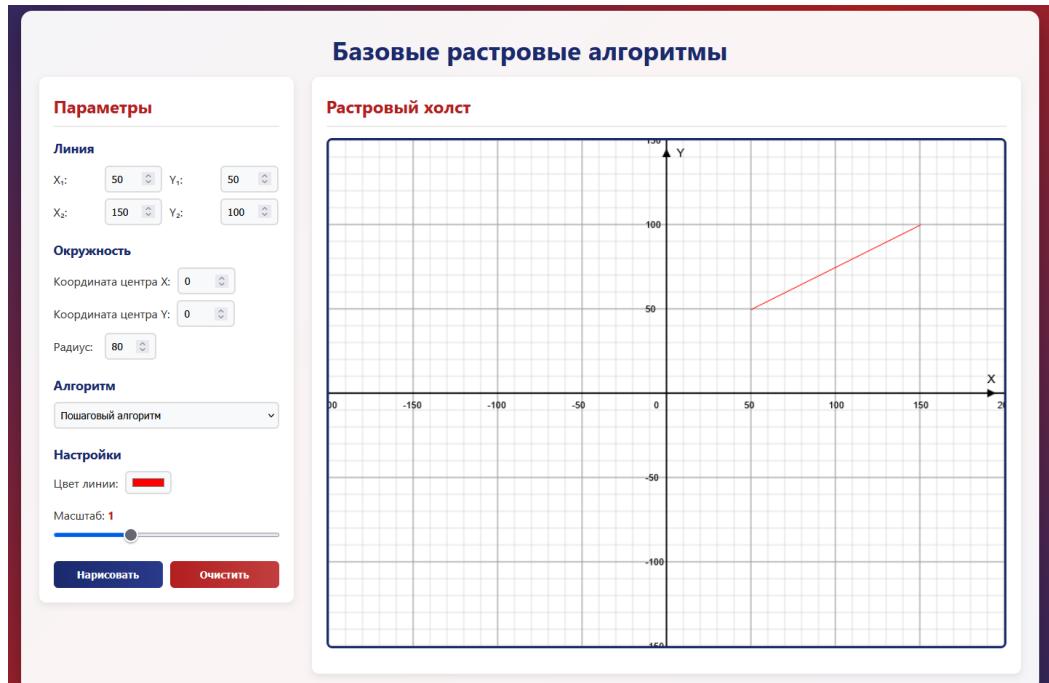


В данной работе изучал применение различных алгоритмов растеризации: пошаговый алгоритм, алгоритм ЦДА, алгоритм Брезенхема для отрисовки отрезка и окружности.

Код на html, css и javascript.

Начальная страница приложения выглядит как-то так:



Помимо основного функционала, у приложения есть возможность регулировать масштаб и выбирать цвет из палитры.

На примере **алгоритма Брезенхема для отрисовки отрезка** разберём, как написана программа и какой алгоритм:

Допустим, мы хотим нарисовать отрезок с концами в точках $P_1 = (x_1, y_1) = (0, 50)$ и $P_2 = (x_2, y_2) = (150, 70)$.

Формально, первым делом я умножаю эти координаты на 2 (как и оси и шкалы), чтобы картинка стала вдвое больше и было лучше видно. Теперь $P'_1 = (0, 100)$, $P'_2 = (300, 140)$.

Далее инициализируем переменные:

$$\begin{aligned}x &\leftarrow x'_1 = 0 \\y &\leftarrow y'_1 = 100 \\\Delta x &\leftarrow |x'_2 - x'_1| = |300 - 0| = 300 \\\Delta y &\leftarrow |y'_2 - y'_1| = |140 - 100| = 40 \\s_x &\leftarrow \text{sign}(x'_2 - x'_1) = 1 \\s_y &\leftarrow \text{sign}(y'_2 - y'_1) = 1 \\\varepsilon &\leftarrow \Delta x - \Delta y = 300 - 40 = 260\end{aligned}$$

После этого используем целочисленную модификацию алгоритма Брезенхема, где мы вначале итерации округляем нашу точку до целой, потом считаем новую ошибку $e2$ и если ей соответствует отклонение по оси Oy больше 0.5, округляем вверх, а меньше – вниз, примерно также, как было сказано в презентации, за тем лишь исключением, что мы проверяем не знак ошибки, а то, больше или меньше она нужной нам погрешности.

```
while (X<=X2) {
    if (E' ≥0) {
        X= X + 1;
        Y= Y + 1;
        E'= E' + 2*(Dy - Dx);
    } else {
        X= X + 1;
        E'= E' + 2*Dy;
    }
    Очередная точка вектора
    PutPixel(X, Y);
}
```

Вычисления для первых 12 шагов

Шаг	(x, y)	ε	$\varepsilon_2 = 2\varepsilon$	$\varepsilon_2 > -\Delta y?$	$\varepsilon_2 < \Delta x?$	Действие
1	(0, 100)	260	520	да	нет	$x \leftarrow 1, \varepsilon \leftarrow 220$
2	(1, 100)	220	440	да	нет	$x \leftarrow 2, \varepsilon \leftarrow 180$
3	(2, 100)	180	360	да	нет	$x \leftarrow 3, \varepsilon \leftarrow 140$
4	(3, 100)	140	280	да	да	$x \leftarrow 4, y \leftarrow 101, \varepsilon \leftarrow 400$
5	(4, 101)	400	800	да	нет	$x \leftarrow 5, \varepsilon \leftarrow 360$
6	(5, 101)	360	720	да	нет	$x \leftarrow 6, \varepsilon \leftarrow 320$
7	(6, 101)	320	640	да	нет	$x \leftarrow 7, \varepsilon \leftarrow 280$
8	(7, 101)	280	560	да	нет	$x \leftarrow 8, \varepsilon \leftarrow 240$
9	(8, 101)	240	480	да	нет	$x \leftarrow 9, \varepsilon \leftarrow 200$
10	(9, 101)	200	400	да	нет	$x \leftarrow 10, \varepsilon \leftarrow 160$
11	(10, 101)	160	320	да	нет	$x \leftarrow 11, \varepsilon \leftarrow 120$
12	(11, 101)	120	240	да	да	$x \leftarrow 12, y \leftarrow 102, \varepsilon \leftarrow 380$

Ниже приведу листинг своего кода:

```
drawBresenhamLine(color) {
    const x1 = 2 * parseInt(document.getElementById('x1').value);
    const y1 = 2 * parseInt(document.getElementById('y1').value);
    const x2 = 2 * parseInt(document.getElementById('x2').value);
    const y2 = 2 * parseInt(document.getElementById('y2').value);

    let x = x1;
    let y = y1;
    let dx = Math.abs(x2 - x1);
    let dy = Math.abs(y2 - y1);
    let sx = (x1 < x2) ? 1 : -1;
    let sy = (y1 < y2) ? 1 : -1;
    let err = dx - dy;

    this.ctx.save();
    const scale = parseFloat(document.getElementById('scale').value)
    this.ctx.scale(scale, scale);

    const centerX = this.canvas.width / (2 * scale);
    const centerY = this.canvas.height / (2 * scale);

    this.ctx.fillStyle = color;

    while (true) {
        const pixelX = Math.round(centerX + x);
        const pixelY = Math.round(centerY - y);
```

```
        this.ctx.fillRect(pixelX, pixelY, 1, 1);

        if (x === x2 && y === y2) break;

        let e2 = 2 * err;
        if (e2 > -dy) {
            err -= dy;
            x += sx;
        }
        if (e2 < dx) {
            err += dx;
            y += sy;
        }
    }

    this.ctx.restore();

    const endTime = performance.now();
    this.timeResults.bresenham = (endTime - startTime).toFixed(3);
    this.updateTimeDisplay();
}
```

Выводы. Таким образом, я разобрался, как выполнять простейшие алгоритмы растеризации, используя идеи приближения целыми точками и 8-связности. Вдобавок, визуальная часть приложения выглядит довольно-таки мило.