МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ
УНИВЕРСИТЕТ

ИНСТИТУТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ
Кафедра системного анализа и информационных технологий

Долгов Дмитрий Александрович

Компьютерная графика

Методическое пособие

КАЗАНЬ, 2025

УДК 004.92, 004.42
ББК 32.97, 32.972.131.2, 32.973

Рекомендовано к изданию
Учебно-методической комиссией ИВМиИТ КФУ
(Протокол № 8 от 18 апреля 2025 года)

Рецензенты:
заведующий кафедры системного анализа и информационных технологий КФУ, доцент, кандидат физико-математических наук **А.В. Васильев**;
кандидат физико-математических наук, доцент кафедры алгебры и математической логики КФУ **М.М. Ямалеев**

Долгов Д.А.
Компьютерная графика: Методическое пособие / Д.А. Долгов. – Казань: Изд-во Казан. ун-та, 2025. – 25 с.

Методическое пособие предназначено для проведения практических занятий по курсу «Компьютерная графика» для студентов, обучающихся по направлениям «Фундаментальная информатика и информационные технологии», «Информационная безопасность». Методическое пособие дает необходимые основы алгебры и геометрии, необходимых для выполнения лабораторных и домашних работ. Здесь представлены примеры рисования геометрических фигур, создание анимации. Программный код написан на языке программирования Python с использованием библиотек NumPy, matplotlib.

УДК 004.92, 004.42
ББК 32.97, 32.972.131.2, 32.973

Содержание

1. Введение

Компьютерная графика предназначена для передачи и манипулирования информацией в графической форме. Информация визуализируется посредством фигур, знаков, цвета. Компьютерная графика представляет собой кросс-дисциплинарную науку. В неё входит математика, физика, биология, дизайн, и др.

Зачем изучать компьютерную графику?

1) Для "правильного" создания и коррекции изображений (неудачная цветопередача, мутное изображение и т.п.).

2) Понимание принципов устройства зрения, обработки цвета компьютером позволяет понять устройство привычных нам вещей: от фотографий и телевидения до алгоритмов сжатия и графических редакторов.

Компютерная графика имеет много разных областей применения:

- Кинематография (спецэффекты)
- Компьютерные игры
- Виртуальная реальность и дополненная реальность
- Разработка ПО (создание интерфейсов)
- Автомобилестроение (разработка эскиза, моделирование поведения деталей)
- Медицина (Томография)
- Архитектура (CAD системы)
- Химия (визуальное представление молекул)
- Вирусология
- Big Data (визуализация данных)
- Стеганография и.т.д.

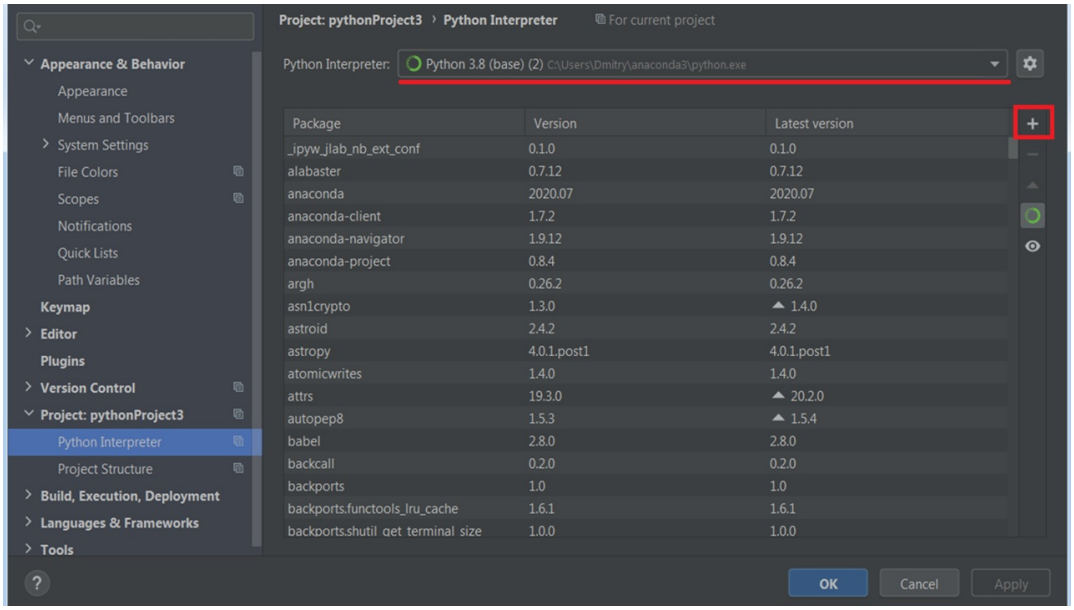
Методическое пособие дает необходимые основы алгебры и геометрии, необходимых для выполнения лабораторных и домашних работ. Здесь представлены примеры рисования геометрических фигур, создание анимации. Программный код написан на языке программирования Python с использованием библиотек NumPy, matplotlib.

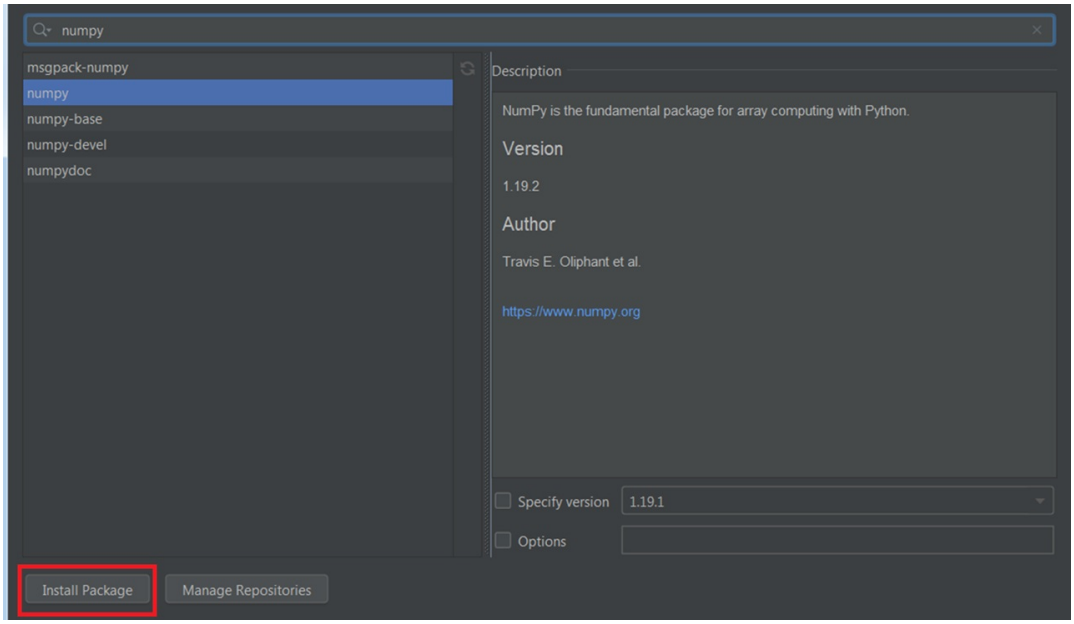
2. Работа с NumPy

NumPy — библиотека с открытым исходным кодом для языка программирования Python. Возможности: поддержка многомерных массивов (включая матрицы); поддержка высокоуровневых математических функций, предназначенных для работы с многомерными массивами. Библиотека NumPy предоставляет реализации вычислительных алгоритмов (в виде функций и операторов), оптимизированные для работы с многомерными массивами. В результате любой алгоритм,р который может быть выражен в виде последовательности операций над массивами (матрицами) и реализованный с использованием NumPy, работает так же быстро, как эквивалентный код, выполняемый в MATLAB.

NumPy – библиотека для работы с массивами. Основная структура - массив (array) определенного типа. Поддерживаются основные типы данных: np.int8, np.int16, np.int32, np.float16, np.float32, float64, np.uint8, np.complex64, np.bool и др.

Для установки NumPy в Pycharm выберите File -> Settings, после чего нажмите на





Установка библиотеки matplotlib происходит таким же образом.

2.1. Создание массива

Пример 1.

```
import numpy as np
a=np.array([[1,2,3],[4,-5,8],[3,6,8]])
print(a)
```

Пример 2.

```
import numpy as np
b = np.array([1,2,3], dtype = np.float32)
print(b)
```

Пример 3. Каждый массив имеет определенный набор атрибутов: размер, количество элементов, тип и др. Для получения значений атрибутов можно воспользоваться следующими функциями shape, size, dtype.

```
import numpy as np
a=np.array([[1,2,3],[4,-5,8],[3,6,8]])
print(a)
print('shape:', a.shape)
print('size:', a.size)
print('type:', a.dtype)
```

Пример 4. Функции zeros, ones позволяют создавать массивы заполненные определенными значениями. В качестве аргумента указываются размеры создаваемого массива.

```
import numpy as np
z=np.zeros((2,3))
print('zero array', z)
o=np.ones((3,4))
print('ones array', o)
```

Пример 5. Для создания массива из последовательных значений можно воспользоваться функциями np.arange() и np.linspace(). Первая функция является аналогом оператора range в языке Python. Может принимать до 3-х аргументов. Формирует массив состоящий из последовательных элементов, определенных аргументами функции: начало, конец, шаг.

```
import numpy as np
ar = np.arange(10)
print(ar)
ar2 = np.arange(10,20)
print(ar2)
ar3 = np.arange(10,20,2)
print(ar3)
```

Пример 6. Функция linspace() позволяет создавать равномерно разбиение интервала на N точек. В качестве аргумента функция принимает начало интервала, конец интервала и количество точек равномерно расположенных точек на интервале. На выходе массив количество элементов в котором совпадает с количеством точек, первый элемент совпадает с началом интервала, последний с концом интервала, а остальные элементы равномерно между началом и концом.

```
import numpy as np
t = np.linspace(0,1,11)
print(t)
```

Пример 7. Следующие функции позволяют создавать массивы со случайными значениями определенного распределения. В качестве аргументов подаются размеры создаваемых массивов. Функция np.random.rand(10,10) задает равномерное распределение U(0,1), получаем матрицу 10*10. Функция np.random.randn(10,10) задает нормальное распределение N(0,1), получаем матрицу 10*10.

```
import numpy as np
r = np.random.rand(10,10)
print(r)
r2 = np.random.randn(10,10)
print(r2)
```

Пример 8. Numpy поддерживает векторные вычисления. Стандартные математические операции +,-,*,/ при этом выполняются поэлементно, необходимо только обеспечивать согласованность размеров.

```
import numpy as np
a = np.arange(5)
b = np.ones((5))
c = a+b
print(c)
```

Пример 9. Надо помнить, что примении * производит поэлементное, а не матричное умножение массивов. Для матричного умножения двух массивов можно воспользоваться функцией np.dot() или оператором @:

```
import numpy as np
r = np.random.rand(10,10)
print(r)
r2 = np.random.randn(10,10)
m = np.dot(r, r2)
print(m)
m2 = r @ r2
print(m2)
```

Пример 10. Кроме этого существует большое количество функций, которые позволяют вычислять определенные характеристики массива.

```
import numpy as np
r2 = np.random.randn(10,10)
print(np.max(r2))
print(np.min(r2))
print(np.mean(r2))
print(np.var(r2))
print(np.median(r2))
```

Пример 11. Обращение по индексам в Numpy происходит аналогично спискам.

```
import numpy as np
```

```
r2 = np.random.randn(10,10)
print(r2[0,0])
print(r2[3,:])
print(r2[2:7, 3:9])
print(r2[2:7:2, 3:9:3])
```

Пример 12. Здесь размеры inds совпадают с размерами r2, значения True стоят в тех ячейках, где элементы r2 больше нуля. В последней строке элементы массива r2, расположенные в тех же позициях, что и элементы True массива inds, обнуляются.

```
import numpy as np
r2 = np.random.randn(10,10)
inds = r2 > 0
r2[inds] = 0
print(r2)
```

Пример 13. В Numpy есть возможность изменить размер массива с помощью функции np.reshape(). При этом нужно обеспечить, чтобы размеры нового массива давали такое же количество элементов. np.hstack() – конкатенация в горизонтальном направлении, а np.vstack() – в вертикальном.

```
import numpy as np
d = np.arange(9)
f = np.reshape(d, (3,3))
print(f)
h = np.hstack((f,f))
print(h)
v = np.vstack((f,f))
print(v)
```

3. Рисование с помощью библиотеки matplotlib

matplotlib – библиотека, которая предоставляет возможность визуализировать данные. Ниже приведен пример построения простого графика параболы.

Пример 1. Функция plt.figure() отвечает за создание графического окна. Функция plt.plot(x,y) отвечает за рисование графика. Функция plt.show() отвечает за отображение на экране.

```
import matplotlib.pyplot as plt
```

```
import numpy as np
x = np.linspace(-3,3,100)
y = x**2
plt.figure()
plt.plot(x,y)
plt.show()
```

Пример 2. Пример отображения зеленого квадрата на черном холсте и его сохранения. При помощи np.zeros создаем черный холст. Потом при помощи заданного среза рисуем зеленый квадрат на черном холсте. Функция plt.imsave отвечает за сохранение изображения.

```
import matplotlib.pyplot as plt
import numpy as np
img = np.zeros((1024, 1024, 3), dtype = np.uint8)
img[300:700, 300:700, 1] = 255
plt.figure()
plt.imshow(img)
plt.show()
plt.imsave('simple_image.png', img)
```

4. Рисование треугольника

4.1. Афинные преобразования

Рассматриваются преобразования с невырожденной матрицей следующего вида:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

В матричной форме преобразование запишется в виде:

$$X' = A * X + B, \det(A) \neq 0$$

Аффинное преобразование применяется к каждой координате геометрической модели. В результате происходит трансформация модели. Результат трансформации зависит от вида матрицы A и вектора B. В общем случае преобразование представляет собой линейное преобразование и параллельный перенос. B – вектор, на который производится параллельный перенос. Если нулевой, то переноса нет. A – матрица линейного преобразования. В зависимости от вида матрицы A осуществля-

ются различные линейные преобразования. Выделяют следующие специальные случаи: A=I - будет линейный перенос на вектор B. A – диагональная. Производится масштабирование фигуры. Элементы диагонали отвечают за масштабирование по соответствующим осям. A – ортогональная. Производится поворот вокруг начала координат!

$$A = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix}$$

$\det(A) > 0$: сохраняется обход вершин. $\det(A) < 0$: обход вершин меняется.

Аффинное преобразование можно компактно записать с помощью проективных координат.

Переход от обычных координат к проективным и наоборот осуществляется с помощью формул:

$$(x, y) = (x, y, 1)$$

$$(x_1, x_2, x_3)\left(\frac{x_1}{x_3}, \frac{x_2}{x_3}\right)$$

Аффинное преобразование в проективных координатах:

$$\begin{pmatrix} x_1{}' \\ x_2{}' \\ x_3{}' \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & b_1 \\ a_{21} & a_{22} & b_2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

Т.к. третья координата x_3 при переходе от декартовых координат равна 1, то представленная формула совпадает по значениям с результатом формулы для декартовых координат. Благодаря компактному представлению в проективных координатах возможно применения аффинного преобразования ко всем вершинам модели представленных в матрице с выписанными по строкам координатами:

$$\begin{pmatrix} x_{11}{}' & \cdots & x_{1n}{}' \\ x_{21}{}' & \cdots & x_{2n}{}' \\ x_{31}{}' & \cdots & x_{3n}{}' \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & b_1 \\ a_{21} & a_{22} & b_2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_{11} & \cdots & x_{1n} \\ x_{21} & \cdots & x_{2n} \\ x_{31} & \cdots & x_{3n} \end{pmatrix}$$

X- матрица, состоящая из координат вершин геометрической модели в проективной форме, выписанных по столбцам.

X'- матрица, состоящая из координат вершин трансформированной геометрической модели в проективной форме, выписанных по столбцам.

С помощью такого представления данной операции можно реализовать распараллеливание процесса вычисления на графических видеокартах.

4.2. Пример

Ниже приведены примеры аффинного преобразования простого геометрического примитива в виде треугольника.

Пример 1.

```
# Transform
import numpy as np
import matplotlib.pyplot as plt


def bVec():
   b = np.array([0, 0])
   return b

def rotMatr(ang):
   mtr = np.array([[np.cos(ang), -np.sin(ang)], [np.sin(ang),
       np.cos(ang)]])
   return mtr

def diagMatr():
   mtr = np.array([[2, 0], [0, 2]])
   return mtr

def detPos():
   mtr = np.array([[1.5, 1], [0.5, 1.2]])
   return mtr

def detNeg():
   mtr = np.array([[0.5, 1], [1.5, -1]])
   return mtr

def to_proj_coords(x):
   r,c = x.shape
   x = np.concatenate([x, np.ones((1,c))], axis = 0)
   return x

def to_cart_coords(x):
   x = x[:-1]/x[-1]
   return x

pt0 = [1, 1]
pt1 = [3, 3]
pt2 = [5, 2]
```

```
x = np.array([pt0, pt1, pt2], dtype = np.float32).T

x_proj = to_proj_coords(x)

# linear transform matrix
a = rotMatr(np.pi/4) # change rotation angle
# a = diagMatr()
# a = detPos()
# a = detNeg()

# shift vector
b = bVec() # change to non-zero

m = np.zeros((3,3))
m[:2,:2] = a
m[:2,-1] = b
m[-1,-1] = 1

x_new_proj = m @ x_proj

x_new = to_cart_coords(x_new_proj)

# drawing
plt.figure()
# plt axes
plt.plot([-10, 10],[0,0], 'k')
plt.plot([0,0],[-10, 10], 'k')
# plot initial figure
plt.plot(x[0, [0,1,2,0]], x[1, [0,1,2,0]], 'r')
plt.plot(x[0,0], x[1,0], 'or')
plt.plot(x[0,1], x[1,1], 'og')
plt.plot(x[0,2], x[1,2], 'ob')
# plot transformed figure
plt.plot(x_new[0, [0,1,2,0]], x_new[1, [0,1,2,0]], 'g')
plt.plot(x_new[0,0], x_new[1,0], 'or')
plt.plot(x_new[0,1], x_new[1,1], 'og')
plt.plot(x_new[0,2], x_new[1,2], 'ob')
plt.show()
```

5. Создание анимации

Рассмотрим задачу анимации отрезка, вращающегося вокруг своего центра. Для решения задачи необходимо задать ряд аффинных преобразований, осуществляющих указанное преобразование. Анимация будет представлять собой набор кадров, на каждом из которых представлен образ отрезка, после применения аффинного преобразования с параметрами меняющимися со временем. Применять матрицу вращения напрямую к координатам отрезка нельзя, т.к. вращение осуществляется вокруг начала координат. Необходимо сначала сдвинуть центр отрезка в начало координат, далее осуществить поворот, после чего сдвинуть начало повернутого отрезка в начальную точку. Получившийся отрезок можно рисовать. Угол поворот должен меняться от кадра к кадру (возрастать), чтобы получилась анимация. Если работать в проективных координатах, то можно записать последовательность преобразований в виде умножения трех матриц. T – матрица сдвига в начало координат

$$T = \begin{pmatrix} 1 & 0 & b_1 \\ 0 & 1 & b_2 \\ 0 & 0 & 1 \end{pmatrix}$$

R – матрица поворота.

$$R = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix}$$

T^{-1} – матрица обратного сдвига.

$$X' = T^{-1}RTX$$

Для объединения набора кадров в анимацию применяются специальные функции из библиотеки Matplotlib, позволяющие сохранить анимацию в виде видеофайла или изображения формата gif. Для создания видеофайла необходимо наличие специальных кодеков. В примере ниже (пример представлен для ОС Windows) явно указывается путь до исполняемого файла с указанными кодеками. Для создание gif файла необходимо наличие установленной библиотеки Pillow. Pillow входит в состав Anoconda и устанавливается вместе с ней.

Ниже приведен пример создания анимации вращающегося отрезка.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from matplotlib.animation import PillowWriter
```

```
plt.rcParams['animation.ffmpeg_path'] = 'ffmpeg.exe'


def shiftMatr(vec):
   mtr = np.array([[1, 0, vec[0]], [0, 1, vec[1]], [0, 0, 1]])
   return mtr

def rotMatr(ang):
   mtr = np.array([[np.cos(ang), -np.sin(ang), 0], [np.sin(ang),
       np.cos(ang), 0], [0, 0, 1]])
   return mtr

def to_proj_coords(x):
   r,c = x.shape
   x = np.concatenate([x, np.ones((1,c))], axis = 0)
   return x

def to_cart_coords(x):
   x = x[:-1]/x[-1]
   return x

pt0 = [50, 50]
pt1 = [50, 150]

x = np.array([pt0, pt1], dtype = np.float32).T
center = np.sum(x, axis=1)/2

x_proj = to_proj_coords(x)

N = 100 # frames count
size = 256
color = np.array([0,255,0], dtype=np.uint8)

frames = []
fig = plt.figure()

for i in range(N):
   # get coords of transformed line
   ang = i*2*np.pi/N
   T = shiftMatr(-center)
   R = rotMatr(ang)
   x_proj_new = np.linalg.inv(T) @ R @ T @ x_proj
   x_new = to_cart_coords(x_proj_new)
```

```
    # draw line
    img = np.zeros((size, size, 3), dtype=np.uint8)
    line_points_count = np.int32(np.max(np.abs(x_new[:,0] -
        x[:,1])) + 1)
    t = np.linspace(0,1,line_points_count)
    a = x_new[:, 0].reshape(-1, 1)
    b = x_new[:, 1].reshape(-1, 1)
    t = t.reshape(1, -1)
    line_points = (1 - t) * a + t * b # not clean lines, use
        Bresehnam instead
    line_points = np.int32(np.round(line_points))
    img[line_points[0], line_points[1]] = color
    im = plt.imshow(img)
    frames.append([im])


print('Frames creation finshed.')


#mp4 animation creation
ani = animation.ArtistAnimation(fig, frames, interval=40,
    blit=True, repeat_delay=0)
Writer = animation.writers['ffmpeg']
writer = Writer(fps=24, metadata=dict(artist='Me'), bitrate=1800)
ani.save('line.mp4', writer)
# ani.save('simple_animation.mp4')


# gif animation creation
ani = animation.ArtistAnimation(fig, frames, interval=40,
    blit=True, repeat_delay=0)
writer = PillowWriter(fps=24)
ani.save("line.gif", writer=writer)
plt.show()
```

6. Построение кубического сплайна

Рассматривается задача построения кубического сплайна (сплайна Эрмита). Даны N+1 точка на плоскости имеющие координаты (x_i, y_i) и значение производной в этой точке $d_i = 0$, $i \in [0, N]$. Предполагается,

что $x_0 < \ldots < x_N$. Требуется для каждого интервала x_i, x_{I+1} определить полином третьей степени, удовлетворяющий следующим условиям:

$$f_i(x) = a_{i0} + a_{i1}x + a_{i2}x^2 + a_{i3}x^3$$
$$f_i(x_i) = y_i$$
$$f_i(x_{i+1}) = y_{i+1}$$
$$f_i'(x_i) = d_i$$
$$f_i'(x_{i+1}) = d_{i+1}$$

Для нахождения коэффициентов полинома предлагается построить систему линейных алгебраических уравнений на основе приведенных выше условий. Получим следующее:

$$a_{i0} + a_{i1}x_i + a_{i2}{x_i}^2 + a_{i3}{x_i}^3 = y_i$$
$$a_{i0} + a_{i1}x_{i+1} + a_{i2}{x_{i+1}}^2 + a_{i3}{x_{i+1}}^3 = y_{i+1}$$

$$a_{i0}0 + a_{i1} + 2a_{i2}x_i + 3a_{i3}{x_i}^2 = d_i$$

$$a_{i0}0 + a_{i1} + 2a_{i2}x_{i+1} + 3a_{i3}{x_{i+1}}^2 = d_{i+1}$$

Данную систему можно переписать в матричной форме:

$$MA = B$$

$$M = \begin{pmatrix} 1 & x_i & {x_i}^2 & {x_i}^3 \\ 1 & x_{i+1} & {x_{i+1}}^2 & {x_{i+1}}^3 \\ 0 & 1 & 2x_i & 3{x_i}^2 \\ 0 & 1 & 2x_{i+1} & 3{x_{i+1}}^2 \end{pmatrix}, A = \begin{pmatrix} a_{i0} \\ a_{i1} \\ a_{i2} \\ a_{i3} \end{pmatrix}, B = \begin{pmatrix} y_i \\ y_{i+1} \\ d_i \\ d_{i+1} \end{pmatrix}$$

Решение системы находится следующим образом:

$$A = M^{-1}B.$$

Ниже приведен пример реализующий представленный метод. Изменяя значения производных можно управлять формой сплайна.

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.image import imsave
```

```
def create_image(heigh, widht, background_color):
img = np.zeros((heigh, widht, 4), np.uint8)
img[:, :, :3] = background_color
img[:, :, 3] = 255

return img


def set_color(img, x, y, color):

img[x, y, :3] = color

return img


def draw_line(img, x0, y0, x1, y1, color):
# Bresenham algorithm
steps_num = int(np.max([np.abs(x0-x1), np.abs(y0-y1)]))
sp = np.linspace(0, 1, steps_num + 1)

x_coords = np.int32(np.round(x0*sp + x1*(1-sp)))
y_coords = np.int32(np.round(y0 * sp + y1 * (1 - sp)))

x_ind = (x_coords>0) & (x_coords < img.shape[0])
y_ind = (y_coords > 0) & (y_coords < img.shape[0])
ind = x_ind & y_ind

x_coords = x_coords[ind]
y_coords = y_coords[ind]

img = set_color(img, x_coords, y_coords, color)

return img


def show_image(img):
img = np.flipud(img)
plt.figure()
plt.imshow(img)
plt.show()

return 0
```

```
def save_image(img, name):
img = np.flipud(img)
imsave(name, img)
return 0


def create_hermite_spline():

h = 1024
w = 1024

black = np.array([0, 0, 0], np.uint8)
green = np.array([0, 255, 0], np.uint8)
red = np.array([255, 0, 0], np.uint8)

img = create_image(h, w, black)

N = 10 # points number

x = np.random.randint(50, w-50, 10)
x = np.sort(x)
y = np.random.randint(50, h-50, 10)
d = 20*np.random.rand(10)-10

for i in range(N-1):
m = np.array([[1, x[i], x[i]**2, x[i]**3], [1, x[i+1], x[i+1]**2,
    x[i+1]**3], [0, 1, 2*x[i], 3*x[i]**2], [0, 1, 2*x[i+1],
    3*x[i+1]**2]])
b = np.array([y[i], y[i+1], d[i], d[i+1]]).T
a = np.linalg.inv(m).dot(b)
for j in range(x[i], x[i+1]+1):
y_j = a[0] + a[1]*j + a[2]*j**2 + a[3]*j**3
y_j_1 = a[0] + a[1]*(j+1) + a[2]*(j+1)**2 + a[3]*(j+1)**3
draw_line(img, j, y_j, j+1, y_j_1, green)

set_color(img, x, y, red)
set_color(img, x+1, y, red)
set_color(img, x-1, y, red)
set_color(img, x, y+1, red)
set_color(img, x, y-1, red)

img = np.transpose(img, axes = (1,0,2))
```

```
#save_image(img, 'hermite_spline.tga')
show_image(img)


if __name__ == '__main__':
   create_hermite_spline()
```

7. Разные примеры

7.1. Имитирование капель дождя на поверхности

Базовый класс анимации animation имеет дело с анимационной частью. Он обеспечивает основу, на которой построены функции анимации. Для этого есть два основных интерфейса: FuncAnimation - создает анимацию, многократно вызывая функцию func. ArtistAnimation - анимация с использованием фиксированного набора объектов Artist.

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation, PillowWriter

# Fixing random state for reproducibility
np.random.seed(19680801)
# Create new Figure and an Axes which fills it.
fig = plt.figure(figsize=(7, 7))
ax = fig.add_axes([0, 0, 1, 1], frameon=False)
ax.set_xlim(0, 1), ax.set_xticks([])
ax.set_ylim(0, 1), ax.set_yticks([])

# Create rain data
n_drops = 50
rain_drops = np.zeros(n_drops, dtype=[('position', float, 2),
('size',    float, 1),
('growth',  float, 1),
('color',   float, 4)])

# Initialize the raindrops in random positions and with
# random growth rates.
rain_drops['position'] = np.random.uniform(0, 1, (n_drops, 2))
rain_drops['growth'] = np.random.uniform(50, 200, n_drops)

# Construct the scatter which we will update during animation
```

```
# as the raindrops develop.
scat = ax.scatter(rain_drops['position'][:, 0],
    rain_drops['position'][:, 1],
s=rain_drops['size'], lw=0.5, edgecolors=rain_drops['color'],
facecolors='none')


def update(frame_number):
# Get an index which we can use to re-spawn the oldest raindrop.
current_index = frame_number % n_drops

# Make all colors more transparent as time progresses.
rain_drops['color'][:, 3] -= 1.0/len(rain_drops)
rain_drops['color'][:, 3] = np.clip(rain_drops['color'][:, 3], 0,
    1)

# Make all circles bigger.
rain_drops['size'] += rain_drops['growth']

# Pick a new position for oldest rain drop, resetting its size,
# color and growth factor.
rain_drops['position'][current_index] = np.random.uniform(0, 1, 2)
rain_drops['size'][current_index] = 5
rain_drops['color'][current_index] = (0, 0, 0, 1)
rain_drops['growth'][current_index] = np.random.uniform(50, 200)

# Update the scatter collection, with the new colors, sizes and
    positions.
scat.set_edgecolors(rain_drops['color'])
scat.set_sizes(rain_drops['size'])
scat.set_offsets(rain_drops['position'])


# Construct the animation, using the update function as the
    animation director.
animation = FuncAnimation(fig, update, interval=10)
#plt.show()
# Construct the animation, using the update function as the
    animation director.
animation = FuncAnimation(fig, update, interval=10)
#plt.show()
animation.save('file.gif',writer=PillowWriter(fps=25))
```

7.2. Анимация функции sin(x)

Воспользуемся интерфейсом FuncAnimation для анимации графика синуса.

В строках (7–9) мы просто создаем окно фигуры с единственной осью на рисунке. Затем мы создаем объект с пустой строкой, который, по сути, и должен быть изменен в анимации. В строках (11–13) мы создаем функцию init, которая будет запускать анимацию. Функция init инициализирует данные, а также устанавливает пределы оси. В строках (14–18) мы, наконец, определяем функцию анимации, которая принимает номер кадра (i) в качестве параметра и создает синусоидальную волну (или любую другую анимацию), которая изменяется в зависимости от значения i. В строке 20 мы создаем фактический объект анимации. Параметр blit гарантирует, что перерисовываются только те части графика, которые были изменены.

```
import numpy as np
from matplotlib import pyplot as plt
from matplotlib.animation import FuncAnimation, PillowWriter
plt.style.use('seaborn-pastel')
fig = plt.figure()
ax = plt.axes(xlim=(0, 4), ylim=(-2, 2))
line, = ax.plot([], [], lw=3)

def init():
   line.set_data([], [])
   return line,
def animate(i):
   x = np.linspace(0, 4, 1000)
   y = np.sin(2 * np.pi * (x - 0.01 * i))
   line.set_data(x, y)
   return line,

anim = FuncAnimation(fig, animate, init_func=init, frames=200,
    interval=20, blit=True)
anim.save('file2.gif',writer=PillowWriter(fps=25))
```

7.3. Еще один вариант рисования графика sin(x))

```
import numpy as np
import matplotlib.pyplot as plt
```

```
from matplotlib.animation import FuncAnimation, PillowWriter

fig, ax = plt.subplots()

x = np.arange(0, 2*np.pi, 0.01)
line, = ax.plot(x, np.sin(x))

def init(): # only required for blitting to give a clean slate.
line.set_ydata([np.nan] * len(x))
return line,

def animate(i):
line.set_ydata(np.sin(x + i / 100)) # update the data.
return line,

ani = animation.FuncAnimation(
fig, animate, init_func=init, interval=2, blit=True,
    save_count=50)

# To save the animation, use e.g.
#
ani.save("movie.mp4")

# To save the animation, use e.g.
#
ani.save("movie.mp4")
## or
## from matplotlib.animation import FFMpegWriter
# writer = FFMpegWriter(fps=15, metadata=dict(artist='Me'),
    bitrate=1800)
# ani.save("movie.mp4", writer=writer)

plt.show()
```

Список литературы

[1] Matplotlib: Visualization with Python. – URL: https://matplotlib.org/ (дата обращения: 01.05.2025).

[2] Numpy: The fundamental package for scientific computing with Python. – URL: https://numpy.org/ (дата обращения: 01.05.2025).

[3] Foley J.D. Computer Graphics: Principles and Practice 3rd Edition. / J.D. Foley. – M.: Addison-Wesley, 2014. – 1263 с.

[4] Marschner S. Fundamentals of computer graphics 3rd edition / S. Marschner, P. Shirley. – M.: Corporate Taylor & Francis Group, 2009. – 804 с.

[5] Роджерс Д. Математические основы машинной графики / Д. Роджерс, Дж. Абрамс. – М.: Мир, 2001. – 604 с.

Методическое пособие

Долгов Дмитрий Александрович

Компьютерная графика