**Лабораторная работа №1.**

**«Работа с матрицами».**

1. Задание

Познакомиться со средствами создания графического пользовательского интерфейса и средствами рисования. Подготовить инструментарий для работы с трехмерной графикой.

Получить удобные средства работы с векторами и матрицами. Написать программу, рисующую перспективную проекцию трехмерного каркасного объекта.

Фигура рисуется с помощью доступных функций рисования отрезка в координатах окна. Фигура состоит из отрезков - ребер. Может генерироваться в программе или загружаться из файла.

Наблюдатель расположен в точке (3 4 5), а объект поднят над плоскостью Oxy.

Пользователь имеет возможность вращать фигуру (2 степени свободы) и включать/выключать перспективу.

Вывод графики осуществляется быстро и плавно. При этом CPU должен быть занят на 100%.

1. Теория

Для отрисовки каркасного объекта понадобится набор точек, по которым будет строится фигура. Но так как нужно будет осуществлять некоторые действия над объектом, нам нужно знать, как устроена работа с матрицами и векторами и как по точкам перейти к векторам.

Для хранения данных о каркасном объекте будем использовать список однородных координат. Модель однородного представления удобна тем, что переход от точки к вектору и обратно осуществляется изменением одной последней координаты. Точки и векторы представляются с помощью одного и того же набора основных базовых объектов. Разница только в последней 4ой координате: векторы имеют 4м компонентом 0, а точки - 1.

В качестве каркасного объекта был выбран тор. Для создания списка его точек используем формулы:

X = D+A\*cos(v))\*cos(u)

Y = (D+A\*cos(v))\*sin(u)

Z = A\*sin(v)

где A, D - константы, u, v – углы [–pi; pi]

Для задания различных преобразований объектов сцены используются операции над матрицами. Выделяют два основных типа матриц: модельно-видовая и матрица проекций. Рассмотрим их:

1) Видовая матрица определяет преобразования объекта в мировых координатах, такие как параллельный перенос, изменение масштаба и поворот.

2) Матрица проекций определяет, как будут проецироваться трехмерные объекты на плоскость экрана (в оконные координаты).

Рассмотрим, какие компоненты включены в каждую матрицу:

1) Матрица поворота:

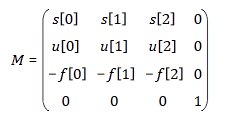


Где с=cos(angle), s = sin(angle), angle – угол поворота.

2) Масштабирования



3) Перемещение положения наблюдателя



При этом:

C:\Users\Victoria\AppData\Local\Temp\msohtmlclip1\01\clip_image001.png - векторное произведение нормированных векторов.

- нормировка



centerXYZ – координата, куда смотрит наблюдатель (координата центра)

eyeXYZ – позиция наблюдателя.

Up вектор направления. По этому вектору происходит наблюдение за объектом. Имеет координаты:



2) Перспективного преобразования



Где:



А все остальные параметры задаются вручную.

Аналогичные матрицы используются в OpenGl методами: gluLookAt, glRotate, glScale, gluPerspective и др.

Для проведения операций достаточно умножить на соответствующую матрицу каждую вершину тора и получить измененные координаты этой вершины.

Для увеличения скорости работы сначала можно подсчитать общую матрицу, а затем каждую вершину умножить на эту общую матрицу.

1. Алгоритм

Работа каркасной модели тора была осуществлена при помощи библиотеки PyQt4, которая включает в себя мощный инструментарий для работы с матрицами и векторами. Для создания окна использовался QtDesigner. После создания .ui-файла, скомпилируем его в Python-код с помощью команды pyuic4. Затем в основную программу пронаследуем сгенерированный класс widget.py и начнем пользоваться стандартными возможностями PyQt.

В программе присутствуют 2 класса:

1) qtMatrix() -- содержит в себе все используемые матрицы

2) Widget -- класс, который отвечает за вывод формы, реакцию на мышь и клавиатуру, за центрирование объекта и установку значений по умолчанию.

Функции formulTorus & makeTorus -- вместе формирует список векторов для тора.

Приведем реализацию:

def formulTorus(v, u, D = 2.0, A = 1.0):

return QtGui.QVector4D((D+A\*cos(v))\*cos(u), (D+A\*cos(v))\*sin(u), A\*sin(v), 1)

def makeTorus():

def frange(start,end,step):

return map(lambda x: x\*step, range(int(start\*1.0/step),int(end\*1.0/step)))

points = []

pi = atan(1)\*4

st = 0.5

st = pi / int(pi / st)

for a in frange(-pi-st, pi+st, st):

for b in frange(-pi-st, pi+st, st):

ltmp = [formulTorus(v,u)

for v,u in ((a,b), (a+st,b),

(a+st,b), (a+st,b+st),

(a+st, b+st), (a+st,b),

(a+st,b), (a,b))

]

points.extend(ltmp)

return points

Для включения/выключения перспективы и изменение наблюдателя используется стандартный метод self.ui.checkBox.isChecked(), который проверяет, выбрана ли галочка в соответствующем блоке или нет.

def paintEvent(self, event):

size = self.size()

ox = size.width()/2

oy = size.height()/2

matr = self.matr

#~ объявление всех матриц для использования

k=60

matrScale = matr.scale(k\*self.sc)

matrRZ = matr.rotateZ(self.alpha)

matrRX = matr.rotateX(self.betta)

vectTrans = QtGui.QVector4D(ox,oy,0,0)

#~ получение "перспективных" вершин

vect\_Torus = self.Torus

if self.ui.checkBox.isChecked():

vect\_Torus = [vect\_perspective(matr,self.Torus[i]) for i in range(0, self.amount+1)]

#~ общая матрица

matrTotal = matrRZ\*matrRX\*matr.exchange\*matrScale

#~ камера применяется почти в конце

if self.ui.checkBox\_2.isChecked():

matrTotal \*= matr.lookAt()

paint = QtGui.QPainter()

paint.begin(self)

paint.setPen(QtCore.Qt.blue)

#~ каждый вектор умножается на общую матрицу и "немного" сдвигается

for i in range(0, self.amount, 2):

v1 = vect\_Torus[i+0]\*matrTotal

v2 = vect\_Torus[i+1]\*matrTotal

v1 += vectTrans

v2 += vectTrans

paint.drawLine(v1.x(),v1.y(),v2.x(),v2.y())

paint.end()

self.update()

1. Выводы

Увидели, как удобно однородное представление координат. Оно и правда очень часто используется, например, в OpenGl все векторы и точки представляются только в однородных координатах. Такие однородные координаты удобно умножать на матрицы (4х4).

Матрицы, которые мы реализовывали в данной лабораторной работе, уже существуют и в библиотеках PyQt и в OpenGl.

1. Литература
2. А.В. Боресков Графика трехмерной компьютерной игры на основе OpenGL.
3. Ф.Хилл OpenGL. Программирование компьютерной графики.
4. Д.Роджерс Алгоритмические основы машинной графики.

**Screen:**

