



UFRR

UNIVERSIDADE FEDERAL DE RORAIMA
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

CLEUSON SOUSA SANTOS

COMPUTAÇÃO GRÁFICA: ALGORITMOS DE PREENCHIMENTO

Boa Vista – Roraima,
Novembro de 2022

CLEUSON SOUSA SANTOS

COMPUTAÇÃO GRÁFICA: ALGORITMOS DE PREENCHIMENTO

Relatório apresentada ao Departamento de Ciência da Computação da Universidade Federal de Roraima, como requisito parcial para a obtenção de nota na Disciplina Computação Gráfica.

Orientador

ProfºDr. Luciano Ferreira Silva

Boa Vista – Roraima,
Novembro de 2022

RELATÓRIO

Discente: Cleuson Sousa Santos

Tema: Preenchimento de polígonos por meios de algoritmos: Flood Fill e Análise de Contorno Geométrico.

O programa desenvolvido permite preencher polígonos por meios dos algoritmos apresentados em sala de aula: Flood Fill e Análise de Contorno Geométrico. Implementado de maneira que se pode ver o trabalho de cada algoritmo e comparar os resultados.

1. Grid de Pixels

Segundo o padrão utilizado pela indústria, e o que consta na literatura, o sistema de de coordenadas de pixel, define a origem de um destino de renderização no canto superior esquerdo, e avança por coordenadas positivas no eixo x para a direita e no eixo y para baixo.

Com a finalidade de tornar a rasterização mais didática e intuitiva, levou-se em consideração o sistema de coordenadas de pixels, no entanto, foi dada preferência e por transformar os sistema de coordenadas para o sistema euclidiano mais próximo do utilizado nas aulas, de tal maneira que torne melhor compreensível o resultado pretendido, tanto na renderização do resultado quanto na comparação de resultados.

A grid foi implemetado como uma malha com 80 pixels de altura, com 80 pixels de largura, mais uma coluna e linha para representar o valor 0. A malha foi dividida em valores variando de -40 a 40, tanto no eixo x, quanto no eixo y.

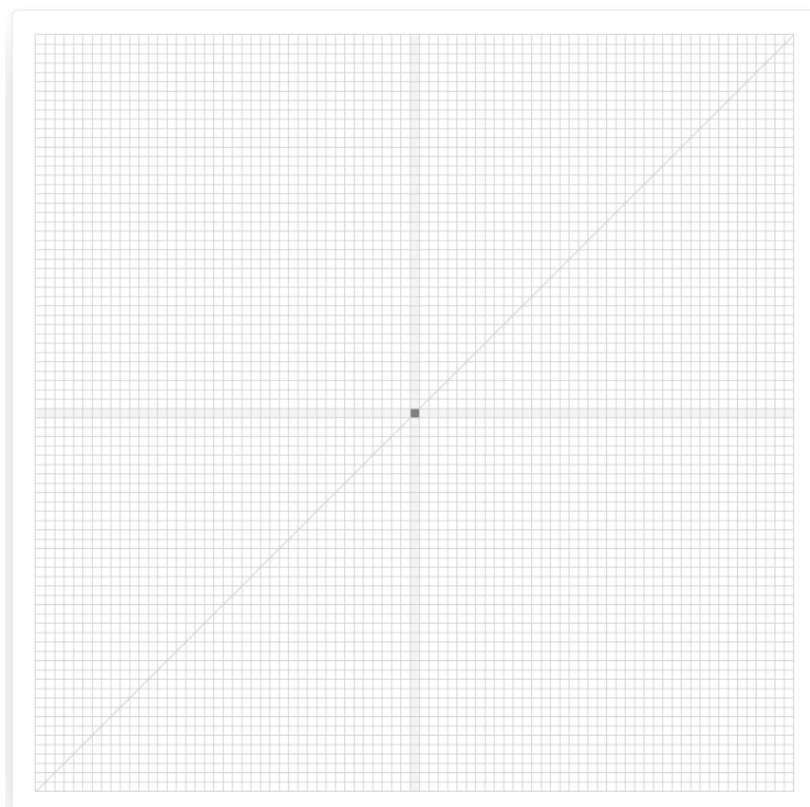


Fig 1. Malha de pixels

2. Preenchimento

A escolha e comparação dos algoritmos, bem como argumentos, devem ser inseridos diretamente no código fonte.

3. Implementação

O programa foi implementado com JavaScript, utilizando o Framework NextJS.

a. Algoritmo Flood Fill

```
export function floodFill(poligon, seed, array) {  
  
    if (seed.x > 40 || seed.x < -40 || seed.y > 40 || seed.y < -40) {  
        return array;  
    }  
  
    array.push(seed);  
  
    if (!contains(poligon, Point(seed.x + 1, seed.y))) {  
        if (!contains(array, Point(seed.x + 1, seed.y))) {  
            floodFill(poligon, Point(seed.x + 1, seed.y), array);  
        }  
    }  
  
    if (!contains(poligon, Point(seed.x - 1, seed.y))) {  
        if (!contains(array, Point(seed.x - 1, seed.y))) {  
            floodFill(poligon, Point(seed.x - 1, seed.y), array);  
        }  
    }  
  
    if (!contains(poligon, Point(seed.x, seed.y + 1))) {  
        if (!contains(array, Point(seed.x, seed.y + 1))) {  
            floodFill(poligon, Point(seed.x, seed.y + 1), array);  
        }  
    }  
  
    if (!contains(poligon, Point(seed.x, seed.y - 1))) {  
        if (!contains(array, Point(seed.x, seed.y - 1))) {  
            floodFill(poligon, Point(seed.x, seed.y - 1), array);  
        }  
    }  
  
    return array;  
}
```

b. Algoritmo Análise de Contorno Geométrico.

```
export function geometric( vertices ) {

    if(vertices.length === 0){
        return [];
    }

    let pixels = [];
    let intersects = [];

    /* Define os Extremos do Poligono */
    let ymin = getYMin(vertices);
    let xmin = getXMin(vertices);

    let ymax = getYMax(vertices);
    let xmax = getXMax(vertices);

    /* Cria Tabela de Aresta Global */
    let arestas = [];
    for (let i = 0; i < vertices.length - 1; i++) {

        // Remove Arestas Horizontais
        if (vertices[i].y !== vertices[i + 1].y) {

            // Inverte Coordenadas no Eixo Y
            if (vertices[i].y < vertices[i + 1].y) {
                arestas.push({ "start": vertices[i], "end": vertices[i + 1]
            });
            } else {
                arestas.push({ "start": vertices[i + 1], "end": vertices[i]
            });
            }
        }
    }

    // Adiciona Ultima Aresta: Do ponto final, para pontos inicial
    if (vertices[0].y !== vertices[vertices.length - 1].y) {
        // Inverte Coordenadas no Eixo Y
        if (vertices[0].y < vertices[vertices.length - 1].y) {
            arestas.push({ "start": vertices[0], "end":
vertices[vertices.length - 1] });
        } else {
            arestas.push({ "start": vertices[vertices.length - 1], "end":
vertices[0] });
        }
    }

    /* Ordenação por ymin e depois por x */
    let arestasOrdenadas = [];
    let i = ymin;
    for (i; i <= ymax; i++) {
        let array = arestas.filter(aresta => aresta.start.y === i);
        arestasOrdenadas = arestasOrdenadas.concat(array);
    }
    arestas = arestasOrdenadas;

    // um cesto para cada linha de varredura

    //Tabela de arestas ativas
    //Somente tocada pela linha de varredura
    //Atualiza a medida que a linha segue
    let arestasAtivas = [];

    //1. set y = ymin
    let y = ymin;
```

```

//2. verifica no y-bucket
//3. insere arestas necessarias no AET
while (y <= ymax) {

    //arestasAtivas = arestasAtivas.concat(arestas.filter(aresta =>
aresta.start.y === y));
    arestasAtivas = arestas.filter(aresta => aresta.start.y === y);

    //5. ordene por valores de X e preencha entre pares sucessivos
    arestasAtivas.forEach(
        element => {

            let xi = element.start.x;
            let xf = element.end.x;
            let yi = element.start.y;
            let yf = element.end.y;

            let dx = element.end.x - xi;
            let dy = element.end.y - element.start.y;

            while (yi <= yf) {
                intersects.push(Point(Math.round(xi), yi));
                xi = xi + dx / dy;
                yi = yi + 1;
            }
        }
    );
    Y++;
}

// Remove aresta das tabelas ativas
arestasAtivas = arestasAtivas.filter(aresta => aresta.end.y < y);

// Preenchimento
y = ymin;
while (y <= ymax) {

    let bucket = intersects.filter(intersect => intersect.y === y);
    let pool = clean(poligonOrderX(bucket)).filter(unique);

    log(pool);

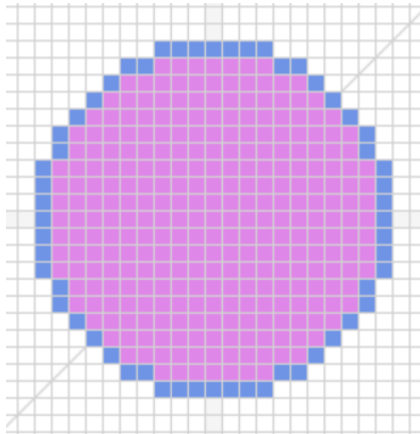
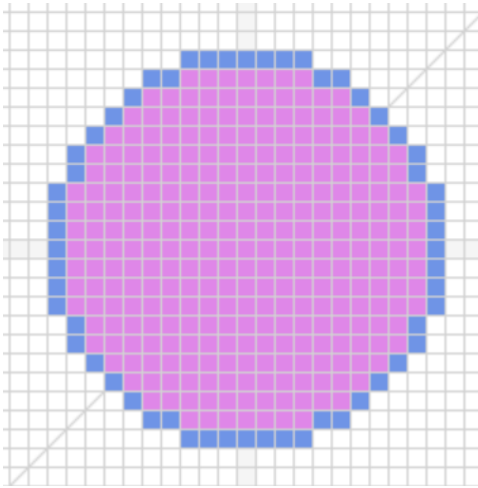
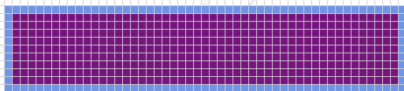
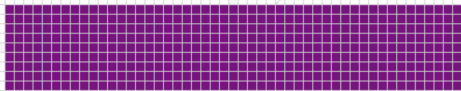
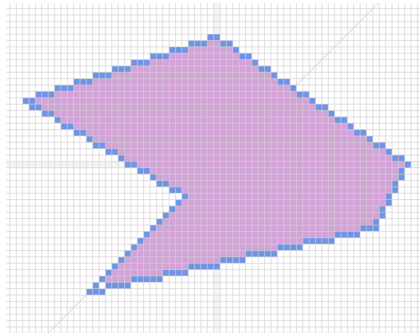
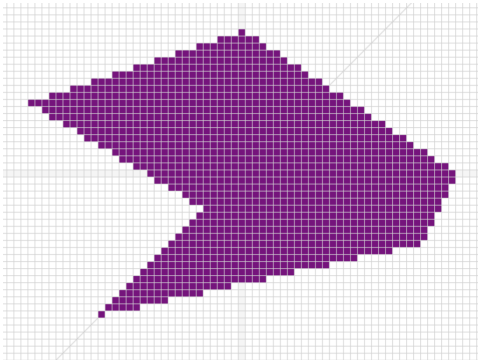
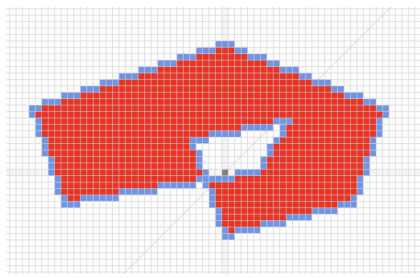
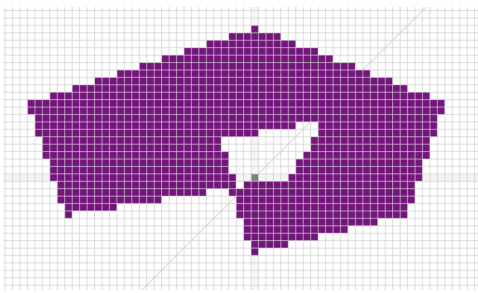
    if (pool.length % 2 === 0) {

        for (let i = 0; i < pool.length; i += 2) {

            let j = pool[i];
            while (j < pool[i + 1]) {
                pixels.push(Point(j, y));
                j++;
            }
        }
    } else {
        for (let i = 0; i < pool.length - 1; i++) {
            let j = pool[i];
            while (j < pool[i + 1]) {
                pixels.push(Point(j, y));
                j++;
            }
        }
    }
    Y++;
}
return intersects = intersects.concat(pixels);
}
}

```

5. Comparação

Coordernada	Flood Fill	Análise de Contorno Geométrico
Circulo		
Retangulo		
Polígono A		
Polígono B		

6. Considerações.

- O Algoritmo Flood Fill, apresenta maior facilidade de operação e operação, no entanto, exige a existência de um contorno prévio para poder fazer o preenchimento. Embora utilize recursividade, não apresentea operações de ponto flutuante.
- O algoritmo de análise de contorno geométrico, apresenta grande complexidade de implementação, e a ainda existe problemas a resolver quanto a vértices fora da forma, o que necessita a implementação de algoritmo de decisão para de o vértice é para

dentro ou para forma da forma. No entanto, apresenta a facilidade de não ser necessário estabelecer contorno para que se obtenha a forma preenchida.