**КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ**

**Институт вычислительной математики и информационных технологий**

**Отчет по дисциплине:**

**“Основы компьютерной графики”**

Выполнил студент: Рахимулин Ринат Р. , гр.09-208

Преподаватель: Нигматуллин Руслан Рафикович

2015г.

**Темы для изучения в течение семестра:**

1. Blender. Написание скриптов в Blender. Основы языка Python.

2. Работа с OpenGL внутри системы Blender. Модуль Draw. Ввод данных. Рисование геометрических примитивов.

3. Модуль Blender.Mathutils. Создание анимации.

4. Аффинные преобразования. Проективные координаты.

5. Сплайны. Интерполяционные многочлены. Кубические сплайны. Кривые Безье 1-го, 2-го, 3-го порядка. Свойства кривых Безье.

6. Система координат в трехмерной графике. Формулы рендеринга и пирамида отсечения. Аффинные преобразования сцены.

7. Установка камеры в желаемую позицию. Анимация камеры. Кватернионы.

8. Полиэдры. Аппроксимация  сферы. Поверхности Безье. Использование функции glMap2f.

9. Освещение. Создание тени.

10. Эффект прозрачности. Работа с материалом.

**2. 1) Работа с OpenGL внутри системы Blender**

**Система Blender** — это  свободный, профессиональный пакет для создания трёхмерной компьютерной графики, включающий в себя средства моделирования, анимации, постобработки и монтажа видео со звуком, компоновки с помощью "узлов", а также для создания интерактивных игр.

Система Blender позволяет вызывать функции из OpenGL и просматривать результат работы на экране. Возможности Blender перекрывают основные функции стандартной системы glut: обработку прерываний от мыши и клавиатуры и перехватывать нажатия рисованных кнопок. Основная программа пишется на языке Python с использованием модулей BGL и Draw. Первый из них содержит функции из OpenGL, а второй позволяет рисовать кнопки и выводить текст на экран и организует основной цикл для работы в интерактивном режиме.

**2. 2) Модуль Draw.**

Модуль позволяет создавать объекты в интерактивном режиме. Он предназначен для создания окон, с помощью которых осуществляется ввод параметров, кнопок, управляющих вызовом функций, выводить текст в основное окно. . Модуль содержит константы, соответствующие нажатию на клавишу. Все они имеют вид: AKEY, BKEY,…, нажатие на специальные клавиши ESCKEY, -нажатие на цифровую клавишу ONEKEY,…. Клавиши мыши: LEFTMOUSE.RIGHTMOUSE, MIDDLEMOUSE, функциональные клавиши и другие. Функции, перехватывающие прерывания от клавиатуры, мыши, и функции, рисующие на экране, должны быть зарегистрированы. Это делается с помощью функции Draw.Register(a,b,c). Вызов этой функции стартует основной цикл обработки. Здесь «а» - имя функции рисования, «в» - имя функции для обработки прерываний от клавиатуры или мыши, «с» - имя функции для обработки нажатия на нарисованную кнопку.

Draw.Register(None, event,None).

Второй параметр функции event определяет, произошло нажатие на клавишу или клавиша отпущена.

Рисование кнопок

Для управления программой можно использовать рисованные кнопки. Нажатие на кнопку обрабатывается специальной программой. В простейшем случае создание кнопки осуществляется командой Draw.PushButton("Здесь находится имя кнопки", 2- идентификатор кнопки, здесь числа 50,10,55,20 означают (50,10 ) координаты вершины кнопки, (55,20) – размеры кнопки,"Здесь то, что мы видим, когда курсор установлен на кнопку")

Перерисовка окна осуществляется функцией gui(), однако прямой вызов функции невозможен. Вместо этого вызывается функция Draw.Redraw(1)

**2. 3) Ввод данных.**

Интерактивный ввод данных

В простейшем случае ввод данных осуществляется функцией retVal=Draw.PupBlock("Надпись на таблице ",block). Хотя значение retVal, как правило, не анализируется, рекомендуется его использовать при создании таблицы ввода. Здесь block список, формирующий данные.

**2. 4) Рисование геометрических примитивов.**

Все геометрические примитивы описываются в терминах своих *вершин* – координат, которые определяют сами точки примитива, конечные точки линии или углы полигона.

**1. Точка**

Точка определяется набором чисел с плавающей точкой, называемым вершиной. Все внутренние вычисления производятся в предположении, что координаты заданы в трехмерном пространстве. Для вершин, которые пользователь определил как двумерные (то есть задал только x- и y-координаты), OpenGL устанавливает z-координату равной 0.

**2. Линии**

В OpenGL термин линия относится к сегменту прямой (отрезку), а не к математическому понятию прямой, которая предполагается бесконечной в обоих направлениях.

**3. Полигоны**

Полигон (или многоугольник) – это область, ограниченная одной замкнутой ломаной, при этом каждый отрезок ломаной описывается вершинами на его концах (вершинами полигона).

В общем случае полигоны могут быть достаточно сложны, поэтому OpenGL накладывает очень серьезные ограничения в отношении того, что считается полигональным примитивом. Во-первых, ребра полигонов в OpenGL не могут пересекаться (в математике полигон, удовлетворяющий этому условию, называется *простым*). Во-вторых, полигоны OpenGL должны быть выпуклыми. Полигон является выпуклым, если отрезок, соединяющий две точки полигона (точки могут быть и внутренними, и граничными) целиком лежит внутри полигона (то есть все его точки также принадлежат полигону).

При использовании OpenGL каждый геометрический объект однозначно описывается в виде упорядоченного набора вершин. Для указания вершины используется команда **glVertex\*()**.

|  |
| --- |
| void **glVertex**{**234**}{**sifd**}(TYPE *coords*); void **glVertex**{**234**}{**sifd**}**v**(const TYPE \**coords*); |

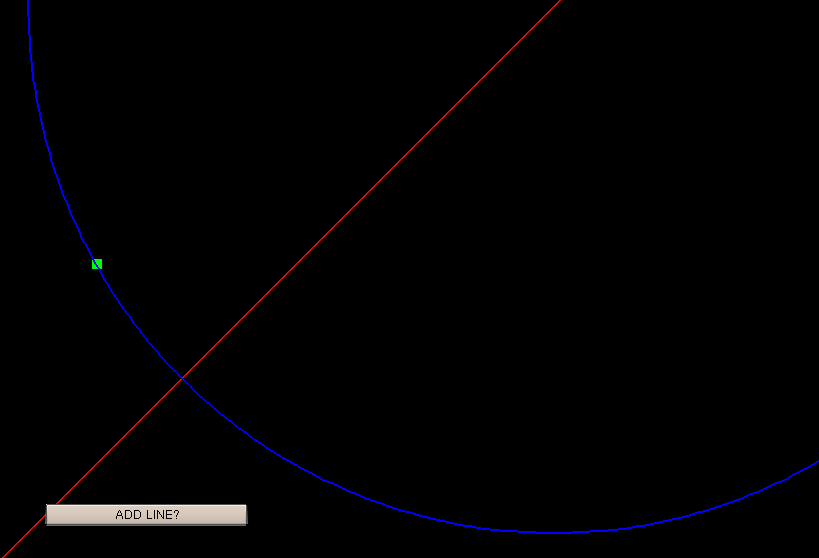
Указывает одну вершину для использования в описании геометрического объекта. Для каждой вершины вы можете указывать от 2 (x,y) до 4 координат (x,y,z,w), выбирая соответствующую версию команды. Если используется версия, где в явном виде не задаются z или w, то z принимается равным 0, а w – равным 1. Вызовы **glVertex\*()** имеют силу только между вызовами команд **glBegin()** и **glEnd()**.

Для того чтобы создать из вершин набор точек, линию или полигон, следует обрамить каждый набор вершин вызовами команд **glBegin()** и **glEnd()**. Аргумент, передаваемый **glBegin()**, определяет, какие графические примитивы создаются на основе вершин.

|  |  |
| --- | --- |
| GL\_POINTS | Рисует точку на месте каждой из *n* вершин |
| GL\_LINES | Рисует серию несоединенных между собой отрезков. Рисуются отрезки между V[0] и V[1] , между V[2] и V[3], и так далее. Если *n* нечетно, то последний отрезок рисуется между V[*n*-3] и V[*n*-2], а вершина *n*-1 игнорируется. |
| GL\_LINE\_STRIP | Рисует отрезок между V[0] и V[1], между V[1] и V[2], и так далее. Последний отрезок рисуется между V[*n*-2] и V[*n*-1]. Таким образом, создается ломаная из *n-1* отрезков. Если *n* < 2 не отображается ничего. Не существует ограничений на вершины, описывающие ломаную (или замкнутую ломаную). Отрезки могут любым образом пересекаться. |
| GL\_LINE\_LOOP | То же, что и GL\_LINE\_STRIP, но, кроме того, рисуется отрезок между V[*n*-1] и V[0], замыкающий ломаную. |
| GL\_TRIANGLES | Рисует серию треугольников (полигонов с тремя сторонами) используя вершины V[0], V[1] и V[2], затем V[3], V[4] и V[5], и так далее. Если *n* не кратно 3, последние 1 или 2 вершины игнорируются. |
| GL\_TRIANGLE\_STRIP | Рисует серию треугольников, используя вершины V[0], V[1] и V[2], затем V[2], V[1] и V[3] (обратите внимание на порядок), затем V[2], V[3], V[4], и так далее. Такой порядок гарантирует, что все треугольники будут иметь одинаковую ориентацию и, таким образом, соединенные треугольники могут сформировать часть поверхности. Сохранение ориентации очень важно для некоторых операций (например, для отсечения нелицевых граней). Для того, чтобы нарисовался хотя бы один треугольник *n* должно быть больше или равно 3. |
| GL\_TRIANGLE\_FAN | То же, что и GL\_TRIANGLE\_STRIP, но порядок вершин следующий: V[0], V[1], V[2], затем V[0], V[2], V[3], затем V[0], V[3], V[4], и так далее. |
| GL\_QUADS | Рисует серию четырехугольников (полигонов с четырьмя сторонами), используя вершины V[0], V[1], V[2], V[3], затем V[4], V[5], V[6], V[7], и так далее. Если *n* не кратно 4, последние 1, 2 или 3 вершины игнорируются. |
| GL\_QUAD\_STRIP | Рисует серию прямоугольников, используя следующий порядок вершин: V[0], V[1], V[3], V[2], затем V[2], V[3], V[5], V[4], затем V[4], V[5], V[7], V[6] и так далее. *n* должно быть не меньше 4, иначе ничего не будет нарисовано. Если *n* нечетно, последняя вершина игнорируется. |
| GL\_POLYGON | Рисует полигон, используя точки V[0], V[1], V[2], ..., V[*n*-1] в качестве вершин. *n* должно быть не меньше 3 или ничего нарисовано не будет. Кроме того, описываемый полигон не должен иметь самопересечений и должен быть выпуклым. Если вершины не удовлетворяют этим условиям, результат непредсказуем. |

**Выполненные семестровые работы:**

**№1.**

***15. Проведение окружности с центром на данной прямой через две данные точки. Одна точка движется***  


import Blender

from Blender import Draw, BGL

from Blender.BGL import \*

import time

from Blender.Mathutils import \*

import math

from math import sin,cos,sqrt,fabs

xy0=Vector([0,0]) # two types of constructors

xy1=Vector([0,0]) #

point0=Vector([0,0])

point1=Vector([0,0])

shag=0

yes=0

a=0

b=0

def getCenter():

global a

global b

global mddl

global xy0

global xy1

Xcenter=0

Ycenter=0

radius=100

y=0

i = 0

while (i < 1000):

y=i\*a+b

i+=1

if (fabs(sqrt((i-xy0[0])\*\*2+(y-xy0[1])\*\*2)-sqrt((i-xy1[0])\*\*2+(y-xy1[1])\*\*2))<10):

radius=sqrt((i-xy0[0])\*\*2+(y-xy0[1])\*\*2)

Xcenter=i

Ycenter=y

break

makeCircle(360.0,radius,Xcenter,Ycenter)

def changeXY1():

global shag

global xy1

n=1000

shag+=1

newVect = (point1-point0)/100

xy1=xy1+newVect

time.sleep(0.1)

if shag > n : Draw.Exit()

Draw.Redraw(1)

def event(evt, val):

if evt == Draw.ESCKEY :

Draw.Exit()

return

def makeCircle(passedNumberOfPoints, passedRadius, XX,YY):

angleStep = 360.0/ passedNumberOfPoints

for ang in range(0, 359,angleStep):

x = passedRadius \* math.cos(math.radians(ang))+XX

y = passedRadius \* math.sin(math.radians(ang))+YY

glVertex2i(x,y)

def gui():

Draw.PushButton("ADD LINE?",1,50,40,200,20,"Add point for line")

global yes

if(yes==1):

glClearColor(0,0,0,1)

glClear(GL\_COLOR\_BUFFER\_BIT)

glLineWidth(2)

glColor3f(1.0,0,0)

glBegin(GL\_LINE\_LOOP)

glVertex2f(\*point0)

glVertex2f(\*point1)

glEnd()

glPointSize(10)

glColor3f(0,1.0,0)

glBegin(GL\_POINTS)

glVertex2f(\*xy0)

glVertex2f(\*xy1)

glEnd()

glColor3f(0,0,1.0)

glBegin(GL\_LINE\_LOOP)

changeXY1()

getCenter()

glEnd()

def button\_event(evt):

if evt == 1:

global yes

global xy0

global xy1

global point0

global point1

global mddl

global a

global b

X0=Draw.Create(0)

Y0=Draw.Create(0)

X1=Draw.Create(0)

Y1=Draw.Create(0)

A=Draw.Create(0)

B=Draw.Create(0)

block=[]

block.append(("X1= ",X0,0,1000))

block.append(("Y1= ",Y0,0,1000))

block.append(("X2= ",X1,0,1000))

block.append(("Y2= ",Y1,0,1000))

block.append(("A= ",A,-1000,1000))

block.append(("B= ",B,-1000,1000))

retVal=Draw.PupBlock("LINE'S POINTS",block)

xy0[0]=X0.val

xy0[1]=Y0.val

xy1[0]=X1.val

xy1[1]=Y1.val

a=A.val

b=B.val

#LINE

point0[0]=1

point0[1]=a\*point0[0]+b

point1[1]=1000

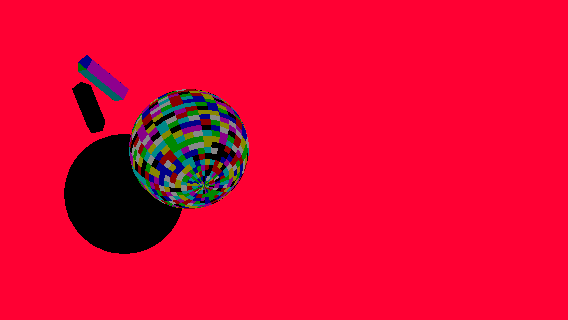
point1[0]=(point1[1]-b)/a

yes=1

return

Draw.Register(gui, event, button\_event)

**№2.**

***3. Пятиугольная призма движется навстречу шару. Останавливается после протыкания шара насквозь. Тень на плоскость х=0***

import Blender as B

import random

import time

import Blender

from Blender import Mathutils

from Blender.Mathutils import \*

from Blender import Draw

from Blender.BGL import \*

from math import sqrt,tan,radians

buffP=Buffer(GL\_FLOAT,4,[0.,0.,20,1]) #initial position of LIGHT0 (proj coords)

ob=Blender.Object.Get('Sphere')

msh=ob.getData(False,True)

fs=msh.faces #grani

a=1

b=0.

l=0.1

vect=Vector(5,5,10)

def key(evt,val):

if evt==Draw.ESCKEY: Draw.Exit()

RADIUS=sqrt(fs[0].verts[0].co.x\*\*2+fs[0].verts[0].co.y\*\*2+fs[0].verts[0].co.z\*\*2)

def changeA():

global a

a +=1

global b

b+=0.01

global buffP

#buffP[1] +=0.2

#buffP[2] +=0.3

time.sleep(0.00001)

Draw.Redraw(1)

def createF(F,shadow):

glFrontFace(GL\_CCW)

glColor3f(0,0,0)

if(shadow):

glTranslatef(2,2,2)

for ff in F:

if len(ff.verts)==4:

if(shadow!=1):

glColor3f(random.randint(0,1),random.randint(0,1),random.randint(0,1))

glBegin(GL\_QUADS)

for pt in ff.verts:

glVertex3f(pt.co.x,pt.co.y,pt.co.z)

#glVertex3f(1,0,1)

glEnd()

if len(ff.verts)==3:

if(shadow!=1):

glColor3f(random.randint(0,1),random.randint(0,1),random.randint(0,1))

glBegin(GL\_TRIANGLES)

for pt in ff.verts:

#glVertex3f(1,0,1)

glVertex3f(pt.co.x,pt.co.y,pt.co.z)

glEnd()

if len(ff.verts)==2:

if(shadow!=1):

glColor3f(random.randint(0,1),random.randint(0,1),random.randint(0,1))

glLineWidth(1)

glBegin(GL\_LINES)

for pt in ff.verts:

#glVertex3f(1,0,1)

glVertex3f(pt.co.x,pt.co.y,pt.co.z)

glEnd()

def myFace(len):

glFrontFace(GL\_CCW)

global b

global RADIUS

global l

if( (4-b) > ( ((-1)\*RADIUS -l\*5) )):

glTranslatef(0,4-b,0)

else:

glTranslatef(0,-1-l\*5,0)

glBegin(GL\_QUADS)

glVertex3f(-1\*len,-5\*len,0)

glVertex3f(1\*len,-5\*len,0)

glVertex3f(1\*len,5\*len,0)

glVertex3f(-1\*len,5\*len,0)

glEnd()

def myCube(shadow):

glPushMatrix()

rad = radians(31);

m=4.5\*l\*0.5 \* tan(rad)

glTranslatef(0,0,-m)#sdvig na 10 edinic -> novaja sistema koordinat

if(shadow!=1):

glColor3f(0,0,1)

myFace(l)

glPopMatrix()#vozvrat

glPushMatrix()#zapomnili)

glRotatef(72\*1,0,1,0)#povorot - dlya sohranenija orientacii normali

glTranslatef(0,0,-m)#svig nazad na 10 otnositelno (0,0,0)

if(shadow!=1):

glColor3f(0,1,0)

myFace(l)

glPopMatrix()#vozvrat

glPushMatrix()#zapomnili)

glRotatef(72\*2,0,1,0)#povorot vokrug 'y'

glTranslatef(0,0,-m)

if(shadow!=1):

glColor3f(0,1,1)

myFace(l)

glPopMatrix()#vozvrat

glPushMatrix()#zapomnili)

glRotatef(72\*3,0,1,0)

glTranslatef(0,0,-m)

if(shadow!=1):

glColor3f(1,0,1)

myFace(l)

glPopMatrix()#vozvrat

glPushMatrix()#zapomnili

glRotatef(72\*4,0,1,0)#povorot vokrug 'y'

glTranslatef(0,0,-m)

if(shadow!=1):

glColor3f(1,1,0)

myFace(l)

glPopMatrix()#vozvrat

glPushMatrix()#zapomnili

glPopMatrix()

glRotatef(90,1,0,0)

glTranslatef(0,0,5\*l)

if(shadow!=1):

glColor3f(1,0,1)

#myZad(m)

def myZad(m):

glBegin(GL\_POLYGON)

glVertex3f(0.5\*m+m\*cos(30)+0.9, 0, 0)

glVertex3f(0.5\*m, m\*sin(30), 0)

glVertex3f((-1)\*0.5\*m, m\*sin(30), 0)

glVertex3f((-1)\*(0.5\*l+m\*cos(30))-1.2, 0, 0)

glVertex3f((-1)\*0.5\*m, (-1)\*m\*sin(30), 0)

glVertex3f(0.5\*m, (-1)\*m\*sin(30), 0)

glEnd()

def plane():

glBegin(GL\_QUADS)

glVertex3f(-20,-20,-50)

glVertex3f(20,-20,-50)

glVertex3f(20,20,-50)

glVertex3f(-20,20,-50) # plane where we see shadow

glEnd()

def key(evt,val):

if evt==Draw.ESCKEY: Draw.Exit()

def gui():

glClearColor(1,0,0.2,1)

glClear(GL\_COLOR\_BUFFER\_BIT|GL\_DEPTH\_BUFFER\_BIT)

glEnable(GL\_DEPTH\_TEST)

glEnable(GL\_LIGHTING)

glFrontFace(GL\_CW)

glCullFace(GL\_FRONT)

glEnable(GL\_COLOR\_MATERIAL);

glViewport(0,0,600,600)

glMatrixMode(GL\_PROJECTION)

glLoadIdentity()

glOrtho(-5,5,-5,5,-500,500)

glMatrixMode(GL\_MODELVIEW)

glLoadIdentity()

glRotatef(60,1,0,1)

createF(fs,0)

myCube(0)

glRotatef(180,0,0,1)

glTranslatef(0,0,-1)

createF(fs,1)

myCube(1)

changeA()

Draw.Register(gui,key,None)