НИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное образовательное учреждение высшего образования
САНКТ – ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ

КАФЕДРА № 42

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

доцент, канд. техн. наук		А.В. Аграновский
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №2

Исследование аффинных преобразований на плоскости

по курсу: Компьютерная графика

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ гр. № 4321		К.А. Лебедев
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2024

СОДЕРЖАНИЕ

1 Цель работы

Целью работы является изучение теоретических основ линейных геометрических преобразований на плоскости. Исследование особенностей практической реализации двумерных аффинных преобразований.

В процессе выполнения лабораторной работы исследуются разные подходы к практической реализации анимации аффинных преобразований. Задачами являются изучение теории принципов применения матриц преобразования для анимации вращения фигур, реализация программы алгоритма этих преобразований и их визуализация.

2 Задание

В рамках данной работы необходимо программу на языке высокого уровня, осуществляющую анимированное аффинное преобразование фигуры на плоскости в соответствии с выбранным вариантом.

Основной задачей является: разработать программу, которая плавно деформирует прямоугольный треугольник, масштабируя его сначала вдоль одного из катетов, затем вдоль гипотенузы. Ни одна из сторон треугольника не должна совпадать с направлением координатных осей.

3 Теоретические сведения

Аффинные преобразования — это линейные преобразования координатных точек, которые сохраняют прямые линии и соотношения между расстояниями, но могут изменять углы и размеры фигур.

Линейные геометрические преобразования изображений

Растровое изображение представляет собой двумерный массив пикселей, где каждый элемент (пиксель) имеет целочисленные координаты x и y. Линейные преобразования описывают формулами

$$x' = ax + by + c \qquad (1)$$

$$y' = dx + ey + f \qquad (2)$$

или в виде матрицы:

$$\begin{matrix} x' \\ y' \\ 1 \end{matrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad (3)$$

где *a, b, c, d, e, f* — параметры преобразования.

Основными типами линейных преобразований на плоскости являются так называемые аффинные преобразования – масштабирование, поворот, перенос (сдвиг), отражение относительно оси. Все рассмотренные аффинные преобразования за исключением сдвига могут быть записаны в виде матрицы.

Поворот на плоскости является одним из важнейших аффинных преобразований для выполнения лабораторной работы. Для поворота объекта относительно начала координат (0, 0) используется следующая матрица поворота:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & co\theta s & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (4)$$

4 Алгоритм выполнения задачи

Реализация анимации вращения прямоугольника достигается с помощью использования матриц аффинных преобразований. Приведем алгоритм выполнения задачи.

4.1 Инициализация

– Определение начальных координат вершин прямоугольника

– Выбор угла прямоугольника для точки вращения

– Настройка параметров анимации (количество кадров, интервал)

– Создание канваса для отображения графики

4.2 Преобразование координат

– Создание матрицы вращения для каждого кадра

– Создание матриц для масштабирования, поворота

– Комбинирование матриц

4.3 Вращение объекта

– Применение последовательности матричных преобразований к координатам вершин

– Вычисление новых величин для каждого кадра анимации

4.4 Обновление графики

– Очистка предыдущего кадра

– Отрисовка нового треугольника

4.5 Анимация

– Использование animate(), которая с помощью requestAnimationFrame рекурсивно вызывает анимацию

– Настройка параметров воспроизведения анимации

5 Язык программирования и используемые библиотеки

Для реализации программы построения фигур и аффинных преобразований был использован язык программирования JavaScript, а также следующие API.

5.1 Язык программирования JavaScript

JavaScript был выбран в качестве основного языка программирования для реализации задачи ввиду его расширяемости, читабельности и мультипарадигменности его предметной области. Браузерный Javascript поддерживает однопоточность и предоставляет API для работы с графикой в браузере (Canvas API), что делает его удобным инструментом для построения сложных математических структур, таких как аффинные преобразования.

5.2 Используемые API

Canvas API — это API для двумерного рисования. Он позволяет рисовать линии, фигуры, изображения и текст прямо в браузере без использования плагинов, таких как Flash или Java. Canvas изначально был создан Apple для своих виджетов, но с тех пор был принят всеми разработчиками основных браузеров и теперь является частью спецификации HTML5.

Browser API — это интерфейс прикладного программирования для веб-сервера или веб-браузера. Это концепция веб-разработки, обычно ограниченная клиентской стороной веб-приложения (включая любые используемые веб-фреймворки), и поэтому обычно не включает детали реализации веб-сервера или браузера, такие как SAPI или API, если они не доступны для общего доступа через удаленное веб-приложение.

5.3 Используемые интерфейсы

document: используется для манипуляциями с DOM (древовидная структура элементов веб страницы).

requestAnimationFrame: указывает браузеру на то, что вы хотите произвести анимацию, и просит его запланировать перерисовку на следующем кадре анимации. В качестве параметра метод получает функцию, которая будет вызвана перед перерисовкой.

6 Описание программы построения фрактала

Программа для построения стохастического фрактала Плазма реализована на языке программирования JavaScript и состоит из нескольких функций и обработчиков событий, которые обеспечивают выполнение алгоритма построения и визуализацию фрактала.

6.1 Инициализация

Программа начинается с получения элемента canvas и настройки контекста рисования. Определяются исходные координаты треугольника.

6.2 Поворот и масштабирование с использованием матриц

Для каждой итерации анимации вычисляются новые углы поворота и коэффициенты масштабирования. Используются функции для создания матриц поворота и масштабирования, а также для их комбинирования.

6.3 Визуализация

Canvas используется для создания графического окна. Треугольник рисуется с помощью преобразованных координат, а центр канваса служит точкой вращения.

6.4 Анимация

Функция requestAnimationFrame обеспечивает плавную анимацию, вызывая функцию обновления для каждого кадра. Изменяются значения углов и масштабов, создавая эффект вращения и деформации.

6.6 Завершение программы

По завершении полного цикла анимации фаза сбрасывается, и программа продолжает выполнять анимацию, создавая бесконечный эффект вращения и масштабирования.

7. Завершение работы

Рисунок 1 – Результат работы программы ч.1

Рисунок 2 – Результат работы программы ч.2

8 Вывод

В ходе выполнения лабораторной работы была успешно реализована программа для анимированного деформирования прямоугольного треугольника с использованием аффинных преобразований.

Основные достижения:

Модульная структура: Разработана четкая структура программы, что обеспечивает хорошую организацию кода и его повторное использование.

Матричные операции: Реализованы основные матричные операции для аффинных преобразований, включая вращение и масштабирование.

Интуитивный интерфейс: Создан визуальный интерфейс для демонстрации процесса деформации треугольника, позволяющий пользователю легко наблюдать изменения.

Плавная анимация: Достигнута плавность анимации благодаря эффективному использованию функции requestAnimationFrame.

Полученные навыки и знания:

Углубленное понимание: Освоены аффинные преобразования и их матричное представление, что позволило лучше понять их применение в графике.

Практический опыт: Получен опыт работы с графическими элементами в JavaScript, что расширяет навыки веб-разработки.

Создание анимаций: получен навык по созданию анимации, применяя принципы матричных преобразований и физику движения.

Возможные улучшения:

Интерактивные элементы: Добавление управления параметрами анимации в реальном времени для повышения интерактивности.

Поддержка других фигур: Расширение функциональности для работы с различными геометрическими фигурами и типами аффинных преобразований.

Оптимизация: Улучшение производительности при работе с более сложными сценами и фигурами.

Заключение:

Выполнение данной лабораторной работы позволило применить теоретические знания о аффинных преобразованиях и матричных операциях на практике. Разработанная программа наглядно демонстрирует процесс деформации объекта и может послужить основой для дальнейшего изучения и реализации более сложных геометрических преобразований в компьютерной графике.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1) ГУАП, документация для учебного процесса. – URL: https://guap.ru/regdocs/docs/uch (дата обращения 10.05.2024)

2) Руководство по JavaScript – URL: https://developer.mozilla.org/ru/docs/Web/JavaScript

3) Мартин Роберт, Чистый код: создание, анализ и рефакторинг / Мартин Роберт. — СПб.: Питер, 2022. — 464 с.

4) Фракталы: что это такое и какие они бывают – URL: https://skillbox.ru/media/code/fraktaly-chto-eto-takoe-i-kakie-oni-byvayut

5) HTML5: Canvas. Основы– URL: https://webmaster.alexanderklimov.ru/html/canvas_basic.php

6) Линейная алгебра: учебник и практикум для вузов / Е. Б. Бурмистрова, С. Г. Лобанов — Москва.: Издательство Юрайт, 2024. — 421 с.

7) Основы компьютерной графики: учебник и практикум для вузов/ А. В. Боресков, Е. В. Шикин — Москва.: Издательство Юрайт, 2024. — 219 с.

ПРИЛОЖЕНИЕ

Исходный код программы

```
const canvas = document.getElementById("canvas");
const ctx = canvas.getContext("2d");

const triangle = [
    [-50, 0],
    [0, 50],
    [50, 0],
];

function multiplyMatrixAndVector(matrix, vector) {
    const [x, y] = vector;
    return [
        matrix[0][0] * x + matrix[0][1] * y + matrix[0][2],
        matrix[1][0] * x + matrix[1][1] * y + matrix[1][2],
        1,
    ];
}

function scaleMatrix(sx, sy) {
    return [
        [sx, 0, 0],
        [0, sy, 0],
        [0, 0, 1],
    ];
}

function rotationMatrix(theta) {
    return [
        [Math.cos(theta), -Math.sin(theta), 0],
        [Math.sin(theta), Math.cos(theta), 0],
        [0, 0, 1],
    ];
```

```
}

function multiplyMatrices(a, b) {
    const result = [];
    for (let i = 0; i < 3; i++) {
        result[i] = [];
        for (let j = 0; j < 3; j++) {
            result[i][j] =
                a[i][0] * b[0][j] + a[i][1] * b[1][j] + a[i][2] * b[2][j];
        }
    }
    return result;
}

function drawTriangle(matrix) {
    ctx.clearRect(0, 0, canvas.width, canvas.height);
    ctx.save();

    ctx.translate(canvas.width / 2, canvas.height / 2);

    ctx.beginPath();
    triangle.forEach(vertex => {
        const transformed = multiplyMatrixAndVector(matrix, [
            vertex[0],
            vertex[1],
            1,
        ]);
        if (vertex === triangle[0]) {
            ctx.moveTo(transformed[0], transformed[1]);
        } else {
            ctx.lineTo(transformed[0], transformed[1]);
        }
    });
    ctx.closePath();
    ctx.fillStyle = "red";
```

```
    ctx.fill();
    ctx.restore();
}

let scaleX = 1;
let scaleY = 1;
let phase = 0;

function animate() {
    phase += 0.02;

    const rotationAngle = Math.PI / 4;

    let transformationMatrix = rotationMatrix(rotationAngle);

    if (phase < Math.PI / 2) {
        scaleX = 1 + Math.sin(phase) * 0.5;
        scaleY = 1;
    } else if (phase < Math.PI) {
        scaleY = 1 + Math.sin(phase - Math.PI / 2) * 0.5;
        scaleX = 1;
    } else {
        phase = 0;
    }

    const scaleMatrixX = scaleMatrix(scaleX, scaleY);
    transformationMatrix = multiplyMatrices(scaleMatrixX, transformationMatrix);

    drawTriangle(transformationMatrix);
    requestAnimationFrame(animate);
}

animate();
```