**`Минобрнауки России**

**Юго-Западный государственный университет**

**Кафедра программной инженерии**

ЛАБОРАТОРНАЯ РАБОТА

по дисциплине «Компьютерная графика»

по теме:

«Растеризация отрезков прямых»

Выполнили: студенты группы ПО-31б

Василевский В.Я.

Пирогов Д.Ю.

Проверил: старший преподаватель

Ефремов В.В.

Курск 2025 г.

**Вариант – 4**

**Цель работы**: изучение алгоритмов растеризации отрезков, создание программы для визуализации работы алгоритмов.

**Задание:**

1. Разработать программу, реализующую алгоритмы растеризации отрезков с их последующей прорисовкой. Вывести рисунок в соответствии с вариантом. Для вывода отрезков использовать:

* алгоритм ЦДА; − алгоритм Брезенхема;
* целочисленный алгоритм Брезенхема;
* встроенные средства языка программирования.

1. Проиллюстрировать разницу между результатами или её отсутствие.
2. Ответить на контрольные вопросы.
3. Оформить отчёт.

Вариант:



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

**Программа на python:**

import tkinter as tk  
from tkinter import messagebox, filedialog  
from math import sqrt, floor  
from PIL import Image, ImageDraw  
  
  
def sign(x):  
 if x > 0:  
 return 1  
 elif x < 0:  
 return -1  
 else:  
 return 0  
  
  
def DrawingLines(x1, y1, x2, y2, Canvas, color):  
 if abs(x1 - x2) < 0.1 and abs(y1 - y2) < 0.1:  
 Canvas.create\_rectangle(x1, y1, x1 + 1, y1 + 1, fill=color, outline=color)  
 return  
  
 dx = x2 - x1  
 dy = y2 - y1  
  
 if abs(dx) >= abs(dy):  
 l = abs(dx)  
 else:  
 l = abs(dy)  
  
 if l < 0.1:  
 return  
  
 dx = dx / l  
 dy = dy / l  
  
 x = x1 + 0.5 \* sign(dx)  
 y = y1 + 0.5 \* sign(dy)  
  
 for i in range(int(l) + 1):  
 Canvas.create\_rectangle(floor(x), floor(y), floor(x) + 1, floor(y) + 1, fill=color, outline=color)  
 x = x + dx  
 y = y + dy  
  
  
def BrezenhemFloat(x1, y1, x2, y2, Canvas, color):  
 if abs(x1 - x2) < 0.1 and abs(y1 - y2) < 0.1:  
 Canvas.create\_rectangle(x1, y1, x1 + 1, y1 + 1, fill=color, outline=color)  
 return  
  
 sx = sign(x2 - x1)  
 sy = sign(y2 - y1)  
  
 dx = abs(x2 - x1)  
 dy = abs(y2 - y1)  
  
 x = x1  
 y = y1  
 flag = 0  
  
 if dy > dx:  
 temp = dx  
 dx = dy  
 dy = temp  
 flag = 1  
  
 if dx < 0.1:  
 return  
  
 f = dy / dx - 0.5  
  
 steps = int(dx) + 1  
  
 for i in range(steps):  
 Canvas.create\_rectangle(int(x), int(y), int(x) + 1, int(y) + 1, fill=color, outline=color)  
  
 if f >= 0:  
 if flag == 1:  
 x = x + sx  
 else:  
 y = y + sy  
 f = f - 1  
  
 if flag == 1:  
 y = y + sy  
 else:  
 x = x + sx  
  
 f = f + dy / dx  
  
  
def BrezenhemInteger(x1, y1, x2, y2, Canvas, color):  
 x1, y1, x2, y2 = int(round(x1)), int(round(y1)), int(round(x2)), int(round(y2))  
  
 if x1 == x2 and y1 == y2:  
 Canvas.create\_rectangle(x1, y1, x1 + 1, y1 + 1, fill=color, outline=color)  
 return  
  
 sx = sign(x2 - x1)  
 sy = sign(y2 - y1)  
  
 dx = abs(x2 - x1)  
 dy = abs(y2 - y1)  
  
 x = x1  
 y = y1  
 flag = 0  
  
 if dy > dx:  
 temp = dx  
 dx = dy  
 dy = temp  
 flag = 1  
  
 if dx == 0:  
 return  
  
 f = 2 \* dy - dx  
  
 for i in range(dx + 1):  
 Canvas.create\_rectangle(x, y, x + 1, y + 1, fill=color, outline=color)  
  
 if f >= 0:  
 if flag == 1:  
 x = x + sx  
 else:  
 y = y + sy  
 f = f - 2 \* dx  
  
 if flag == 1:  
 y = y + sy  
 else:  
 x = x + sx  
  
 f = f + 2 \* dy  
  
  
def draw\_line\_builtin(x1, y1, x2, y2, Canvas, color):  
 Canvas.create\_line(x1, y1, x2, y2, fill=color, width=1)  
  
  
def DrawingLines\_pil(x1, y1, x2, y2, draw, color):  
 if abs(x1 - x2) < 0.1 and abs(y1 - y2) < 0.1:  
 draw.point((x1, y1), fill=color)  
 return  
  
 dx = x2 - x1  
 dy = y2 - y1  
  
 if abs(dx) >= abs(dy):  
 l = abs(dx)  
 else:  
 l = abs(dy)  
  
 if l < 0.1:  
 return  
  
 dx = dx / l  
 dy = dy / l  
  
 x = x1 + 0.5 \* sign(dx)  
 y = y1 + 0.5 \* sign(dy)  
  
 for i in range(int(l) + 1):  
 draw.point((floor(x), floor(y)), fill=color)  
 x = x + dx  
 y = y + dy  
  
  
def BrezenhemFloat\_pil(x1, y1, x2, y2, draw, color):  
 if abs(x1 - x2) < 0.1 and abs(y1 - y2) < 0.1:  
 draw.point((x1, y1), fill=color)  
 return  
  
 sx = sign(x2 - x1)  
 sy = sign(y2 - y1)  
  
 dx = abs(x2 - x1)  
 dy = abs(y2 - y1)  
  
 x = x1  
 y = y1  
 flag = 0  
  
 if dy > dx:  
 temp = dx  
 dx = dy  
 dy = temp  
 flag = 1  
  
 if dx < 0.1:  
 return  
  
 f = dy / dx - 0.5  
  
 steps = int(dx) + 1  
  
 for i in range(steps):  
 draw.point((int(x), int(y)), fill=color)  
  
 if f >= 0:  
 if flag == 1:  
 x = x + sx  
 else:  
 y = y + sy  
 f = f - 1  
  
 if flag == 1:  
 y = y + sy  
 else:  
 x = x + sx  
  
 f = f + dy / dx  
  
  
def BrezenhemInteger\_pil(x1, y1, x2, y2, draw, color):  
 x1, y1, x2, y2 = int(round(x1)), int(round(y1)), int(round(x2)), int(round(y2))  
  
 if x1 == x2 and y1 == y2:  
 draw.point((x1, y1), fill=color)  
 return  
  
 sx = sign(x2 - x1)  
 sy = sign(y2 - y1)  
  
 dx = abs(x2 - x1)  
 dy = abs(y2 - y1)  
  
 x = x1  
 y = y1  
 flag = 0  
  
 if dy > dx:  
 temp = dx  
 dx = dy  
 dy = temp  
 flag = 1  
  
 if dx == 0:  
 return  
  
 f = 2 \* dy - dx  
  
 for i in range(dx + 1):  
 draw.point((x, y), fill=color)  
  
 if f >= 0:  
 if flag == 1:  
 x = x + sx  
 else:  
 y = y + sy  
 f = f - 2 \* dx  
  
 if flag == 1:  
 y = y + sy  
 else:  
 x = x + sx  
  
 f = f + 2 \* dy  
  
  
def draw\_line\_builtin\_pil(x1, y1, x2, y2, draw, color):  
 draw.line([(x1, y1), (x2, y2)], fill=color, width=1)  
  
  
def create\_rhombus():  
 try:  
 x1 = float(entry\_x1.get())  
 y1 = float(entry\_y1.get())  
 x2 = float(entry\_x2.get())  
 y2 = float(entry\_y2.get())  
 other\_diag\_len = float(entry\_len.get())  
  
 if other\_diag\_len <= 0:  
 messagebox.showerror("Ошибка", "Длина диагонали должна быть положительной")  
 return  
  
 Canvas.delete("all")  
  
 offsets = [(0, 0), (3, 3), (6, 6), (9, 9)]  
 algorithms = [  
 ("ЦДА", "blue", DrawingLines),  
 ("Брезенхем (вещ.)", "red", BrezenhemFloat),  
 ("Брезенхем (цел.)", "green", BrezenhemInteger),  
 ("Встроенный", "black", draw\_line\_builtin)  
 ]  
  
 for i, (offset\_x, offset\_y) in enumerate(offsets):  
 algorithm\_name, color, algorithm\_func = algorithms[i]  
  
 shifted\_x1 = x1 + offset\_x - 50  
 shifted\_y1 = y1 + offset\_y - 30  
 shifted\_x2 = x2 + offset\_x - 50  
 shifted\_y2 = y2 + offset\_y - 30  
  
 center\_rhomb\_x = (shifted\_x1 + shifted\_x2) / 2  
 center\_rhomb\_y = (shifted\_y1 + shifted\_y2) / 2  
  
 dx = shifted\_x2 - shifted\_x1  
 dy = shifted\_y2 - shifted\_y1  
  
 diag1\_len = sqrt(dx \* dx + dy \* dy)  
  
 if diag1\_len < 0.1:  
 continue  
  
 perp\_dx = -dy  
 perp\_dy = dx  
  
 perp\_len = sqrt(perp\_dx \* perp\_dx + perp\_dy \* perp\_dy)  
 if perp\_len < 0.1:  
 continue  
  
 perp\_dx = perp\_dx / perp\_len \* other\_diag\_len / 2  
 perp\_dy = perp\_dy / perp\_len \* other\_diag\_len / 2  
  
 x3 = center\_rhomb\_x + perp\_dx  
 y3 = center\_rhomb\_y + perp\_dy  
 x4 = center\_rhomb\_x - perp\_dx  
 y4 = center\_rhomb\_y - perp\_dy  
  
 algorithm\_func(shifted\_x1, shifted\_y1, x3, y3, Canvas, color)  
 algorithm\_func(x3, y3, shifted\_x2, shifted\_y2, Canvas, color)  
 algorithm\_func(shifted\_x2, shifted\_y2, x4, y4, Canvas, color)  
 algorithm\_func(x4, y4, shifted\_x1, shifted\_y1, Canvas, color)  
  
 Canvas.create\_text(250, 350 + i \* 20, text=algorithm\_name, fill=color, font=("Arial", 8))  
  
 except ValueError:  
 messagebox.showerror("Ошибка", "Пожалуйста, введите корректные числовые значения")  
  
  
def save\_image():  
 try:  
 file\_path = filedialog.asksaveasfilename(  
 defaultextension=".png",  
 filetypes=[("PNG files", "\*.png"), ("All files", "\*.\*")]  
 )  
  
 if file\_path:  
 image = Image.new('RGB', (600, 400), 'white')  
 draw = ImageDraw.Draw(image)  
  
 x1 = float(entry\_x1.get())  
 y1 = float(entry\_y1.get())  
 x2 = float(entry\_x2.get())  
 y2 = float(entry\_y2.get())  
 other\_diag\_len = float(entry\_len.get())  
  
 colors\_pil = {  
 "blue": (0, 0, 255),  
 "red": (255, 0, 0),  
 "green": (0, 255, 0),  
 "black": (0, 0, 0)  
 }  
  
 algorithms\_pil = [  
 ("ЦДА", "blue", DrawingLines\_pil),  
 ("Брезенхем (вещ.)", "red", BrezenhemFloat\_pil),  
 ("Брезенхем (цел.)", "green", BrezenhemInteger\_pil),  
 ("Встроенный", "black", draw\_line\_builtin\_pil)  
 ]  
  
 offsets = [(0, 0), (3, 3), (6, 6), (9, 9)]  
  
 for i, (offset\_x, offset\_y) in enumerate(offsets):  
 algorithm\_name, color, algorithm\_func = algorithms\_pil[i]  
 color\_pil = colors\_pil[color]  
  
 shifted\_x1 = x1 + offset\_x - 50  
 shifted\_y1 = y1 + offset\_y - 30  
 shifted\_x2 = x2 + offset\_x - 50  
 shifted\_y2 = y2 + offset\_y - 30  
  
 center\_rhomb\_x = (shifted\_x1 + shifted\_x2) / 2  
 center\_rhomb\_y = (shifted\_y1 + shifted\_y2) / 2  
 dx = shifted\_x2 - shifted\_x1  
 dy = shifted\_y2 - shifted\_y1  
 diag1\_len = sqrt(dx \* dx + dy \* dy)  
  
 if diag1\_len >= 0.1:  
 perp\_dx = -dy  
 perp\_dy = dx  
 perp\_len = sqrt(perp\_dx \* perp\_dx + perp\_dy \* perp\_dy)  
  
 if perp\_len >= 0.1:  
 perp\_dx = perp\_dx / perp\_len \* other\_diag\_len / 2  
 perp\_dy = perp\_dy / perp\_len \* other\_diag\_len / 2  
  
 x3 = center\_rhomb\_x + perp\_dx  
 y3 = center\_rhomb\_y + perp\_dy  
 x4 = center\_rhomb\_x - perp\_dx  
 y4 = center\_rhomb\_y - perp\_dy  
  
 algorithm\_func(shifted\_x1, shifted\_y1, x3, y3, draw, color\_pil)  
 algorithm\_func(x3, y3, shifted\_x2, shifted\_y2, draw, color\_pil)  
 algorithm\_func(shifted\_x2, shifted\_y2, x4, y4, draw, color\_pil)  
 algorithm\_func(x4, y4, shifted\_x1, shifted\_y1, draw, color\_pil)  
  
 image.save(file\_path)  
 messagebox.showinfo("Успех", f"Изображение сохранено как {file\_path}")  
  
 except Exception as e:  
 messagebox.showerror("Ошибка", f"Не удалось сохранить изображение: {str(e)}")  
  
  
root = tk.Tk()  
root.title("Растеризация отрезков прямых")  
root.geometry("600x500")  
  
title\_label = tk.Label(root, text="Растеризация отрезков прямых (вариант с ромбами)", font=("Arial", 14, "bold"))  
title\_label.pack(pady=5)  
  
input\_frame = tk.Frame(root)  
input\_frame.pack(pady=10)  
  
tk.Label(input\_frame, text="x1").grid(row=0, column=0, padx=5, sticky="e")  
entry\_x1 = tk.Entry(input\_frame, width=8)  
entry\_x1.grid(row=0, column=1, padx=5)  
entry\_x1.insert(0, "200")  
  
tk.Label(input\_frame, text="y1").grid(row=0, column=2, padx=5, sticky="e")  
entry\_y1 = tk.Entry(input\_frame, width=8)  
entry\_y1.grid(row=0, column=3, padx=5)  
entry\_y1.insert(0, "150")  
  
tk.Label(input\_frame, text="x2").grid(row=1, column=0, padx=5, sticky="e")  
entry\_x2 = tk.Entry(input\_frame, width=8)  
entry\_x2.grid(row=1, column=1, padx=5)  
entry\_x2.insert(0, "400")  
  
tk.Label(input\_frame, text="y2").grid(row=1, column=2, padx=5, sticky="e")  
entry\_y2 = tk.Entry(input\_frame, width=8)  
entry\_y2.grid(row=1, column=3, padx=5)  
entry\_y2.insert(0, "150")  
  
tk.Label(input\_frame, text="Длина второй диагонали").grid(row=2, column=0, columnspan=2, padx=5, pady=5, sticky="e")  
entry\_len = tk.Entry(input\_frame, width=8)  
entry\_len.grid(row=2, column=2, padx=5)  
entry\_len.insert(0, "200")  
  
button\_frame = tk.Frame(root)  
button\_frame.pack(pady=10)  
  
btn\_create = tk.Button(button\_frame, text="Создать", command=create\_rhombus,  
 width=10, height=1)  
btn\_create.pack(side=tk.LEFT, padx=10)  
  
btn\_save = tk.Button(button\_frame, text="Сохранить", command=save\_image,  
 width=10, height=1)  
btn\_save.pack(side=tk.LEFT, padx=10)  
  
Canvas = tk.Canvas(root, width=600, height=400, bg="white", relief="solid", bd=1)  
Canvas.pack(pady=10)  
  
root.mainloop()

**Порядок работы:  
1.Запуск приложения**

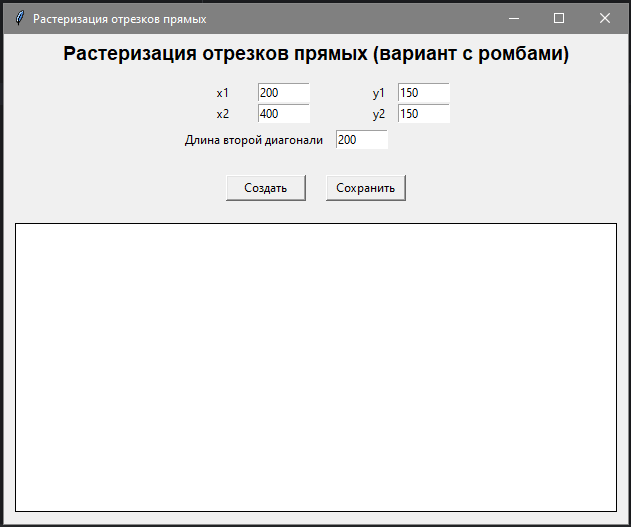
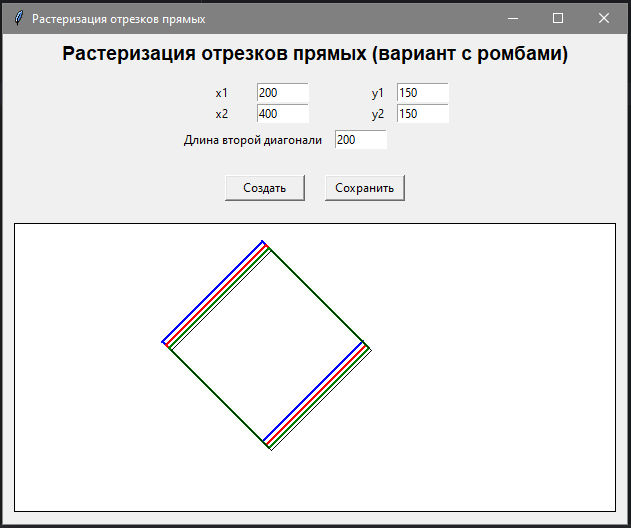


Рис.1 – Запуск приложения

**2.Нажатие на кнопку «Создать» с указанными точками**

****Рис.2 – Создание ромбов в соответствии с алгоритмами

**3.Обработка изображения**

**4.Сохранение результата в удобном формате**

**5.Сохранение файла в формате PBM**

**Ответы на контрольные вопросы**

**1. Что такое графика?**

Графика — это область визуального искусства и технологии, связанная с созданием, обработкой и отображением изображений. В широком смысле графика делится на:

Компьютерную графику: изображения, созданные, обработанные или отображаемые с помощью computational устройств (компьютеров).

Статическую графику: неподвижные изображения (рисунки, фотографии, чертежи).

Динамическую графику: движущиеся изображения (анимация, видео).

**2. Что такое оптическое изображение?**

Оптическое изображение — это картина, созданная световыми лучами, которые исходят от объекта (или кажутся исходящими) после их взаимодействия с оптической системой (например, линзой, зеркалом, глазом). Бывает двух видов:

Действительное изображение: образуется реальными лучами, его можно спроецировать на экран (например, изображение в кинопроекторе).

Мнимое изображение: образуется продолжениями лучей, его нельзя спроецировать на экран (например, изображение в обычном плоском зеркале).

**3. Что такое сцена в компьютерной графике?**

Сцена — это виртуальное трёхмерное пространство, содержащее все объекты, источники света, камеры и другие данные, необходимые для формирования конечного изображения или анимации. Это цифровая модель мира, которую нужно визуализировать.

**4. Что такое визуализация?**

Визуализация — это процесс преобразования абстрактных данных или идей в визуальную форму (изображение, анимацию, интерактивную среду) с целью облегчения их понимания, анализа и коммуникации. Примеры: рендеринг 3D-сцены, построение графиков и диаграмм, медицинские снимки.

**5. Зачем визуализировать то, что нереально воплотить физически?**

Визуализация невозможного в реальности преследует несколько ключевых целей:

Научные исследования и образование: моделирование явлений, которые нельзя увидеть (структура атома, черные дыры, процессы в ядре Земли) или которые опасны для воссоздания (взрывы, ядерные реакции).

Проектирование и дизайн: создание концептов будущих продуктов, зданий, интерьеров до их физического производства и строительства.

Искусство и развлечения: реализация творческих замыслов, создание фантастических миров и персонажей в кино и играх.

Анализ данных: представление больших и сложных массивов данных в наглядной форме для выявления скрытых закономерностей.

**6. Что такое рендеринг?**

Рендеринг (визуализация) — это финальный технологический процесс в компьютерной графике, в ходе которого 3D-сцена (описанная математическими моделями) преобразуется в плоское 2D-изображение (растровую картинку), которое мы видим на экране. Этот процесс включает расчет геометрии, освещения, теней, материалов, текстур и эффектов.

**7. Что понимается под 3-D графикой на 2-D мониторе?**

Это иллюзия трехмерности, создаваемая на плоском (2D) экране с помощью специальных методов:

Геометрические преобразования: перспектива, чтобы удаленные объекты казались меньше.

Работа с светом: расчет теней, бликов, затенения для передачи объема.

Стереоскопия: использование специальных очков для создания разного изображения для каждого глаза (3D-кино).

Параллакс: смещение более близких объектов относительно далеких при движении камеры (как в играх).

Монитор остается двумерным, но мы обманываем мозг, заставляя его воспринимать картинку как объемную.

**8. Как получить движущееся изображение?**

Движущееся изображение (анимация, видео) создается по принципу персистенции — инерции человеческого зрения. Быстрая смена последовательности статичных кадров (обычно 24, 25, 30 или 60 кадров в секунду) создает у зрителя иллюзию плавного движения.

**9. Как компьютерная графика различается по назначению?**

Растровая графика: для обработки и хранения фотографий, сложных изображений с плавными переходами цветов.

Векторная графика: для создания логотипов, чертежей, схем, шрифтов, где важны четкие контуры и возможность масштабирования без потерь.

3D-графика: для моделирования объектов, создания анимации, визуальных эффектов в кино, архитектурной визуализации и видеоиграх.

Научная визуализация: для графического отображения научных данных (графики, диаграммы, модели молекул).

Инфографика: для наглядного представления информации и статистики.

**10. Каковы особенности векторной и растровой графики?**

Векторная графика строится на математическом описании простых геометрических фигур (примитивов) — точек, линий, окружностей, многоугольников и кривых. Эти объекты хранятся в памяти компьютера в виде формул и параметров (например, координаты центра, радиус, цвет заливки). Главное преимущество векторной графики — она не зависит от разрешения. Это означает, что изображение можно увеличивать или уменьшать до любого размера без малейшей потери качества. Такие изображения имеют относительно небольшой размер файла, который зависит не от площади, а от сложности рисунка (количества объектов). Векторная графика идеально подходит для создания логотипов, иконок, шрифтов, инженерных чертежей и схем, где важны четкие, гладкие контуры.

Растровая графика, напротив, состоит из множества маленьких квадратных элементов — пикселей, которые образуют собой подобие сетки (растра). Каждый пиксель имеет свой строго определенный цвет и положение. Ключевая особенность растровых изображений — их зависимость от разрешения, то есть от общего количества пикселей по ширине и высоте. При значительном увеличении такого изображения пиксели становятся заметными, что приводит к потере четкости и появлению «зубчатости». Файлы растровой графики, особенно высокого качества, обычно имеют большой размер, который напрямую зависит от количества пикселей и глубины цвета. Этот формат незаменим для работы с фотографиями, цифровыми картинами и любыми изображениями с плавными переходами оттенков и сложной текстурой.

**11. Как отобразить 4-D объекты?**

Поскольку мы не можем воспринять 4-е измерение (чаще всего под ним понимают время или дополнительное пространственное измерение), используются специальные методы:

Проецирование: "отбрасывание тени" 4D-объекта в наше 3D-пространство (аналогично тому, как 3D-объект отбрасывает 2D-тень).

Сечения (Слайсы): показ 3D-сечений 4D-объекта.

Изменение во времени (Анимация): если 4-м измерением является время, то анимация показывает, как 3D-объект меняется.

Цвет или другие атрибуты: использование цвета, прозрачности или текстуры для обозначения значения в четвертом измерении.

**12. Как кодируется цвет?**

Цвет кодируется комбинацией числовых значений, описывающих его в определенной цветовой модели.

Основные модели:

RGB (Red, Green, Blue): Аддитивная модель. Цвет представляется как сумма интенсивностей красного, зеленого и синего каналов. Используется в мониторах и телевизорах. Например, `(255, 0, 0)` — ярко-красный.

CMYK (Cyan, Magenta, Yellow, Key/black): Субтрактивная модель. Цвет описывается количеством краски каждого цвета. Используется в полиграфии.

HSV/HSL (Hue, Saturation, Value/Lightness): Модель, более intuitive для человека. Цвет описывается тоном (оттенком), насыщенностью (чистотой цвета) и значением/яркостью.

**13. Чем стереоизображения отличаются от трёхмерных?**

Трехмерное (3D) изображение (на мониторе): Это монокулярное изображение (для одного глаза), которое создает иллюзию объема за счет перспективы, теней и т.д. Его можно увидеть без каких-либо приспособлений.

Стереоизображение: Это всегда бинокулярное изображение, использующее два отдельных кадра — для левого и правого глаза. Мозг объединяет их в одну картинку с ощущением настоящей глубины. Для просмотра требуются специальные средства (очки, шлемы VR).

**14. Что такое фрактал?**

Фрактал — это геометрическая фигура, обладающая свойством самоподобия. Это означает, что ее часть является уменьшенной копией целого. Какую бы маленькую часть фрактала вы ни увеличили, вы будете видеть похожую, но не обязательно идентичную структуру. Фракталы имеют дробную (не целочисленную) размерность и описываются рекурсивными математическими формулами. Классические примеры: множество Мандельброта, снежинка Коха.

**15. Что такое разрешение?**

Разрешение — это величина, определяющая количество деталей изображения или степень его четкости. В компьютерной графике обычно означает:

Для растровых изображений: количество пикселей по ширине и высоте (например, 1920×1080).

Для устройств вывода (мониторов, принтеров): плотность этих точек (пикселей или точек) на единицу длины.

**16. В каких единицах измеряется разрешающая способность?**

Разрешающая способность измеряется по-разному для разных устройств:

Мониторы, экраны: PPI (Pixels Per Inch) — количество пикселей на дюйм.

Принтеры, сканеры: DPI (Dots Per Inch) — количество точек на дюйм. Одна точка принтера (капля краски) не всегда равна одному пикселю изображения.

Изображения: Абсолютное разрешение в пикселях (например, 4000×3000 px) и/или относительное разрешение в PPI (например, 300 PPI), которое важно при печати.