МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Доцент |  |  |  | А.В. Аграновский |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ЛАБОРАТОРНАЯ РАБОТА №2  Исследование аффинных преобразований на плоскости |
|  |
| по дисциплине: Компьютерная графика |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4329 |  |  |  | Д.С. Шаповалова |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2024

Содержание

[1. Цель работы: 3](#_Toc178542351)

[2. Задание: 3](#_Toc178542352)

[3. Теоретические сведения: 3](#_Toc178542353)

[3.1 Аффинные преобразования в общем 3](#_Toc178542354)

[3.2. Композиция аффинных преобразований на плоскости 4](#_Toc178542355)

[3.3. Создание анимированных изображений 5](#_Toc178542356)

[3.4 Особенности применения аффинных преобразований на плоскости 5](#_Toc178542357)

[4. Алгоритм преобразований: 6](#_Toc178542358)

[5. Язык программирования и используемые библиотеки 6](#_Toc178542359)

[6. Описание разработанной программы: 7](#_Toc178542360)

[7. Скриншоты, иллюстрирующие результаты работы программы: 9](#_Toc178542361)

[8. Вывод: 10](#_Toc178542362)

[8.1 Полученные теоретические знания и навыки: 10](#_Toc178542363)

[8.2 Возникшие проблемы и пути их решения: 11](#_Toc178542364)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 13](#_Toc178542365)

[ПРИЛОЖЕНИЕ А 14](#_Toc178542366)

# 1. Цель работы:

Изучение теоретических основ линейных геометрических преобразований на плоскости. Исследование особенностей практической реализации двумерных аффинных преобразований.

# 2. Задание:

Написать программу на языке высокого уровня, осуществляющую анимированное аффинное преобразование фигуры на плоскости в соответствии с вариантом 17 - Изобразить вращающиеся часовую и минутную стрелки.

Рисование фигур (точек, отрезков) на экране монитора можно, но не обязательно, осуществлять с помощью специализированных библиотек, например, GLUT. Аффинные преобразования необходимо выполнять только путем композиции рассмотренных в методическом пособии матричных вычислений (можно, но не обязательно, использовать библиотеки программ для матричных вычислений). Использование созданных другими авторами программ аффинных преобразований не допускается.

# 3. Теоретические сведения:

## 3.1 Аффинные преобразования в общем

Аффинные преобразования – основные типы линейных преобразований – масштабирование, поворот, перенос, отражение относительно оси. Также отображение плоскости или пространства в себя, при котором параллельные прямые переходят в параллельные прямые, пересекающиеся — в пересекающиеся, скрещивающиеся — в скрещивающиеся.

Отображение в себя – значит, что если мы находились в пространстве , то после образования мы должны остаться в нем же. Например: если мы применили какое-то преобразование к прямоугольнику и получили параллелепипед, то мы вышли из в . А вот если из прямоугольника у нас получился другой прямоугольник, то все хорошо, мы отобразили исходное пространство в себя. Формально это описывается так: «преобразование отображает пространство в ». Если записать с помощью формул: .

Cкрещивающиеся прямые – не лежат в одной плоскости. Мы должны остаться в той же плоскости: значит мы представляем себе 2D декартову систему координат. Здесь речь идет о нескольких прямых, так что давайте представим 2 параллельных линии. Из определения мы понимаем, что после преобразования эти линии должны остаться параллельными. Просто сдвигаем их куда-нибудь из исходного местоположения. (один из видов аффинных преобразований – сдвиг)

Преобразование плоскости называется аффинным, если оно непрерывно, взаимно однозначно и образом любой прямой является прямая.

Преобразование называется непрерывным, если «близкие точки переходят в близкие». Т.е. иначе - если у нас есть две точки и они находятся рядом, то после преобразования они все равно будут находиться где-то поблизости друг от друга.

Далее - преобразование взаимооднозначно, если разные точки переводятся в разные точки и в каждую точку переводится какая-то точка. Например: если мы отобразили отрезок и он слипся в точку - это не взаимооднозначное преобразование. Из отрезка мы должны получить ровно такой же отрезок, тогда будет взаимооднозначно (если это сработает для всех отрезков, конечно).

Пусть у нас есть исходная система координат. Точка в этой системе характеризуется двумя числами - x и y. Совершить переход к новым координатам x' и y'мы можем с помощью следующей системы:

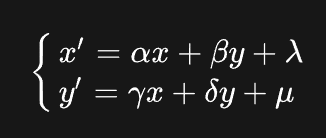


Рисунок 1.1 Система координат

При этом, числа должны образовывать невырожденную матрицу:

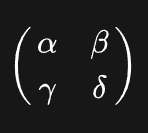


Рисунок 1.2 Невырожденная матрица

Матрица называется невырожденной, если ее определитель не равен нулю, т.е.

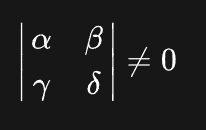


Рисунок 1.3 Определитель матрицы равен 0

Можно записать и в более общем в виде.

Аффинное преобразование - преобразование вида , где - обратимая матрица, а . В данном случае , само собой, -мерный вектор.

## 3.2. Композиция аффинных преобразований на плоскости

Математически сложное аффинное преобразование выражается через последовательное умножение матриц, представляющих каждое простое преобразование. В результате получается итоговая матрица преобразований.

Суперпозиция аффинных преобразований сохраняет прямые линии, отношения длин отрезков на одной прямой или параллельных прямых, а также отношения площадей фигур. Также она гарантирует, что параллельные прямые останутся параллельными.

Важно отметить, что все рассмотренные преобразования выполняются относительно начала координат или координатной оси. Если необходимо выполнить аналогичные преобразования относительно другой точки или прямой, сначала нужно совместить их с началом координат или осью с помощью сдвига и, возможно, поворота.

## 3.3. Создание анимированных изображений

При создании анимированных изображений с помощью аффинных преобразований важно учитывать, что для плавного движения фигур необходимо, чтобы частота смены кадров была не менее 24 кадров в секунду. Если продолжительность анимации составляет Т секунд, то для достижения этой частоты преобразование нужно разделить на 24\*Т итераций. Каждая итерация должна занимать 1/24 секунды.

Можно использовать метод двойной буферизации кадра. При этом изображение выводится на экран из одного буфера, а расчет нового кадра происходит в другом буфере. Затем буферы меняются местами с нужной частотой кадров. При использовании двойной буферизации рекомендуется синхронизировать смену кадров с началом каждой новой итерации. Также важно помнить об удалении фигуры с предыдущей позиции перед началом следующей итерации.

## 3.4 Особенности применения аффинных преобразований на плоскости

Любое аффинное преобразование имеет обратное преобразование, которое также является аффинным и описывается обратной матрицей. Произведение прямой и обратной матриц дает единичную матрицу, то есть все пиксели остаются на своих местах.

Однако, математически точное описание аффинных преобразований для обработки цифровых изображений затруднительно из-за их растровой природы. Большинство преобразований могут привести к нецелым значениям x' и y', которые нужно округлять. При повороте изображения могут возникать неопределенные пиксели, которые не преобразуются ни в один другой пиксель.

Эти проблемы не возникают при преобразованиях, где каждый пиксель отображается в один пиксель (например, отражение относительно оси или поворот на 90 или 180 градусов). В остальных случаях приходится использовать более сложные алгоритмы, такие как методы интерполяции яркости пикселя или алгоритм поворота растровых изображений Оуэна и Македона.

# 4. Алгоритм преобразований:

1. **Матрицы вращения**:

Для вращения в 2D используется матрица вращения. Если угол  задан в радианах, то матрица вращения выглядит так:

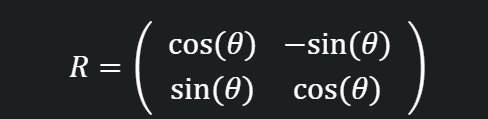


Рисунок 2.1 Матрица вращения (в написанной программе с использованием данной матрицы положительный угол – против часовой)

При умножении этой матрицы на вектор координат (например, ) происходит вращение точки на угол .

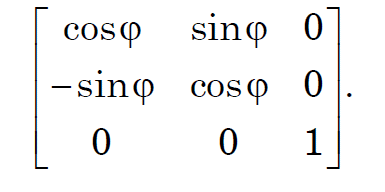


Рисунок 2.2 Матрица вращения (в написанной программе с использованием данной матрицы положительный угол – по часовой)

1. **Векторы стрелок**:

В коде каждая стрелка представляется вектором, например, для часовой стрелки:



Рисунок 2.3 Вектор стрелки

1. **Применение матрицы вращения**:

Для получения нового положения точки, соответствующей стрелке, мы используем: new\_position=R⋅original\_positionnew\_position=R⋅original\_position

Это осуществляется в коде через использование функции rotation\_matrix.

1. **Анимация**:

Каждую итерацию анимации вычисляется новое положение стрелок на основе их угловой скорости и времени. Изменение угла в секунды преобразуется в радианы, после чего применяется аффинное преобразование.

# 5. Язык программирования и используемые библиотеки

В качестве языка программирования был выбран язык Python. В коде используются следующие библиотеки:

1) numpy (import numpy as np):

Библиотека Python, которую применяют для математических вычислений: начиная с базовых функций и заканчивая линейной алгеброй. В данном случае используется для создания и манипулирования массивами, а именно, для создания двумерного массива, матрицы, благодаря которой можно представить пиксели в двумерной системе координат.

2) matplotlib ((import matplotlib.pyplot as plt) и (import matplotlib.animation as animation)):

Библиотека на языке Python для визуализации данных. В ней можно построить двумерные (плоские) и трехмерные графики. В данном случае используется для отображения перемещений точек. В свою очередь «matplotlib.animation» используется для анимации перемещений.

# 6. Описание разработанной программы:

**Определение длины стрелок:**

Устанавливаем длину минутной стрелки в 1.0 и часовой стрелки в 0.5.

**Определение угловой скорости:**

Вычисляем угловую скорость для минутной стрелки (360 градусов за 60 секунд) и часовой стрелки (360 градусов за 12 часов). Это равно соответственно.

**Функция для создания матрицы вращения:**

Определяем функцию, которая возвращает 2D матрицу вращения на угол ( theta ). Используются тригонометрические функции «cos» и «sin» для вычисления значений в матрице.

**Функция обновления:**

Создаем функцию «update», которая будет вызываться для каждого кадра анимации. Сначала очищаем текущий график с помощью «plt.clf()».

**Вычисление углов:**

Для каждой итерации (кадра) вычисляем текущий угол минутной стрелки и часовой стрелки. Углы выражаются в радианах, основываясь на текущем кадре (номер «frame»).

**Вычисление позиций стрелок:**

Используя матрицы вращения, умножаем их на векторы, представляющие положение конца стрелок. Векторы задаются в виде «[0, длина стрелки]», что означает, что стрелки начинаются от начала координат (0, 0).

**Рисование стрелок:**

Используем «plt.plot» для рисования стрелок. Первая координата начинает с точки (0, 0) и заканчивается в позиции, полученной для каждой стрелки. Цвет и ширина линий задаются через «'g'« (зеленый для минутной) и «'b'« (синий для часовой).

*Установка границ графика:*

Управляем размерами отображаемого графика по осям X и Y, чтобы было место для отображения стрелок.

*Установка соотношения сторон:*

Обеспечиваем равные масштабы по обеим осям, чтобы фигуры не искажались.

*Добавление осей координат:*

Рисуем горизонтальную и вертикальную оси в центре графика для наглядности, используя параметры цвета и толщины линии.

*Добавление заголовка и сетки:*

Устанавливаем заголовок для графика, и включаем сетку для лучшей читабельности.

*Добавление легенды:*

Создаем легенду в верхнем правом углу графика, чтобы было понятно, какая стрелка за какую отвечает.

*Создание фигур и анимации:*

Создаем пустую фигуру «fig» и настраиваем анимацию «ani», указывая, что «update» должен вызываться 3600 раз (количество кадров) с частотой 60 кадров в секунду (1000/60 миллисекунд на кадр). Параметры изменяются на усмотрение пользователя.

**Запуск анимации:**

Вызываем «plt.show()», чтобы отобразить анимацию на экране.

# 7. Скриншоты, иллюстрирующие результаты работы программы:

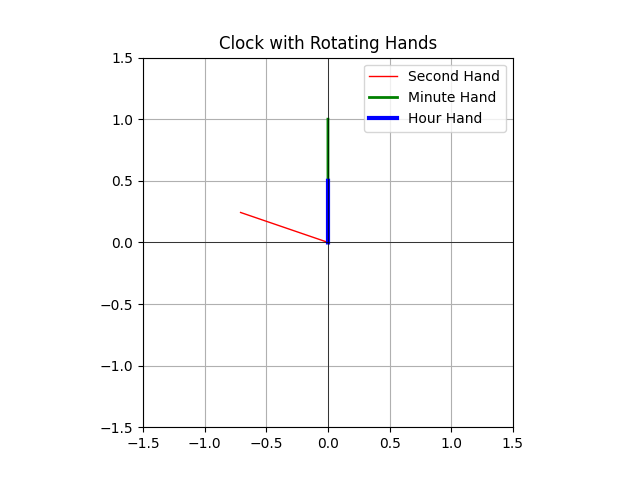


Рис 3.1. Изображение секундной, минутной и часовой стрелки. От запуска программы прошло 47 секунд.

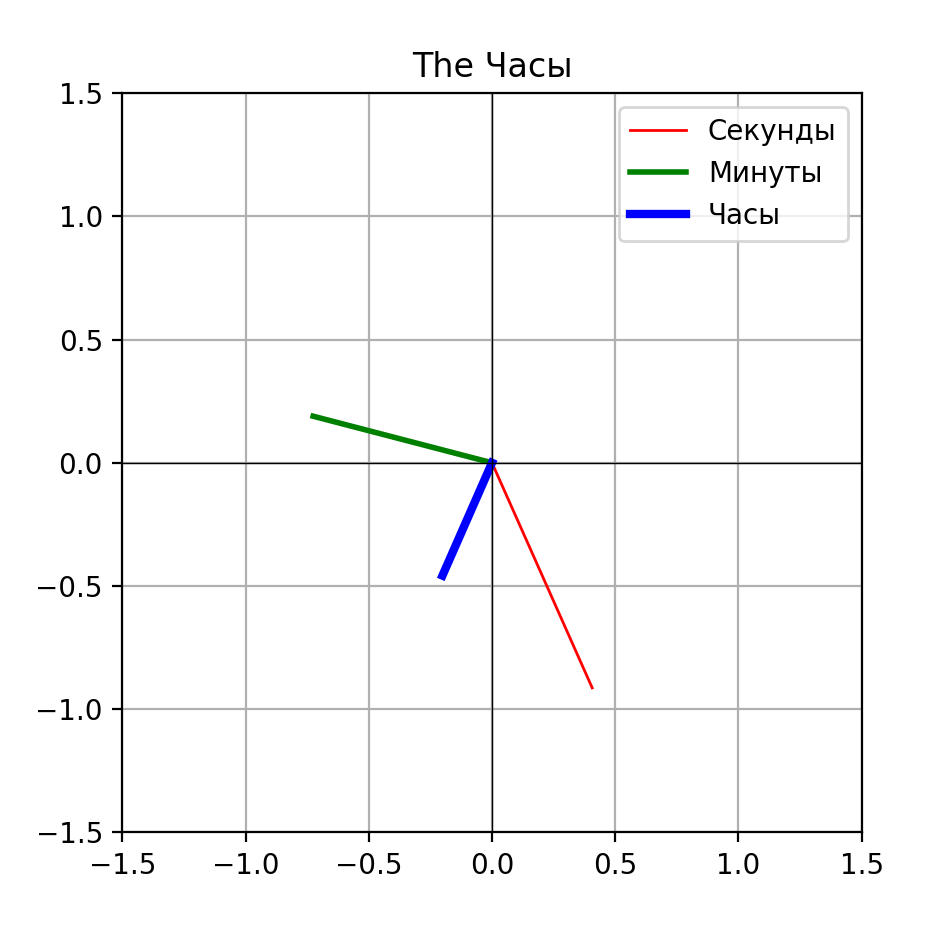


Рис 3.2. Изображение секундной, минутной и часовой стрелки. От запуска программы прошло около 40 минут + было применено весомое ускорение (koef = 1).

# 8. Вывод:

## 8.1 Полученные теоретические знания и навыки:

В процессе выполнения этой работы были получены следующие теоретические знания и навыки:

1. Аффинные преобразования:

Были изучены основы линейных геометрических преобразований на плоскости, особенности практической реализации двумерных аффинных преобразований их виды(поворот, сдвиг, растяжение-сжатие). Была вспомнена линейная алгебра, матрицы, векторы. Были освоены способы их переноса на компьютерный код, реализация манипуляций с пикселями в компьютерной программе.

1. Язык программирования высокого уровня Python:

В процессе выполнения работы был использован язык программирования Python. Был изучен алгоритм создания векторов и визуализации их на экран, посредством графика и математическим операциям с матрицами.

Так же были подробнее изучены некоторые библиотеки Python, такие как numpy и matplotlib, которые в будущем могут пригодиться в работе с графикой и с которыми мы работали в прошлых лабораторных.

1. Библиотеки numpy, matplotlib:

Использование этих библиотек позволило реализовать алгоритм перемещения часовых стрелок и визуализировать это в отдельном окошке. А конкретнее, модуль numpy был использован чтобы вычислять новые координаты точек для вектора. В общем позволяет нам работать с многомерными массивами данных (матрицы). В свою очередь matplotlib был использован для визуализации вычисленных данных на протяжении установленного количества кадров(длины анимации).

1. Вставка формул в Microsoft Word

В процессе написания отчёта была выявлена необходимость вставлять формулы адекватно. Потому пару минут было затрачено на получение этого навыка, хотя бы в базовом виде. Пригодится в дальнейших работах.

## 8.2 Возникшие проблемы и пути их решения:

В процессе выполнения работы возникали определенные трудности, а именно:

1. Стрелки двигались слишком быстро.

Проблема возникала в связи с быстротой отображения изменений на каждый кадр. А именно тем, что в строках, подобных – second\_angle = (frame/koef) \* second\_hand\_speed – «koef» должен быть что-то около 16. Это было вычислено экспериментально и это приблизительное значение. Связано с особенностями устройства часов и времени.

1. Работа с библиотеками:

Matplotlib, использованная в программе – это большая библиотека с обширным функционалом и большим количеством функций. Сходу разобраться тяжеловато, а после ещё и код читать, используя функции, которые написал не ты сам. В этом же и лёгкость – нет необходимости самостоятельно писать функции, что значительно сокращает время реализации программы.

1. Неполная остановка программы.

Программу приходится останавливать 2 раза, потому что после закрытия окошка с анимацией, процессором всё ещё проводятся вычисления. Просто небольшая неприятность, решается 2-мя Ctrl+F2 в редакторе. Возможно, проблема только в PyCharm.

В результате выполнения этой работы, был получен практический опыт в создании алгоритма перемещения векторов на основании математических операций с функциями, аффинных преобразований и визуализации всего этого с использованием языка Python и его библиотек. Такой опыт может пригодиться для решения других задач в сфере информационных технологий, науке о данных и других областях, где используется вычислительное моделирование и визуализация, анимация посредство компьютерных мощностей.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Что за зверь — аффинные преобразования? – URL:  <https://habr.com/ru/articles/539420/> (дата обращения 28.09.2024)
2. Д. Роджерс – Математические основы машинной графики / Д. Роджерс, Дж. Адамс: Пер. с англ. – П.А. Монахова Г.В. Олохтоновой, Д.В. Волкова – М.: Мир, 2001. – 604с., ил.
3. Научная графика в python / Шабанов П.А. –URL: <https://github.com/whitehorn/Scientific_graphics_in_python>  (дата обращения 15.09.2024)
4. Аграновский А. В. Использование методов преобразования координат для формирования растровых изображений: учеб .- метод. Пособие / А. В. Аграновский. – СПб.: ГУАП, 2024. – 40 с.

# ПРИЛОЖЕНИЕ А

Листинг Программы

import numpy as np  
import matplotlib.pyplot as plt  
import matplotlib.animation as animation  
  
# Задаем размеры стрелок (длина линии)  
minute\_hand\_length = 0.75  
hour\_hand\_length = 0.5  
second\_hand\_length = 1  
  
# Угловые скорости стрелок (в радианах за секунду)  
second\_hand\_speed = -(2 \* np.pi) / 60 # 360° за 60 секунды  
minute\_hand\_speed = -(2 \* np.pi) / 3600 # 360° за 3600 секунды (1 час)  
hour\_hand\_speed = -(2 \* np.pi) / 43200 # 360° за 43200 секунд (12 часов)  
  
  
# Функция для создания матрицы вращения  
def rotation\_matrix(theta):  
 return np.array([[np.cos(theta), -np.sin(theta)],  
 [np.sin(theta), np.cos(theta)]])  
# theta - угол на который вращаем  
# array - создаёт матрицу  
  
# Функция для обновления рисунка  
def update(frame):  
 plt.clf() # Очистка текущего рисунка  
 koef = 16 # вычислено экспериментально, чтобы секундная стрелка проходила 360 градусов за 60 секунд.  
 # Чем меньше, тем быстрее двигаются стрелки.  
 # Вычисляем углы для стрелок  
 # каждая итерация цикла (каждый кадр) увеличивает угол на величину, соответствующую угловой скорости  
 second\_angle = (frame/koef) \* second\_hand\_speed # каждый кадр  
 minute\_angle = (frame/(koef^2)) \* minute\_hand\_speed  
 hour\_angle = (frame/(koef^3)) \* hour\_hand\_speed  
  
 # Вектора стрелок  
 second\_hand = rotation\_matrix(second\_angle).dot(np.array([0, second\_hand\_length]))  
 minute\_hand = rotation\_matrix(minute\_angle).dot(np.array([0, minute\_hand\_length]))  
 hour\_hand = rotation\_matrix(hour\_angle).dot(np.array([0, hour\_hand\_length]))  
 # dot - матричное умножение текущей позиции на повёрнутую матрицу => новые координаты вектора.  
 # => координаты конца стрелки в новой позиции, которая была получена путём вращения на угол тета.  
  
 # Рисуем стрелки  
 # Используя матрицы вращения, умножаем их на векторы, представляющие положение конца стрелок.  
 # Векторы задаются в виде [0,длина стрелки] => стрелки начинаются от начала координат (0,0) => вращение вокруг 0,0.  
 plt.plot([0, second\_hand[0]], [0, second\_hand[1]], 'r', lw=1, label='Секунды')  
 plt.plot([0, minute\_hand[0]], [0, minute\_hand[1]], 'g', lw=2, label='Минуты')  
 plt.plot([0, hour\_hand[0]], [0, hour\_hand[1]], 'b', lw=3, label='Часы')  
  
 # Настройки графика  
 plt.xlim(-1.5, 1.5)  
 plt.ylim(-1.5, 1.5)  
 plt.gca().set\_aspect('equal', adjustable='box')  
 plt.axhline(0, color='black', lw=0.5)  
 plt.axvline(0, color='black', lw=0.5)  
 plt.title("The Часы")  
 plt.grid()  
 plt.legend(loc='upper right')  
  
  
# Настройка анимации  
fig = plt.figure()  
ani = animation.FuncAnimation(fig, update, frames=(60\*60\*60), interval=(1000/60))  
# interval - время в миллисекундах между обновлениями кадров анимации.  
# frames - длина анимации в кадрах  
  
# Показ анимации  
plt.show()