UNIVERSIDADE FEDERAL DE RORAIMA

CENTRO DE CIÊNCIAS E TECNOLOGIA

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

COMPUTAÇÃO GRÁFICA

# ALGORITMOS DE RESTERIZAÇÃO DE LINHAS, CIRCUNFERÊNCIAS E PREENCHIMENTO

FABIO VITOR DE OLIVEIRA NORONHA

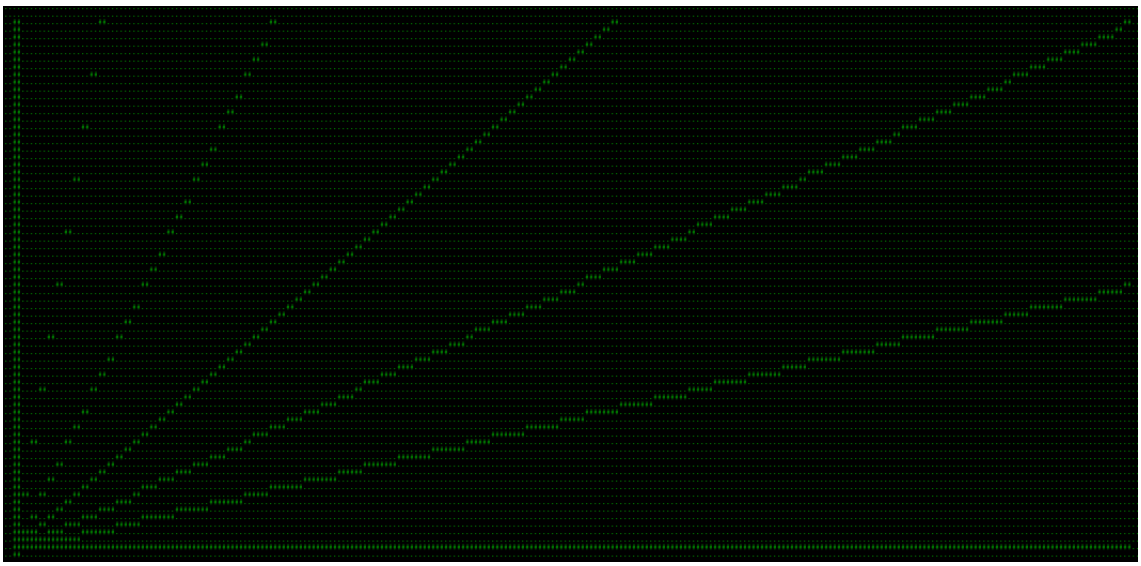
BOA VISTA, RR

NOVEMBRO DE 2017

# RASTERIZAÇÃO DE LINHAS

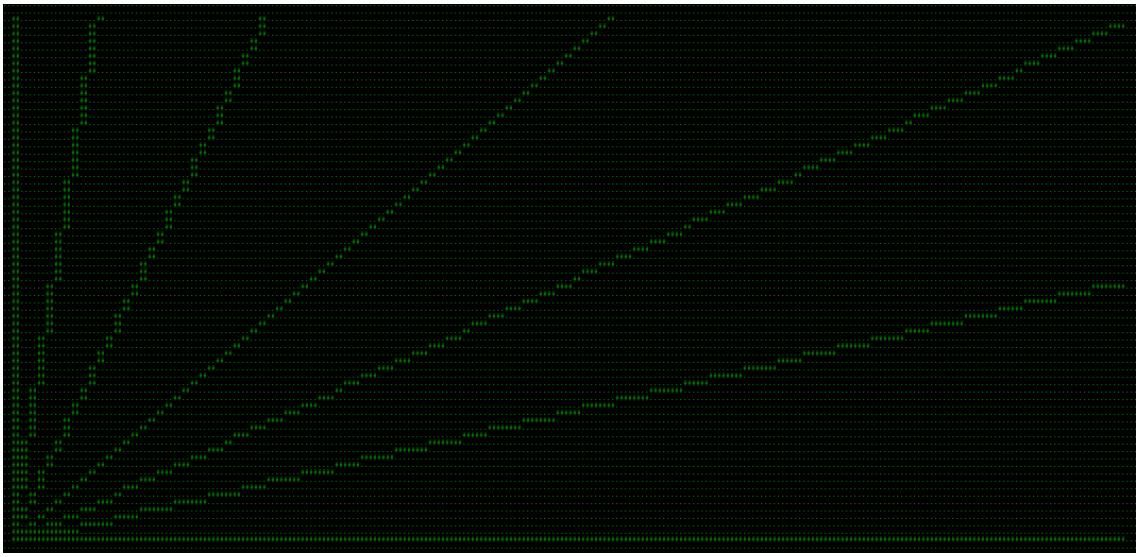
## MÉTODO ANALÍTICO

O método analítico mostrou-se aceitável apenas em linhas cuja angulação varia de 0º a 45º. Como mostra a imagem abaixo pode-se notar que as retas com angulação entre 45º e 90º ficam com falhas. As demais linhas estão representando bem as retas desejadas, porém seu ponto fraco é: O algoritmo trabalha com muitos cálculos com pontos flutuantes (O que é custoso para o computador) e como a tela é uma malha de pixels com posições inteiras, o mesmo faz arredondamentos desses números reais, o que causa, talvez, pequenos erros.



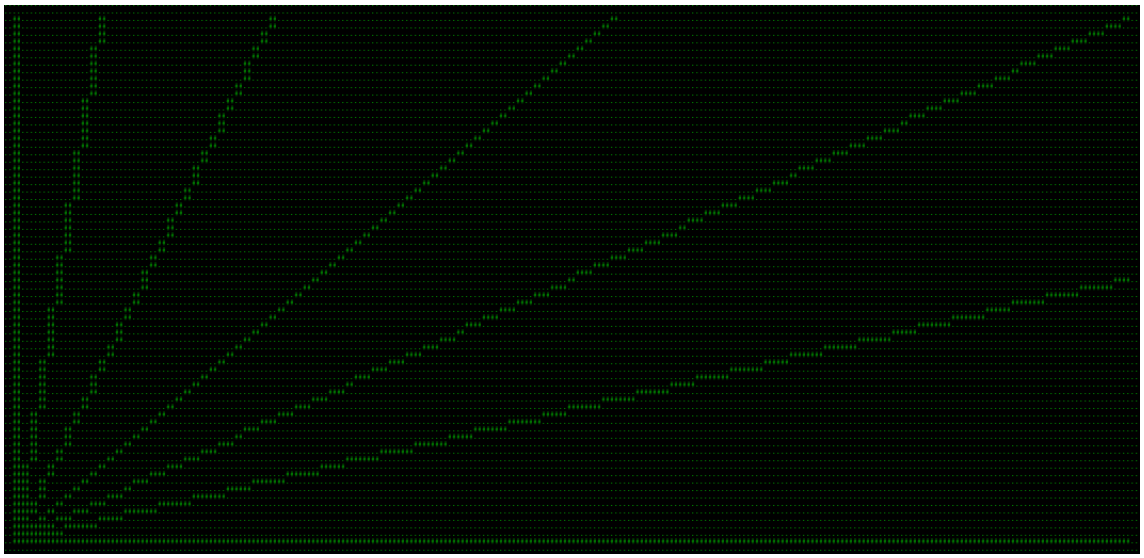
## MÉTODO DDA

Já o método DDA funciona aceitavelmente bem para retas de todas as inclinações como mostra a imagem abaixo. Seu ponto fraco é que, por ter uma abordagem parecida com o método analítico o método DDA trabalha com números de ponto flutuante e arredonda os resultados, o que é pesado de processar pelo computador e causa pequenos erros.



## ALGORITMO DE BRESENHAM

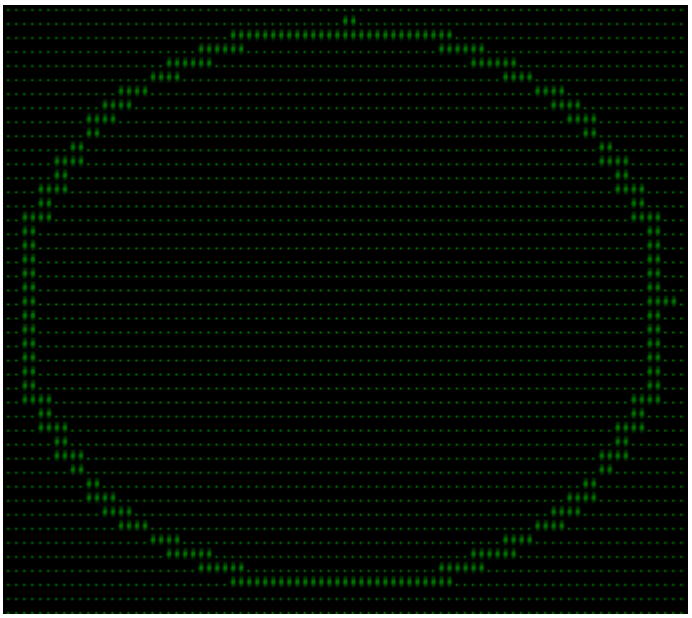
O algoritmo de Bresenham, por sua vez, além de funcionar para retas de todas as angulações, trabalha apenas com operações envolvendo números inteiros, sendo assim o melhor em desempenho entre os três algoritmos estudados. Outro fator positivo é que ele toma as decisões “conscientemente”, ou seja, sem arredondamentos. Atualmente é o mais usado para a rasterização de retas. A aplicação deste algoritmo pode ser vista na imagem abaixo. Repare que é bem similar ao resultado do método DDA, salvo por algumas pequenas mudanças causadas pelo arredondamento que o método DDA faz.



## RASTERIZAÇÃO DE CIRCUNFERÊNCIAS

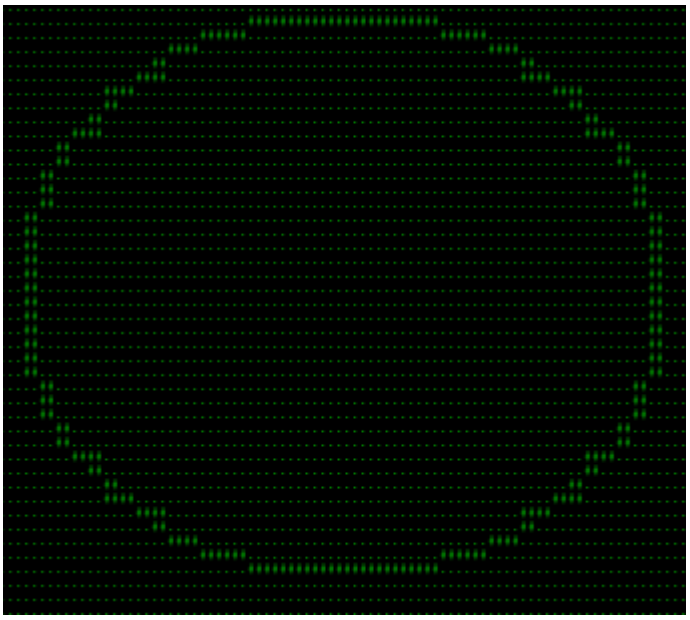
## EQUAÇÃO PARAMÉTRICA

O algoritmo que utiliza a equação paramétrica como base mostrou-se o menos eficiente, porém aceitável. Seus pontos fracos foram que ele não possui total simetria em alguns casos como mostrado abaixo na imagem, representa todas as circunferências utilizando o mesmo número de pontos (360 pontos), calculando todos estes sempre, resultado: o mesmo pixel pode ser pintado muitas vezes em casos de circunferências pequenas, falhas podem acontecer no caso de grandes circunferências e a mesma quantidade de operações com pontos flutuantes é realizada.



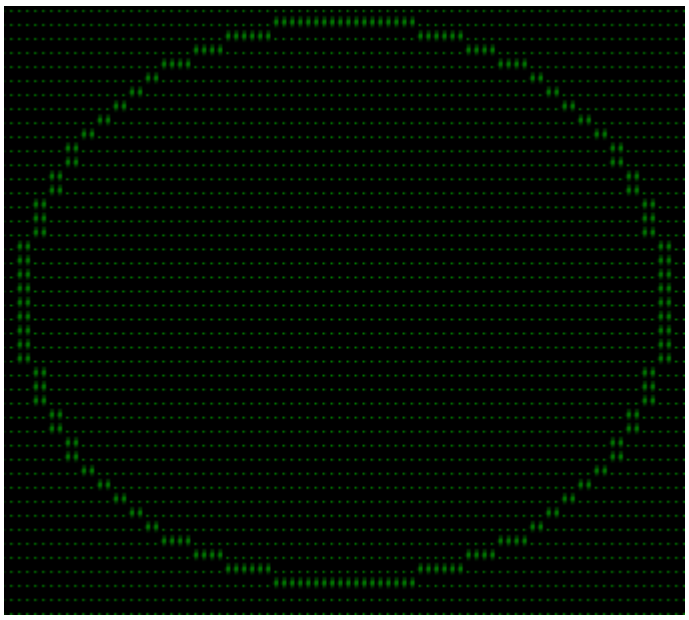
## ALGORITMO INCREMENTAL COM SIMETRIA

Possui uma abordagem mais inteligente em relação ao anterior, representa bem as circunferências de todos os tamanhos. Este, por sua vez, calcula apenas 1/8 da circunferência e utiliza da simetria da mesma para encontrar os pixels nos outros octetos. O problema é que ele, assim como o da equação paramétrica, faz muitas operações com números de ponto flutuante, o que consome muito tempo de processamento.

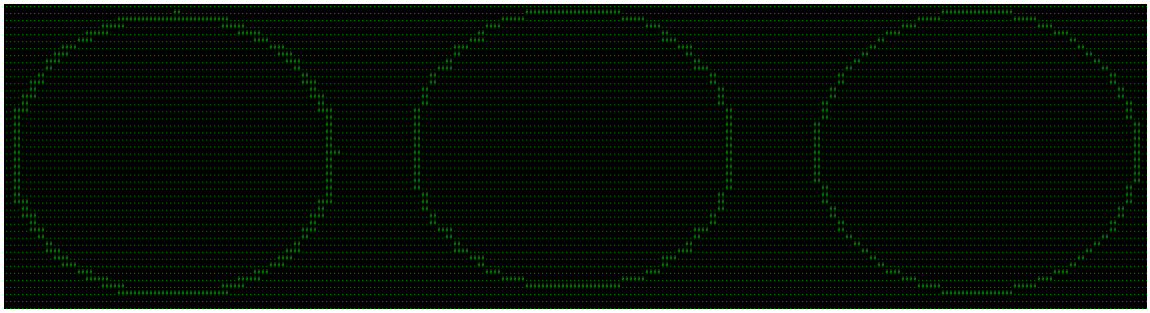


## ALGORITMO DE BRESENHAM PARA CIRCUNFERÊNCIAS

Este foi algoritmo mais satisfatório dentre os três algoritmos de rasterização de circunferências. Como utiliza apenas cálculos com números inteiros é o mais eficiente e assim como anterior calcula apenas 1/8 da circunferência. Atualmente é o mais utilizado para este tipo de rasterização.



Abaixo podemos ver os três exemplos em comparação, da esquerda para a direita o algoritmo da equação paramétrica, incremental por simetria e o de Bresenham.



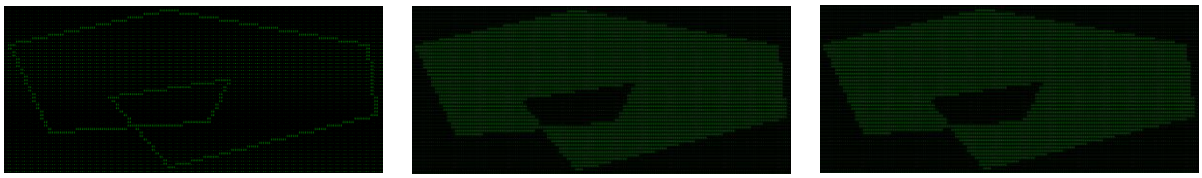
## ALGORITMOS DE PREENCHIMENTO

## *FLOOD FILL* E VARREDURA COM ANÁLISE GEOMÉTRICA

Os dois algoritmos possuem premissas diferentes no que se diz respeito ao preenchimento de figuras. O *flood fill* obteve resultados mais aceitáveis nas figuras requeridas enquanto o de varredura com análise geométrica obteve algumas inconsistências que serão comentadas mais à frente.

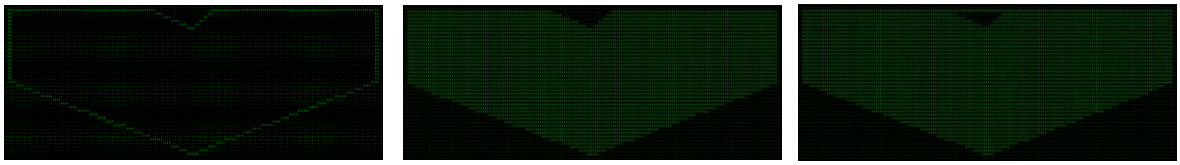
Na imagem abaixo vemos a primeira figura que deveria ser pintada e logo abaixo a aplicação dos dois algoritmos em ordem de apresentação. Note que o resultado é idêntico para a mesma.

A segunda figura, mostrada abaixo, também obteve o mesmo resultado com a aplicação dos dois algoritmos.

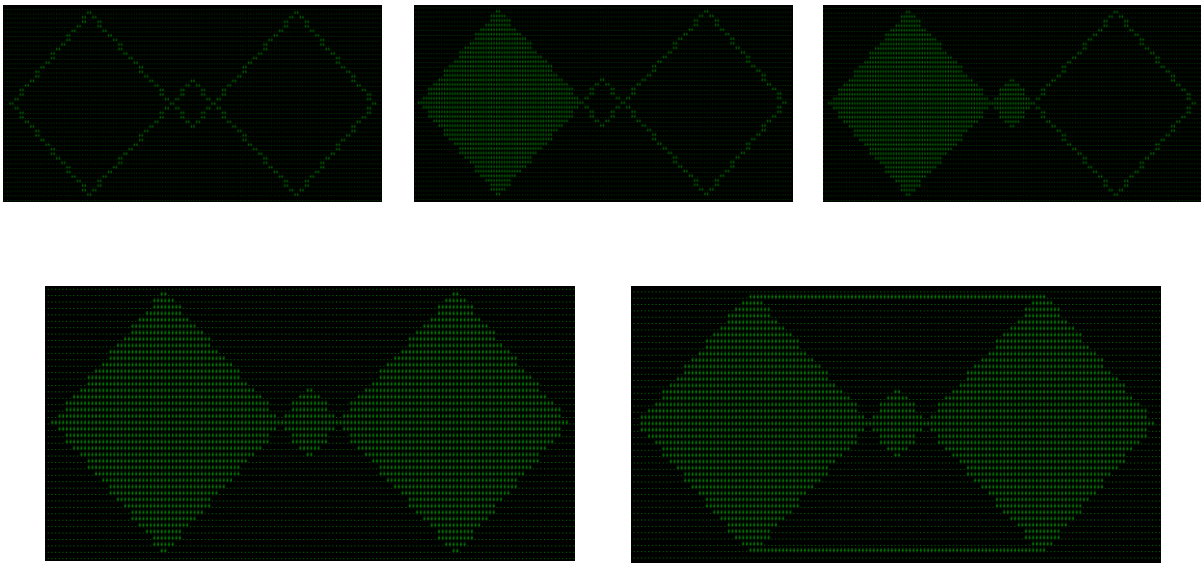


A terceira imagem mostra uma diferença: uma linha horizontal na primeira fileira de pixels da imagem que pode ser vista na terceira imagem. Isso se deve

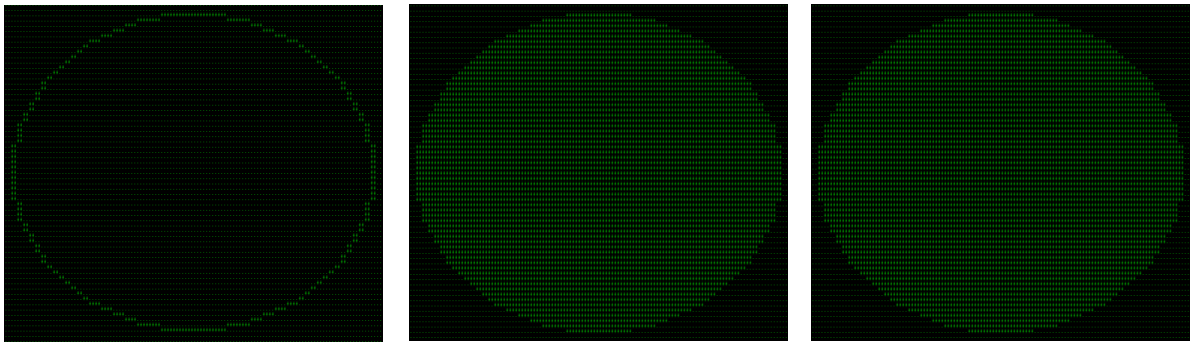
ao fato do algoritmo não cessar o preenchimento quando encontro o último ponto antes da abertura pois o mesmo fazia parte das coordenadas da figura. O *flood fill* funcionou corretamente.



Abaixo vemos a próxima figura que deve ser preenchida. Em sequencia o algoritmo de Flood fill sendo aplicado. Como o mesmo precisa de um ponto inicial para começar o preenchimento, seguem os 3 passos necessários para realizar o preenchimento completo da figura. Na última imagem vemos o algoritmo de varredura com análise geométrica. Observe que existem duas linhas horizontais causadas pelo mesmo problema da figura anterior.



Tanto na circunferência como no retângulo os resultados obtidos visualmente foram exatamente os mesmos como é mostrado nas imagens abaixo.



Como dito anteriormente o Flood Fill obteve resultados visuais mais consistentes em relação ao de varredura com análise geométrica na aplicação sobre essas imagens.

Em relação à custo de processamento temos a inversão desse cenário uma vez que o flood fill faz muitas comparações em um mesmo pixel enquanto que o de varredura com análise geométrica é mais objetivo e direto no cálculo do preenchimento.