Computação Gráfica com IA: Comparação e Melhoria de Algoritmos Tradicionais

MELHORANDO ALGORITMOS DE RASTERIZAÇÃO COM INTELIGÊNCIA ARTIFICIAL

NOME DOS AUTORES:

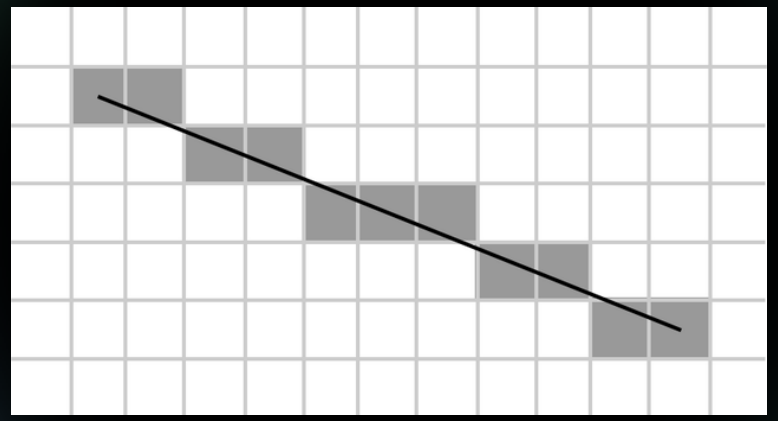
- FELIPE TADEU GÓES GUIMARÃES
- MATEUS AULER LIMA E SILVA

Soluções Tradicionais em Computação Gráfica

Exemplo de algoritmo de Computação Gráfica clássico usado para rasterização de linhas: **Algoritmo de Bresenham**

Características:

- Desempenho eficiente.
- Menor uso de memória.
- Implementação simples.



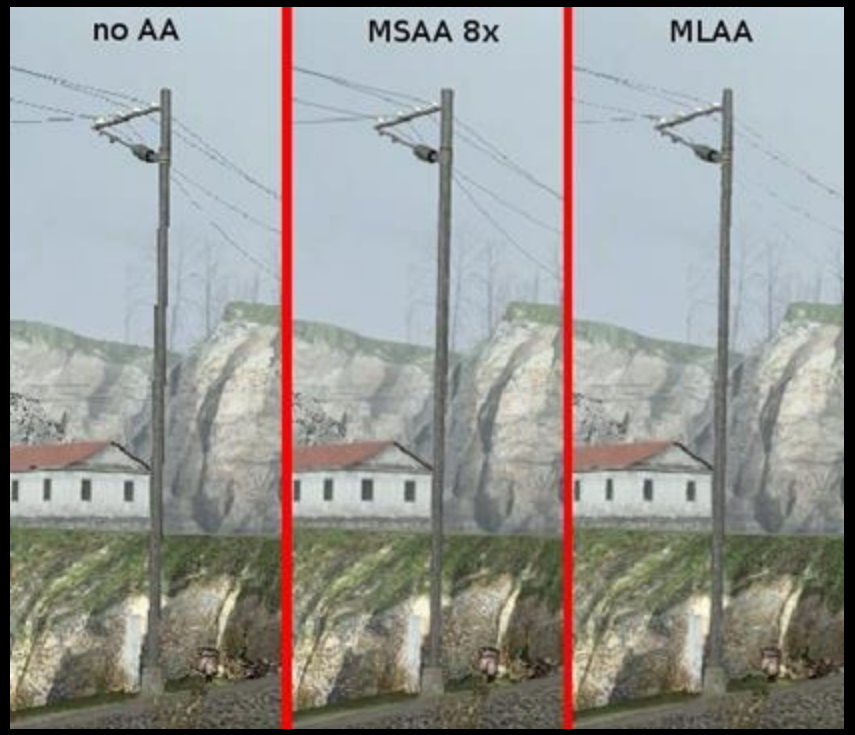
Soluções Atuais com Inteligência Artificial

Técnica moderna que utiliza IA para melhorar a qualidade e eficiência:

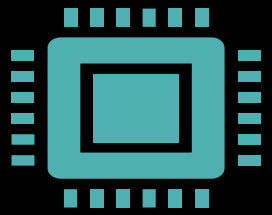
<u>Redes neurais para suavização de bordas e anti-aliasing.</u>

Características:

- Melhor qualidade de imagem.
- Processamento paralelo.
- Maior uso de memória e poder computacional.



Recursos Computacionais Necessários



Hardware:

Processador: Intel Core i5 ou equivalente (mínimo), i7 ou equivalente (recomendado).

Memória RAM: 8GB (mínimo), 16GB (recomendado).

GPU: NVIDIA GTX 1050 ou equivalente (mínimo), RTX 2060 ou equivalente (recomendado).

Software:

Sistema Operacional: Windows 10 ou superior, ou qualquer distribuição Linux recente.

Ferramentas de Desenvolvimento: Python, TensorFlow/PyTorch, OpenCV.

IDE: Visual Studio Code, PyCharm.

Desenvolvimento da Demo

Algoritmo Tradicional:

Bresenham: Eficiência e precisão na rasterização de linhas.

Melhoria com IA:

Suavização: Aplicação de filtro neural para suavizar bordas.

```python
# Algoritmo tradicional de rasterização de linha (Bresenham)
def bresenham(x1, y1, x2, y2):
    points = []
    dx = abs(x2 - x1)
    dy = abs(y2 - y1)
    sx = 1 if x1 < x2 else -1
    sy = 1 if y1 < y2 else -1
    err = dx - dy

    while True:
        points.append((x1, y1))
        if x1 == x2 and y1 == y2:
            break
        e2 = err * 2
        if e2 > -dy:
            err -= dy
            x1 += sx
        if e2 < dx:
            err += dx
            y1 += sy
    return points
```

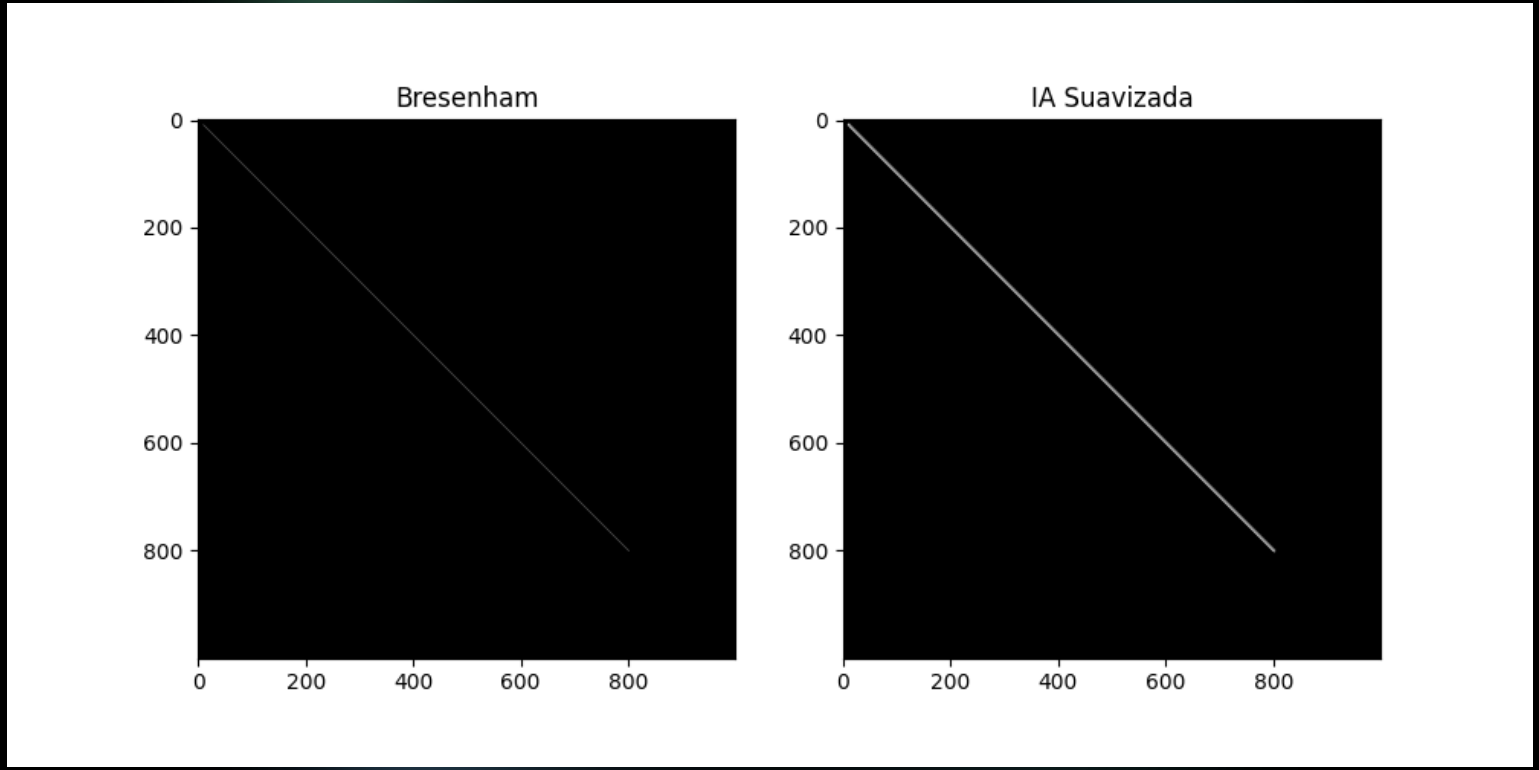
```python
# Define uma rede neural simples para suavização
def create_model():
    model = models.Sequential()
    model.add(layers.Input(shape=(None, None, 1)))  # Usar Input como a primeira camada
    model.add(layers.Conv2D(16, (3, 3), activation='relu', padding='same'))
    model.add(layers.Conv2D(1, (3, 3), padding='same'))
    model.compile(optimizer='adam', loss='mean_squared_error')
    return model

# Função para suavizar a imagem usando a rede neural
def smooth_line_with_cnn(image, points, model):
    for point in points:
        image[point[1], point[0]] = 255
    image = image.astype('float32') / 255.0
    image_tensor = np.expand_dims(np.expand_dims(image, axis=0), axis=-1)
    smoothed_tensor = model.predict_on_batch(image_tensor)
    smoothed_image = (smoothed_tensor.squeeze() * 255.0).astype('uint8')
    return smoothed_image
```

Desenvolvimento da Demo

Implementação:

Comparação de tempo de execução e uso de memória.

Resultados visuais da qualidade da imagem.



Resultados Comparativos

Tabela : Desempenho, memória e qualidade.

	Algoritmo	Tempo (s)	Memoria (MB)	Qualidade (Intensidade)
	Bresenham	0.003519	0.085328	201705
IA	Suavizada	0.161054	13.399831	985546

Conclusão

Observações

- A IA oferece melhorias significativas na qualidade da imagem.
- Maior uso de memória e tempo de processamento.

Impacto:

Aplicações em jogos, simulações e renderização de alta qualidade.

Próximos Passos:

- Exploração de mais algoritmos.
- Otimizações para reduzir o uso de recursos.