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 №3

по дисциплине: Компьютерная графика

тема: «Аффинные преобразования на плоскости»

Выполнил: ст. группы ПВ-223

Дмитриев Андрей Александрович

Проверил:

Осипов Олег Васильевич

Белгород 2024 г.

**Цель работы:** получение навыков выполнения аффинных преобразований на плоскости и создание графического приложения на языке C++ для создания простейшей анимации.

**Порядок выполнения работы**

1. Разработать алгоритм и составить программу для построения на экране изображения в соответствии с номером варианта. В качестве исходных данных взять указанные в таблице №1.

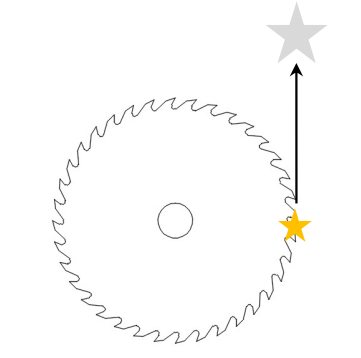
**Требования к программе**

1. Разработать модуль для выполнения аффинных преобразований на плоскости с помощью матриц. В модуле должны быть реализованы перегруженные операции действия с матрицами (умножение), с векторами и матрицами (умножение вектора-строки на матрицу), конструкторы различных матриц (переноса, масштабирования, переноса, отражения).
2. Раскрасить (залить) примитивы (круги, многоугольники и др.) по собственному усмотрению.

**Содержание отчѐта**

1. Название темы.
2. Цель работы.
3. Постановка задачи.
4. Вывод необходимых геометрических формул для построения изображения и расчѐта цвета. Также указать, какие матрицы используются и в какой последовательности они умножаются для реализации анимации.
5. Текст программы для рисования фигур.
6. Результат работы программы (снимки экрана).
7. Выводы о проделанной работе.

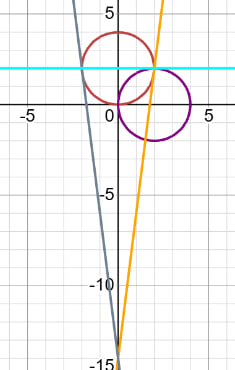
Вариант 2:



Ход работы:

**Пила** состоит из шейдеров, примененных на квадрат. Шейдер задаётся на плоскости 14 на 14 (для простоты счёта) и описывается неравенствами:

График:



Код программы (Шейдер):

COLOR main(float wx, float wy) {  
 if (  
 wy <= 2  
 && wy >= 8 \* wx - 14  
 && wy >= -8 \* wx - 14  
 && ((wx - 0) \* (wx - 0) + (wy - 2) \* (wy - 2) >= 2\*2  
 || (wx - 2) \* (wx - 2) + (wy - 0) \* (wy - 0) <= 2\*2)  
 ) {  
 return C0;  
 }  
 return {0, 0, 0, 0};  
}  
  
COLOR color(float x, float y) {  
  
 float h0 = ((y1 - y2) \* (x - x2) + (x2 - x1) \* (y - y2)) /S;  
 float h1 = ((y2 - y0) \* (x - x2) + (x0 - x2) \* (y - y2)) /S;  
 float h2 = ((y0 - y1) \* (x - x1) + (x1 - x0) \* (y - y1)) /S;  
 if (h0 < -1E-6 || h1 < -1E-6 || h2 < -1E-6) {  
 return COLOR(0, 0, 0, 0); // Ошибка алгоритма растеризации, если рисуется чёрный пиксель  
 }  
 float wx = h0 \* wx0 + h1 \* wx1 + h2 \* wx2;  
 float wy = h0 \* wy0 + h1 \* wy1 + h2 \* wy2;  
 return main(wx, wy);  
}

Код программы (Отрисовка пилы):

void drawSaw\_1(  
 Frame &frame,  
 Matrix &WS,  
 float xC, float yC,  
 float globalAngle  
) {  
 globalAngle = globalAngle \* 180.0f / M\_PI \* 2;  
  
 float maxAngle = 360.0f;  
 float rotateAngle = 10.0f;  
 float angle = 0.0f;  
 while (angle < maxAngle) {  
  
 // точки для рисования зубьев  
 Vector p[] = {  
 Vector(-0.5f, 0), Vector(0.5f, 0),  
 Vector(0.5f, 1), Vector(-0.5f, 1)};  
  
 for (int i = 0; i < \_countof(p); i++) {  
 p[i] = p[i] \* Matrix::rotation(angle + globalAngle) \* WS;  
 }  
   
 // применяем шейдер  
 TriangleShader shader1(  
 p[0].x() + xC, p[0].y() + yC,  
 p[1].x() + xC, p[1].y() + yC,  
 p[2].x() + xC, p[2].y() + yC,  
 -14, -14, 14, -14, 14, 14,  
 sawColor  
 );  
 frame.Triangle(  
 p[0].x() + xC, p[0].y() + yC,  
 p[1].x() + xC, p[1].y() + yC,  
 p[2].x() + xC, p[2].y() + yC,  
 shader1  
 );  
 TriangleShader shader2(  
 p[2].x() + xC, p[2].y() + yC,  
 p[3].x() + xC, p[3].y() + yC,  
 p[0].x() + xC, p[0].y() + yC,  
 14, 14, -14, 14, -14, -14,  
 sawColor  
 );  
 frame.Triangle(  
 p[2].x() + xC, p[2].y() + yC,  
 p[3].x() + xC, p[3].y() + yC,  
 p[0].x() + xC, p[0].y() + yC,  
 shader2  
 );  
  
 // отклоняем угол для рисования следующего зуба  
 angle += rotateAngle;  
 }  
}

**Искры** сделаны из фигуры «Звезда», правая сторона фигуры выложена на мировой координатной плоскости и отражена по горизонтали. Фигура в мировых координатах вплоскости 1 на 1:

**{**0.0f, 0.75**}**  
**{**0.0f, 0.25**}**  
**{**0.0f, -0.3**}**  
**{**0.5, -0.75**}**  
**{**-0.5, -0.75**}**  
**{**0.75, 0.25**}**  
**{**-0.75, 0.25**}**

Перед тем как позиционировать звезду координатная плоскость сначала рандомно поворачивается, затем удаляется на расстояние, зависящее от глобального поворота угла. После искра поворачивается в зависимости от глобального угла и увеличивается. Чтобы сделать затухание используем барицентрическую интерполяцию, будет брать один цвет из палитры и применять на фигуру от жёлтого к серому.

Код программы:

int currH = 0;  
float currAngle = 0;  
  
void drawStars(  
 Frame &frame,  
 Matrix WS,  
 float xC, float yC,  
 float globalAngle  
) {  
 globalAngle = globalAngle \* 180.0f / M\_PI;  
  
 // получаем переменную для просчёта удаления искр  
 int h = (int) globalAngle % 40;  
 // условие для повторного воспроизведения  
 if (currH > h) currAngle = rand() % 40 - 10;  
 currH = h;  
  
  
 // изменяем положение координатной плоскости  
 Matrix toss =  
 Matrix::rotation(currAngle)  
 \* Matrix::transform(0, h \* 0.075f)  
 \* Matrix::rotation(globalAngle)  
 \* Matrix::scale(h \* 0.02f);  
  
 // применяем матрицу на точки для построения звезды  
 Vector p1 = **{**0.0f, 0.75, 1.0f**}**;  
 p1 = p1 \* toss \* WS;  
 Vector p2 = **{**0.0f, 0.25, 1.0f**}**;  
 p2 = p2 \* toss \* WS;  
 Vector p3 = **{**0.0f, -0.3, 1.0f**}**;  
 p3 = p3 \* toss \* WS;  
 Vector p4 = **{**0.5, -0.75, 1.0f**}**;  
 Vector p4\_ = **{**-0.5, -0.75, 1.0f**}**;  
 p4 = p4 \* toss \* WS;  
 p4\_ = p4\_ \* toss \* WS;  
 Vector p5 = **{**0.75, 0.25, 1.0f**}**;  
 Vector p5\_ = **{**-0.75, 0.25, 1.0f**}**;  
 p5 = p5 \* toss \* WS;  
 p5\_ = p5\_ \* toss \* WS;  
  
 // интерополируем цвета  
 BarycentricInterpolator interpolator(  
 xC, yC + 10,  
 0, 0,  
 frame.width - 1, 0,  
 {255, 255, 0},  
 {180, 180, 180},  
 {180, 180, 180}  
 );  
 // получаем цвет из интерполятора  
 COLOR starColor = interpolator.color(p3.x() + xC, p3.y() + yC);  
  
 // рисуем звезду  
 frame.Triangle(  
 p5.x() + xC, p5.y() + yC,  
 p2.x() + xC, p2.y() + yC,  
 p3.x() + xC, p3.y() + yC,  
 starColor  
 );  
 frame.Triangle(  
 p4.x() + xC, p4.y() + yC,  
 p3.x() + xC, p3.y() + yC,  
 p1.x() + xC, p1.y() + yC,  
 starColor  
 );  
 frame.Triangle(  
 p5\_.x() + xC, p5\_.y() + yC,  
 p2.x() + xC, p2.y() + yC,  
 p3.x() + xC, p3.y() + yC,  
 starColor  
 );  
 frame.Triangle(  
 p4\_.x() + xC, p4\_.y() + yC,  
 p3.x() + xC, p3.y() + yC,  
 p1.x() + xC, p1.y() + yC,  
 starColor  
 );  
}

Код полной отрисовки 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;  
  
 Assets::drawChessBg(frame);  
 Matrix WS = Matrix::WorldToScreen(  
 W / 2 - a, H / 2 - a,  
 W / 2 + a, H / 2 + a,  
 -1, -1, 1, 1  
 );  
 float globalAngle = global\_angle;  
 drawSaw(  
 frame,  
 WS,  
 W / 2, H / 2,  
 -globalAngle  
 );  
 drawStars(  
 frame,  
 WS,  
 W \* 5 / 8.2, H / 2,  
 globalAngle  
 );  
 }

**Вывод:** в ходе работы получены навыки выполнения аффинных преобразований на плоскости и создания графического приложения на языке C++ для создания простейшей анимации.