МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА» (БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №2

по дисциплине: Компьютерная графика

тема: «Растровая заливка геометрических фигур»

Выполнил: ст. группы ПВ-223

Дмитриев Андрей Александрович

Проверил:

Осипов Олег Васильевич

Белгород 2024 г.

**Цель работы:** изучение алгоритмов растровой заливки основных геометрических фигур: кругов, многоугольников.

**Порядок выполнения работы**

1. Изучить растровые алгоритмы заливки геометрических фигур.

2. Разработать алгоритм и составить программу для построения на

экране изображения в соответствии с номером варианта (по журналу

старосты). В качестве исходных данных взять указанные в таблице

№1 лаб. работы №1.

**Требования к программе**

1. Программа должна быть написана на языках Си или C++.

2. Фигуры, нарисованные в первой лабораторной работе, необходимо заполнить цветом. Реализовать следующие способы заливки:

барицентрическая, радиальная, секторная. Изучить пример программы lab\_2\_colored\_square.vcxproj. Реализовать возможность выбора пользователем (например, клавишами) различных способов заливки для каждой геометрической фигуры.

3. Программа должна реагировать на выделение пользователем фигур, когда он выбирает их с использованием кнопок мыши. Фигура должна подсвечиваться другой текстурой или цветом, когда она выбрана после наведения (клика) на неё курсора мыши.

4. В программе должна быть предусмотрена возможность изменять прозрачность фигур. Фигуры должны быть нарисованы в порядке убывания площади, чтобы большие фигуры не закрывали маленькие.

5. Изображение должно масштабироваться строго по центру окна с радиусом 7/8 относительно размера окна (см. пример проекта lab\_1\_basics.vcxproj).

6. Пользовать должен иметь возможность менять размер окна и

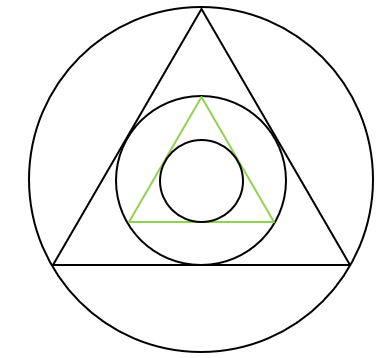
изменять разрешение пикселей. См. пример проекта lab\_1\_basics.vcxproj, в котором разрешение изменяется клавишами F2/F3.

7. Если в задании указано, что требуется реализовать анимацию (например, вращение), то перерисовку изображения нужно выполнять

по таймеру 30 раз в секунду.

8. Цвет примитивов выбрать по собственному усмотрению

Вариант 2:



Позиционирование точек треугольника и окружностей представлена в работе №1.

Основной алгоритм закрашивания треугольника представлен в теоретических сведениях к работе. Алгоритм удалось оптимизировать. Следующие строки кода в цикле:

int X0 = (int) ((y - y0) / (y1 - y0) \* (x1 - x0) + x0 + 0.5f);

int X1 = (int) ((y - y0) / (y2 - y0) \* (x2 - x0) + x0 + 0.5f);

– можно посчитать перед итерацией и складывать с изменяющейся частью (выделено в коде):

**int rawX0 = (Y0 + 0.5f - y0) / (y1 - y0) \* (x1 - x0) + x0;  
int rawX1 = (Y0 + 0.5f - y0) / (y2 - y0) \* (x2 - x0) + x0;**int X0, X1;  
  
// Рисование верхней части треугольника  
for (float y = Y0 + 0.5f; y < Y1; y++) {  
 X0 = rawX0;  
 X1 = rawX1;  
  
 if (X0 > X1) swap(X0, X1);  
 if (X0 < 0) X0 = 0;  
 if (X1 > width) X1 = width;  
  
 for (int x = X0; x < X1; x++) {  
 // f(x + 0.5, y)  
 COLOR color = Interpolator.color(x + 0.5f, y);  
  
 SetPixel(x, y, getColorByAlpha(color, x, y));  
 }  
  
 **rawX0 += (x1 - x0) / (y1 - y0);  
 rawX1 += (x2 - x0) / (y2 - y0);**  
}

Аналогично и с нижней частью треугольника.

Для закрашивания круга следующий оптимизированный код:

void Circle(int x0, int y0, int r, COLOR color) {  
 int x = 0, y = r;  
 int D = x \* x + y \* y - r \* r +  
 x \* x + (y - 1) \* (y - 1) - r \* r;  
  
 while (x < y) {  
 if (D >= 0) {  
 y--;  
 D -= 4 \* y - 2;  
 }  
  
 for (int i = x0 - x; i < x0 + x; i++) {  
 SetPixel(i, y0 + y,

getColorByAlpha(color, i, y0 + y));  
 SetPixel(i, y0 - y,

getColorByAlpha(color, i, y0 - y));  
 }  
 for (int i = x0 - y; i < x0 + y; i++) {  
 SetPixel(i, y0 + x,

getColorByAlpha(color, i, y0 + x));  
 SetPixel(i, y0 - x,

getColorByAlpha(color, i, y0 - x));  
 }  
  
 x++;  
 D += 4 \* x;  
 }  
}

Оптимизация заключается в использовании алгоритма Бразенхейма, как в лабораторной работе №1.

Функция getColorByAlpha применяет параметр прозрачности к цвету:

COLOR getColorByAlpha(COLOR color, int x, int y) {  
 // Для рисования полупрозрачных фигур будем использовать альфа-смешивание  
 if (color.ALPHA < 255) {  
 COLOR written = matrix[(int) y][x]; // Уже записанное в буфере кадра значение цвета, т.е. цвет фона  
 float a = color.ALPHA / 255.0f, b = 1 - a;  
 color.RED = color.RED \* a + written.RED \* b;  
 color.GREEN = color.GREEN \* a + written.GREEN \* b;  
 color.BLUE = color.BLUE \* a + written.BLUE \* b;  
 }  
  
 return color;  
}

Барицентрическая и радиальная интерполяция перенесены из теоретических сведений к работе.

Секторная заливка реализована следующим образом на основании формата HSV:

SectorInterpolator(float \_x0, float \_y0) :  
 cx(\_x0), cy(\_y0) {  
}  
  
COLOR color(float x, float y) override {  
 int r = 0;  
 int g = 0;  
 int b = 0;  
  
 float angle = 180.f + atan2(y - cy, x - cx) \* 180 / M\_PI;  
 if (angle >= 0 && angle < 60) {  
 r = 255;  
 g = (angle / 60.0) \* 255;  
 }  
 else if (angle >= 60 && angle < 120) {  
 angle -= 60;  
 r = (1 - angle / 60.0) \* 255;  
 g = 255;  
 }  
 else if (angle >= 120 && angle < 180) {  
 angle -= 120;  
 g = 255;  
 b = (angle / 60.0) \* 255;  
 }  
 else if (angle >= 180 && angle < 240) {  
 angle -= 180;  
 g = (1 - angle / 60.0) \* 255;  
 b = 255;  
 }  
 else if (angle >= 240 && angle < 300) {  
 angle -= 240;  
 r = (angle / 60.0) \* 255;  
 b = 255;  
 }  
 else if (angle >= 300 && angle <= 360) {  
 angle -= 300;  
 r = 255;  
 b = (1 - angle / 60.0) \* 255;  
 }  
  
 return COLOR(r, g, b, global\_alpha);  
}

Управление заливкой реализована через изменение глобальных переменных:

float global\_angle = 0;  
  
struct PIXEL {  
 int x;  
 int y;  
} global\_clicked\_pixel = {-1, -1};  
  
int global\_alpha = 255;  
  
int global\_action = 0;

Через левую и правую кнопку мыши можно изменять заливку: левой – менять интерполяцию, правой – подсвечивать фигуры:

case WM\_LBUTTONDOWN:  
 global\_clicked\_pixel = {  
 LOWORD(lParam) / pixelSize,  
 HIWORD(lParam) / pixelSize  
 };  
 global\_action = painter.*CHANGE*;  
 InvalidateRect(hWnd, NULL, false);  
 break;  
  
case WM\_RBUTTONDOWN:  
 global\_clicked\_pixel = {  
 LOWORD(lParam) / pixelSize,  
 HIWORD(lParam) / pixelSize  
 };  
 global\_action = painter.*HIGHLIGHT*;  
 InvalidateRect(hWnd, NULL, false);  
 break;

Управление заливками реализована через метод, который подставляет нужную интерполяцию или цвет:

BaseInterpolator &getInterpolator(  
 int indexMode,  
 int action,  
 OneColorInterpolator &oneColorInterpolator,  
 SectorInterpolator &sectorInterpolator,  
 RadialInterpolator &radialInterpolator,  
 BarycentricInterpolator &barycentricInterpolator  
) {  
 if (action == *CHANGE*) {  
 modes[indexMode] = spinDrawMode(modes[indexMode]);  
 highlighted[indexMode] = false;  
 } else if (action == *HIGHLIGHT*) {  
 for (auto &h: highlighted) h = false;  
 highlighted[indexMode] = true;  
 }  
  
 if (highlighted[indexMode]) {  
 return oneColorInterpolator;  
 }  
  
 if (modes[indexMode] == *SECTOR*) {  
 return sectorInterpolator;  
 } else if (modes[indexMode] == *RADIAL*) {  
 return radialInterpolator;  
 } else if (modes[indexMode] == *BARYCENTRIC*) {  
 return barycentricInterpolator;  
 } else {  
 std::cerr << "Unknown draw mode\n";  
 throw std::exception();  
 }  
}

Метод отрисовки задания варианта выглядит следующим образом:

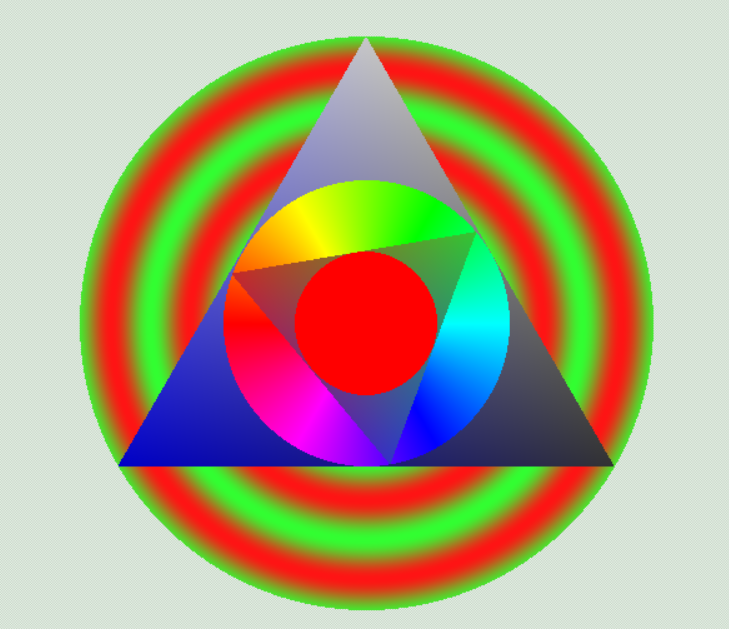
void Draw(Frame &frame) {  
 int W = frame.width, H = frame.height;  
  
 // Размер рисунка возьмём меньше (7 / 8), чтобы он не касался границ экрана  
 float a = 7.0 / 8.0 \* ((W < H) ? W - 1 : H - 1) / 2;  
  
 // Инициализируем исходные координаты центра и вершин квадрата  
 struct {  
 float x;  
 float y;  
 }  
 C = {W / 2 + 0.5f, H / 2 + 0.5f},  
 A\_st[3] = {  
 {C.x, C.y - a},  
 {static\_cast<float>(+a \* sin(2. \* M\_PI / 3) + C.x),  
 static\_cast<float>( -a \* cos(2. \* M\_PI / 3) + C.y)},  
 {static\_cast<float>(+a \* sin(-2. \* M\_PI / 3) + C.x),  
 static\_cast<float>(-a \* cos(-2. \* M\_PI / 3) + C.y)}},  
 B\_mv[3] = {  
 {A\_st[0].x, A\_st[0].y},  
 {A\_st[1].x, A\_st[1].y},  
 {A\_st[2].x, A\_st[2].y}  
 };  
  
 // Поворачиваем все вершины треугольника вокруг точки C на угол global\_angle

for (int i = 0; i < 3; i++) {  
 double xi = B\_mv[i].x, yi = B\_mv[i].y;  
 B\_mv[i].x = (xi - C.x) \* cos(global\_angle) - (yi - C.y) \* sin(global\_angle) + C.x;  
 B\_mv[i].y = (xi - C.x) \* sin(global\_angle) + (yi - C.y) \* cos(global\_angle) + C.y;  
 }  
  
  
 Assets::drawChessBg(frame);

// инициализация интерполяций

// ...  
   
 if (global\_action != *NOTHING* && global\_clicked\_pixel.x != -1  
 && global\_clicked\_pixel.y != -1  
 ) {  
 actionOnFigure[0] = frame.IsPointInCircle(  
 C.x, C.y, a,  
 global\_clicked\_pixel.x,  
 global\_clicked\_pixel.y  
 ) ? global\_action : *NOTHING*;  
 actionOnFigure[1] = frame.IsPointInTriangle(  
 A\_st[0].x, A\_st[0].y,  
 A\_st[1].x, A\_st[1].y,  
 A\_st[2].x, A\_st[2].y,  
 global\_clicked\_pixel.x, global\_clicked\_pixel.y  
 ) ? global\_action : *NOTHING*;  
 actionOnFigure[2] = frame.IsPointInCircle(  
 C.x, C.y, a / 2,  
 global\_clicked\_pixel.x,  
 global\_clicked\_pixel.y  
 ) ? global\_action : *NOTHING*;  
 actionOnFigure[3] = frame.IsPointInTriangle(  
 (B\_mv[0].x + C.x) / 2, (B\_mv[0].y + C.y) / 2,  
 (B\_mv[1].x + C.x) / 2, (B\_mv[1].y + C.y) / 2,  
 (B\_mv[2].x + C.x) / 2, (B\_mv[2].y + C.y) / 2,  
 global\_clicked\_pixel.x, global\_clicked\_pixel.y  
 ) ? global\_action : *NOTHING*;  
 actionOnFigure[4] = frame.IsPointInCircle(  
 C.x, C.y, a / 4,  
 global\_clicked\_pixel.x,  
 global\_clicked\_pixel.y  
 ) ? global\_action : *NOTHING*;  
 }  
  
 bool isFirstFigureChange = false;  
 for (int i = 4; i >= 0; i--) {  
 if (!isFirstFigureChange && actionOnFigure[i] != *NOTHING*) {  
 isFirstFigureChange = true;  
 } else {  
 actionOnFigure[i] = *NOTHING*;  
 }  
 }  
  
 // Рисуем внешний круг  
 frame.Circle(  
 C.x,  
 C.y,  
 a,  
 getInterpolator(  
 0,  
 actionOnFigure[0],  
 highlightColor,  
 sectorInterpolator,  
 radialInterpolator1,  
 barycentricInterpolator1  
 )  
 );  
  
 // Рисуем статичный треугольник  
 frame.Triangle(  
 A\_st[0].x, A\_st[0].y,  
 A\_st[1].x, A\_st[1].y,  
 A\_st[2].x, A\_st[2].y,  
 getInterpolator(  
 1,  
 actionOnFigure[1],  
 highlightColor,  
 sectorInterpolator,  
 radialInterpolator2,  
 barycentricInterpolator2  
 )  
 );  
  
 // Рисуем средний круг  
 frame.Circle(  
 C.x, C.y,  
 a / 2,  
 getInterpolator(  
 2,  
 actionOnFigure[2],  
 highlightColor,  
 sectorInterpolator,  
 radialInterpolator3,  
 barycentricInterpolator3  
 )  
 );  
  
 // Рисуем вращающийся внутренний треугольник  
 frame.Triangle(  
 (B\_mv[0].x + C.x) / 2, (B\_mv[0].y + C.y) / 2,  
 (B\_mv[1].x + C.x) / 2, (B\_mv[1].y + C.y) / 2,  
 (B\_mv[2].x + C.x) / 2, (B\_mv[2].y + C.y) / 2,  
 getInterpolator(  
 3,  
 actionOnFigure[3],  
 highlightColor,  
 sectorInterpolator,  
 radialInterpolator4,  
 barycentricInterpolator4  
 )  
 );  
 frame.Circle(  
 C.x, C.y,  
 a / 4,  
 {255, 0, 0, global\_alpha}  
 );  
  
 global\_clicked\_pixel = {-1, -1};  
 for (auto &action: actionOnFigure) {  
 action = *NOTHING*;  
 }  
}

Пример работы программы:





**Вывод:** в ходе работы изучены алгоритмы растровой заливки основных геометрических фигур: кругов, многоугольников.