ГУАП

КАФЕДРА № 42

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ _____

ПРЕПОДАВАТЕЛЬ

канд. техн. наук, доцент		А.В. Аграновский
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №7

Исследование артефактов аффинных преобразований

по курсу: КОМПЬЮТЕРНАЯ ГРАФИКА

РАБОТУ ВЫПОЛНИЛ

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

Санкт-Петербург 2024

Содержание

1. Цель работы:

Изучение артефактов, возникающих при аффинных преобразованиях.

2. Задание:

Написать на языке программирования высокого уровня программу, которая выполняет поворот изображения, используя аффинные преобразования, а также использует алгоритм Оуэна и Македона.

Для этого необходимо:

1. Перевести представленную формулу Оуэна и Македона в однородные координаты.
2. Выбрать для последующего преобразования изображение, использование которого не имеет ограничений, связанных с авторским правом. Рекомендуется использовать изображения, позволяющие увидеть артефакты, возникающие при осуществлении поворота.
3. Выбрать вариант задания в соответствии со своим порядковым номером в списке группы: осуществить поворот изображения на угол в градусах, равный сумме цифр порядкового номера; затем на данный угол, умноженный на три и исходный угол, умноженный на пять. Если артефакты аффинного поворота не проявляются, измените углы поворота, умножая исходный угол не на 3 и 5, а на 0,3 и 0,5 соответственно.
4. Написать программу на любом языке высокого уровня, реализующую преобразование изображения в соответствии со своим вариантом задания с использованием матрицы аффинного преобразования (поворота) и комбинации матриц Оуэна и Македона.
5. Ввод исходных и вывод преобразованных изображений производить в одинаковом формате (предпочтительно, BMP).

Вариант 17 – углы 8, 24, 40 градусов.

3. Теоретические сведения:

3.1 Аффинные преобразования

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 R^n, 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 R^2 в R^3. А вот если из прямоугольника у нас получился другой прямоугольник, то все хорошо, мы отобразили исходное пространство в себя. Формально это описывается так: «преобразование f отображает пространство R^n в R^n». Если записать с помощью формул: $f: R^n \to R^n$.

Скрещивающиеся 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:

$$\begin{cases} x' = \alpha x + \beta y + \lambda \\ y' = \gamma x + \delta y + \mu \end{cases}$$

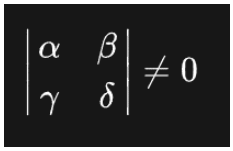
Рисунок 1.1 - Система координат

При этом, числа $\alpha, \beta, \gamma, \mu$ должны образовывать невырожденную матрицу:

$$\begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix}$$

Рисунок 1.2 – Невырожденная матрица

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



$$\begin{vmatrix} \alpha & \beta \\ \gamma & \delta \end{vmatrix} \neq 0$$

Рисунок 1.3 – Определитель матрицы равен 0

Можно записать и в более общем в виде.

Аффинное преобразование $f: R^n \rightarrow R^n$- преобразование вида $f(x) = Mx + v$, где M- обратимая матрица, а $v \in R^n$. В данном случае x, само собой, n-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. Методы улучшения растровых изображений при реализации аффинных преобразований

В процессе практического применения аффинных преобразований для обработки растровых графических объектов возникает сложность, связанная с неоднозначностью определения значений пикселей после преобразования.

Например, при использовании матрицы поворота к фотографии без дополнительной обработки могут появиться пиксели с нулевым значением. Это происходит из-за того, что в процессе вычисления по известной матрице поворота эти пиксели не получают никаких значений и окрашиваются в цвет фона. Очевидно, что на светлых участках изображения такие артефакты заметны и значительно ухудшают восприятие преобразованного изображения.

Аналогичные проблемы могут возникать и при других аффинных преобразованиях. Неопределённые пиксели могут быть закрашены любым цветом фона или интерполированы каким-либо образом, но их присутствие всё равно является дефектом преобразованного изображения.

Для устранения эффекта, возникающего при повороте, был предложен алгоритм Оуэна и Македона. Этот алгоритм позволяет избежать появления «дырок» без интерполяции. Суть алгоритма заключается в повороте цифровых изображений путём последовательного сдвига строк, столбцов и затем снова строк изображения.

Вариант разложения матрицы Оуэна и Македона:

$$R(\alpha) = \begin{bmatrix} cos\ \alpha & -\sin\alpha \\ sin\ \alpha & \cos\alpha \end{bmatrix} = \begin{bmatrix} 1 & -tg\ \alpha/2 \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ \sin\alpha & 1 \end{bmatrix} \times \begin{bmatrix} 1 & -tg\ \alpha/2 \\ 0 & 1 \end{bmatrix}, (1)$$

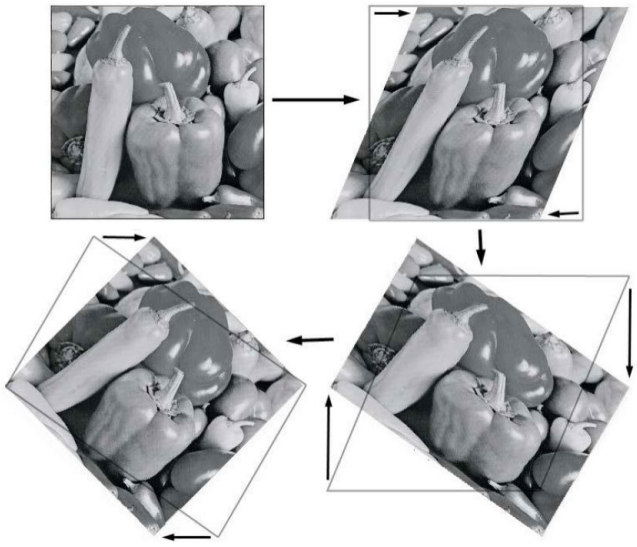


Рисунок 2 – Графическая интерпретация алгоритма Оуэна и Македона

3.3 Однородные координаты

Однородные координаты – это математический механизм, связанный с определением положения точек в пространстве. Привычный аппарат декартовых координат, не подходит для решения некоторых важных задач в силу следующих соображений:

В декартовых координатах невозможно описать бесконечно удаленную точку. А многие математические и геометрические концепции значительно упрощаются, если в них используется понятие бесконечности. Например, "бесконечно удаленный источник света".

С точки зрения алгебраических операций, декартовы координаты не позволяют провести различия межу точками и векторами в пространстве. Действительно, (1,-2,5) - это направление или точка?

Невозможно использовать унифицированный механизм работы с матрицами для выражения преобразований точек. С помощью матриц 3x3 можно описать вращение и масштабирование, однако описать смещение (x=x+a) нельзя.

Аналогично, декартовы координаты не позволяют использовать матричную запись для задания перспективного преобразования (проекции) точек.

Как видно из вышесказанного двумерные преобразования имеют различный вид. Сдвиг реализуется сложением, а масштабирование и поворот - умножением. Это различие затрудняет формирование суммарного преобразования и устраняется использованием двумерных однородных координат точки, имеющих вид: (x,y,w). Здесь w - произвольный множитель, не равный 0. Число w так же называется масштабным множителем.

В иной форме записи преобразование из однородных координат в обычные: (x/w,y/w,w/w)->(X,Y,1). Однородные координаты для 2D случая можно представить как промасштабированные с коэффициентом w значения двумерных координат, расположенные в плоскости с Z = w.

Два набора чисел, представляющих однородные координаты, соответствуют одной точке Декартового пространства если они могут быть получены один из другого умножением на некоторый множитель. – (2,5,3) и (4,10,6) представляют одну точку.

В силу произвольности значения w в однородных координатах не существует единственного представления точки, заданной в декартовых координатах.

Как минимум одно число из тройки должно быть отлично от нуля – точка (0,0,0) не будет определена.

Если w != 0, то деление на неё даст Декартовы координаты (x/w,y/w,1).

Если w=0, то точка находится в бесконечности.

В общем случае осуществляется переход от n-мерного пространства к (n+1)-мерному. Обратное преобразование называется проекцией однородных координат.

4. Описание алгоритма решения поставленной задачи

1. Импорт необходимых библиотек.
2. Функция rotate_image_with_artifacts:

Входные параметры: путь к исходному изображению (input_path), путь для сохранения выходного изображения (output_path), угол вращения (angle).

Чтение изображения: открывается изображение и получаются его размеры.

Конвертация угла: угол вращения преобразуется из градусов в радианы для математических расчетов.

Создание нового пустого изображения: создается новое изображение такой же ширины и высоты, как и исходное, все пиксели изначально устанавливаются в None.

Определение центра изображения: вычисляется центр изображения для корректного вращения.

Цикл по каждому пикселю изображения: для каждого пикселя осуществляется:

Смещение пикселя, чтобы центр изображения находился в координатах (0, 0).

Применение матрицы вращения к смещенным координатам, чтобы получить новые координаты пикселя.

Если новые координаты пикселя находятся в пределах границ изображения, копирование цвета из исходного изображения в новое изображение.

Сохранение выходного изображения: новое изображение с артефактами сохраняется по указанному пути.

3. Функция apply_transformation:

Входные параметры: входное изображение, новые размеры, матрица преобразования и центр.

Создание черного изображения для перемещения.

Все пиксели перемещаются в центре новосозданного изображения, чтобы их можно было дальше преобразовать.

Для каждого пикселя нового изображения вычисляются новые координаты, используя переданную матрицу преобразования.

Мы создаём матрицы преобразования, которые действуют на точки в однородных координатах.

Точки представляются как векторы, состоящие из трех компонентов: $(x,y,1)(x,y,1)$. Число 1 в третьей компоненте представляет однородные координаты. Здесь же мы смещаем координаты, чтобы центр изображения оказался в начале координат, что позволяет корректно применять преобразования.

Если новые координаты соответствуют границам изображения, цвет пикселя устанавливается.

4. Функция crop_image:

Входные параметры: входное изображение и координаты (x, y) для верхнего левого угла прямоугольной области обрезки, а также ширина и высота обрезки.

Использует метод crop библиотеки Pillow для возврата обрезанного изображения.

5. Функция rotate_image_without_artifacts:

Входные параметры: входное изображение, угол поворота и базовое имя файла для сохранения.

Увеличиваются размеры изображения, чтобы избежать обрезки на краях.

Создаются две матрицы преобразования, используемые для двух операций поворота, и последовательно применяются к изображению.

Обрезка каждого промежуточного изображения до оригинальных размеров для улучшения конечного результата с помощью crop_image.

Выходное обрезанное изображение сохраняется с уникальным именем.

6. Использование:

Задаются параметры, после чего функция rotate_image_with_artifacts вызывается для вращения изображений с определенным углом, и конечное изображение сохраняется с соответствующим именем.

5. Выбор языка программирования и библиотек:

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) Pillow (from PIL import Image):

Библиотека Pyhton, которая используется для чтения, обработки и сохранения изображений.

3) Math (import math):

Библиотека Pythin, которая используется для математических функций, таких как преобразование градусов в радианы.

6. Скриншоты, иллюстрирующие результаты работы программы:

Рисунок 3.1 – Исходное изображение

Рисунок 3.2 – Изображение повёрнуто на 8 градусов аффинными преобразованиями

Рисунок 3.3 – Изображение повёрнуто на 24 градуса аффинными преобразованиями

Рисунок 3.4 – Изображение повёрнуто на 40 градусов аффинными преобразованиями

Рисунок 3.5 – Изображение повёрнуто на 8 градусов алгоритмом Оуэна и Македона – шаг 1

Рисунок 3.6 – Изображение повёрнуто на 8 градусов алгоритмом Оуэна и Македона – шаг 2

Рисунок 3.7 – Изображение повёрнуто на 8 градусов алгоритмом Оуэна и Македона – шаг 3

Рисунок 3.8 – Изображение повёрнуто на 24 градуса алгоритмом Оуэна и Македона – шаг 1

Рисунок 3.9 – Изображение повёрнуто на 24 градуса алгоритмом Оуэна и Македона – шаг 2

Рисунок 3.9 – Изображение повёрнуто на 24 градуса алгоритмом Оуэна и Македона – шаг 3

Рисунок 3.10 – Изображение повёрнуто на 40 градусов алгоритмом Оуэна и Македона – шаг 1

Рисунок 3.11 – Изображение повёрнуто на 40 градусов алгоритмом Оуэна и Македона – шаг 2

Рисунок 3.12 – Изображение повёрнуто на 40 градусов алгоритмом Оуэна и Македона – шаг 3

7. Вывод:

В данной работе мы подробно рассмотрели поворот изображения с помощью аффинных преобразований, а также артефакты, которые появляются в процессе. В ходе выполнения лабораторной мы изучили алгоритм Оуэна и Македона, который путём трёх последовательных сдвигов строк, столбцов и затем снова строк картинки позволяет повернуть изображение без артефактов.

В результате выполнения лабораторной работы удалось достигнуть поставленной цели – изучить артефакты, возникающие при аффинных преобразованиях.

В процессе работы с изображениями, преобразованными с использованием матрицы вращения, становится заметным появление пробелов — пикселей, цвет которых совпадает с цветом фона. Появление этих артефактов связано с тем, что после преобразования координаты точек изображения могут стать нецелыми, что не соответствует целочисленным координатам пикселей в сетке.

Из результатов работы программы становится очевидным, что с увеличением угла поворота количество неопределённых пикселей также увеличивается. Это связано с тем, что при увеличении угла увеличивается смещение пикселей, что повышает вероятность появления зазоров.

Один из способов решения этой проблемы — заполнение пробелов цветом, рассчитанным на основе соседних пикселей. Минус подхода в том, что это может привести к погрешностям или эффекту зазубренности вдоль краёв изображения, а мы ведь максимально стараемся избегать всяких артефактов.

В данной работе рассматривается эффективный метод, позволяющий устранить подобные искажения — алгоритм Оуэна и Македона, основанный на разбиении матрицы вращения на произведение трёх матриц сдвигов.

Последовательное умножение на эти матрицы приводит к искажению вдоль осей: сначала вдоль оси Ox, затем вдоль оси Oy, и снова вдоль Ox. Такой подход позволяет выполнять все преобразования в рамках сетки пикселей, что значительно снижает вероятность возникновения неопределённости в координатах.

По результатам работы программы можно заметить, что метод правда эффективен. Кроме того, угол поворота изображения оказался идентичен углу поворота изображения при использовании стандартной матрицы вращения. Свойства аффинных преобразований дают понять: последовательность нескольких аффинных операций равна одному аффинному преобразованию.

Алгоритм был изучен на практике, благодаря написанию программы, которая использует статические массивы в качестве матриц аффинного преобразования.

В ходе лабораторной были получены практические навыки работы с библиотеками языка Python – Pillow, Math и Numpy. Основные функции, с которыми происходила работа:

Image.open() – открывает изображение по заданному пути;

.save() – сохраняет изображение по заданному пути;

.putpixel() – позволяет изменять цвет пикселя. Для этого нужно указать координаты пикселя (в виде кортежа из двух элементов) и цвет;

math.radians() – конвертирует заданный угол из градусов в радианы.

В ходе работы возникли проблемы. Одна из основных – неконтролируемый поворот изображения, то есть не относительно центра, из-за чего картинка оказывалась непонятно где и была странно обрезана. Решением стало перемещение центра изображения в начало координат, которое было упущено по невнимательности.

Также присутствует проблема некоторой медлительности программы из-за использования метода Image.putpixel(), так что для более масштабных изображений лучше использовать другие функции, например ImageDraw. В ходе лабораторной работы скорость полученной программы меня вполне устраивает.

Таким образом, в процессе лабораторной работы был изучен и применён на практике алгоритм предотвращения артефактов, который может пригодится в дальнейшем для решения других задач в сфере информационных технологий, науке о данных и других областях, где используется деформация изображений или вычислительное моделирование.

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. Библиотека Pillow в Python – URL: https://docs-python.ru/packages/biblioteka-pillow-python/ (дата обращения 01.12.2024)
4. Morgan D. Цифровая обработка сигналов и изображений – URL: https://www.bsu.by/upload/page/353593.pdf (дата обращения 27.09.2024)
5. Выжимка из документации Numpy: функции, методы и примеры. – URL: https://pythonru.com/biblioteki/rukovodstvo-po-ispolzovaniju-python-biblioteki-numpy (дата обращения 01.12.2024)
6. Owen Ch.B., Makedon F. High quality alias free image rotation // Proceeding of 30th Asilomar Conference on Signals, Systems, and Computers Pacific Grove, California, November 2—6, 1996. URL: https://digitalcommons.dartmouth.edu/cgi/viewcontent.cgi?article=5030&context=facoa (дата посещения 01.12.2024).

ПРИЛОЖЕНИЕ А

Листинг Программы

```
import numpy as np
from PIL import Image
import math

def rotate_image_with_artifacts(input_path, output_path, angle):
    # Открываем исходное изображение
    input_image = Image.open(input_path)
    width, height = input_image.size

    # Конвертируем угол в радианы
    angle_rad = math.radians(angle)

    # Создаем новое пустое изображение (начальное состояние пикселей - None)
    output_image = Image.new("RGB", (width, height))

    # Центр изображения
    cx, cy = width // 2, height // 2

    for y in range(height):
        for x in range(width):
            # Перемещаем координаты так, чтобы центр изображения был в (0,0)
            x_shifted = x - cx
            y_shifted = y - cy

            # Применяем матрицу поворота
            new_x = round(x_shifted * math.cos(angle_rad) - y_shifted *
math.sin(angle_rad))
            new_y = round(x_shifted * math.sin(angle_rad) + y_shifted *
math.cos(angle_rad))

            # Перемещаем координаты обратно
            new_x += cx
            new_y += cy

            # Копируем пиксель, если он попадает в границы изображения
            if 0 <= new_x < width and 0 <= new_y < height:
                output_image.putpixel((new_x, new_y),
input_image.getpixel((x, y)))

    # Сохраняем изображение с артефактами
    output_image.save(output_path)
    print(f"Изображение сохранено: {output_path}")

def apply_transformation(input_image, width, height, matrix, center):
    size = input_image.size
    new_size = (width, height)

    # Создаем черное изображение для перемещения
    output = Image.new('RGB', new_size, (0, 0, 0))



    # Перемещение изображения в центр области
    for y in range(size[1]):
        for x in range(size[0]):
            output.putpixel((x + new_size[0] // 2 - size[0] // 2, y +
new_size[1] // 2 - size[1] // 2), input_image.getpixel((x, y)))

    transformed = Image.new('RGB', new_size, (0, 0, 0))
```

```
    for y in range(new_size[1]):
        for x in range(new_size[0]):
            # Преобразование координат в систему с центром
            pos = np.array([x - new_size[0] / 2.0, y - new_size[1] / 2.0, 1])
            result = np.zeros(3)

            # Применение матрицы преобразования
            for i in range(3):
                result[i] = sum(matrix[i][j] * pos[j] for j in range(3))

            # Возвращение в исходную систему координат
            new_x = int(result[0] + new_size[0] / 2)
            new_y = int(result[1] + new_size[1] / 2)

            # Проверка границ изображения
            if 0 <= new_x < new_size[0] and 0 <= new_y < new_size[1]:
                transformed.putpixel((new_x, new_y), output.getpixel((x, y)))

    return transformed

def crop_image(input_image, x, y, width, height):
    """ Обрезает изображение до заданных размеров """
    return input_image.crop((x, y, x + width, y + height))

def rotate_image_without_artifacts(input_image, angle, base_filename):
    size = input_image.size
    # Размеры увеличенной области для преобразований
    width = size[0] * 2
    height = size[1] * 2

    temp1 = Image.new('RGB', (width, height), (0, 0, 0))
    temp2 = Image.new('RGB', (width, height), (0, 0, 0))
    output = Image.new('RGB', (width, height), (0, 0, 0))

    rad = np.deg2rad(angle)
    center = (size[0] / 2.0, size[1] / 2.0)

    # Первая матрица
    shift_matrix1 = np.array([
        [1, -np.tan(rad) / 2, 0],
        [0, 1, 0],
        [0, 0, 1]
    ])

    # Вторая матрица
    shift_matrix2 = np.array([
        [1, 0, 0],
        [np.sin(rad), 1, 0],
        [0, 0, 1]
    ])

    # Применение первой матрицы
    temp1 = apply_transformation(input_image, width, height, shift_matrix1,
center)
    temp1_cropped = crop_image(temp1, size[0] / 2.0, size[1] / 2.0, size[0],
size[1])
    temp1_cropped.save(base_filename + "_crop_step1.bmp")

    # Применение второй матрицы
    temp2 = apply_transformation(temp1, width, height, shift_matrix2, center)
    temp2_cropped = crop_image(temp2, size[0] / 2.0, size[1] / 2.0, size[0],
size[1])
    temp2_cropped.save(base_filename + "_crop_step2.bmp")
```

```
    # Применение первой матрицы
    output = apply_transformation(temp2, width, height, shift_matrix1,
center)
    output_cropped = crop_image(output, size[0] / 2.0, size[1] / 2.0,
size[0], size[1])
    output_cropped.save(base_filename + "_crop_step3.bmp")

    # Обрезка окончательного изображения
    #output_cropped = crop_image(output, size[0] / 2.0, size[1] / 2.0,
size[0], size[1])
    #output_cropped.save(base_filename + "_cropped.bmp")

    return output_cropped


# Задаём параметры:
a, a3, a5=8, 8*3, 8*5
x=a
# Поворот через аффинные, с артефактами
rotate_image_with_artifacts("розовый-танк.bmp", f"output_affin_{x}.bmp", x)

# Поворот через Оуэна-Македона, без артефактов
input_image = Image.open("розовый-танк.bmp")
rotated_image = rotate_image_without_artifacts(input_image, x,
f"rotated_{x}")
```