МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет автоматики и вычислительной техники
Кафедра автоматики

ЛАБОРАТОРНАЯ РАБОТА №2

по дисциплине: «Компьютерная графика»
Фильтрация

Выполнили:
Студенты гр. АВТ-418
Юрков Владислав Юрьевич
Куриленко Платон Семенович
«___» _____ 2025 г.

Проверил:
Надежницкая В. А.

_____
(оценка, подпись)

Новосибирск
2025

Цель работы: Изучить масочную фильтрацию

Задание:

1. Сгенерировать изображение, используя масочную фильтрацию (взвешенная маска).
2. Изменить маску и ещё раз сгенерировать изображение.

Теоретические основы:

Усредняющая маска перемещается по изображению с шагами, равными ее размеру, улучшая качество изображения. Усреднение можно за счет использования весов, задающих влияние отдельных подпикселов на атрибут пиксела экрана

Практическая часть:

Реализовано применение мисочной фильтрации с помощью WEB-технологий: каждый алгоритм рисует один и тот же отрезок разным цветом.

Используемые технологии

Typescript, Vue, Nuxt, Tailwind CSS, canvas, Daysi UI

Организация кода

- Nuxt архитектура с SSR для просмотра всех лабораторных работ
- Серверные компоненты для реализации подсветки синтаксиса и интеграции кода в компоненты для отрисовки

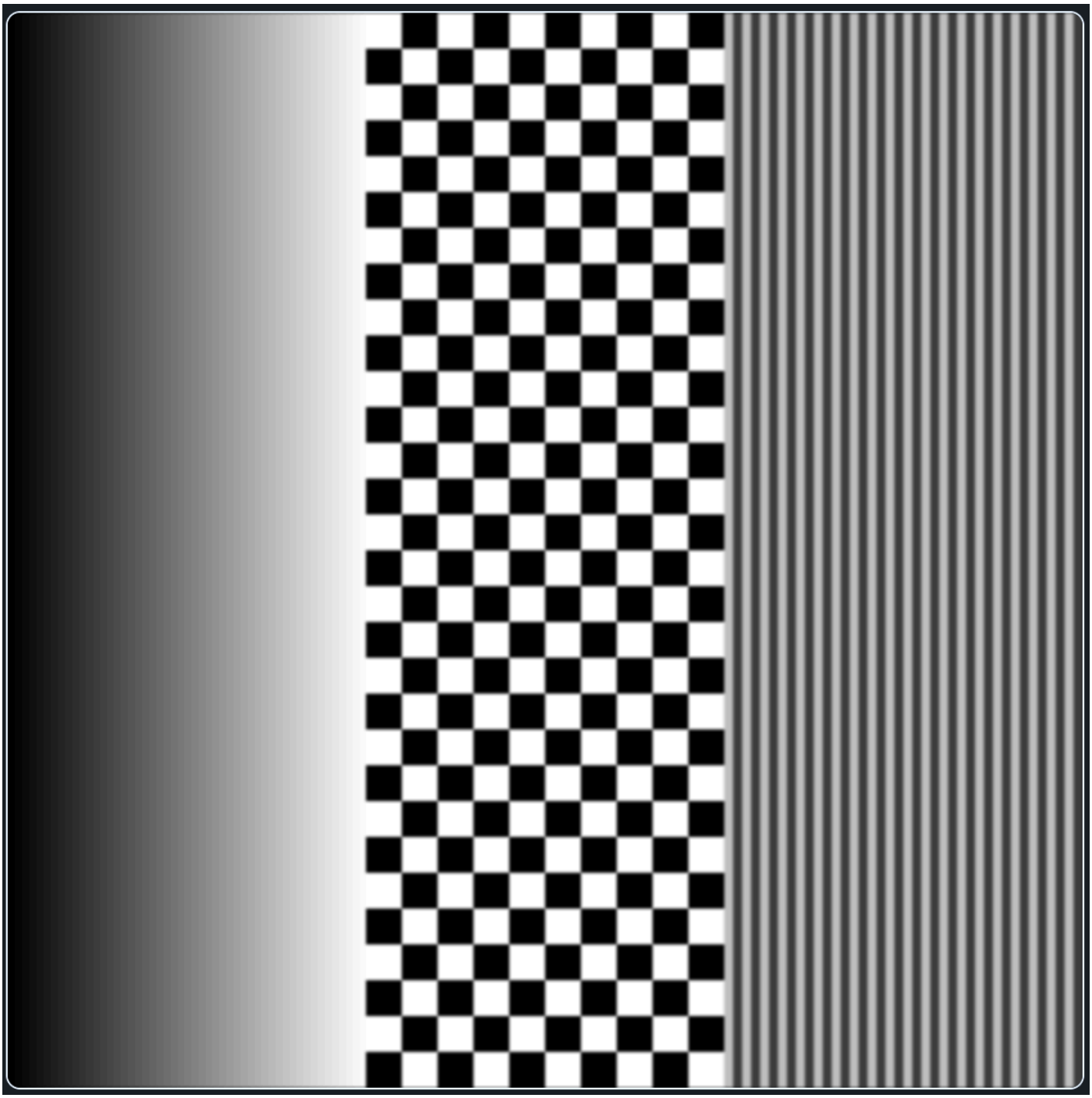

Рисунок 1 - Изображение со взвешенной маской 3x3

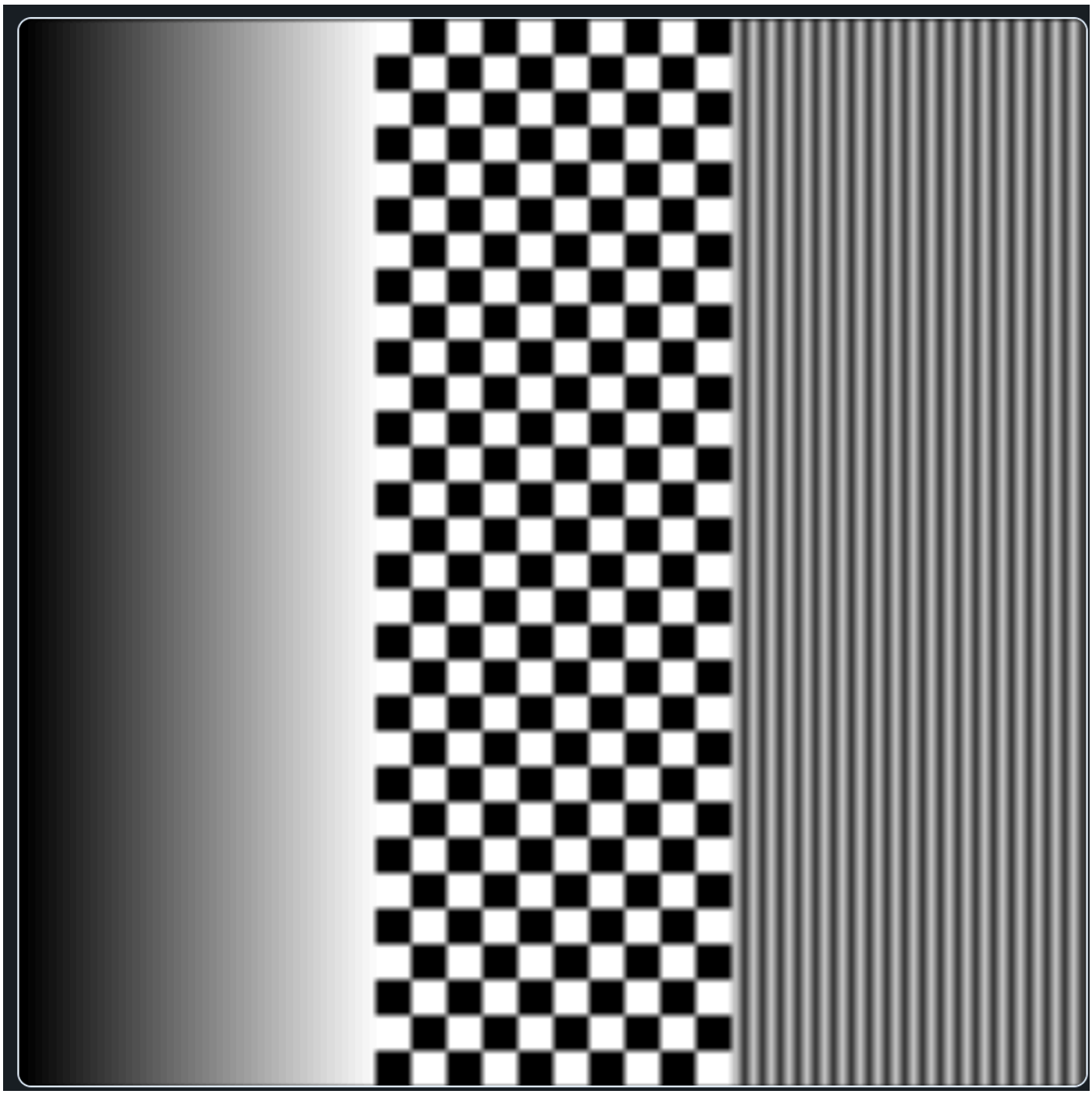

Рисунок 2 - Изображение с усредненной маской 4x4

Вывод: В ходе лабораторной работы была изучена маточная фильтрация различных видов

Приложение

```ts
export enum MaskType {
    ORIGINAL = 'o',
    UNIFORM_2X2 = 'u2',
    UNIFORM_4X4 = 'u4',
    WEIGHTED_3X3 = 'w3',
    WEIGHTED_5X5 = 'w5',
}

const masks: Record<Exclude<MaskType, MaskType.ORIGINAL>, number[][]> = {
    [MaskType.UNIFORM_2X2]: [
        [1 / 4, 1 / 4],
        [1 / 4, 1 / 4],
    ],
    [MaskType.UNIFORM_4X4]: [
        [1 / 16, 1 / 16, 1 / 16, 1 / 16],
        [1 / 16, 1 / 16, 1 / 16, 1 / 16],
        [1 / 16, 1 / 16, 1 / 16, 1 / 16],
        [1 / 16, 1 / 16, 1 / 16, 1 / 16],
    ],
    [MaskType.WEIGHTED_3X3]: [
        [1 / 16, 2 / 16, 1 / 16],
        [2 / 16, 4 / 16, 2 / 16],
        [1 / 16, 2 / 16, 1 / 16],
    ],
    [MaskType.WEIGHTED_5X5]: [
        [1 / 25, 2 / 25, 3 / 25, 2 / 25, 1 / 25],
        [2 / 25, 4 / 25, 6 / 25, 4 / 25, 2 / 25],
        [3 / 25, 6 / 25, 9 / 25, 6 / 25, 3 / 25],
        [2 / 25, 4 / 25, 6 / 25, 4 / 25, 2 / 25],
        [1 / 25, 2 / 25, 3 / 25, 2 / 25, 1 / 25],
    ],
};

const clamp = (min: number, value: number, max: number) => Math.max(min, Math.min(max, Math.round(value)));
```

```ts
export default function render(
    ctx: CanvasRenderingContext2D,
    maskType: MaskType = MaskType.ORIGINAL,
) {
    const { width, height } = ctx.canvas;

    function generateTestImage(): ImageData {
        const img = ctx.createImageData(width, height);
        const d = img.data;
        for (let y = 0; y < height; y++) {
            for (let x = 0; x < width; x++) {
                const i = (y * width + x) * 4;
                if (x < width / 3) {
                    const g = Math.floor((x / (width / 3)) * 255);
                    d[i] = d[i + 1] = d[i + 2] = g;
                }
                else if (x < 2 * width / 3) {
                    const size = 20;
                    const c = ((Math.floor(x / size) + Math.floor(y / size)) % 2) ? 0 : 255;
                    d[i] = d[i + 1] = d[i + 2] = c;
                }
                else {
                    const stripeWidth = 5;
                    const c = (Math.floor(x / stripeWidth) % 2) ? 64 : 192;
                    d[i] = d[i + 1] = d[i + 2] = c;
                }
                d[i + 3] = 255;
            }
        }
        return img;
    }

    function applyMask(img: ImageData, mask: number[][]): ImageData {
        const src = img.data;
        const dst = new Uint8ClampedArray(src);

        const mh = mask.length;
        const mw = mask[0]!.length;

        const oy = Math.floor(mh / 2), ox = Math.floor(mw / 2);

        for (let y = 0; y < height; y++) {
            for (let x = 0; x < width; x++) {
                let r = 0, g = 0, b = 0;
                for (let my = 0; my < mh; my++) {
                    for (let mx = 0; mx < mw; mx++) {
                        const sy = y + my - oy, sx = x + mx - ox;
                        if (sy >= 0 && sy < height && sx >= 0 && sx < width) {
                            const si = (sy * width + sx) * 4;
                            const w = mask[my]![mx]!;
                            r += src[si]! * w;
                            g += src[si + 1]! * w;
                            b += src[si + 2]! * w;
                        }
                    }
                }
                const di = (y * width + x) * 4;
                dst[di] = clamp(0, r, 255);
                dst[di + 1] = clamp(0, g, 255);
                dst[di + 2] = clamp(0, b, 255);
            }
        }
        return new ImageData(dst, width, height);
    }

    const origin = generateTestImage();

    if (maskType === MaskType.ORIGINAL) {
        ctx.putImageData(origin, 0, 0);
        return;
    }

    const mask = masks[maskType];
    const filtered = applyMask(origin, mask);
    ctx.putImageData(filtered, 0, 0);
}
```