**Лабораторная работа №3.**

**«Модели освещения»**

1. Задание

Создать программу, рисующую освещенный объект без применения специализированных средств. Для рисования допускается использовать только функцию закраски полигона, заданного в координатах окна. Цвет каждой грани рассчитывать самостоятельно, по известным моделям освещения. Дать возможность вращать объект в 3 степенях свободы. Ориентацию объекта хранить в матрице.

Объект: конус.

1. Теория

Освещение подразделяется на изотропное и анизотропное.

Изотропная модель характеризуется тем, что при повороте поверхности вокруг вектора нормали, освещение в точке не изменяется. Однако есть целый ряд материалов (например поверхность компакт-диска) для который это условие не выполняется.

Анизотропное освещение -- освещение, используемое для поверхностей, у которых свойства этой поверхности в точках не зависят от ее ориентации. При этом, для того чтобы можно было учитывать поворот грани вокруг вектора нормали, на поверхности нужно ввести поле касательных векторов, т.е. каждой точке поверхности сопоставляется касательный вектор, перпендикулярный нормали. По касательному вектору и нормали можно найти еще один вектор, перпендикулярный нормали, так называемую бинормаль.

Взятые вместе, нормаль, касательная и бинормаль образуют так называемый касательный базис.

Простейшая анизотропная модель освещения основана на довольно простой модели - вся поверхность считается состоящей из бесконечно тонких нитей. Тогда в качестве касательного вектора в точке выступает касательная к нити, проходящей через данную точку.

К анизотропным поверхностям обычно относят: компакт-диск, металлические поверхности, материалы одежды и др.

1. Алгоритм

Для освещения конуса использовалась изотропная модель Варда.

Будем использовать следующие обозначения:

http://chart.apis.google.com/chart?cht=tx&chl=n - единичный вектор нормали в точке.

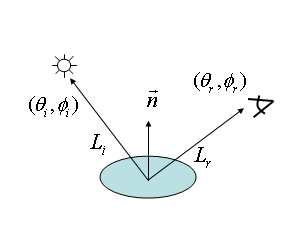
http://chart.apis.google.com/chart?cht=tx&chl=l - единичный вектор направления на источник света.

http://chart.apis.google.com/chart?cht=tx&chl=v - единичный вектор направления на камеру (наблюдателя).

http://chart.apis.google.com/chart?cht=tx&chl=h%20=%20\frac%7bl%20%2b%20v%7d%7b|l%20%2b%20v|%7d - биссектор векторов на источник света и камеру.

http://chart.apis.google.com/chart?cht=tx&chl=\cos\theta_l=(n,l) - косинус угла между нормалью к поверхности в точке и направлением на источник света.

http://chart.apis.google.com/chart?cht=tx&chl=\cos\theta_v=(n,v) - косинус угла между нормалью к поверхности в точке и направлением на камеру.



Бликовое освещение определяется по следующей формуле:

http://chart.apis.google.com/chart?cht=tx&chl=I=\exp%5e%7b-k%20\frac%7b1-(h,n)%5e2%7d%7b(h,n)%5e2%7d%7d

В этой формуле параметр k отвечает за неровность поверхности.

Приведем реализацию на PyOpenGl:

def SetLightWard(n,l,v,k=10.0,

diffColor=array([0.0, 0.5, 0.0, 1.0]),

specColor=array([0.7, 0.7, 0.0, 1.0])):

""" Возвращает цвет с освещением по методу Варда """

n2 = normalize(n) #normalize

l2 = normalize(l)

h = l2 + v

h2 = normalize(h)

hn = dot(h2,n2)

hn2 = hn\*\*2

diff = diffColor \* max(dot(n2, l2),0.0)

spec = specColor \* exp(-k\*(1.0 - hn2)/hn2)

resultColor = diff+spec

return resultColor[0], resultColor[1], resultColor[2], resultColor[3]

Пераметры n,l,v -- высчитываются для каждой области полигона. Нормали высчитываются прямым произведением векторов каждой отдельно взятой области. Малая область составляется из четырех соседних вершин.

Позиция света равна (100,100,100), позиция наблюдателя располагается в точке (0,0,3). С помощью усредненного значения малой области считаем нужные нам вектора l и v и передаем их в функцию SetLightWard. Функция возвращает координаты цвета. Зная отдельные координаты конуса и цвет мелких полигонов, закрашиваем полигон с помощью GL\_QUADS. Для увеличения производительности, закрашиваем только видимую часть конуса.

Соответствующая реализация:

def DrawFigure(figure):

figure = Rotate(figure)

lenFigure = len(figure)

for i in xrange(0,len(figure),4):

v1, v2, v3, v4 = figure[i], figure[i+1], figure[i+2], figure[i+3]

#~ norm = cross(v1-(v2+v3)\*0.5, v1-(v3+v4)\*0.5)

norm = cross(v1-v2, v1-v4)

avg = (v1+v2+v3+v4)/4.0

c1,c2,c3,c4 = SetNoLight()

#~ рисуем только видимые грани

if norm[2]>0.0:

vectLight = initLight - avg

vectEye = initEye - avg

c1,c2,c3,c4 = SetLightWard(norm, vectLight, vectEye)

glColor4f(c1,c2,c3,c4)

glBegin(GL\_QUADS)

glVertex3f(v1[0], v1[1], v1[2]); glVertex3f(v2[0], v2[1], v2[2]); glVertex3f(v3[0], v3[1], v3[2]); glVertex3f(v4[0], v4[1], v4[2]); glEnd()

1. Выводы

В этой лаюбораторной работе мы научились делать освещение самостоятельно. Увидели множество новых моделей освещения и убедились в том, что одними стандартными моделями освещения OpenGl не обойтись. В OpenGl из всего множества моделей, реализованы только метод диффузного отражения Ламберта и метод тонирования Гуро.

Стандарт OpenGl накладывает ограничение на количество источников света (8), в то время как мы сами можем использовать намного больше.

Однако то освещение, которое мы научились делать, использует в основном CPU, в то время как на GPU почти не производится вычислений. Было бы эффективнее ускорить наложение освещения, используя GPU.

Используемая модель освещения зависит от потребностей, материалов и других свойств освещаемого объекта.

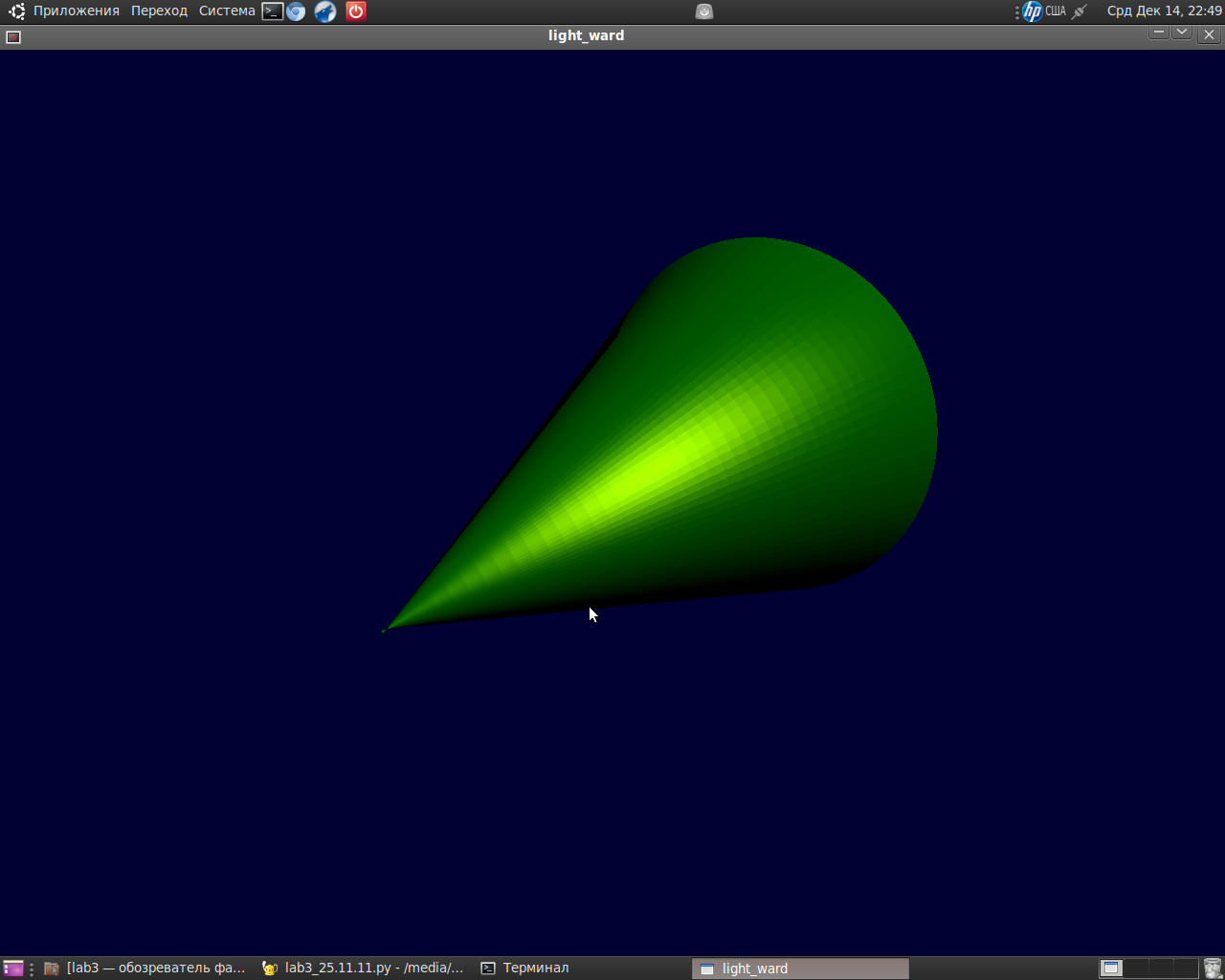
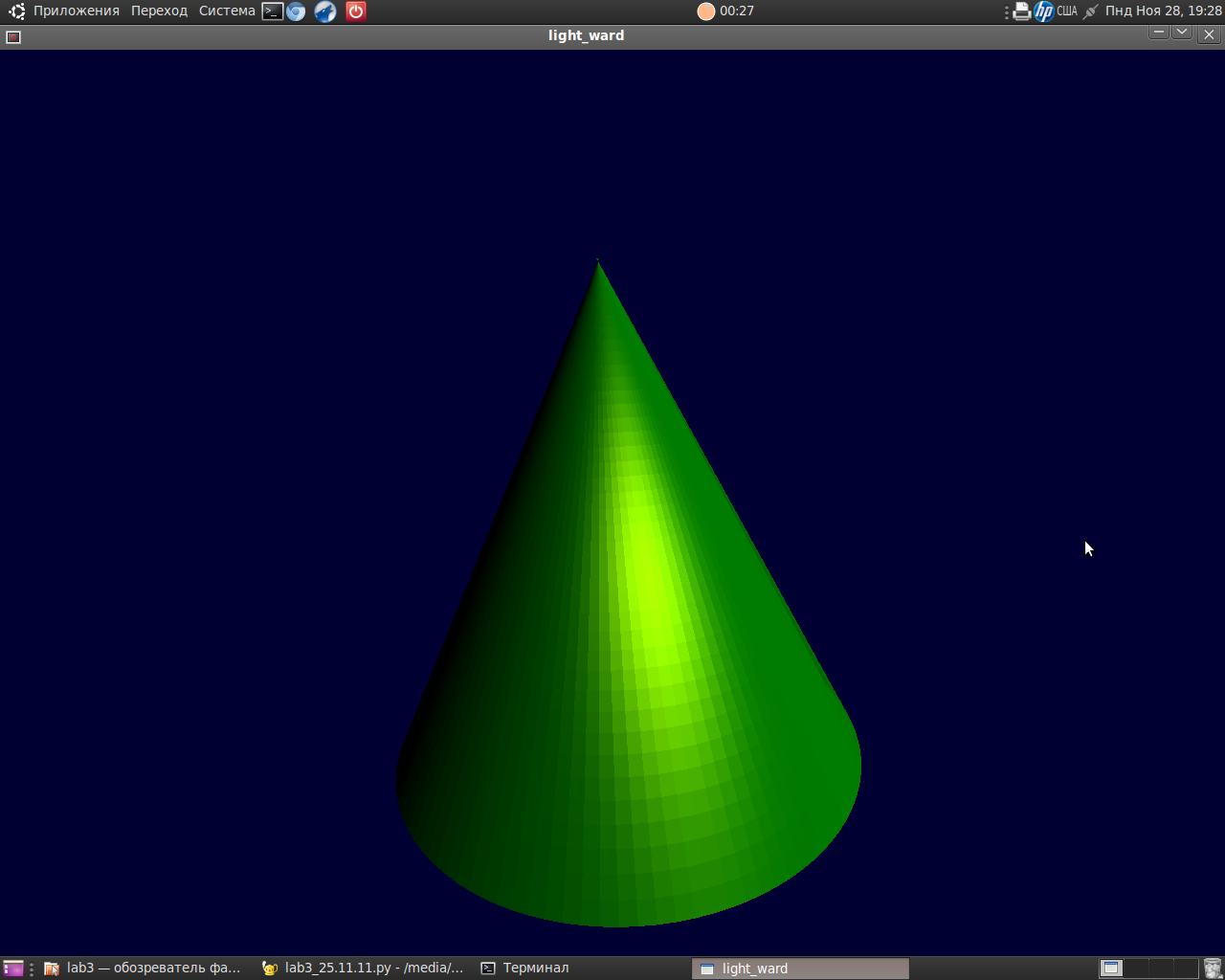
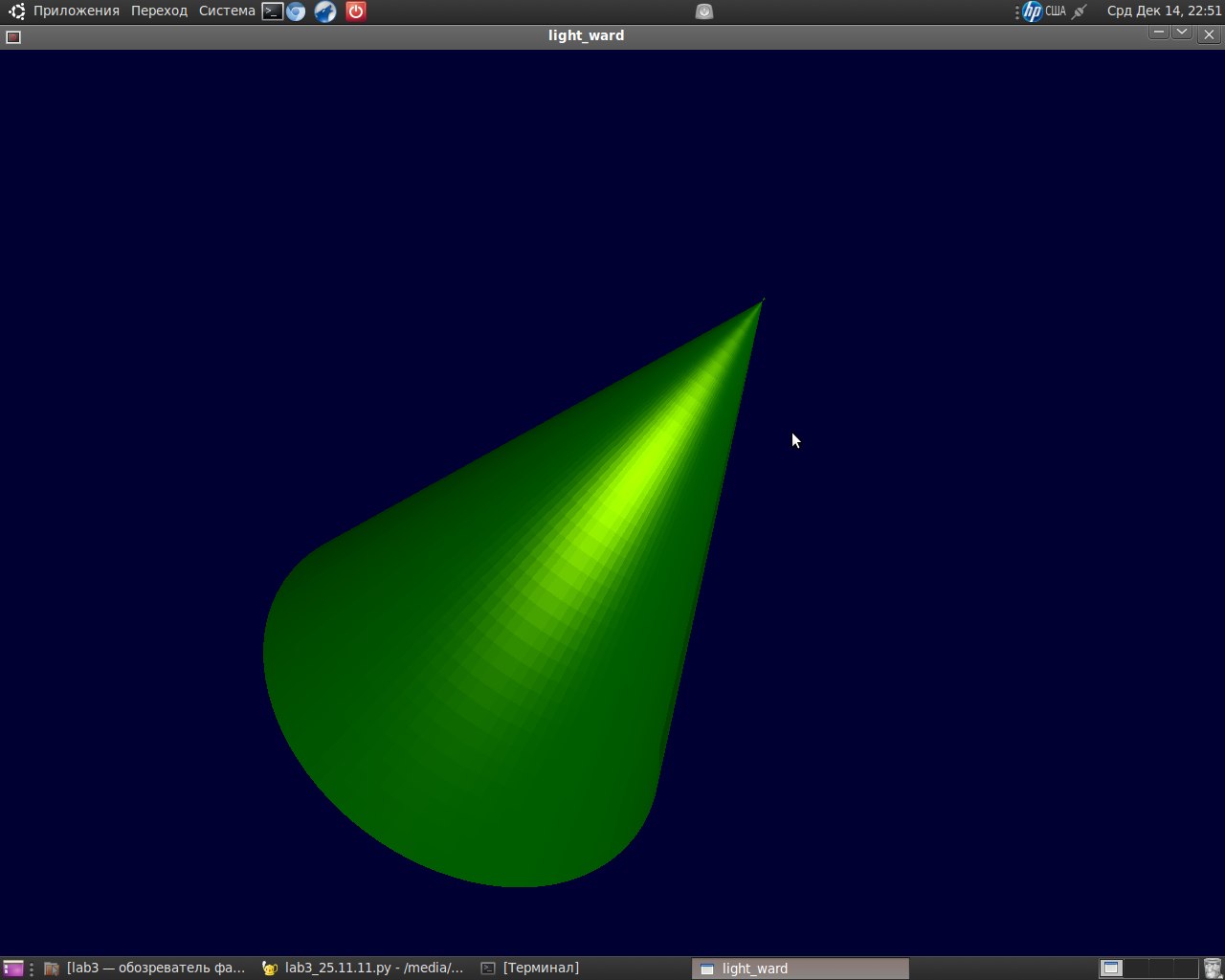
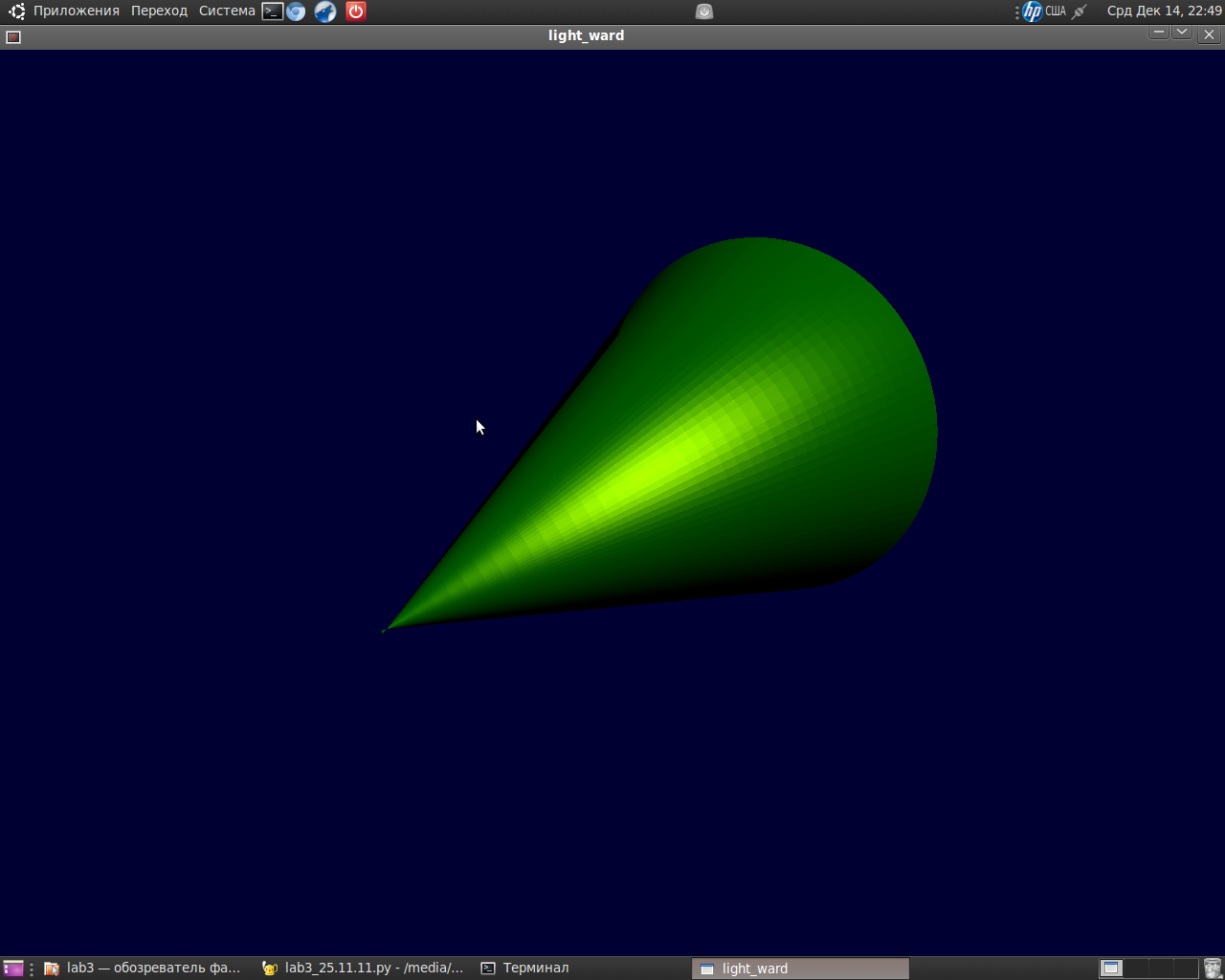
Помимо изотропной и анизотропной, выделяют диффузные, бликовые, текстурованные модели освещения.

К искусственным моделям освещения относятся такие интересные:

* toon shading -- получение изображения, выглядящее как нарисованное человеком.
* subsurface scattering -- эффект подповерхностного рассеивания.
* bidirectional lighting -- освещение с двух противоположных сторон источниками разных цветов.
* и еще множество других методов.

1. Литература
2. А.В. Боресков "Графика трехмерной компьютерной игры на основе OpenGL"
3. А.В. Боресков "Разработка и отладка шейдеров"
4. <http://steps3d.narod.ru/tutorials/lighting-tutorial.html>

**Screen:**

****