**`Минобрнауки России**

**Юго-Западный государственный университет**

**Кафедра программной инженерии**

ЛАБОРАТОРНАЯ РАБОТА

по дисциплине «Компьютерная графика»

по теме:

«Программа для формирования изображений»

Выполнили: студенты группы ПО-31б

Василевский В.Я.

Пирогов Д.Ю.

Проверил: старший преподаватель

Ефремов В.В.

Курск 2025 г.

**Вариант – 4**

**Цель работы**– совершенствование базовых навыков обработки растровых изображений с помощью алгоритмических языков программирования.

**Задание:**

1. Разработать алгоритм и программу для чтения из файла, преобразования и записи в файл графической информации по варианту. Программа включает в себя: - создание нового изображения; - чтение изображения из файла; - добавление фрагмента заданной формы из заданной позиции изображения, загруженного из файла в заданную позицию созданного нового изображения попиксельно; - добавление осей системы координат и графика функции; - сохранение созданного изображения в файл на диске.

2. Ответить на контрольные вопросы.

3. Оформить отчёт.

Вариант:



**Ссылка на репозиторий:**<https://github.com/Vados-rb26/CompGraphLabs>

**Программа на python:**

import tkinter as tk

from tkinter import filedialog, messagebox, ttk

from PIL import Image, ImageTk, ImageDraw

import math

class ImageProcessor:

def \_\_init\_\_(self, root):

self.root = root

self.root.title("Лабораторная работа №2 - Вариант 4")

self.root.geometry("1000x700")

# Переменные

self.source\_image = None

self.result\_image = None

# Создание интерфейса

self.create\_interface()

def create\_interface(self):

# Главный фрейм

main\_frame = ttk.Frame(self.root)

main\_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

# Фрейм для управления

control\_frame = ttk.LabelFrame(main\_frame, text="Управление - Вариант 4", padding=10)

control\_frame.pack(fill=tk.X, pady=(0, 10))

# Информация о варианте

info\_label = ttk.Label(control\_frame,

text="Фрагмент: КВАДРАТ из ЛЕВОГО НИЖНЕГО угла → в ПРАВЫЙ НИЖНИЙ угол | Функция: x\*sin(x)",

font=("Arial", 10, "bold"))

info\_label.pack(pady=5)

# Поля ввода параметров

params\_frame = ttk.Frame(control\_frame)

params\_frame.pack(fill=tk.X, pady=5)

ttk.Label(params\_frame, text="Ширина нового изображения:").grid(row=0, column=0, padx=5, sticky="w")

self.width\_entry = ttk.Entry(params\_frame, width=10)

self.width\_entry.insert(0, "600")

self.width\_entry.grid(row=0, column=1, padx=5)

ttk.Label(params\_frame, text="Высота нового изображения:").grid(row=0, column=2, padx=5, sticky="w")

self.height\_entry = ttk.Entry(params\_frame, width=10)

self.height\_entry.insert(0, "600")

self.height\_entry.grid(row=0, column=3, padx=5)

ttk.Label(params\_frame, text="Сторона квадрата:").grid(row=1, column=0, padx=5, sticky="w")

self.square\_size\_entry = ttk.Entry(params\_frame, width=10)

self.square\_size\_entry.insert(0, "150")

self.square\_size\_entry.grid(row=1, column=1, padx=5)

# Кнопки управления

buttons\_frame = ttk.Frame(control\_frame)

buttons\_frame.pack(fill=tk.X, pady=5)

ttk.Button(buttons\_frame, text="1. Создать новое изображение",

command=self.create\_new\_image).grid(row=0, column=0, padx=5, pady=2)

ttk.Button(buttons\_frame, text="2. Открыть исходное изображение",

command=self.open\_source\_image).grid(row=0, column=1, padx=5, pady=2)

ttk.Button(buttons\_frame, text="3. Перенести квадратный фрагмент",

command=self.transfer\_square\_fragment).grid(row=0, column=2, padx=5, pady=2)

ttk.Button(buttons\_frame, text="4. Нарисовать оси и график x\*sin(x)",

command=self.draw\_axes\_graph).grid(row=0, column=3, padx=5, pady=2)

ttk.Button(buttons\_frame, text="5. Сохранить результат в BMP",

command=self.save\_result).grid(row=0, column=4, padx=5, pady=2)

# Области для изображений

images\_frame = ttk.LabelFrame(main\_frame, text="Изображения", padding=10)

images\_frame.pack(fill=tk.BOTH, expand=True)

# Фрейм для исходного изображения

source\_frame = ttk.Frame(images\_frame)

source\_frame.pack(side=tk.LEFT, fill=tk.BOTH, expand=True, padx=5)

ttk.Label(source\_frame, text="Исходное изображение",

font=("Arial", 10, "bold")).pack(pady=5)

# Canvas для исходного изображения с прокруткой

source\_canvas\_frame = ttk.Frame(source\_frame)

source\_canvas\_frame.pack(fill=tk.BOTH, expand=True)

self.source\_canvas = tk.Canvas(source\_canvas\_frame, bg="lightgray", width=400, height=400)

source\_vscroll = ttk.Scrollbar(source\_canvas\_frame, orient=tk.VERTICAL, command=self.source\_canvas.yview)

source\_hscroll = ttk.Scrollbar(source\_canvas\_frame, orient=tk.HORIZONTAL, command=self.source\_canvas.xview)

self.source\_canvas.configure(yscrollcommand=source\_vscroll.set, xscrollcommand=source\_hscroll.set)

source\_vscroll.pack(side=tk.RIGHT, fill=tk.Y)

source\_hscroll.pack(side=tk.BOTTOM, fill=tk.X)

self.source\_canvas.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

# Фрейм для результата

result\_frame = ttk.Frame(images\_frame)

result\_frame.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True, padx=5)

ttk.Label(result\_frame, text="Результат",

font=("Arial", 10, "bold")).pack(pady=5)

# Canvas для результата с прокруткой

result\_canvas\_frame = ttk.Frame(result\_frame)

result\_canvas\_frame.pack(fill=tk.BOTH, expand=True)

self.result\_canvas = tk.Canvas(result\_canvas\_frame, bg="lightgray", width=400, height=400)

result\_vscroll = ttk.Scrollbar(result\_canvas\_frame, orient=tk.VERTICAL, command=self.result\_canvas.yview)

result\_hscroll = ttk.Scrollbar(result\_canvas\_frame, orient=tk.HORIZONTAL, command=self.result\_canvas.xview)

self.result\_canvas.configure(yscrollcommand=result\_vscroll.set, xscrollcommand=result\_hscroll.set)

result\_vscroll.pack(side=tk.RIGHT, fill=tk.Y)

result\_hscroll.pack(side=tk.BOTTOM, fill=tk.X)

self.result\_canvas.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

# Информационная панель

info\_frame = ttk.LabelFrame(main\_frame, text="Информация", padding=5)

info\_frame.pack(fill=tk.X, pady=(10, 0))

self.info\_label = ttk.Label(info\_frame, text="Готов к работе. Создайте новое изображение и загрузите исходное.")

self.info\_label.pack()

def create\_new\_image(self):

try:

width = int(self.width\_entry.get())

height = int(self.height\_entry.get())

if width <= 0 or height <= 0:

messagebox.showerror("Ошибка", "Размеры изображения должны быть положительными числами")

return

self.result\_image = Image.new('RGB', (width, height), 'white')

self.update\_result\_display()

self.info\_label.config(text=f"Создано новое изображение {width}x{height} пикселей")

except ValueError:

messagebox.showerror("Ошибка", "Введите корректные размеры изображения")

def open\_source\_image(self):

file\_path = filedialog.askopenfilename(

title="Выберите исходное изображение",

filetypes=[("Image files", "\*.jpg \*.jpeg \*.png \*.bmp \*.gif \*.tiff")]

)

if file\_path:

try:

self.source\_image = Image.open(file\_path).convert('RGB')

self.update\_source\_display()

self.info\_label.config(text=f"Загружено изображение: {file\_path}")

except Exception as e:

messagebox.showerror("Ошибка", f"Не удалось загрузить изображение: {str(e)}")

def transfer\_square\_fragment(self):

if self.source\_image is None or self.result\_image is None:

messagebox.showerror("Ошибка", "Сначала создайте новое изображение и загрузите исходное")

return

try:

square\_size = int(self.square\_size\_entry.get())

if square\_size <= 0:

messagebox.showerror("Ошибка", "Сторона квадрата должна быть положительным числом")

return

src\_width, src\_height = self.source\_image.size

res\_width, res\_height = self.result\_image.size

# Проверяем, что квадрат помещается в исходное изображение

if square\_size > min(src\_width, src\_height):

messagebox.showerror("Ошибка", "Квадрат слишком большой для исходного изображения")

return

# ЛЕВЫЙ НИЖНИЙ угол исходного изображения

src\_x1 = 0 # левый верхний угол квадрата (X) - ЛЕВЫЙ край

src\_y1 = src\_height - square\_size # левый верхний угол квадрата (Y) - НИЖНИЙ край

src\_x2 = square\_size - 1 # правый нижний угол квадрата (X)

src\_y2 = src\_height - 1 # правый нижний угол квадрата (Y)

# Правый нижний угол нового изображения

res\_x1 = res\_width - square\_size # левый верхний угол квадрата (X)

res\_y1 = res\_height - square\_size # левый верхний угол квадрата (Y)

res\_x2 = res\_width - 1 # правый нижний угол квадрата (X)

res\_y2 = res\_height - 1 # правый нижний угол квадрата (Y)

# Создаем копию результата для работы

result\_img = self.result\_image.copy()

# Копирование квадратного фрагмента

for y in range(square\_size):

for x in range(square\_size):

# Координаты в исходном изображении (ЛЕВЫЙ нижний угол)

src\_x = src\_x1 + x

src\_y = src\_y1 + y

# Координаты в новом изображении (ПРАВЫЙ нижний угол)

res\_x = res\_x1 + x

res\_y = res\_y1 + y

# Проверяем границы

if (0 <= src\_x < src\_width and 0 <= src\_y < src\_height and

0 <= res\_x < res\_width and 0 <= res\_y < res\_height):

pixel = self.source\_image.getpixel((src\_x, src\_y))

result\_img.putpixel((res\_x, res\_y), pixel)

# Рисуем красную границу вокруг квадрата

draw = ImageDraw.Draw(result\_img)

draw.rectangle([res\_x1, res\_y1, res\_x2, res\_y2], outline=(255, 0, 0), width=2)

# Также рисуем границу на исходном изображении для наглядности

self.source\_image\_with\_border = self.source\_image.copy()

source\_draw = ImageDraw.Draw(self.source\_image\_with\_border)

source\_draw.rectangle([src\_x1, src\_y1, src\_x2, src\_y2], outline=(255, 0, 0), width=2)

self.result\_image = result\_img

self.update\_result\_display()

self.update\_source\_display() # Обновляем исходное изображение с границей

self.info\_label.config(text=f"Квадратный фрагмент {square\_size}x{square\_size} перенесен из левого нижнего в правый нижний угол")

except ValueError:

messagebox.showerror("Ошибка", "Введите корректный размер стороны квадрата")

def draw\_axes\_graph(self):

if self.result\_image is None:

messagebox.showerror("Ошибка", "Сначала создайте новое изображение")

return

result\_img = self.result\_image.copy()

draw = ImageDraw.Draw(result\_img)

width, height = result\_img.size

# Рисование осей координат

center\_x = width // 4 # Смещаем график левее

center\_y = height // 2

# Ось X

draw.line([(50, center\_y), (width-50, center\_y)], fill=(0, 0, 0), width=2)

# Ось Y

draw.line([(center\_x, 50), (center\_x, height-50)], fill=(0, 0, 0), width=2)

# Стрелки осей

draw.line([(width-50, center\_y), (width-60, center\_y-5)], fill=(0, 0, 0), width=2)

draw.line([(width-50, center\_y), (width-60, center\_y+5)], fill=(0, 0, 0), width=2)

draw.line([(center\_x, 50), (center\_x-5, 60)], fill=(0, 0, 0), width=2)

draw.line([(center\_x, 50), (center\_x+5, 60)], fill=(0, 0, 0), width=2)

# Подписи осей

draw.text((width-40, center\_y-20), "X", fill=(0, 0, 0))

draw.text((center\_x+10, 40), "Y", fill=(0, 0, 0))

# Деления на осях

for i in range(0, width, 50):

if i > 50 and i < width-50:

draw.line([(i, center\_y-3), (i, center\_y+3)], fill=(0, 0, 0), width=1)

if i % 100 == 0: # Подписи каждые 100 пикселей

draw.text((i-10, center\_y+10), str((i-center\_x)//20), fill=(0, 0, 0))

for i in range(0, height, 50):

if i > 50 and i < height-50:

draw.line([(center\_x-3, i), (center\_x+3, i)], fill=(0, 0, 0), width=1)

if i % 100 == 0: # Подписи каждые 100 пикселей

draw.text((center\_x+10, i-10), str((center\_y-i)//30), fill=(0, 0, 0))

# Рисование графика функции x\*sin(x) - вариант 4

points = []

scale\_x = 20 # Масштаб по X

scale\_y = 30 # Масштаб по Y

# Рисуем график для положительных x

for i in range(1, (width - center\_x - 50) \* 10 // scale\_x):

x = i / 10.0

try:

y = x \* math.sin(x) # Функция для варианта 4: x\*sin(x)

pixel\_x = center\_x + int(x \* scale\_x)

pixel\_y = center\_y - int(y \* scale\_y)

if 0 <= pixel\_x < width and 0 <= pixel\_y < height:

points.append((pixel\_x, pixel\_y))

except:

continue

# Рисуем график синей линией

if len(points) > 1:

for i in range(len(points) - 1):

draw.line([points[i], points[i+1]], fill=(0, 0, 255), width=3)

# Подпись графика

draw.text((center\_x + 100, 100), "y = x\*sin(x)", fill=(0, 0, 255), font=None)

self.result\_image = result\_img

self.update\_result\_display()

self.info\_label.config(text="Оси координат и график функции x\*sin(x) нарисованы")

def save\_result(self):

if self.result\_image is None:

messagebox.showerror("Ошибка", "Нет изображения для сохранения")

return

file\_path = filedialog.asksaveasfilename(

title="Сохранить результат в формате BMP",

defaultextension=".bmp",

filetypes=[("BMP files", "\*.bmp")]

)

if file\_path:

try:

# Сохраняем в формате BMP

self.result\_image.save(file\_path, format='BMP')

messagebox.showinfo("Успех", f"Изображение сохранено как {file\_path}")

self.info\_label.config(text=f"Изображение сохранено в формате BMP: {file\_path}")

except Exception as e:

messagebox.showerror("Ошибка", f"Не удалось сохранить: {str(e)}")

def update\_source\_display(self):

if self.source\_image:

display\_size = 400

# Используем изображение с границей, если оно есть

if hasattr(self, 'source\_image\_with\_border'):

display\_image = self.source\_image\_with\_border.copy()

else:

display\_image = self.source\_image.copy()

# Масштабируем для отображения с сохранением пропорций

ratio = min(display\_size/display\_image.width, display\_size/display\_image.height)

new\_size = (int(display\_image.width \* ratio), int(display\_image.height \* ratio))

display\_image = display\_image.resize(new\_size, Image.Resampling.LANCZOS)

photo = ImageTk.PhotoImage(display\_image)

# Очищаем canvas и добавляем изображение

self.source\_canvas.delete("all")

self.source\_canvas.create\_image(0, 0, anchor=tk.NW, image=photo)

self.source\_canvas.image = photo # Сохраняем ссылку

# Настраиваем прокрутку

self.source\_canvas.configure(scrollregion=self.source\_canvas.bbox("all"))

def update\_result\_display(self):

if self.result\_image:

display\_size = 400

display\_image = self.result\_image.copy()

# Масштабируем для отображения с сохранением пропорций

ratio = min(display\_size/display\_image.width, display\_size/display\_image.height)

new\_size = (int(display\_image.width \* ratio), int(display\_image.height \* ratio))

display\_image = display\_image.resize(new\_size, Image.Resampling.LANCZOS)

photo = ImageTk.PhotoImage(display\_image)

# Очищаем canvas и добавляем изображение

self.result\_canvas.delete("all")

self.result\_canvas.create\_image(0, 0, anchor=tk.NW, image=photo)

self.result\_canvas.image = photo # Сохраняем ссылку

# Настраиваем прокрутку

self.result\_canvas.configure(scrollregion=self.result\_canvas.bbox("all"))

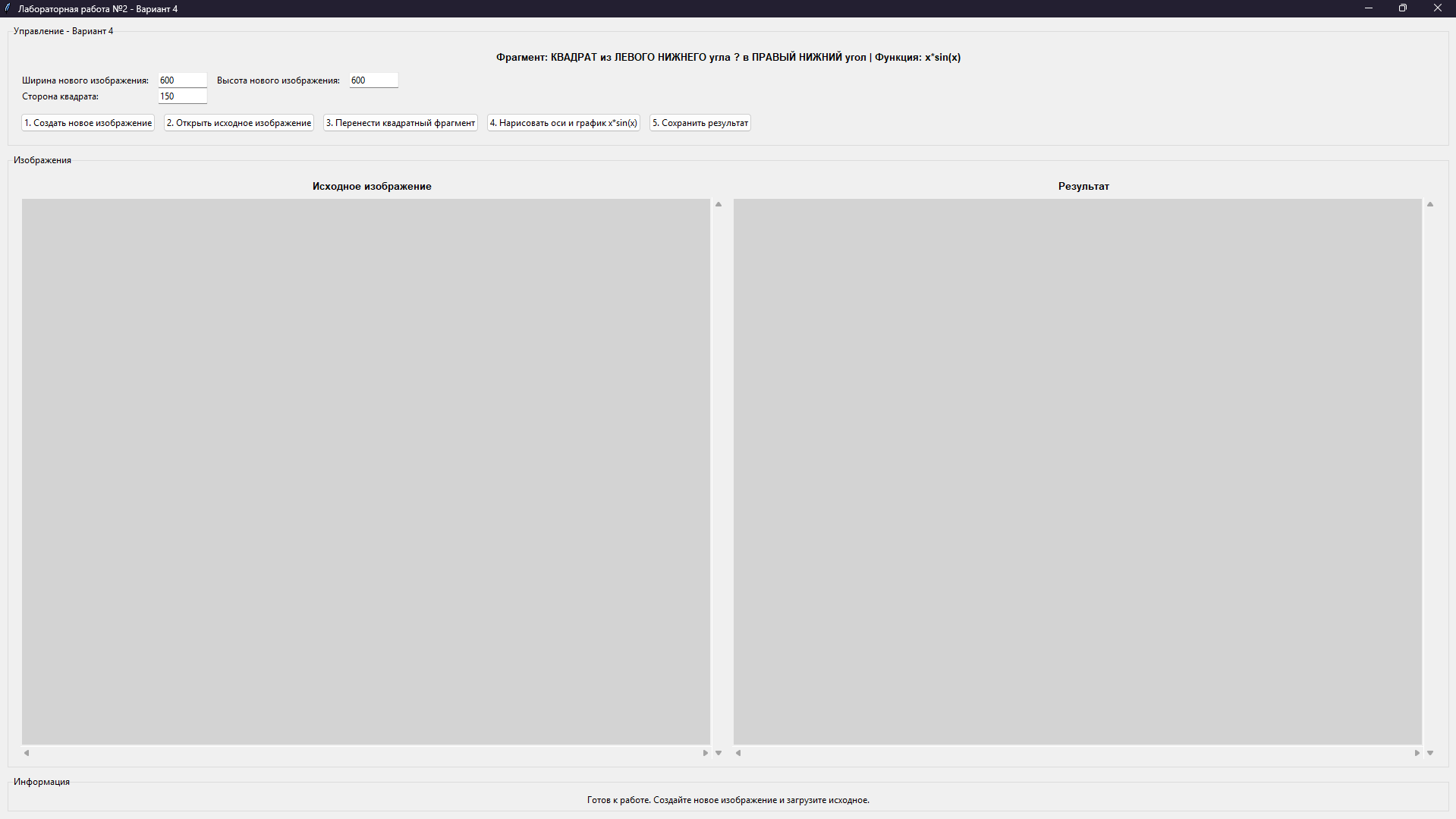
if \_\_name\_\_ == "\_\_main\_\_":

root = tk.Tk()

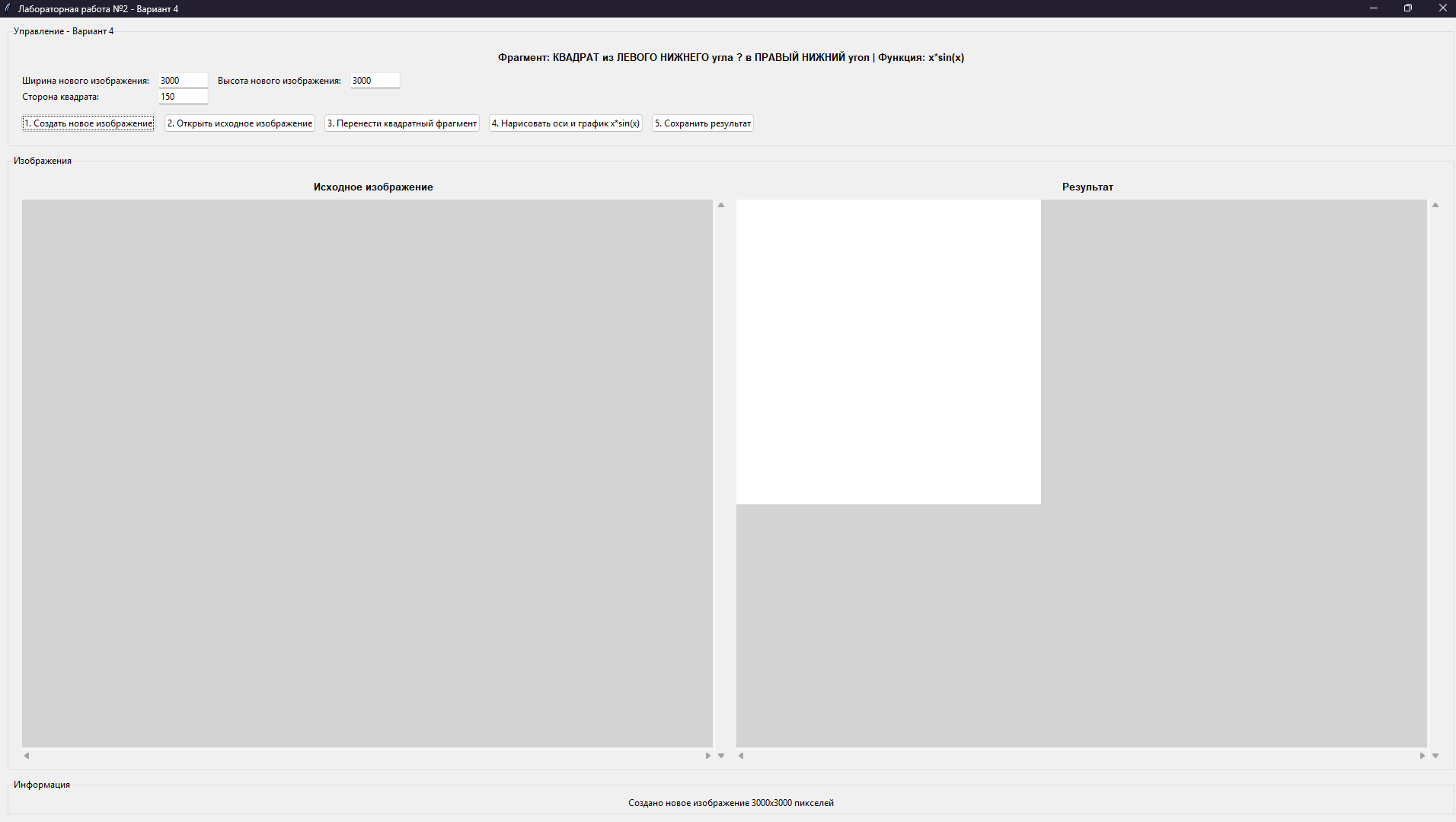
app = ImageProcessor(root)

root.mainloop()

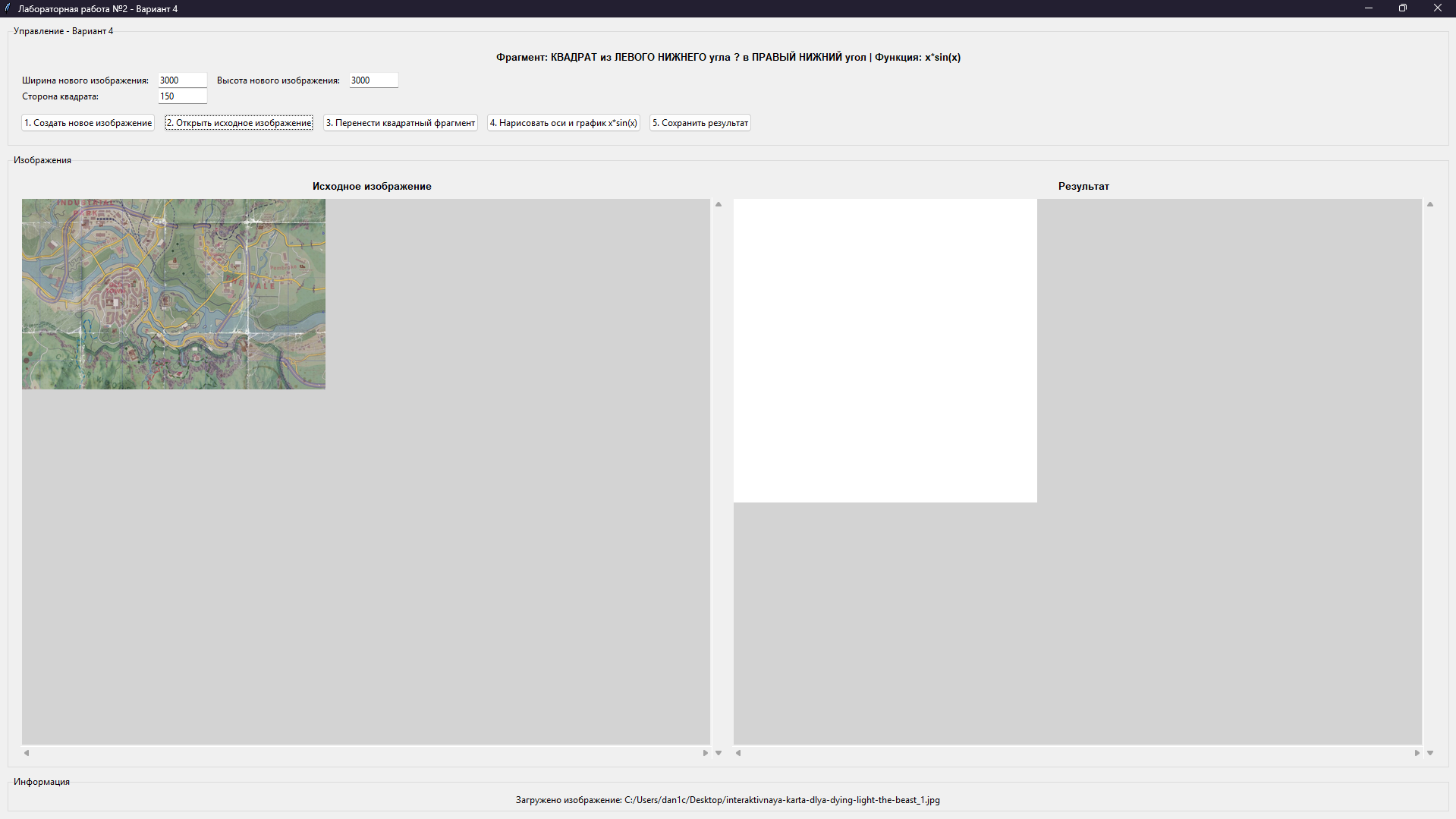
**Порядок работы:  
1.Запуск приложения**

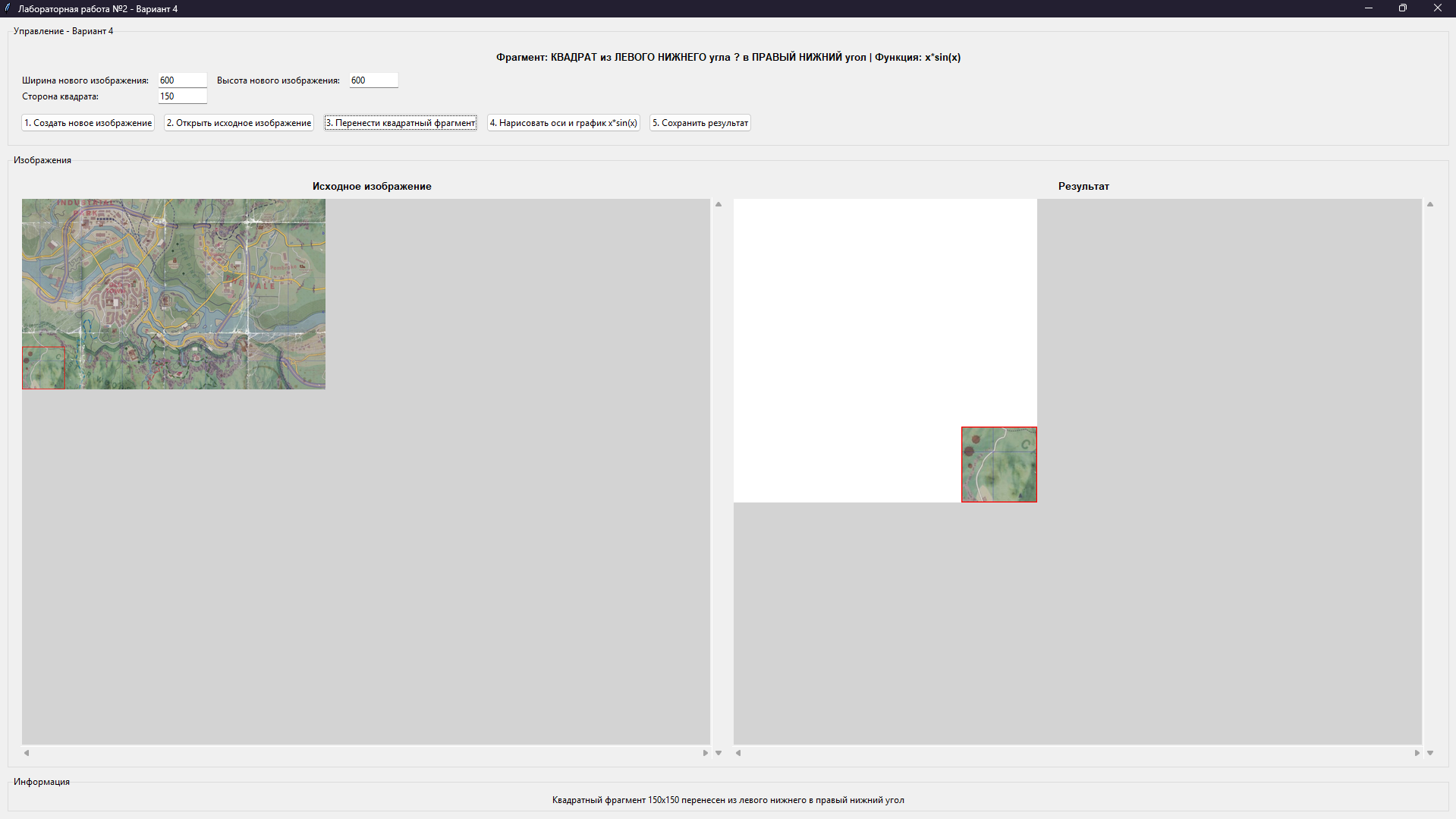


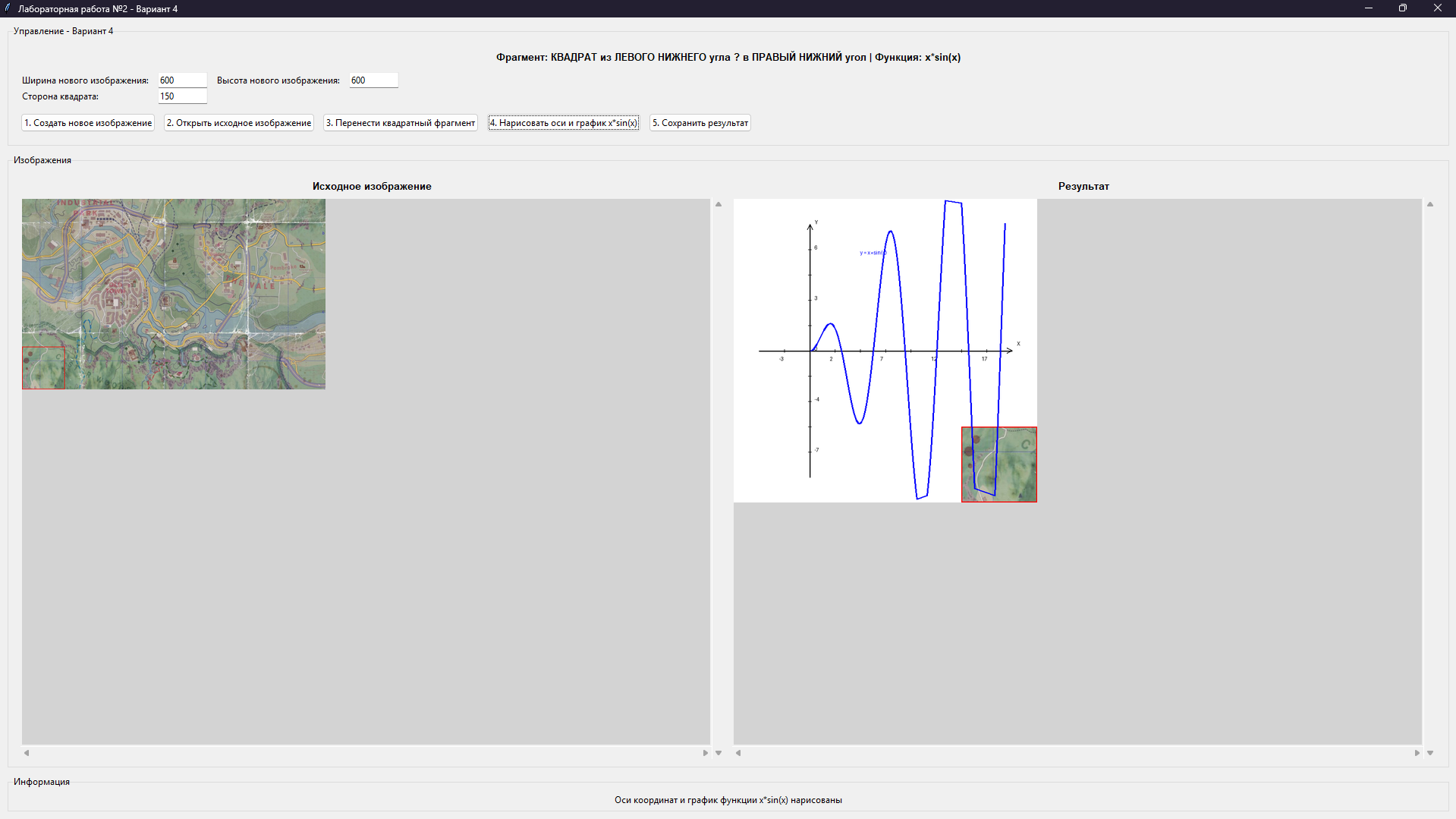
**2.Создание нового изображения**



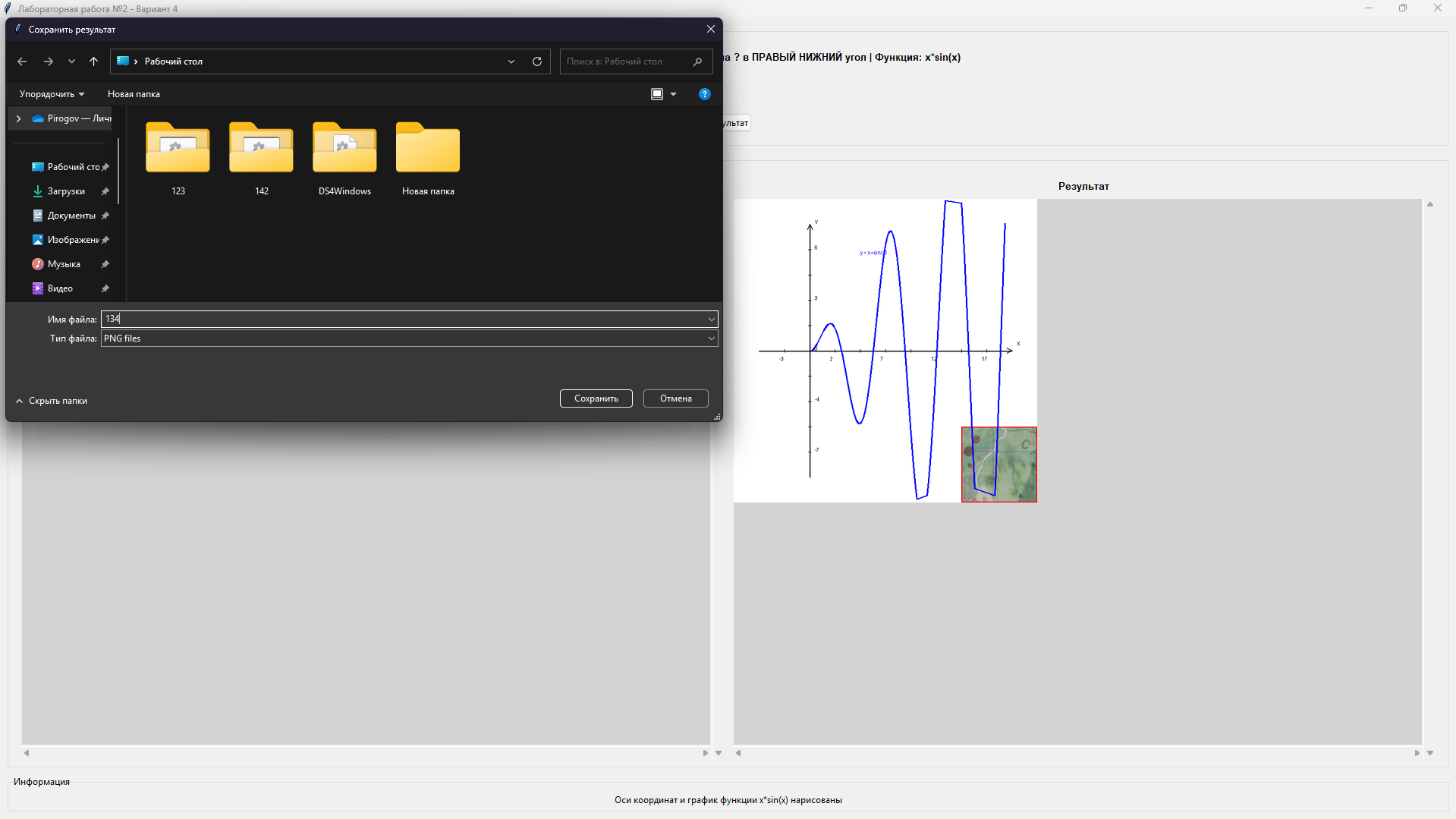
**3.Открытие исходного изображения**

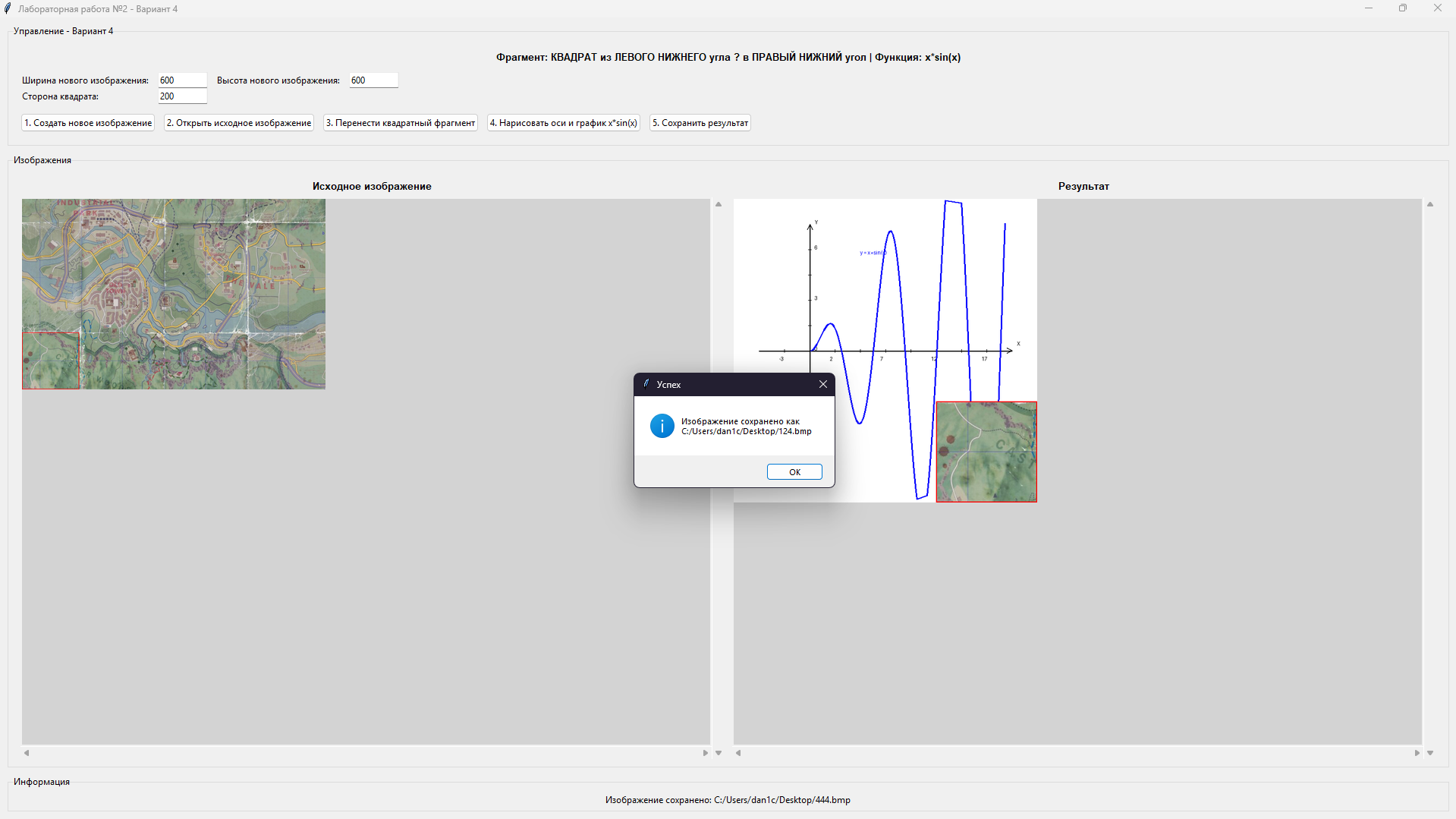


**4.Переносим фрагмент исходного изображения   
**

**5.Рисуем график sin(x)  
**

**6. Сохраняем результат**

****

****

**Ответы на контрольные вопросы**

## 1. Отличия оптического излучения

Оптическое излучение отличается от других видов излучений прежде всего своей способностью непосредственно восприниматься человеческим зрением без дополнительных преобразователей. Видимый свет занимает узкий диапазон электромагнитного спектра (380-780 нм), что обусловлено особенностями атмосферы Земли и спектром солнечного излучения. Именно в этом диапазоне достигается оптимальное сочетание разрешающей способности и энергетической эффективности для зрительного восприятия.

## 2. Важность зрения и его замена

## Зрение является основным каналом получения информации об окружающем мире, обеспечивая около 80% поступающих данных. Оно позволяет мгновенно оценивать форму, цвет, расстояние и движение объектов. Для замены зрительного восприятия разрабатываются тактильные интерфейсы, системы звуковой навигации и электронные сенсоры, но ни одна из этих технологий пока не может полностью воспроизвести богатство зрительного опыта.

## 3. Причины восприятия видимого света

Человек воспринимает именно видимый свет благодаря эволюционной адаптации к условиям Земли. Атмосфера нашей планеты максимально прозрачна для этого диапазона, а солнечное излучение имеет в нем пик интенсивности. Кроме того, фотоны видимого света обладают оптимальной энергией для эффективного протекания фотохимических реакций в сетчатке глаза.

## 4. Самосветящиеся источники

Наиболее распространенными самосветящимися источниками являются Солнце (тепловое излучение с непрерывным спектром), лампы накаливания (также тепловые источники) и светодиоды (электролюминесценция с узким спектром). Солнце обеспечивает естественное освещение, лампы накаливания дают теплый свет с хорошей цветопередачей, а светодиоды отличаются высокой энергоэффективностью и долговечностью.

## 5. Коррекция спектра источника

Коррекция спектра источника излучения имеет смысл при необходимости точной цветопередачи, например в музейном освещении, фотографии или медицинской диагностике. Также спектр корректируют для энергосбережения, создания специфических световых эффектов или обеспечения оптимальных условий для роста растений в теплицах.

## 6. Анизотропный и изотропный рефлекторы

Анизотропный диффузный рефлектор создается использованием шероховатых поверхностей с определенной ориентацией микронеровностей, что приводит к неравномерному распределению отраженного света. Для достижения изотропности необходимо создать поверхность со статистически равномерным распределением неровностей, обеспечивающим одинаковое рассеяние во всех направлениях.

## 7. Непрерывность функции яркости

Функция яркости является непрерывной, поскольку описывает усредненное поведение огромного количества дискретных фотонов. За счет статистического усреднения и конечного времени интеграции в зрительной системе дискретная природа света воспринимается как непрерывная величина. Это аналогично тому, как дискретные молекулы воды создают впечатление непрерывного потока.

## 8. Разделение света на зоны

Разделение света на зоны может осуществляться пространственным, временным или спектральным методами. Пространственное разделение просто в реализации, но ограничено дифракцией. Временное разделение обеспечивает высокую точность, но требует сложного оборудования. Спектральное разделение позволяет анализировать цветовые характеристики, но чувствительно к внешним помехам.

## 9. Регулярное vs хаотическое размещение

Регулярное размещение чувствительных элементов, характерное для матриц, обеспечивает простоту обработки сигнала и калибровки, но может приводить к муару. Хаотическое размещение, как в сетчатке глаза, уменьшает артефакты и повышает равномерность восприятия, однако требует более сложных алгоритмов обработки. Каждый подход имеет свои преимущества в зависимости от применения.

## 10. Эффективность оптической системы глаза

Оптическая система глаза достигает высокой эффективности благодаря миллионам лет эволюционной оптимизации. Хрусталик с переменным фокусным расстоянием, адаптивная диафрагма (зрачок) и специализированные фоторецепторы работают как единая система. Мозг осуществляет сложную постобработку изображения, компенсируя оптические недостатки.

## 11. Параметры искусственного датчика

Искусственный датчик, превосходящий глаз, должен обладать расширенным спектральным диапазоном, большим динамическим диапазоном (свыше 24 ступеней экспозиции), высочайшим разрешением и чувствительностью. Важны также адаптивные возможности, быстрая обработка данных и способность работать в различных условиях освещенности без потери качества изображения.

## 12. Приоритеты улучшения параметров

При улучшении дискретизации и квантования в первую очередь следует повышать разрешающую способность и битовую глубину, поскольку они непосредственно влияют на детализацию и цветовую точность изображения. Затем важно работать над уменьшением шумов и расширением динамического диапазона, что особенно критично в условиях сложного освещения.

## 13. Система координат холста

Система координат холста в компьютерной графике использует декартову систему с началом в левом верхнем углу. Ось X направлена вправо, ось Y - вниз, что отличается от математической традиции. Координаты измеряются в пикселях, причем каждый пиксель имеет целочисленные координаты. Такая система удобна для работы с растровыми изображениями.

## 14. Вывод движущихся изображений

Для вывода движущихся изображений на холст программа должна последовательно отрисовывать кадры с определенной частотой. Это достигается использованием таймеров или циклов анимации, которые обновляют положение объектов и перерисовывают сцену. Важно оптимизировать код для обеспечения плавности анимации и минимизации мерцания.