**Минобрнауки России**

**Федеральное государственное образовательное учреждение высшего образования «Юго-Западный государственный университет»**

**Факультет фундаментальной и прикладной информатики**

**кафедра Программной инженерии**

По дисциплине «Компьютерная графика»

**ЛАБОРАТОРНАЯ РАБОТА №2**

КОМПЬЮТЕРНАЯ ГРАФИКА: Программа для формирования изображений

Выполнили: ст. гр. ПО-33б

Лукьянчиков А.Е.

Проверил: Ефремов В. В.

Курск, 2025

# Вариант 6

Каждый вариант индивидуальных заданий (табл. 1) включает в себя графическое изображение формы и позиции фрагмента на исход ном изображении, размеченном проходящими через центр вертикаль ной и горизонтальной линиями, позицию для размещения фрагмента на новом изображении и функцию, график которой нужно нарисовать на новом изображении. Конкретные числовые параметры, детально определяющие форму и положение фрагментов, выбираются произ вольно, но так чтобы не нарушить содержание условия задания.

Ссылка на открытый гит репозиторий https://github.com/xxyofu/GraphicsLaba2

# 2. Программа

# #include <iostream>

# #include <fstream>

# #include <vector>

# #include <string>

# #include <sstream>

# #include <cmath>

# #define PI 3.14159265358979323846

# using namespace std;

# struct RGB {

# unsigned char r, g, b;

# };

# struct Point {

# int x, y;

# };

# class Image1

# {

# public:

# int width, height;

# std::string magic\_number;

# std::vector<RGB> pixels;

# Image1(const string& filepath)

# {

# ifstream stream(filepath, ios::binary);

# if (!stream) {

# cout << "Cant open file" << endl;

# return;

# }

# stream >> magic\_number;

# stream.ignore(100, '\n');

# stream >> width >> height;

# stream.ignore(100, '\n');

# stream.ignore(100, '\n');

# pixels.resize(width \* height);

# if(magic\_number=="P6"){

# stream.read(reinterpret\_cast<char\*>(pixels.data()), pixels.size() \* sizeof(RGB));

# }else if (magic\_number=="P3"){

# for (int i = 0; i < width \* height; i++) {

# int r, g, b;

# stream >> r >> g >> b;

# pixels[i] = {static\_cast<unsigned char>(r),

# static\_cast<unsigned char>(g),

# static\_cast<unsigned char>(b)};

# }

# }else{

# return;

# }

# 

# }

# void Replace\_Pixel(int x, int y, unsigned char r, unsigned char g, unsigned char b)

# {

# pixels[y \* width + x] = { r, g, b };

# }

# bool Save\_image(const string& filepath)

# {

# ofstream stream(filepath, ios::binary);

# if (!stream) {

# cout << "Cant make file" << endl;

# return false;

# }

# stream << "P6\n" << width << " " << height << "\n255\n";

# stream.write(reinterpret\_cast<const char\*>(pixels.data()), pixels.size() \* sizeof(RGB));

# return true;

# }

# };

# class Canvas1 {

# public:

# int width, height;

# std::vector<RGB> pixels;

# Canvas1(const int& w, const int& h)

# {

# width = w;

# height = h;

# pixels.assign(width \* height,{255,255,255});

# }

# void Replace\_Pixel(int x, int y, RGB& color)

# {

# pixels[y \* width + x] = color;

# }

# void DrawLine(Point p1, Point p2, RGB& color)

# {

# int stepx = abs(p2.x - p1.x);

# int stepy = abs(p2.y - p1.y);

# int d = (stepx > stepy) ? stepx : stepy;

# int sx = (p1.x < p2.x) ? 1 : -1;

# int sy = (p1.y < p2.y) ? 1 : -1;

# int bx = 0;

# int by = 0;

# while (true) {

# if (p1.x == p2.x && p1.y == p2.y) break;

# Replace\_Pixel(p1.x, p1.y, color);

# bx += stepx;

# by += stepy;

# p1.x += sx\*(bx / d);

# p1.y += sy \* (by / d);

# bx %= d;

# by %= d;

# }

# 

# }

# void CopyLine(Point p1, Point p2, Point i1, Point i2, Image1 image)

# {

# int stepx = abs(p2.x - p1.x);

# int stepy = abs(p2.y - p1.y);

# int d = (stepx > stepy) ? stepx : stepy;

# int sx = (p1.x < p2.x) ? 1 : -1;

# int sy = (p1.y < p2.y) ? 1 : -1;

# int bx = 0;

# int by = 0;

# int istepx = abs(i2.x - i1.x);

# int istepy = abs(i2.y - i1.y);

# int id = (istepx > istepy) ? istepx : istepy;

# int isx = (i1.x < i2.x) ? 1 : -1;

# int isy = (i1.y < i2.y) ? 1 : -1;

# int ibx = 0;

# int iby = 0;

# while (true) {

# if (p1.x == p2.x && p1.y == p2.y) break;

# Replace\_Pixel(p1.x, p1.y, image.pixels[i1.y\*image.width+i1.x]);

# bx += stepx;

# by += stepy;

# p1.x += sx \* (bx / d);

# p1.y += sy \* (by / d);

# bx %= d;

# by %= d;

# ibx += istepx;

# iby += istepy;

# i1.x += isx \* (ibx / id);

# i1.y += isy \* (iby / id);

# ibx %= id;

# iby %= id;

# }

# }

# void CopyPart(Image1& image, Point destcircle\_center, Point circle\_center, int radius)

# {

# //Point e{ place.x + end.x - start.x, place.y + end.y - start.y };

# //(i-radius)/radius

# // for (int i = 0; i < radius\*2; i++)

# // {

# //     CopyLine({destcircle\_center.x+(i-radius),destcircle\_center.y+int(acos((i-radius)/radius)\*radius/PI/2)}, {destcircle\_center.x+(i-radius),destcircle\_center.y-int(acos((i-radius)/radius)\*radius/PI/2)}, {circle\_center.x+(i-radius),circle\_center.y+int(acos((i-radius)/radius)\*radius/PI/2)}, {circle\_center.x+(i-radius),circle\_center.y-int(acos((i-radius)/radius)\*radius/PI/2)}, image);

# // }

# for (int y = -radius; y <= radius; y++)

# {

# for (int x = -radius; x <= radius; x++)

# {

# 

# if (x\*x + y\*y <= radius\*radius)

# {

# 

# int src\_x = circle\_center.x + x;

# int src\_y = circle\_center.y + y;

# int dest\_x = destcircle\_center.x + x;

# int dest\_y = destcircle\_center.y + y;

# 

# 

# if (src\_x >= 0 && src\_x < image.width &&

# src\_y >= 0 && src\_y < image.height &&

# dest\_x >= 0 && dest\_x < image.width &&

# dest\_y >= 0 && dest\_y < image.height)

# {

# Replace\_Pixel(dest\_x, dest\_y, image.pixels[src\_y \* image.width + src\_x]);

# }

# }

# }

# }

# }

# void DrawFunk(  Point zero, int l, RGB color)

# {

# for (int i = 1; i <= l; i += 1)

# {

# DrawLine({zero.x+i,zero.y-int(log(double(i)/100.0))}, {zero.x+i+1,zero.y-int(log(double(i+1)/100.0))}, color);

# }

# }

# bool Save\_Canvas(const string& filepath, std::string magic\_number)

# {

# ofstream stream(filepath, ios::binary);

# if (!stream) {

# cout << "Cant make file" << endl;

# return false;

# }

# if(magic\_number == "P6"){

# stream << "P6\n" << width << " " << height << "\n255\n";

# stream.write(reinterpret\_cast<const char\*>(pixels.data()), pixels.size() \* sizeof(RGB));

# return true;

# }else if (magic\_number=="P3"){

# stream << "P3\n" << width << " " << height << "\n255\n";

# for (int i = 0; i < height; i++) {

# for (int j = 0; j < width; j++) {

# const RGB& pixel = pixels[i \* width + j];

# stream << static\_cast<int>(pixel.r) << " "

# << static\_cast<int>(pixel.g) << " "

# << static\_cast<int>(pixel.b);

# if (j < width - 1) stream << " ";

# }

# stream << "\n";

# }

# return true;

# }else{

# return false;

# }

# 

# }

# };

# int main(int argc, char \*argv[])

# {

# string input = "input.ppm";

# string output = "output.ppm";

# if(argc==2){

# input = argv[1];

# }else if(argc==3){

# input = argv[1];

# output = argv[2];

# }

# Canvas1 canvas(600, 300);

# Image1 img(input);

# Point place{ 300,170 };

# Point p1{ 50,150 };

# Point p2{300,100 };

# RGB color = { 0,0,0 };

# RGB bl = { 0,0,255 };

# canvas.CopyPart(img, {0+40, canvas.height/2},{img.width/2-40, img.height/2-40},40);

# canvas.DrawLine({ 0, canvas.height / 2 }, { canvas.width-1, canvas.height / 2 }, color);

# canvas.DrawLine({ canvas.width / 2, 0 }, { canvas.width / 2, (canvas.height-1)}, color);

# canvas.DrawLine({ canvas.width - 1, canvas.height / 2 }, { canvas.width - 1 - canvas.width / 20, canvas.height / 2 + canvas.height / 20 }, color);

# canvas.DrawLine({ canvas.width - 1, canvas.height / 2 }, { canvas.width - 1 - canvas.width / 20, canvas.height / 2 - canvas.height / 20 }, color);

# canvas.DrawLine({ canvas.width / 2 + canvas.width / 20 ,canvas.height / 20, }, { canvas.width / 2, 0 }, color);

# canvas.DrawLine({ canvas.width / 2 - canvas.width / 20 ,canvas.height / 20, }, { canvas.width / 2, 0 }, color);

# canvas.DrawFunk({ canvas.width / 2, canvas.height / 2 }, canvas.width / 2 - 1,bl);

# canvas.Save\_Canvas(output, "P3");

# }

# 3. Форма программы

Программа выполнена на языке C++ и запускается в консоли. Вызов осуществляется из командной строки с указанием входного и выходного файла:



# 4. Исходное и преобразованное изображения

Исходное изображение

  
Преобразованное изображение:



# 5. Ответы на контрольные вопросы

1. В чём основные отличия свойств оптического излучения от других видов излучения?

Главное отличие — энергия квантов (фотонов) и соответствующая ей длина волны, которая определяет характер взаимодействия излучения с веществом.

* Оптическое излучение (ультрафиолетовое, видимое, инфракрасное) имеет длину волны roughly от 10 нм до 1 мм. Энергии фотонов достаточно для электронных переходов в атомах и молекулах, возбуждения люминесценции, но недостаточно для ионизации атомов (за исключением жесткого УФ).
* Отличие от радиоизлучения: Длина волны больше, энергия фотонов мала. Легко проходит через атмосферу, не взаимодействуя с большинством материалов, что идеально для связи.
* Отличие от рентгеновского и гамма-излучения: Длина волны меньше, энергия фотонов гораздо выше. Эти виды излучения обладают высокой проникающей способностью и ионизирующей способностью, что опасно для живых тканей.

Ключевые свойства оптического излучения:

* Преломление и отражение: Подчиняется законам геометрической оптики.
* Интерференция и дифракция: Явно проявляются благодаря соизмеримости длины волны с препятствиями.
* Поглощение и испускание: Связаны с переходами электронов в веществе, что определяет цвет и спектр.

2. Почему зрение так важно для человека? Чем его можно заменить для восприятия той же информации об окружающем мире?

Важность зрения:

* Дистанционность и скорость: Информация воспринимается мгновенно и на расстоянии, что критично для ориентации в пространстве, избегания опасностей и добычи пищи.
* Информационная емкость: Зрение предоставляет огромный объем данных одновременно: форма, цвет, движение, текстура, глубина.
* Основной канал: До 80-90% информации об окружающем мире человек получает через зрение.

Замещение:  
Полностью заменить зрение невозможно, но можно дублировать ключевую информацию через другие каналы:

* Слух: Звуковые сигналы (эхолокация), озвучка текста, пространственное звучание для навигации.
* Тактильные ощущения: Шрифт Брайля, вибрационные сигналы (например, в смартфонах для навигации), тактильные карты.
* Прочие технологии: Сенсорные интерфейсы, устройства, преобразующие изображение в звук или тактильный образ (например, для слепых).

3. Почему человек воспринимает излучение в пределах именно этой небольшой части оптического диапазона, называемой видимым светом?

Это результат эволюционной адаптации.

* Атмосферное окно: Атмосфера Земли максимально прозрачна для излучения в диапазоне примерно 400-700 нм. Излучение других длин волн (УФ, ИК) сильно поглощается или рассеивается.
* Солнечный спектр: Пик излучения Солнца (при его температуре ~5800 K) приходится именно на видимый диапазон. Эволюционно выгодно было развить чувствительность к самому интенсивному потоку излучения.
* Биохимические возможности: Молекулы ретиналя в фоторецепторах сетчатки оптимально поглощают фотоны именно этой энергии. Фотоны с меньшей энергией (ИК) не вызывают нужных фотохимических реакций, а с большей (УФ) повреждают ткани.

4. Опишите характеристики трёх самосветящихся источников, которые чаще всего попадаются на глаза.

1. Солнце:
   * Тип: Тепловой излучатель.
   * Спектр: Непрерывный, близкий к спектру абсолютно черного тела с температурой ~5800 K.
   * Яркость: Очень высокая.
   * Цветовая температура: ~5500-6000 K (белый с желтоватым оттенком).
2. Лампа накаливания:
   * Тип: Тепловой излучатель.
   * Спектр: Непрерывный, смещенный в красную область (температура нити ~2700 K).
   * Яркость: Средняя.
   * Цветовая температура: ~2700 K (теплый белый).
3. Светодиод (LED-лампа или экран смартфона):
   * Тип: Люминесцентный (холодный) излучатель.
   * Спектр: Узкополосный у монохромных светодиодов; у белых — широкий, состоящий из пика синего светодиода и спектра люминофора.
   * Яркость: Регулируемая, от низкой до очень высокой.
   * Цветовая температура: Может быть любой (от теплой до холодной).

5. В каких случаях имеет смысл корректировать спектр источника излучения?

* Точная цветопередача: В музеях, полиграфии, фотографии, дизайне, где важно точное воспроизведение цветов (используются источники с высоким индексом цветопередачи CRI).
* Фотобиологические процессы: В растениеводстве (фитолампы) спектр корректируют под пики поглощения хлорофилла (синий и красный). В медицине — для лечения определенных состояний.
* Визуальный комфорт и здоровье: Уменьшение доли синего света вечером для нормализации циркадных ритмов (режим "Night Shift" в гаджетах).
* Привлечение внимания: Создание специфических спектров для сигнальных огней, рекламы.

6. Как получить анизотропный диффузный рефлектор и как сделать его изотропным?

* Анизотропный диффузный рефлектор отражает свет с разной яркостью в разных направлениях. Его можно получить, создав поверхность с упорядоченным микрорельефом (например, мелкие параллельные бороздки, рифление, brushing на металле).
* Чтобы сделать его изотропным (рассеивающим свет одинаково во все стороны), нужно разрушить эту упорядоченность. Это можно сделать:
  1. Шлифовкой: Создать хаотичный, беспорядочный микрорельеф (например, матовая поверхность).
  2. Покрытием: Нанести слой материала с хаотичными частицами-рассеивателями (краска, матовая пленка, белая бумага).

7. Почему функция яркости непрерывна, в то время как количество излучаемых фотонов дискретно?

Это проявление перехода от квантового описания к классическому.

* Фотоны испускаются дискретно, но в макроскопических процессах их количество чрезвычайно велико. Например, обычная лампа мощностью 100 Вт испускает порядка 10²⁰ фотонов в секунду.
* Яркость — это усредненная по времени и пространству энергетическая характеристика потока этих фотонов. Из-за закона больших чисел флуктуации, связанные с дискретностью, становятся пренебрежимо малы, и среднее значение ведет себя как непрерывная величина.

8. Опишите достоинства и недостатки каждого из способов разделения света на зоны.

* Последовательное (временное) разделение (например, вращающийся диск с цветными фильтрами):
  + *Достоинства*: Простота конструкции, высокое спектральное разрешение.
  + *Недостатки*: Низкое быстродействие, невозможность регистрации быстрых процессов, чувствительность к движению объекта.
* Пространственное разделение (например, призма или дифракционная решетка + несколько детекторов):
  + *Достоинства*: Высокое быстродействие (все зоны регистрируются одновременно).
  + *Недостатки*: Сложность и стоимость, меньшая светосила, возможные геометрические искажения.
* Матричное (фильтровое) разделение (например, массив Байера в цифровых камерах):
  + *Достоинства*: Компактность, низкая стоимость, высокое пространственное разрешение.
  + *Недостатки*: Интерполяция цвета (потеря истинного разрешения по цвету), меньшая спектральная точность.

9. Какой из способов размещения чувствительных элементов фотоприёмника предпочтителен – регулярный (как на матрице) или хаотический (как на сетчатке глаза)?

Регулярный (матричный) предпочтительнее для большинства технических применений.

* Достоинства регулярного размещения:
  + Простота изготовления и калибровки.
  + Легкость обработки сигнала (изображение представляет собой регулярную сетку пикселей).
  + Предсказуемость и равномерность характеристик по полю.
* Недостатки хаотического (как в сетчатке):
  + Сложность производства.
  + Сложность обработки сигнала, требующая "пересчета" в мозгу.
  + Нерегулярность может приводить к артефактам.

Глаз использует хаотическое размещение из-за эволюционных компромиссов (например, необходимость наличия слепого пятна — места выхода зрительного нерва, где нет рецепторов). Мозг компенсирует эти недостатки сложной постобработкой.

10. Почему оптическая система глаза, состоя из небольшого количества элементов, обеспечивает сравнительно высокое качество изображения?

* Адаптивность: Глаз — не статичная линза. Хрусталик меняет свою форму (аккомодация), а зрачок — диаметр, адаптируясь к расстоянию и освещенности.
* Постобработка в мозге: Мозг не просто "фотографирует" изображение. Он активно его обрабатывает, исправляя аберрации (хроматические, сферические), компенсируя дефекты (слепое пятно), усиливая контраст и края.
* Биологическая оптимизация: Роговица и хрусталик имеют сложный градиент показателя преломления, что минимизирует аберрации. Качество изображения максимально в центральной ямке (fovea), а периферическое зрение — менее качественное, но этого достаточно для выживания.

11. Какими параметрами должен обладать искусственный датчик излучения, чтобы не уступать глазу?

* Динамический диапазон: Способность одновременно различать детали в очень темных и очень светлых областях сцены (~24 f-стопа у глаза).
* Чувствительность: Возможность работать при очень низкой освещенности (до нескольких фотонов).
* Спектральный диапазон: Чувствительность в видимом диапазоне (400-700 нм).
* Разрешение: Высокая плотность пикселей, особенно в центре поля зрения.
* Адаптация: Возможность быстрой адаптации к изменению освещенности (как палочки и колбочки).
* Скорость обработки: Минимальная задержка между событием и его восприятием.

12. Какие параметры, влияющие на дискретизацию и квантование, следует улучшать в первую очередь?

Приоритет зависит от задачи, но обычно:

1. Разрешение (количество пикселей) и их размер — определяет уровень детализации и наличие муара.
2. Глубина цвета (битность квантования) — определяет плавность цветовых переходов и точность цветопередачи, особенно в тенях и светах.
3. Скорость дискретизации (частота кадров) — критична для регистрации быстрых процессов.

Для обычной фотографии приоритет: (1) -> (2). Для высокоскоростной съемки: (3) -> (1).

13. Какова система координат холста?

В компьютерной графике система координат холста обычно декартова, но с началом в левом верхнем углу.

* Ось X направлена слева направо.
* Ось Y направлена сверху вниз (в отличие от математической системы, где Y направлен вверх).
* Единицы измерения — пиксели.

14. Как с помощью программы вывести на холст движущиеся изображения?

Общий алгоритм (на примере игрового цикла или анимации):

1. Инициализация: Создать холст, загрузить ресурсы (изображения).
2. Главный цикл (выполняется много раз в секунду):
   * Очистка холста: Стереть предыдущий кадр.
   * Обновление состояния: Пересчитать позиции, скорости, состояния всех объектов (например, x = x + velocityX).
   * Отрисовка: Нарисовать все объекты на их новых позициях.
   * Синхронизация: Сделать паузу для поддержания заданного FPS (кадров в секунду).
3. Завершение: Выйти из цикла по условию.