МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное

учреждение высшего образования

«Ижевский государственный технический университет имени М. Т. Калашникова»

Факультет «Информатика и вычислительная техника»

Кафедра «Программное обеспечение»

Работа защищена с оценкой

«\_\_\_\_\_\_\_»

Дата\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КУРСОВАЯ РАБОТА

по дисциплине «Компьютерная графика»

на тему «Пространственная трансформация объектов с использованием аппарата однородных координат»

Выполнил:

Студент группы Б03-191-1з О.В.Кузнецов

Проверил:

доцент Л.Н.Левицкая

Рецензия:

степень достижения поставленной цели работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

полнота разработки темы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

уровень самостоятельности работы обучающегося\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

недостатки работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Содержание

Описание задания3

Анализ предметной области4

Выбор и обоснование языка7

Анализ решения8

Код программы9

Описание задания

Необходимо написать программу, поворачивающую фигуру (вытянутый прямоугольник, стоящий у левого края экрана на наклонной линии), на угол ***r*** (90 градусов) по часовой стрелке вокруг первой вершины фигуры, затем на тот же угол вокруг второй вершины и т. д. ***k*** раз (каждый раз в качестве точки поворота выбирать нижний правый угол очередного прямоугольника. Одну из сторон фигуры (короткую) выделить другим цветом.

Фигуру, подвергающуюся преобразованию, задавать массивом векторов,

где каждый вектор представляет координаты одной из вершин фигуры.

Необходимое преобразование задавать матрицей 3**×**3. Следует помнить, что пространственные преобразования реализуются в трехмерном пространстве однородных координат, которое соответствует двухмерному пространству декартовых координат.

В зависимости от условия задачи, подключить ресурсы (или меню, или кнопки) для управления анимацией.

Пример выполнения.



Анализ предметной области

Компьютерная геометрия есть математический аппарат, положенный в основу компьютерной графики. В свою очередь, основу компьютерной геометрии составляют различные преобразования точек и линий. При использовании машинной графики можно по желанию изменять масштаб изображения, вращать его, смещать и трансформировать для улучшения наглядности перспективного изображения. Все эти преобразования можно выполнить на основе математических методов, которые мы будем рассматривать далее.

Для начала заметим, что точки на плоскости задаются с помощью двух ее координат. Таким образом, геометрически каждая точка задается значениями координат вектора относительно выбранной системы координат. Координаты точек можно рассматривать как элементы матрицы [x,y], т. е. в виде вектор-строки или вектор-столбца. Положением этих точек управляют путем преобразования матрицы.

Четыре вектора положения точек заданной фигуры записываются в виде матрицы:

|  |  |
| --- | --- |
| xa | y­a |
| xb | yb |
| xc | yc |
| xd | yd |

Матрица преобразований представляется в виде:

|  |  |
| --- | --- |
| a | b |
| c | d |

Применение матричного преобразования к заданной фигуре приводит к виду:

|  |  |
| --- | --- |
| xa | y­a |
| xb | yb |
| xc | yc |
| xd | yd |

|  |  |
| --- | --- |
| xa\* a + ya \* c | xa\* b + ya \* d |
| xb\* a + yb\*c | xb\* b + yb\*d |
| xc\* a + yc\*c | xc\* b + yc\*d |
| xd\* a + yd\*c | xd\* b + yd\*d |

|  |  |
| --- | --- |
| a | b |
| c | d |

\* =

Преобразования переноса, масштабирования и поворота записываются в матричной форме в виде P\*=P + T, P\* = P \* S, P\* = P \* T. Очевидно, что перенос, в отличие от масштабирования и поворота, реализуется с помощью сложения. Это обусловлено тем, что вводить константы переноса внутрь структуры общей матрицы размером 2×2 не представляется возможным. Желательным является представление преобразований в единой форме – с помощью умножения матриц. Эту проблему можно решить за счет введения третьей компоненты в векторы точек [x y] и [x\* y\*] , т. е. представляя их в виде [x y 1] и [x\* y\* 1]. Матрица преобразования после этого становится матрицей размером 3×2. Это необходимо, поскольку число столбцов в матрице, описывающей точку, должно равняться числу строк в матрице преобразования для выполнения операции умножения матриц.

|  |  |
| --- | --- |
| a | b |
| c | d |
| m | n |

Поскольку матрица 3×2 не является квадратной, она не имеет обратной матрицы. Эту трудность можно обойти, дополнив 3. Например,×матрицу преобразования до квадратной размером 3×3

|  |  |  |
| --- | --- | --- |
| a | b | 0 |
| c | d | 0 |
| m | n | 1 |

Однородные координаты обеспечивают поворот изображения вокруг точек, отличных от начала координат. В общем случае вращение около произвольной точки может быть выполнено путем переноса центра вращения в начало координат, поворотом относительно начала координат, а затем переносом точки вращения в исходное положение. Таким образом, поворот вектора положения [х у 1 ] около точки (m, n) на произвольный угол может быть выполнен с помощью преобразования

Выполнив две операции умножения матриц, можно записать

Выбор и обоснование языка

Для решения поставленной задачи подходят многие языки программирования, но самым оптимальным выбором является язык Python. Для такого выбора есть несколько причин.

Во первых, язык Python поддерживает объектно-ориентированной подход, благодаря чему, создаваемую систему управления можно разделить на отдельные модули и при этом программа остается поддерживаемой и читаемой, что позволяет при необходимости вносить изменения в программный код с меньшей сложностью, чем при использовании других подходов программирования, таких как структурное программирование.

Во вторых, в языке Python есть графическая библиотека tkinter, которая содержит самые часто используемые графические примитивы и функции для их рисования. С ее помощью повышается производительность работы программиста, а код становится более надежным, коротким и читаемым. Также для использования данной библиотеки нет необходимости устанавливать дополнительной программное обеспечение, что позволяет не нагружать систему.

Анализ решения

Для решения поставленной задачи был реализован класс MyRectangle, который содержит следующие методы:

init – конструктор класса, в нем сохраняются координаты заданной фигуры и поверхность для рисования, также происходит инициализация матрицы преобразований.

step – метод, который совершает вращение фигуры на 90 градусов, вокруг заданной точки, с помощью умножения на матрицу преобразований.

print – метод, который стирает или рисует заданную фигуру.

is\_exited\_borders – метод для проверки выхода за границы окна.

move – метод вызывающий step, до тех пор пока фигура не доходит до границ окна.

Код программы

import tkinter as tk

import time

class MyRectangle:

def \_\_init\_\_(self, canv, coord\_matrix):

self.canv = canv

self.coord\_matrix = []

self.base\_angle = 1

side\_1 = ((coord\_matrix[0][0] - coord\_matrix[3][0])\*\*2

+ (coord\_matrix[0][1] - coord\_matrix[3][1])\*\*2)\*\*0.5

side\_2 = ((coord\_matrix[0][0] - coord\_matrix[1][0])\*\*2

+ (coord\_matrix[0][1] - coord\_matrix[1][1])\*\*2)\*\*0.5

self.max\_side = max(side\_1, side\_2)

for i in coord\_matrix:

self.coord\_matrix.append([])

insert\_index = len(self.coord\_matrix)

for j in i:

self.coord\_matrix[insert\_index - 1].append(j)

self.coord\_matrix[len(self.coord\_matrix)-1].append(1)

bx = self.coord\_matrix[self.base\_angle][0];

by = self.coord\_matrix[self.base\_angle][1];

self.moving\_matrix = [[],[],[]]

self.moving\_matrix[0].extend([0, -1, bx + by])

self.moving\_matrix[1].extend([1, 0, -bx + by])

self.moving\_matrix[2].extend([0, 0, 1])

def step(self):

self.print(1)

new\_matrix = [[], [], [], []]

for i in range(4):

new\_matrix[i].append(

self.coord\_matrix[i][0]\*self.moving\_matrix[0][0] +

self.coord\_matrix[i][1]\*self.moving\_matrix[0][1] +

self.coord\_matrix[i][2]\*self.moving\_matrix[0][2])

new\_matrix[i].append(

self.coord\_matrix[i][0]\*self.moving\_matrix[1][0] +

self.coord\_matrix[i][1]\*self.moving\_matrix[1][1] +

self.coord\_matrix[i][2]\*self.moving\_matrix[1][2])

new\_matrix[i].append(

self.coord\_matrix[i][0]\*self.moving\_matrix[2][0] +

self.coord\_matrix[i][1]\*self.moving\_matrix[2][1] +

self.coord\_matrix[i][2]\*self.moving\_matrix[2][2])

for i in range(len(new\_matrix)):

for j in range(len(new\_matrix[i])):

self.coord\_matrix[i][j] = new\_matrix[i][j]

self.base\_angle += 1

if self.base\_angle > 3:

self.base\_angle = 0

bx = self.coord\_matrix[self.base\_angle][0];

by = self.coord\_matrix[self.base\_angle][1];

self.moving\_matrix[0][2] = bx + by

self.moving\_matrix[1][2] = -bx + by

self.print()

def print(self, is\_deleted = 0):

small\_side\_color = "lightgray"

color = "lightgray"

if is\_deleted == 0:

color = "red"

small\_side\_color = "green"

canv.create\_line(

self.coord\_matrix[0][0],

self.coord\_matrix[0][1],

self.coord\_matrix[1][0],

self.coord\_matrix[1][1],

width = 3, fill = small\_side\_color)

canv.create\_line(

self.coord\_matrix[1][0],

self.coord\_matrix[1][1],

self.coord\_matrix[2][0],

self.coord\_matrix[2][1],

width = 3, fill = color)

canv.create\_line(

self.coord\_matrix[2][0],

self.coord\_matrix[2][1],

self.coord\_matrix[3][0],

self.coord\_matrix[3][1],

width = 3, fill = color)

canv.create\_line(

self.coord\_matrix[3][0],

self.coord\_matrix[3][1],

self.coord\_matrix[0][0],

self.coord\_matrix[0][1],

width = 3, fill = color)

def is\_exited\_borders(self, main\_width, main\_height):

max\_y = self.coord\_matrix[self.base\_angle][1]

buf = self.base\_angle

if buf > 3:

buf = 0

max\_x = self.coord\_matrix[buf][1]

if max\_x + self.max\_side >= main\_width

or max\_y + self.max\_side >= main\_height:

return True

return False

def move(self, root, main\_width, main\_height):

if self.is\_exited\_borders(main\_width, main\_height):

return;

self.step()

wait\_time = 500

if self.base\_angle == 1:

wait\_time = 2000

root.after(wait\_time, lambda: self.move(root, main\_width, main\_height))

main\_width = 700

main\_height = 700

coord\_matrix = [[100, 100], [150, 150], [250, 50], [200, 0]]

root = tk.Tk()

root.title("Test example")

root.minsize(width = main\_width, height = main\_height)

canv = tk.Canvas(root, width = main\_width, height = main\_height, bg = 'lightgray')

rect = MyRectangle(canv, coord\_matrix)

rect.print()

root.after(2000, lambda: rect.move(root, main\_width, main\_height))

canv.pack()

root.mainloop()