

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

Игнатьев Артур Олегович

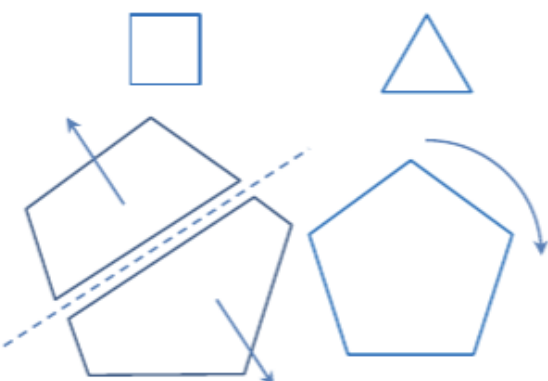
Проверил:

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.

### Вариант 3

3,13,23		<p>На экране в случайных местах непрерывно появляются равносторонние многоугольники. Постепенно скорость вращения каждого многоугольника относительно своего центра увеличивается и сам многоугольник тоже плавно увеличивается. При достижении определённой угловой скорости многоугольник случайным образом «трескается» пополам относительно центра. Два осколка разлетаются с равной скоростью перпендикулярно рассекающей прямой (как показано на рисунке). Осколки исчезают за пределами экрана. Цвет каждой точки на фигуре должен зависеть от расстояния до края фигуры.</p>
---------	---	--

### Вывод формул:

Пусть центр многоугольника находится в начале координат, а первая вершина имеет координаты  $A(0, 1)$ . Для определения координат всех вершин необходимо умножить координаты точки  $A$  на матрицы поворота:

$$P_i = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} \begin{pmatrix} \cos \alpha & \sin \alpha & 0 \\ -\sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

где параметр  $\alpha = \frac{2\pi}{n} + \beta$ , а  $\beta$  — дополнительный угол для анимации.

Далее определим новую мировую систему координат  $(x, y)$ , руководствуясь следующим образом:

Отступим от края экрана и введем новую мировую систему координат  $(X, Y)$ .

Построение матрицы преобразования мировых координат (x, y) в экранные (X, Y):

Выделим квадрат со стороной  $a = 7/8 * \min(W, H)$  и сопоставим точке  $(\frac{W}{2} + \frac{a}{2}, \frac{H}{2} + \frac{a}{2})$  мировые координаты (1, -1), а противоположной точке  $(\frac{W}{2} - \frac{a}{2}, \frac{H}{2} - \frac{a}{2})$  - координаты (-1, 1). Тогда для построения матрицы WS для преобразования мировых координат в экранные используем следующие параметры:

$$\begin{aligned} X_1 &= \frac{W}{2} - \frac{a}{2}, & Y_1 &= \frac{H}{2} - \frac{a}{2}, \\ X_2 &= \frac{W}{2} + \frac{a}{2}, & Y_2 &= \frac{H}{2} + \frac{a}{2}, \\ x1 &= -1, & y1 &= -1, & x2 &= 1, & y2 &= 1 \end{aligned}$$

Для получения окончательных экранных координат достаточно умножить все координаты  $P_i$  на матрицу WS.

### Пример функции:

```
void DrawRegularPolygon(Frame& frame, int sides, const Matrix& WS, float beta, COLOR color1, COLOR color2) {
    Vector A(0, 0.375f); // Координаты первой вершины многоугольника в мировой системе координат
    Vector O(0, 0); // Координаты центра многоугольника в мировой системе координат
    std::vector<Vector> V(sides); // Вектор для хранения экранных координат вершин

    // Преобразуем центр многоугольника в экранные координаты
    Vector C = O * WS;

    // Вычисляем координаты вершин многоугольника с учетом вращения
    for (int i = 0; i < sides; i++) {
        // Угол для текущей вершины с учетом поворота на угол beta
        Matrix M = Matrix::Rotation(360.0f / sides * i - beta) * WS;
        V[i] = A * M; // Координаты i-й вершины в экранной системе координат
    }

    // Рисуем многоугольник, используя треугольники между центром и парами соседних вершин
    for (int i = 0; i < sides; i++) {
        Vector T1 = V[i];
        Vector T2 = V[(i + 1) % sides]; // Следующая вершина (с циклическим индексом)

        // Создаем шейдер для градиента внутри треугольников
        TriangleShaderVertexToEdge shader(C.x, C.y, T1.x, T1.y, T2.x, T2.y, color1, color2);

        // Рисуем треугольник
        frame.Triangle(C.x, C.y, T1.x, T1.y, T2.x, T2.y, shader);
    }
}
```

## Класс для шейдера:

// Класс шейдера для отрисовки треугольников с градиентом от вершины к противоположной границе  
class TriangleShaderVertexToEdge {

float x0, y0, x1, y1, x2, y2; // Координаты вершин треугольника  
float S; // Площадь треугольника  
COLOR C0, C1; // Начальный и конечный цвет для градиента

public:

// Конструктор шейдера, инициализирует координаты вершин и цвета  
TriangleShaderVertexToEdge(float \_x0, float \_y0, float \_x1, float \_y1, float \_x2, float \_y2,  
COLOR \_C0, COLOR \_C1)  
: x0(\_x0), y0(\_y0), x1(\_x1), y1(\_y1), x2(\_x2), y2(\_y2),  
C0(\_C0), C1(\_C1),  
S((\_y1 - \_y2) \* (\_x0 - \_x2) + (\_x2 - \_x1) \* (\_y0 - \_y2)) // Вычисляем площадь треугольника  
{}

// Метод для вычисления цвета градиента от вершины к противоположной границе  
COLOR mainGradient(float h0) {  
// h0 — это барицентрическая координата, соответствующая вершине C0  
// Она изменяется от 1 (в вершине C0) до 0 (на противоположной стороне)  
float d = pow(h0, 0.375); // Используем h0 как коэффициент для линейной интерполяции

// Интерполируем цвет от C0 к C1 на основе d  
return COLOR(  
C0.RED \* d + C1.RED \* (1.0f - d),  
C0.GREEN \* d + C1.GREEN \* (1.0f - d),  
C0.BLUE \* d + C1.BLUE \* (1.0f - d),

);  
}

// Метод для получения цвета в точке (x, y) на треугольнике  
COLOR color(float x, float y) {  
// Вычисляем барицентрические координаты h0, h1, h2  
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;

// Если точка (x, y) находится за пределами треугольника, возвращаем черный цвет  
if (h0 < -0.00000001 || h1 < -0.00000001 || h2 < -0.00000001) {  
return COLOR(0, 0, 0); // Черный цвет для точек вне треугольника  
}

// Используем h0 для определения цвета в точке с градиентом от вершины к границе  
return mainGradient(h0);  
}

};

## Метод для реализации разбиения фигуры пополам с последующим вылетом за пределы экрана:

```
void DrawPolygonHalf(Frame& frame, int sides, const Matrix& WS, float beta, COLOR color1, COLOR color2, bool isLeftHalf, float offset) {
    Vector A(0, 0.375f); // Начальная вершина многоугольника в мировой системе координат. Расстояние от центра.
    Vector O(0, 0); // Центр многоугольника в мировой системе координат.
    std::vector<Vector> V(sides); // Массив для хранения экранных координат всех вершин многоугольника.
    Vector C = O * WS; // Экранные координаты центра многоугольника, преобразованные через матрицу WS.

    // Обновляем координаты вершин с учетом угла вращения beta.
    for (int i = 0; i < sides; i++) {
        Matrix M = Matrix::Rotation(360.0f / sides * i - beta) * WS; // Вычисляем матрицу вращения для текущей вершины и масштабируем по WS.
        V[i] = A * M; // Получаем экранные координаты текущей вершины, применяя матрицу к начальной вершине A.
    }

    // Смещение для половин.
    float direction = isLeftHalf ? -1.0f : 1.0f; // Определяем направление смещения для левой или правой половины.
    Vector explode_offset(offset * direction, 0); // Вычисляем вектор смещения для эффекта разлетающейся половины.

    // Рисуем половину фигуры.
    int mid = sides / 2; // Находим индекс середины многоугольника, чтобы разделить его на две половины.
    for (int i = 0; i < mid; i++) {
        int index1 = isLeftHalf ? (i + mid); // Выбираем начальную вершину для текущей половины (либо левую, либо правую).
        int index2 = (index1 + 1) % sides; // Определяем индекс следующей вершины для создания грани.

        Vector T1 = V[index1] + explode_offset; // Применяем смещение к первой вершине текущей грани.
        Vector T2 = V[index2] + explode_offset; // Применяем смещение ко второй вершине текущей грани.

        // Создаем шейдер для градиента между половинами.
        TriangleShaderVertexToEdge shader(C.x + explode_offset.x, C.y + explode_offset.y, T1.x, T1.y, T2.x, T2.y, color1, color2);

        // Отрисовка половины фигуры.
        frame.Triangle(C.x + explode_offset.x, C.y + explode_offset.y, T1.x, T1.y, T2.x, T2.y, shader); // Рисуем треугольник с градиентом от центра до грани.
    }
}
```

## Основная функция программы:

// Метод для рисования многоугольников на экране

```
void Draw(Frame& frame) {
    float W = frame.width, H = frame.height;

    static float centerXTriangle = W / 4, centerYTriangle = H / 2; // Triangle (left)
    static float centerXSquare = 3 * W / 4, centerYSquare = H / 2; // Square (right)
    static float centerXPentagon = W / 3, centerYPentagon = H / 3; // Pentagon (fixed position)
    static float centerXHexagon = W / 2, centerYHexagon = 2 * H / 3; // Hexagon (new position)

    if (rotation_speed < max_rotation_speed) {
        //обновление положения фигур
        UpdatePolygonCenter(W, H, centerXSquare, centerYSquare);
        UpdatePolygonCenterTriangle(W, H, centerXTriangle, centerYTriangle);
        UpdatePolygonCenterPentagon(W, H, centerXPentagon, centerYPentagon);
        UpdatePolygonCenterHexagon(W, H, centerXHexagon, centerYHexagon);
    }

    float a = 7.0 / 8.0 * ((W < H) ? W : H);
    global_angle += rotation_speed;
    float beta = global_angle;

    // Увеличиваем rotation_speed, если она не достигла max_rotation_speed
    rotation_speed = min(rotation_speed + 0.01f, max_rotation_speed);

    // Коэффициент масштабирования на основе скорости вращения
    float scale_factor = 1.0f + 0.2f * (rotation_speed / max_rotation_speed);

    Matrix WS1 = Matrix::WorldToScreen(centerXSquare - a / 2, centerYSquare - a / 2, centerXSquare + a / 2,
centerYSquare + a / 2, -1, -1, 1, 1) * Matrix::Scale(scale_factor, scale_factor);
    Matrix WS2 = Matrix::WorldToScreen(centerXTriangle - a / 2, centerYTriangle - a / 2, centerXTriangle + a / 2,
centerYTriangle + a / 2, -1, -1, 1, 1) * Matrix::Scale(scale_factor, scale_factor);
    Matrix WS3 = Matrix::WorldToScreen(centerXPentagon - a / 2, centerYPentagon - a / 2, centerXPentagon + a / 2,
centerYPentagon + a / 2, -1, -1, 1, 1) * Matrix::Scale(scale_factor, scale_factor);
    Matrix WS4 = Matrix::WorldToScreen(centerXHexagon - a / 2, centerYHexagon - a / 2, centerXHexagon + a / 2,
centerYHexagon + a / 2, -1, -1, 1, 1) * Matrix::Scale(scale_factor, scale_factor);

    if (rotation_speed >= max_rotation_speed) { // Проверка: если скорость вращения достигла максимального
значения
        static float explode_offset = 0; // Статическая переменная для смещения разлетающихся половин
        explode_offset += 10.0f; // Увеличиваем смещение, чтобы части фигуры расходились

        // Отрисовка разлетающихся половин с цветами
        DrawPolygonHalf(frame, 4, WS1, global_angle, COLOR(0, 0, 255), COLOR(255, 255, 255), true, explode_offset);
        // Левая половина квадрата (синий)
        DrawPolygonHalf(frame, 4, WS1, global_angle, COLOR(0, 0, 255), COLOR(255, 255, 255), false, explode_offset);
        // Правая половина квадрата (синий)
        DrawPolygonHalf(frame, 3, WS2, global_angle, COLOR(255, 105, 180), COLOR(255, 255, 255), true,
explode_offset); // Левая половина треугольника (розовый)
        DrawPolygonHalf(frame, 3, WS2, global_angle, COLOR(255, 105, 180), COLOR(255, 255, 255), false,
explode_offset); // Правая половина треугольника (розовый)
        DrawPolygonHalf(frame, 5, WS3, global_angle, COLOR(0, 0, 255), COLOR(255, 255, 255), true, explode_offset);
        // Левая половина пятиугольника (синий)
        DrawPolygonHalf(frame, 5, WS3, global_angle, COLOR(0, 0, 255), COLOR(255, 255, 255), false, explode_offset);
        // Правая половина пятиугольника (синий)
        DrawPolygonHalf(frame, 6, WS4, global_angle, COLOR(128, 0, 128), COLOR(255, 255, 255), true,
explode_offset); // Левая половина шестиугольника (фиолетовый)
```

```

    DrawPolygonHalf(frame, 6, WS4, global_angle, COLOR(128, 0, 128), COLOR(255, 255, 255), false,
explode_offset); // Правая половина шестиугольника (фиолетовый)
    }
else {
    // Продолжаем обычное вращение с цветами
    DrawRegularPolygon(frame, 4, WS1, global_angle, COLOR(0, 0, 255), COLOR(255, 255, 255)); // Квадрат
(синий)
    DrawRegularPolygon(frame, 3, WS2, global_angle, COLOR(255, 105, 180), COLOR(255, 255, 255)); //
Треугольник (розовый)
    DrawRegularPolygon(frame, 5, WS3, global_angle, COLOR(0, 0, 255), COLOR(255, 255, 255)); // Пятиугольник
(синий)
    DrawRegularPolygon(frame, 6, WS4, global_angle, COLOR(128, 0, 128), COLOR(255, 255, 255)); //
Шестиугольник (фиолетовый)
    }

    // Пауза в 30 миллисекунд для обновления анимации
    std::this_thread::sleep_for(std::chrono::milliseconds(30));
}

```

**Вывод:** на этой лабораторной работе мы получили навыки выполнения аффинных преобразований на плоскости и создания графического приложения на языке C++ для создания простейшей анимации.