1))
        #glMaterialfv(GL_FRONT, GL_SHININESS, 1)
        #glColorMaterial(self.face[0], GL_AMBIENT_AND_DIFFUSE)

        glLightfv(GL_LIGHT0, GL_AMBIENT, self.ambient)  # цвет фонового освещения
        glLightfv(GL_LIGHT0, GL_DIFFUSE, self.diffuse)  # цвет диффузного освещения
        glLightfv(GL_LIGHT0, GL_SPECULAR, self.specular)  # цвет зеркального отражения

        glLightModelfv(GL_LIGHT_MODEL_AMBIENT, self.model_ambient)  # цвет глобального фонового света
        glLightModelfv(GL_LIGHT_MODEL_LOCAL_VIEWER, self.model_local_viewer)  # является ли точка наблюдения локальной
        # или удаленной
        glLightModelfv(GL_LIGHT_MODEL_TWO_SIDE, self.model_two_side)  # правильно ли происходит закрашивание обеих
        # сто-рон полигона

        if self.light_sample[0] == 0:

            glColor4f(0, 0, 255, 1)

            self.position[3] = 0

            glLightfv(GL_LIGHT0, GL_POSITION, self.position)
            glLightfv(GL_LIGHT0, GL_SPOT_CUTOFF, 180)

            glLineWidth(4)
            glBegin(GL_LINES)
            glVertex3f(0, 0, 0)
            glVertex3f(x, y, z)
            glEnd()
```

```
    elif self.light_sample[0] == 1:

        glColor4f(255, 255, 255, 1)

        self.position[3] = 1

        glLightfv(GL_LIGHT0, GL_POSITION, self.position)  # задает положение источника света в пространстве.
        # Чем дальше источник света от поверхности, тем больше затухание света.
        glLight(GL_LIGHT0, GL_CONSTANT_ATTENUATION, self.constant)  # задает постоянное затухание света в
        # зависимости от расстояния между источником света и поверхностью. Чем больше это значение,
        # тем меньше затухание.
        glLightfv(GL_LIGHT0, GL_LINEAR_ATTENUATION, self.linear)  # задает линейное затухание света. Оно
        # увеличивает затухание света линейно в зависимости от расстояния между источником света и поверхностью
        glLightfv(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, self.quadratic)  # задает квадратичное затухание света.
        # Оно увеличивает затухание света квадратично в зависимости от расстояния между источником света
        # и поверхностью.

        glPointSize(50)
        glBegin(GL_POINTS)
        glVertex3f(x, y, z)
        glEnd()

    elif self.light_sample[0] == 2:

        self.position[3] = 1

        direction = self.direction
        position = self.position

        glLightfv(GL_LIGHT0, GL_POSITION, position)
        glLightfv(GL_LIGHT0, GL_CONSTANT_ATTENUATION, self.constant)
        glLightfv(GL_LIGHT0, GL_LINEAR_ATTENUATION, self.linear)
        glLightfv(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, self.quadratic)

        glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, direction)  # направление света прожектора.
        glLightfv(GL_LIGHT0, GL_SPOT_CUTOFF, self.cutoff)  # угловая ширина светового луча.
        glLightfv(GL_LIGHT0, GL_SPOT_EXPONENT, self.exponent)  # концентрация светового луча.

        glPointSize(50)
        glBegin(GL_LINES)
        glColor4f(0, 255, 255, 1)
```

```
            glVertex3f(x, y, z)
            glColor4f(255, 255, 0, 1)
            glVertex3f(self.direction[0], self.direction[1], self.direction[2])
            glEnd()

            gluLookAt(-2, 2, -6, 0, 0, 0, 0, 1, 0)

        glPopMatrix()

    def render(self):
        glMatrixMode(GL_MODELVIEW)
        glPushMatrix()
        glMultMatrixf(self._rotation_mat)

        glPushMatrix()
        glLoadIdentity()
        self._rotation_mat = glGetFloatv(GL_MODELVIEW_MATRIX)
        glPopMatrix()
        self.draw()
        glPopMatrix()
```

файл view.main.py

```
from tkinter import *
from pyopengltk import OpenGLFrame
from OpenGL.GL import *
from OpenGL.GLU import *
from model.axis import Axis
from model.camera import Camera
from model.vertex import Vertex
from model.edge import Edge
from model.figure import Figure
from model.lighter import Lighter
from tkinter.ttk import Combobox
from tkinter.ttk import Scale
import math


def click_fill_figure():
    figures[figure_id].change_fill()


def get_figure():
    temp_figure = combo_select_figure.get()
    global figure_id
    if temp_figure == "Куб":
        figure_id = 0
```

```
    elif temp_figure == "Пирамида":
        figure_id = 1
    elif temp_figure == "Конус":
        figure_id = 2
    elif temp_figure == "Цилиндр":
        figure_id = 3


def get_light():
    temp_light = combo_select_light.get()
    global light_sample
    if temp_light == "Направленный":
        light_sample[0] = 0
    elif temp_light == "Точечный":
        light_sample[0] = 1
    elif temp_light == "Прожектор":
        light_sample[0] = 2


def set_light_default(event):
    if light_sample[0] == 0:
        position[0] = 0
        position[1] = 0
        position[2] = 0
    elif light_sample[0] == 1:
        position[0] = 0
        position[1] = 0
        position[2] = -13.20000000000001
    elif light_sample[0] == 2:
        direction[0] = -1.1102230246251565e-16
        direction[1] = 0.2999999999999999
        direction[2] = 0.0

        position[0] = 1.1102230246251565e-16
        position[1] = -0.6000000000000003
        position[2] = -10.500000000000004


def change_ambient():
    global check_ambient
    if check_ambient < 2:
        check_ambient += 1
    else:
        check_ambient = 0
    if check_ambient == 0:

        label_GL_AMBIENT.config(text="LIGHT_AMBIENT")
        label_GL_DIFFUSE.config(text="LIGHT_DIFFUSE")
        label_GL_SPECULAR.config(text="LIGHT_SPECULAR")
```

```
    elif check_ambient == 1:

        label_GL_AMBIENT.config(text="MODEL_AMBIENT")
    elif check_ambient == 2:
        label_GL_AMBIENT.config(text="MATERIAL_AMBIENT")
        label_GL_DIFFUSE.config(text="MATERIAL_DIFFUSE")
        label_GL_SPECULAR.config(text="MATERIAL_SPECULAR")


def change_local_viewer():
    if local_viewer[0] == 0:
        button_local_viewer.config(text="local_viewer_on")
        local_viewer[0] = 1
    else:
        button_local_viewer.config(text="local_viewer_off")
        local_viewer[0] = 0


def change_two_side():
    if two_side[0] == 0:
        button_two_side.config(text="two_side_on")
        two_side[0] = 1
    else:
        button_two_side.config(text="two_side_off")
        two_side[0] = 0


def change_face():
    global check_face
    if check_face < 2:
        check_face += 1
    else:
        check_face = 0
    if check_face == 0:
        button_face.config(text="FRONT_AND_BACK")
        face[0] = GL_FRONT_AND_BACK
    elif check_face == 1:
        button_face.config(text="FRONT")
        face[0] = GL_FRONT
    else:
        button_face.config(text="BACK")
        face[0] = GL_BACK


def change_cutoff(event):
    global check_cutoff
    check_cutoff = not check_cutoff
    if check_cutoff:
```

```
        button_GL_SPOT_CUTOFF.config(text="180")
    else:
        button_GL_SPOT_CUTOFF.config(text="<-")
    cutoff[0] = 180


def plus_lighter_direction_x(event):
    direction[0] += 0.3


def plus_lighter_x(event):
    position[0] += 0.3


def minus_lighter_direction_x(event):
    direction[0] -= 0.3


def minus_lighter_x(event):
    position[0] -= 0.3


def plus_lighter_direction_y(event):
    direction[1] += 0.3


def plus_lighter_y(event):
    position[1] += 0.3


def minus_lighter_direction_y(event):
    direction[1] -= 0.3


def minus_lighter_y(event):
    position[1] -= 0.3


def plus_lighter_direction_z(event):
    direction[2] += 0.3


def plus_lighter_z(event):
    position[2] += 0.3


def minus_lighter_direction_z(event):
    direction[2] -= 0.3
```

```
def minus_lighter_z(event):
    position[2] -= 0.3


def camera_plus_tx(event):
    camera.tx = 0.1


def camera_minus_tx(event):
    camera.tx = -0.1


def camera_plus_tz(event):
    camera.tz = 0.1


def camera_minus_tz(event):
    camera.tz = -0.1


def camera_plus_ty(event):
    camera.ty = 0.1


def camera_minus_ty(event):
    camera.ty = -0.1


def camera_plus_ry(event):
    camera.ry = 1.0


def camera_minus_ry(event):
    camera.ry = -1.0


def camera_plus_rx(event):
    camera.rx = 1.0


def camera_minus_rx(event):
    camera.rx = -1.0


def camera_plus_rz(event):
    camera.rz = 1.0
```

```
def camera_minus_rz(event):
    camera.rz = -1.0


def plus_axes_x(event):
    figures[figure_id].tx = 0.01


def minus_axes_x(event):
    figures[figure_id].tx = -0.01


def plus_axes_y(event):
    figures[figure_id].ty = 0.01


def minus_axes_y(event):
    figures[figure_id].ty = -0.01


def plus_axes_z(event):
    figures[figure_id].tz = 0.01


def minus_axes_z(event):
    figures[figure_id].tz = -0.01


def plus_axes_rx(event):
    figures[figure_id].rx = 0.5


def minus_axes_rx(event):
    figures[figure_id].rx = -0.5


def plus_axes_ry(event):
    figures[figure_id].ry = 0.5


def minus_axes_ry(event):
    figures[figure_id].ry = -0.5


def plus_axes_rz(event):
    figures[figure_id].rz = 0.5


def minus_axes_rz(event):
```

```
    figures[figure_id].rz = -0.5


def plus_axes_sxyz(event):
    figures[figure_id].sxyz = 1.01


def minus_axes_sxyz(event):
    figures[figure_id].sxyz = 1 / 1.01


def camera_stop(event):
    if camera.tx > 0:
        camera.tx = 0
    elif camera.tx < 0:
        camera.tx = 0
    elif camera.tz > 0:
        camera.tz = 0
    elif camera.tz < 0:
        camera.tz = 0
    elif camera.ty > 0:
        camera.ty = 0.0
    elif camera.ty < 0:
        camera.ty = 0.0
    elif camera.ry > 0:
        camera.ry = 0.0
    elif camera.ry < 0:
        camera.ry = 0.0
    elif camera.rx < 0:
        camera.rx = 0.0
    elif camera.rx > 0:
        camera.rx = 0.0
    elif camera.rz < 0:
        camera.rz = 0.0
    elif camera.rz > 0:
        camera.rz = 0.0


def figure_stop(event):
    figures[figure_id].stop()


def createCircle(shift_x, shift_y, shift_z, R):
    global vertexes_circle
    steps = 100
    angle = math.pi * 2 / steps

    for i in range(steps):
        newX = R * math.sin(angle * i) + shift_x
```

```
        newY = -R * math.cos(angle * i) + shift_y
        vertexes_circle.append([newX, newY, shift_z])

    newX = R * math.sin(angle * 1000) + shift_x
    newY = -R * math.cos(angle * 1000) + shift_y
    vertexes_circle.append([newX, newY, shift_z])


def resize(width, height):
    glViewport(0, 0, width, height)


def change_select_matrix_projection():
    global select_matrix_projection
    select_matrix_projection = not select_matrix_projection
    if select_matrix_projection:
        label_select_matrix.config(text="Перспективная проекция")
    else:
        label_select_matrix.config(text="Параллельная проекция")


class DrawingWindow(OpenGLFrame):  # создание класса на основе пакета pyopengltk

    def initgl(self):  # инициализация
        resize(*SCREEN_SIZE)

        global light_sample
        glEnable(GL_LIGHTING)
        glEnable(GL_COLOR_MATERIAL)
        glEnable(GL_NORMALIZE)
        glEnable(GL_DEPTH_TEST)
        glEnable(GL_CULL_FACE)

        # glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)
        # glEnable(GL_BLEND)

    def redraw(self):  # перерисовка
        glEnable(GL_LIGHT0)
        glMatrixMode(GL_PROJECTION)
        glLoadIdentity()
        if not select_matrix_projection:
            glOrtho(-4, 4, -4, 4, -150, 150)  # left right bottom top near far # ортографическая
            # left - координата левой границы видимой области вдоль оси X;
            # right - координата правой границы видимой области вдоль оси X;
            # bottom - координата нижней границы видимой области вдоль оси Y;
            # top - координата верхней границы видимой области вдоль оси Y;
            # near - координата ближней плоскости отсечения;
            # far - координата дальней плоскости отсечения.
        else:
```

```
    gluPerspective(40.0, float(SCREEN_SIZE[0] / SCREEN_SIZE[1]), 0.1, 50.0)  # перспективная
glMatrixMode(GL_MODELVIEW)
glLoadIdentity()

glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)

glPushMatrix()

get_light()
constant[0] = float(scale_GL_CONSTANT_ATTENUATION.get())
linear[0] = float(scale_GL_LINEAR_ATTENUATION.get())
quadratic[0] = float(scale_GL_QUADRATIC_ATTENUATION.get())
if check_cutoff:
    cutoff[0] = float(scale_GL_SPOT_CUTOFF.get())
exponent[0] = float(scale_GL_SPOT_EXPONENT.get())
shininess[0] = float(scale_SHININESS.get())

if check_ambient == 0:
    ambient[0] = float(scale_GL_AMBIENT_r.get())
    ambient[1] = float(scale_GL_AMBIENT_g.get())
    ambient[2] = float(scale_GL_AMBIENT_b.get())
    #ambient[3] = float(scale_GL_AMBIENT_a.get())

    diffuse[0] = float(scale_GL_DIFFUSE_r.get())
    diffuse[1] = float(scale_GL_DIFFUSE_g.get())
    diffuse[2] = float(scale_GL_DIFFUSE_b.get())
    #diffuse[3] = float(scale_GL_DIFFUSE_a.get())

    specular[0] = float(scale_GL_SPECULAR_r.get())
    specular[1] = float(scale_GL_SPECULAR_g.get())
    specular[2] = float(scale_GL_SPECULAR_b.get())
    #specular[3] = float(scale_GL_SPECULAR_a.get())
if check_ambient == 1:
    model_ambient[0] = float(scale_GL_AMBIENT_r.get())
    model_ambient[1] = float(scale_GL_AMBIENT_g.get())
    model_ambient[2] = float(scale_GL_AMBIENT_b.get())
    #model_ambient[3] = float(scale_GL_AMBIENT_a.get())
else:
    material_ambient[0] = float(scale_GL_AMBIENT_r.get())
    material_ambient[1] = float(scale_GL_AMBIENT_g.get())
    material_ambient[2] = float(scale_GL_AMBIENT_b.get())
    #material_ambient[3] = float(scale_GL_AMBIENT_a.get())

    material_diffuse[0] = float(scale_GL_DIFFUSE_r.get())
    material_diffuse[1] = float(scale_GL_DIFFUSE_g.get())
    material_diffuse[2] = float(scale_GL_DIFFUSE_b.get())
    #material_diffuse[3] = float(scale_GL_DIFFUSE_a.get())

    material_specular[0] = float(scale_GL_SPECULAR_r.get())
```

```
            material_specular[1] = float(scale_GL_SPECULAR_g.get())
            material_specular[2] = float(scale_GL_SPECULAR_b.get())
            #material_specular[3] = float(scale_GL_SPECULAR_a.get())

        camera.render()
        axis.render()
        lighter.render()

        get_figure()
        for i in range(len(figures)):
            figures[i].render()

        glPopMatrix()
        glDisable(GL_LIGHT0)


root = Tk()  # главное окно
root.resizable(False, False)

axis = Axis()
camera = Camera()
lighter = Lighter()

select_matrix_projection = True
position = lighter.position
light_sample = lighter.light_sample
constant = lighter.constant
linear = lighter.linear
quadratic = lighter.quadratic
cutoff = lighter.cutoff
check_cutoff = True
exponent = lighter.exponent
direction = lighter.direction
ambient = lighter.ambient
diffuse = lighter.diffuse
specular = lighter.specular
local_viewer = lighter.model_local_viewer
two_side = lighter.model_two_side
model_ambient = lighter.model_ambient
check_ambient = 0
face = lighter.face
check_face = 0
shininess = lighter.shininess
material_ambient = lighter.material_ambient
material_diffuse = lighter.material_diffuse
material_specular = lighter.material_specular

vertex0_cube = Vertex(1, -1, -1)
vertex1_cube = Vertex(1, 1, -1)
```

```
vertex2_cube = Vertex(-1, 1, -1)
vertex3_cube = Vertex(-1, -1, -1)
vertex4_cube = Vertex(1, -1, 1)
vertex5_cube = Vertex(1, 1, 1)
vertex6_cube = Vertex(-1, -1, 1)
vertex7_cube = Vertex(-1, 1, 1)

edges_cube = (Edge(vertex0_cube, vertex1_cube),
        Edge(vertex0_cube, vertex3_cube),
        Edge(vertex0_cube, vertex4_cube),
        Edge(vertex2_cube, vertex1_cube),
        Edge(vertex2_cube, vertex3_cube),
        Edge(vertex2_cube, vertex7_cube),
        Edge(vertex6_cube, vertex3_cube),
        Edge(vertex6_cube, vertex4_cube),
        Edge(vertex6_cube, vertex7_cube),
        Edge(vertex5_cube, vertex1_cube),
        Edge(vertex5_cube, vertex4_cube),
        Edge(vertex5_cube, vertex7_cube),
        )

vertex1_pyramid = Vertex(-1, 0, 0)
vertex2_pyramid = Vertex(0, 0, 1)
vertex3_pyramid = Vertex(1, 0, 0)
vertex4_pyramid = Vertex(0, 0, -1)
vertex5_pyramid = Vertex(0, 1, 0)

edges_pyramid = (Edge(vertex1_pyramid, vertex5_pyramid),
            Edge(vertex2_pyramid, vertex5_pyramid),
            Edge(vertex3_pyramid, vertex5_pyramid),
            Edge(vertex4_pyramid, vertex5_pyramid),
            Edge(vertex1_pyramid, vertex4_pyramid),
            Edge(vertex3_pyramid, vertex4_pyramid),
            Edge(vertex2_pyramid, vertex3_pyramid),
            Edge(vertex1_pyramid, vertex2_pyramid))

vertexes_circle = []
createCircle(0, 0, 0, 1)
edge_cone = []
vertex_cone = Vertex(0, 1, 0)

for i in range(len(vertexes_circle) - 1):
   temp_vertex1 = Vertex(vertexes_circle[i][0], vertexes_circle[i][2], vertexes_circle[i][1])
   temp_vertex2 = Vertex(vertexes_circle[i + 1][0], vertexes_circle[i + 1][2], vertexes_circle[i + 1][1])
   edge_cone.append(Edge(temp_vertex1, temp_vertex2))
   if i % 5 == 0:
      edge_cone.append(Edge(temp_vertex1, vertex_cone))

edge_cylinder = []
```

```
temp_circle = vertexes_circle.copy()
vertexes_circle.clear()
createCircle(0, 0, 1, 1)

for i in range(len(temp_circle) - 1):
    temp_vertex1 = Vertex(temp_circle[i][0], temp_circle[i][2], temp_circle[i][1])
    temp_vertex2 = Vertex(temp_circle[i + 1][0], temp_circle[i + 1][2], temp_circle[i + 1][1])

    temp1_vertex1 = Vertex(vertexes_circle[i][0], vertexes_circle[i][2], vertexes_circle[i][1])
    temp1_vertex2 = Vertex(vertexes_circle[i + 1][0], vertexes_circle[i + 1][2], vertexes_circle[i + 1]
[1])

    edge_cylinder.append(Edge(temp_vertex1, temp_vertex2))
    edge_cylinder.append(Edge(temp_vertex1, temp1_vertex1))
    edge_cylinder.append(Edge(temp1_vertex1, temp1_vertex2))

cylinder = Figure(edge_cylinder, [1, 0, 0], [-2, 2, -6, 1, 1.5, 0, 0, 1, 0])
cone = Figure(edge_cone, [0, 100, 0], [-5, 2, -6, 3, -1.5, 0, 0, 1, 0])
cube = Figure(edges_cube, [1, 0, 1], [-10, 2, -6, -1.6499988, -1.77999955, 0, 0, 1, 0])
pyramid = Figure(edges_pyramid, [1, 25, 0], [-2, 2, -6, -1.9699985, 0.36999992, 0, 0, 1, 0])

figures = [cube, pyramid, cone, cylinder]
figure_id = 0

window_width = 800  # размеры окна (ширина и высота)
window_height = 600
SCREEN_SIZE = (window_width, window_height)

app = DrawingWindow(root, width=window_width, height=window_height)  # создание окна для
отрисови
app.pack(fill=BOTH, expand=YES)  # отобразить

root.bind('a', camera_plus_tx)
root.bind('d', camera_minus_tx)
root.bind('w', camera_plus_tz)
root.bind('s', camera_minus_tz)
root.bind('q', camera_plus_ty)
root.bind('e', camera_minus_ty)
root.bind('<Right>', camera_plus_ry)
root.bind('<Left>', camera_minus_ry)
root.bind('<Up>', camera_minus_rx)
root.bind('<Down>', camera_plus_rx)
root.bind('x', camera_plus_rz)
root.bind('z', camera_minus_rz)
root.bind('<KeyRelease>', camera_stop)
root.bind('r', set_light_default)

label_select_figure = Label(text="Выбор фигуры")
label_select_figure.place(x=685, y=0)
```

```
combo_select_figure = Combobox(app)
combo_select_figure['values'] = ("Куб", "Пирамида", "Конус", "Цилиндр")
combo_select_figure['state'] = 'readonly'
combo_select_figure.current(0)
combo_select_figure.place(x=655, y=25)

label_select_axis = Label(text="Перемещение по осям")
label_select_axis.place(x=660, y=50)

label_select_x = Label(text="x", width=2, height=1)
label_select_x.place(x=665, y=77)
button_plus_axes_x = Button(app, text="+", width=4)
button_plus_axes_x.place(x=690, y=75)
button_minus_axes_x = Button(app, text="-", width=4)
button_minus_axes_x.place(x=730, y=75)
button_plus_axes_x.bind('<Button-1>', minus_axes_x)
button_plus_axes_x.bind('<ButtonRelease-1>', figure_stop)
button_minus_axes_x.bind('<Button-1>', plus_axes_x)
button_minus_axes_x.bind('<ButtonRelease-1>', figure_stop)

label_select_y = Label(text="y", width=2, height=1)
label_select_y.place(x=665, y=102)
button_plus_axes_y = Button(app, text="+", width=4)
button_plus_axes_y.place(x=690, y=100)
button_minus_axes_y = Button(app, text="-", width=4)
button_minus_axes_y.place(x=730, y=100)
button_plus_axes_y.bind('<Button-1>', plus_axes_y)
button_plus_axes_y.bind('<ButtonRelease-1>', figure_stop)
button_minus_axes_y.bind('<Button-1>', minus_axes_y)
button_minus_axes_y.bind('<ButtonRelease-1>', figure_stop)

label_select_z = Label(text="z", width=2, height=1)
label_select_z.place(x=665, y=127)
button_plus_axes_z = Button(app, text="+", width=4)
button_plus_axes_z.place(x=690, y=125)
button_minus_axes_z = Button(app, text="-", width=4)
button_minus_axes_z.place(x=730, y=125)
button_plus_axes_z.bind('<Button-1>', minus_axes_z)
button_plus_axes_z.bind('<ButtonRelease-1>', figure_stop)
button_minus_axes_z.bind('<Button-1>', plus_axes_z)
button_minus_axes_z.bind('<ButtonRelease-1>', figure_stop)

label_select_axis = Label(text="Вращение по осям")
label_select_axis.place(x=665, y=155)

label_select_rx = Label(text="x", width=2, height=1)
label_select_rx.place(x=665, y=182)
button_plus_axes_rx = Button(app, text="+", width=4)
button_plus_axes_rx.place(x=690, y=180)
```

```
button_minus_axes_rx = Button(app, text="-", width=4)
button_minus_axes_rx.place(x=730, y=180)
button_plus_axes_rx.bind('<Button-1>', plus_axes_rx)
button_plus_axes_rx.bind('<ButtonRelease-1>', figure_stop)
button_minus_axes_rx.bind('<Button-1>', minus_axes_rx)
button_minus_axes_rx.bind('<ButtonRelease-1>', figure_stop)

label_select_ry = Label(text="y", width=2, height=1)
label_select_ry.place(x=665, y=207)
button_plus_axes_ry = Button(app, text="+", width=4)
button_plus_axes_ry.place(x=690, y=205)
button_minus_axes_ry = Button(app, text="-", width=4)
button_minus_axes_ry.place(x=730, y=205)
button_plus_axes_ry.bind('<Button-1>', plus_axes_ry)
button_plus_axes_ry.bind('<ButtonRelease-1>', figure_stop)
button_minus_axes_ry.bind('<Button-1>', minus_axes_ry)
button_minus_axes_ry.bind('<ButtonRelease-1>', figure_stop)

label_select_rz = Label(text="z", width=2, height=1)
label_select_rz.place(x=665, y=232)
button_plus_axes_rz = Button(app, text="+", width=4)
button_plus_axes_rz.place(x=690, y=230)
button_minus_axes_rz = Button(app, text="-", width=4)
button_minus_axes_rz.place(x=730, y=230)
button_plus_axes_rz.bind('<Button-1>', plus_axes_rz)
button_plus_axes_rz.bind('<ButtonRelease-1>', figure_stop)
button_minus_axes_rz.bind('<Button-1>', minus_axes_rz)
button_minus_axes_rz.bind('<ButtonRelease-1>', figure_stop)

label_select_axis = Label(text="Масштабирование")
label_select_axis.place(x=665, y=260)
label_select_sxyz = Label(text="xyz", width=3, height=1)
label_select_sxyz.place(x=665, y=285)
button_plus_axes_sxyz = Button(app, text="+", width=4)
button_plus_axes_sxyz.place(x=695, y=285)
button_minus_axes_sxyz = Button(app, text="-", width=4)
button_minus_axes_sxyz.place(x=735, y=285)
button_plus_axes_sxyz.bind('<Button-1>', plus_axes_sxyz)
button_plus_axes_sxyz.bind('<ButtonRelease-1>', figure_stop)
button_minus_axes_sxyz.bind('<Button-1>', minus_axes_sxyz)
button_minus_axes_sxyz.bind('<ButtonRelease-1>', figure_stop)

label_select_matrix = Label(text="Перспективная проекция")
label_select_matrix.place(x=0, y=0)
button_select_matrix_projection = Button(app, text="Сменить",
command=change_select_matrix_projection)
button_select_matrix_projection.place(x=0, y=23)

label_select_light = Label(text="Тип источника света")
```

```
label_select_light.place(x=660, y=315)
combo_select_light = Combobox(app)
combo_select_light['values'] = ("Направленный", "Точечный", "Прожектор")
combo_select_light['state'] = 'readonly'
combo_select_light.current(0)
combo_select_light.place(x=660, y=340)

label_GL_POSITION = Label(text="POSITION")
label_GL_POSITION.place(x=690, y=365)

label_select_lighter_x = Label(text="x", width=2, height=1)
label_select_lighter_x.place(x=639, y=390)
button_plus_lighter_x = Button(app, text="+", width=1)
button_plus_lighter_x.place(x=657, y=390)
button_minus_lighter_x = Button(app, text="-", width=1)
button_minus_lighter_x.place(x=675, y=390)
button_plus_lighter_x.bind('<Button-1>', minus_lighter_x)
button_minus_lighter_x.bind('<Button-1>', plus_lighter_x)

label_select_lighter_y = Label(text="y", width=2, height=1)
label_select_lighter_y.place(x=693, y=390)
button_plus_lighter_y = Button(app, text="+", width=1)
button_plus_lighter_y.place(x=711, y=390)
button_minus_lighter_y = Button(app, text="-", width=1)
button_minus_lighter_y.place(x=729, y=390)
button_plus_lighter_y.bind('<Button-1>', plus_lighter_y)
button_minus_lighter_y.bind('<Button-1>', minus_lighter_y)

label_select_lighter_y = Label(text="z", width=2, height=1)
label_select_lighter_y.place(x=747, y=390)
button_plus_lighter_y = Button(app, text="+", width=1)
button_plus_lighter_y.place(x=765, y=390)
button_minus_lighter_y = Button(app, text="-", width=1)
button_minus_lighter_y.place(x=783, y=390)
button_plus_lighter_y.bind('<Button-1>', minus_lighter_z)
button_minus_lighter_y.bind('<Button-1>', plus_lighter_z)

label_GL_CONSTANT_ATTENUATION = Label(text="CONSTANT")
label_GL_CONSTANT_ATTENUATION.place(x=639, y=420)
scale_GL_CONSTANT_ATTENUATION = Scale(from_=0, to=1, orient=HORIZONTAL, value=0,
length=85)
scale_GL_CONSTANT_ATTENUATION.place(x=710, y=420)

label_GL_LINEAR_ATTENUATION = Label(text="LINEAR")
label_GL_LINEAR_ATTENUATION.place(x=639, y=445)
scale_GL_LINEAR_ATTENUATION = Scale(from_=0, to=1, orient=HORIZONTAL, value=0, length=85)
scale_GL_LINEAR_ATTENUATION.place(x=710, y=445)

label_GL_QUADRATIC_ATTENUATION = Label(text="QUADRATIC")
```

```
label_GL_QUADRATIC_ATTENUATION.place(x=639, y=470)
scale_GL_QUADRATIC_ATTENUATION = Scale(from_=0, to=1, orient=HORIZONTAL, value=0,
length=85)
scale_GL_QUADRATIC_ATTENUATION.place(x=710, y=470)

label_GL_SPOT_CUTOFF = Label(text="CUTOFF")
label_GL_SPOT_CUTOFF.place(x=639, y=495)
button_GL_SPOT_CUTOFF = Button(app, text="180", width=2, height=1)
button_GL_SPOT_CUTOFF.bind('<Button-1>', change_cutoff)
button_GL_SPOT_CUTOFF.place(x=690, y=495)
scale_GL_SPOT_CUTOFF = Scale(from_=0, to=90, orient=HORIZONTAL, value=0, length=80)
scale_GL_SPOT_CUTOFF.place(x=715, y=495)

label_GL_SPOT_EXPONENT = Label(text="EXPONENT")
label_GL_SPOT_EXPONENT.place(x=639, y=520)
scale_GL_SPOT_EXPONENT = Scale(from_=0, to=128, orient=HORIZONTAL, value=0, length=85)
scale_GL_SPOT_EXPONENT.place(x=710, y=520)

label_GL_SPOT_DIRECTION = Label(text="DIRECTION")
label_GL_SPOT_DIRECTION.place(x=690, y=545)

label_select_lighter_direction_x = Label(text="x", width=2, height=1)
label_select_lighter_direction_x.place(x=639, y=570)
button_plus_lighter_direction_x = Button(app, text="+", width=1)
button_plus_lighter_direction_x.place(x=657, y=570)
button_minus_lighter_direction_x = Button(app, text="-", width=1)
button_minus_lighter_direction_x.place(x=675, y=570)
button_plus_lighter_direction_x.bind('<Button-1>', minus_lighter_direction_x)
button_minus_lighter_direction_x.bind('<Button-1>', plus_lighter_direction_x)

label_select_lighter_direction_y = Label(text="y", width=2, height=1)
label_select_lighter_direction_y.place(x=693, y=570)
button_plus_lighter_direction_y = Button(app, text="+", width=1)
button_plus_lighter_direction_y.place(x=711, y=570)
button_minus_lighter_direction_y = Button(app, text="-", width=1)
button_minus_lighter_direction_y.place(x=729, y=570)
button_plus_lighter_direction_y.bind('<Button-1>', plus_lighter_direction_y)
button_minus_lighter_direction_y.bind('<Button-1>', minus_lighter_direction_y)

label_select_lighter_direction_y = Label(text="z", width=2, height=1)
label_select_lighter_direction_y.place(x=747, y=570)
button_plus_lighter_direction_y = Button(app, text="+", width=1)
button_plus_lighter_direction_y.place(x=765, y=570)
button_minus_lighter_direction_y = Button(app, text="-", width=1)
button_minus_lighter_direction_y.place(x=783, y=570)
button_plus_lighter_direction_y.bind('<Button-1>', minus_lighter_direction_z)
button_minus_lighter_direction_y.bind('<Button-1>', plus_lighter_direction_z)

label_GL_AMBIENT = Label(text="LIGHT_AMBIENT")
```

```
label_GL_AMBIENT.place(x=0, y=50)

label_GL_AMBIENT_r = Label(text="R")
label_GL_AMBIENT_r.place(x=0, y=104)
scale_GL_AMBIENT_r = Scale(from_=-1, to=1, orient=HORIZONTAL, value=0, length=50)
scale_GL_AMBIENT_r.place(x=15, y=102)

label_GL_AMBIENT_g = Label(text="G")
label_GL_AMBIENT_g.place(x=0, y=129)
scale_GL_AMBIENT_g = Scale(from_=-1, to=1, orient=HORIZONTAL, value=0, length=50)
scale_GL_AMBIENT_g.place(x=15, y=127)

label_GL_AMBIENT_b = Label(text="B")
label_GL_AMBIENT_b.place(x=0, y=154)
scale_GL_AMBIENT_b = Scale(from_=-1, to=1, orient=HORIZONTAL, value=0, length=50)
scale_GL_AMBIENT_b.place(x=15, y=152)

#label_GL_AMBIENT_a = Label(text="A")
#label_GL_AMBIENT_a.place(x=0, y=179)
#scale_GL_AMBIENT_a = Scale(from_=-1, to=1, orient=HORIZONTAL, value=1, length=50)
#scale_GL_AMBIENT_a.place(x=15, y=177)

label_GL_DIFFUSE = Label(text="LIGHT_DIFFUSE")
label_GL_DIFFUSE.place(x=0, y=205)

label_GL_DIFFUSE_r = Label(text="R")
label_GL_DIFFUSE_r.place(x=0, y=229)
scale_GL_DIFFUSE_r = Scale(from_=-1, to=1, orient=HORIZONTAL, value=1, length=50)
scale_GL_DIFFUSE_r.place(x=15, y=227)

label_GL_DIFFUSE_g = Label(text="G")
label_GL_DIFFUSE_g.place(x=0, y=254)
scale_GL_DIFFUSE_g = Scale(from_=-1, to=1, orient=HORIZONTAL, value=1, length=50)
scale_GL_DIFFUSE_g.place(x=15, y=252)

label_GL_DIFFUSE_b = Label(text="B")
label_GL_DIFFUSE_b.place(x=0, y=279)
scale_GL_DIFFUSE_b = Scale(from_=-1, to=1, orient=HORIZONTAL, value=1, length=50)
scale_GL_DIFFUSE_b.place(x=15, y=277)

#label_GL_DIFFUSE_a = Label(text="A")
#label_GL_DIFFUSE_a.place(x=0, y=304)
#scale_GL_DIFFUSE_a = Scale(from_=-1, to=1, orient=HORIZONTAL, value=1, length=50)
#scale_GL_DIFFUSE_a.place(x=15, y=302)

label_GL_SPECULAR = Label(text="LIGHT_SPECULAR")
label_GL_SPECULAR.place(x=0, y=330)

label_GL_SPECULAR_r = Label(text="R")
```

```
label_GL_SPECULAR_r.place(x=0, y=354)
scale_GL_SPECULAR_r = Scale(from_=-1, to=1, orient=HORIZONTAL, value=1, length=50)
scale_GL_SPECULAR_r.place(x=15, y=352)

label_GL_SPECULAR_g = Label(text="G")
label_GL_SPECULAR_g.place(x=0, y=379)
scale_GL_SPECULAR_g = Scale(from_=-1, to=1, orient=HORIZONTAL, value=1, length=50)
scale_GL_SPECULAR_g.place(x=15, y=377)

label_GL_SPECULAR_b = Label(text="B")
label_GL_SPECULAR_b.place(x=0, y=404)
scale_GL_SPECULAR_b = Scale(from_=-1, to=1, orient=HORIZONTAL, value=1, length=50)
scale_GL_SPECULAR_b.place(x=15, y=402)

#label_GL_SPECULAR_a = Label(text="A")
#label_GL_SPECULAR_a.place(x=0, y=429)
#scale_GL_SPECULAR_a = Scale(from_=-1, to=1, orient=HORIZONTAL, value=1, length=50)
#scale_GL_SPECULAR_a.place(x=15, y=427)

button_switch_model_light = Button(text="Сменить", command=change_ambient)
button_switch_model_light.place(x=0, y=74)

button_local_viewer = Button(text="local_viewer_off", command=change_local_viewer)
button_local_viewer.place(x=0, y=455)

button_two_side = Button(text="two_side_off", command=change_two_side)
button_two_side.place(x=0, y=481)

button_face = Button(text="FRONT_AND_BACK", command=change_face)
button_face.place(x=0, y=507)

label_SHININESS = Label(text="SHININESS")
label_SHININESS.place(x=0, y=541)

scale_SHININESS = Scale(from_=0, to=128, orient=HORIZONTAL, value=1, length=47)
scale_SHININESS.place(x=65, y=538)

fill_figure = Button(app, text="Закрашивание", command=click_fill_figure)
fill_figure.place(x=0, y=570)

app.animate = 1
app.mainloop()
```