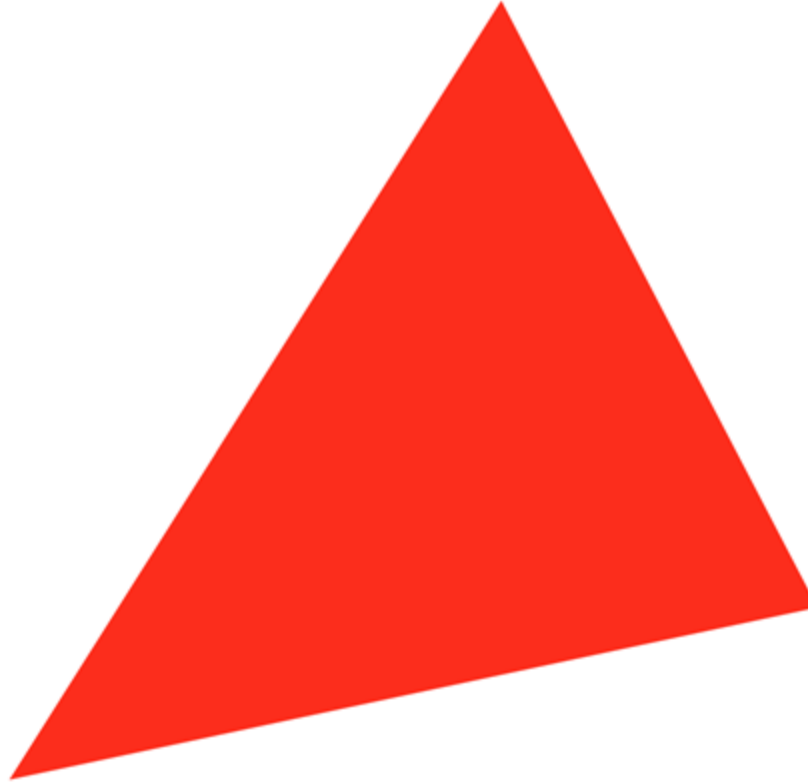


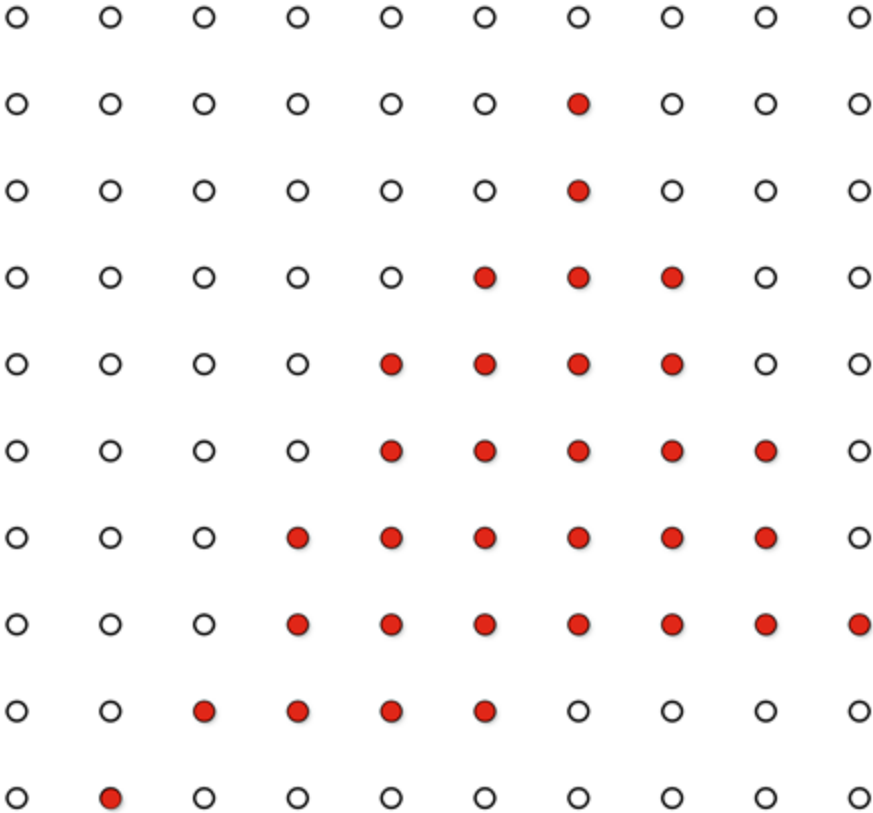
# Computação Gráfica

Aula 11: Anti-aliasing e Visibilidade

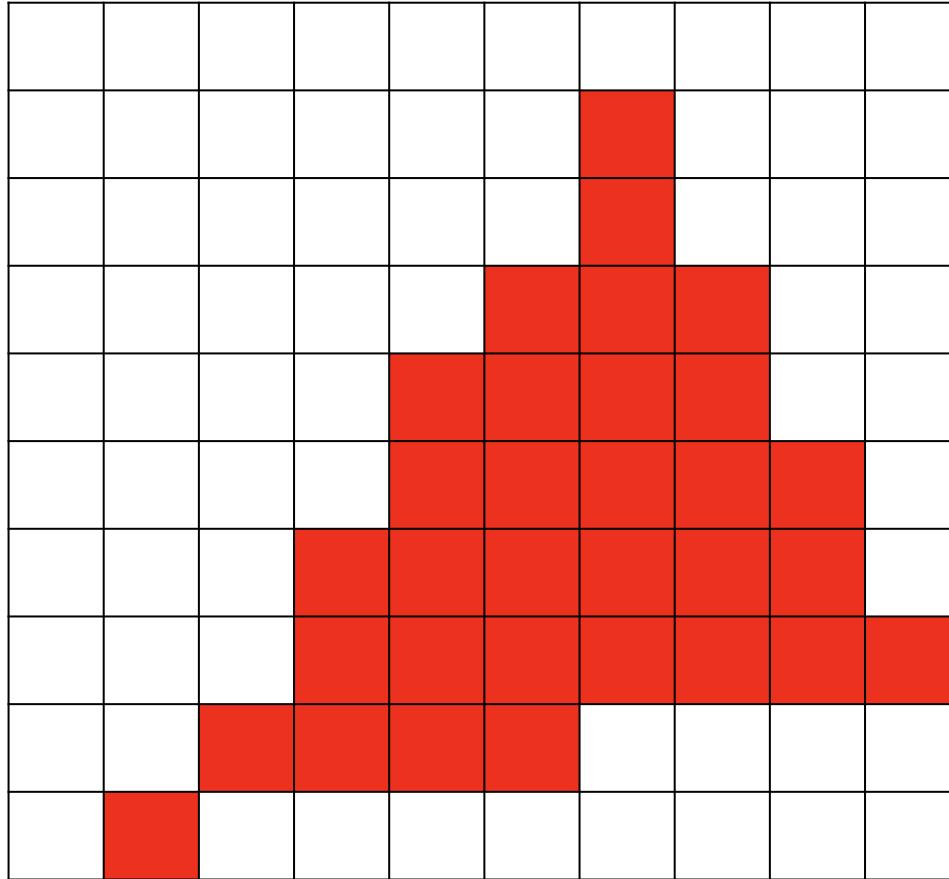
# Para desenhar um triângulo assim



# Coletamos as seguintes amostras



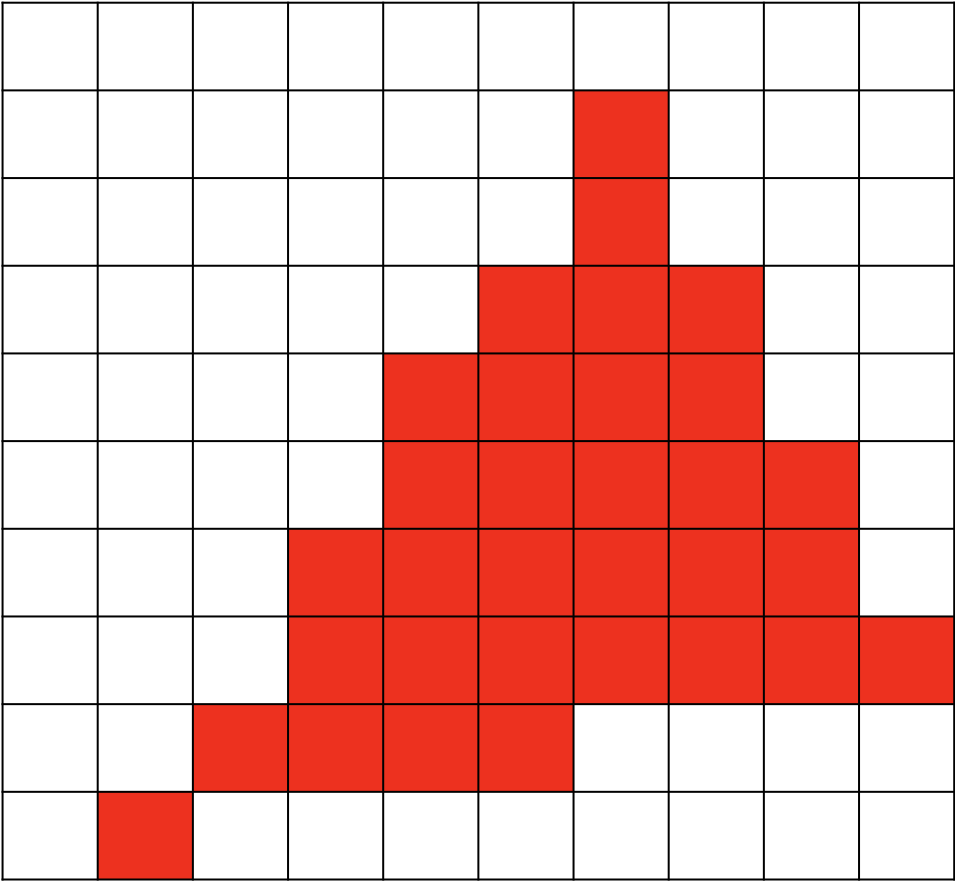
# O display exibirá a seguinte imagem



Mas nosso triângulo "contínuo" seria



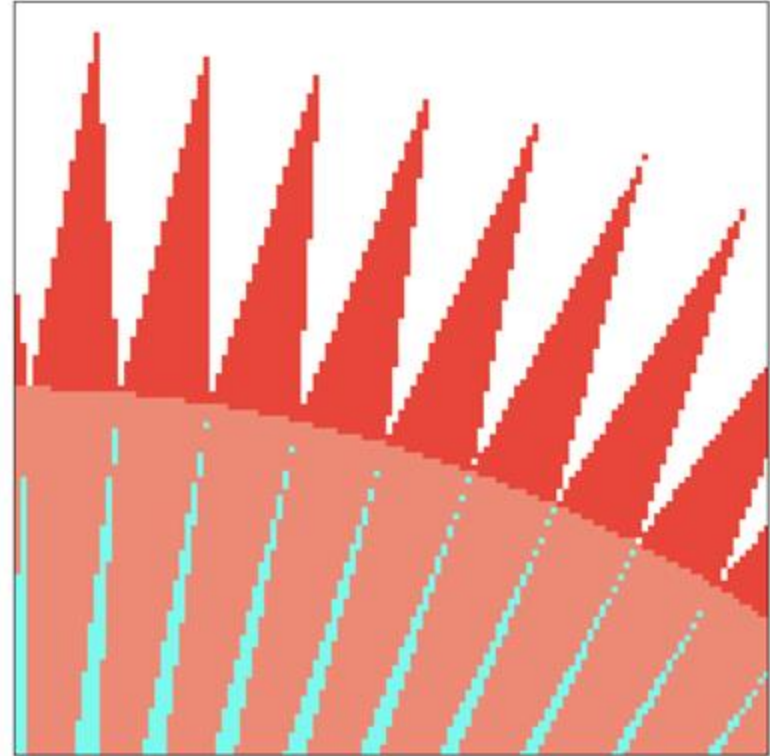
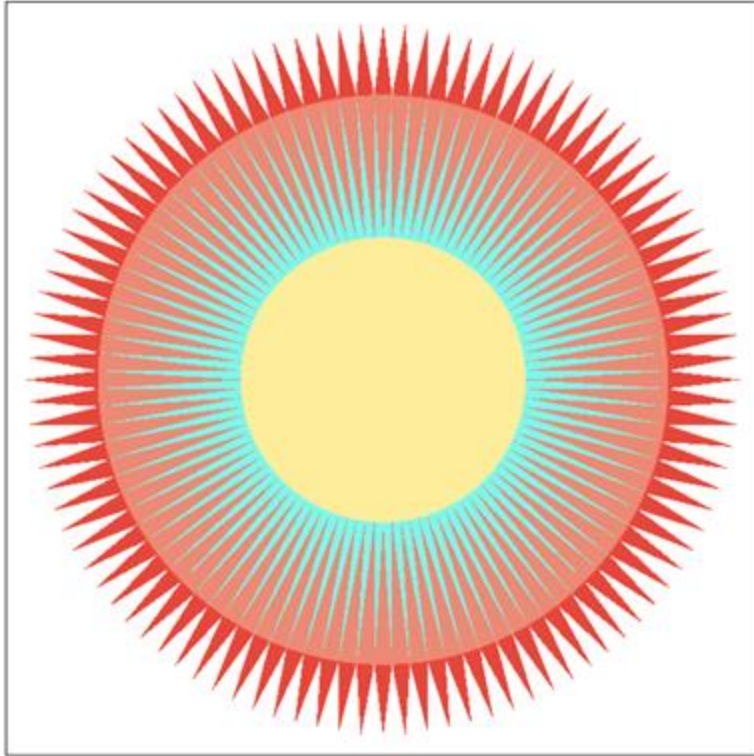
Qual um dos problemas com essa reconstrução?



Serrilhamento (Jaggies)



# Será que esse é o melhor que se pode fazer?



**Não**

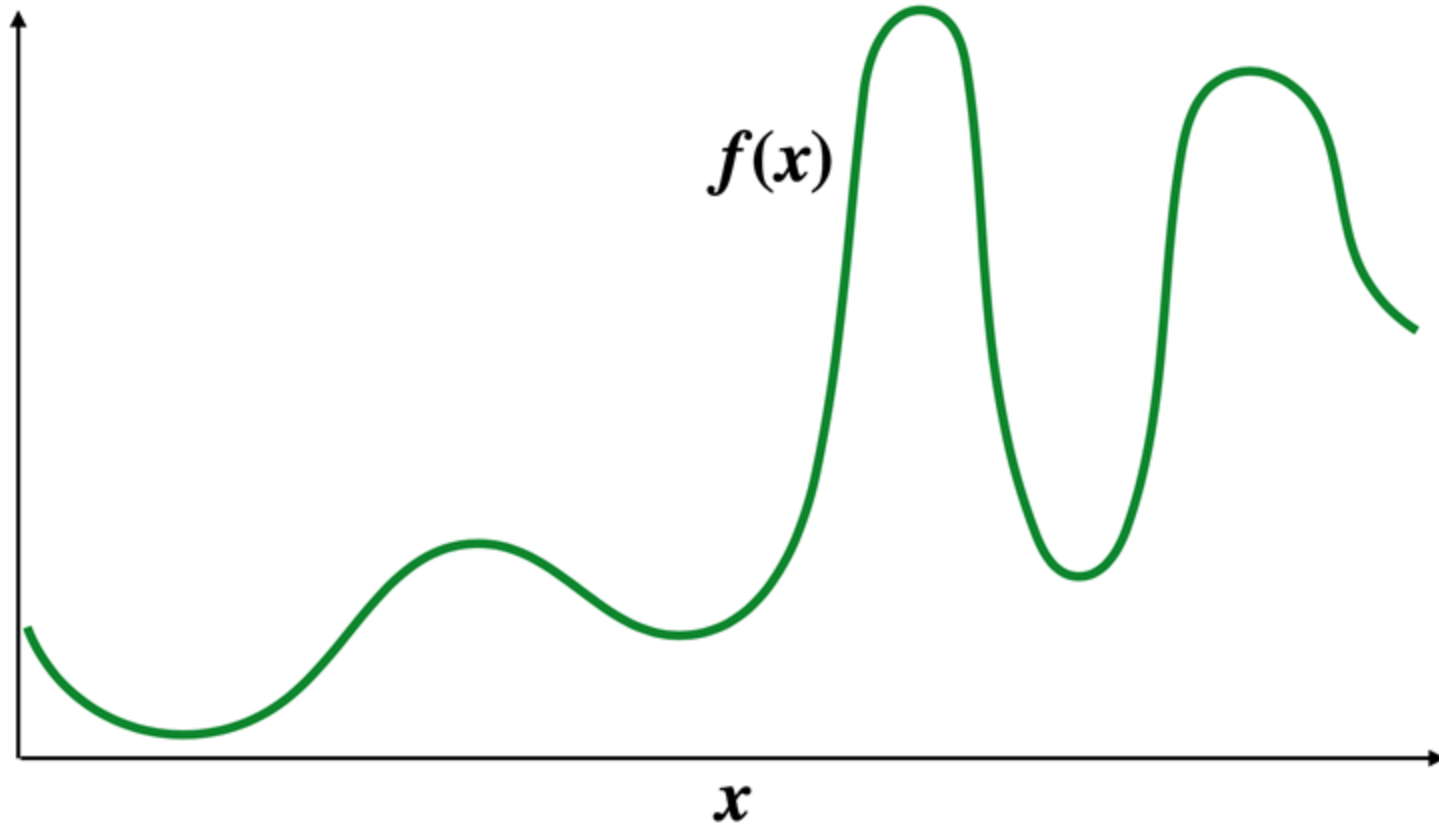
# Arquivos de Áudio

Armazenam as amostras em 1D (por exemplo a 44.1Khz)





Considere um sinal 1D:  $f(x)$



# Amostrando uma função

Avaliar uma função em um ponto é pegar o valor da função.

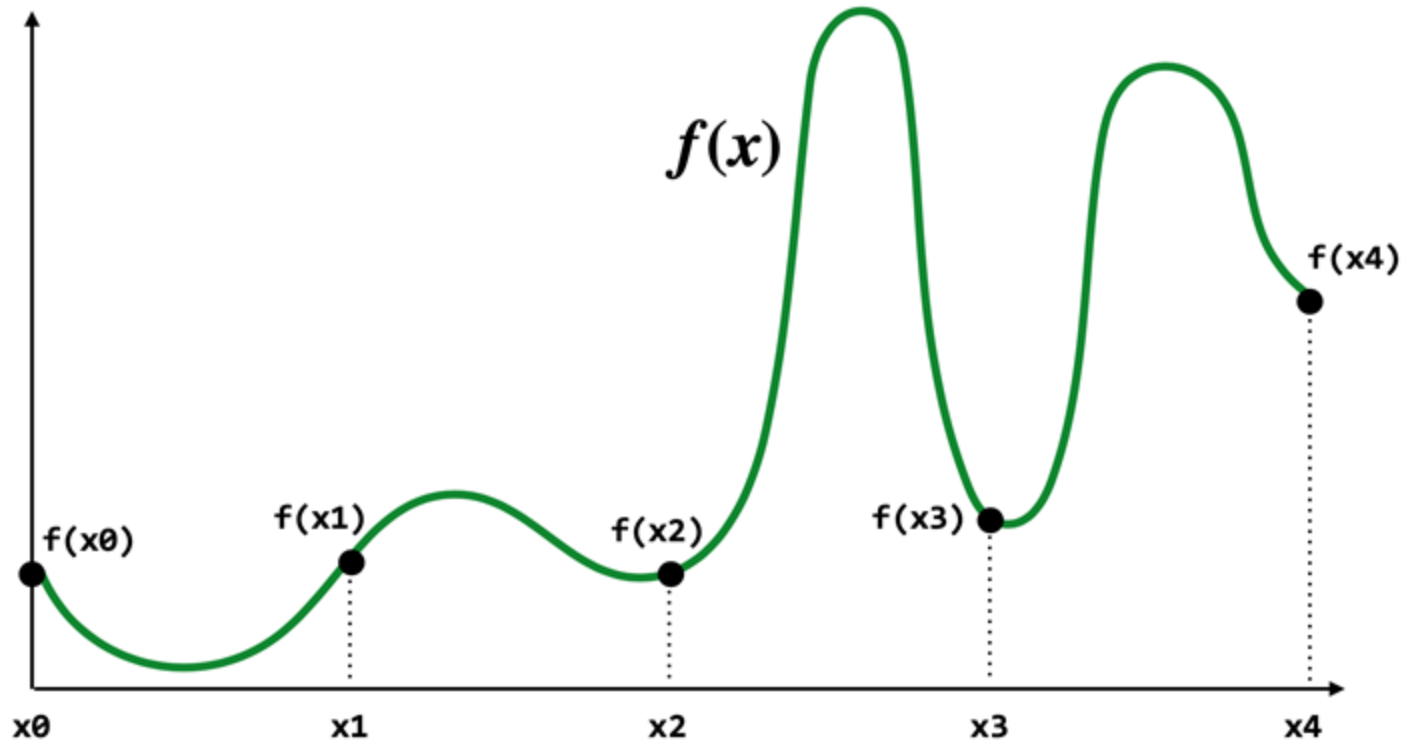
Podemos discretizar uma função por amostragem periódica

```
for(int x = 0; x < xmax; x++)  
    output[x] = f(x);
```

A amostragem é uma ideia central em gráficos. Iremos amostrar o tempo (1D), área (2D), ângulo (2D), volume (3D), etc ...

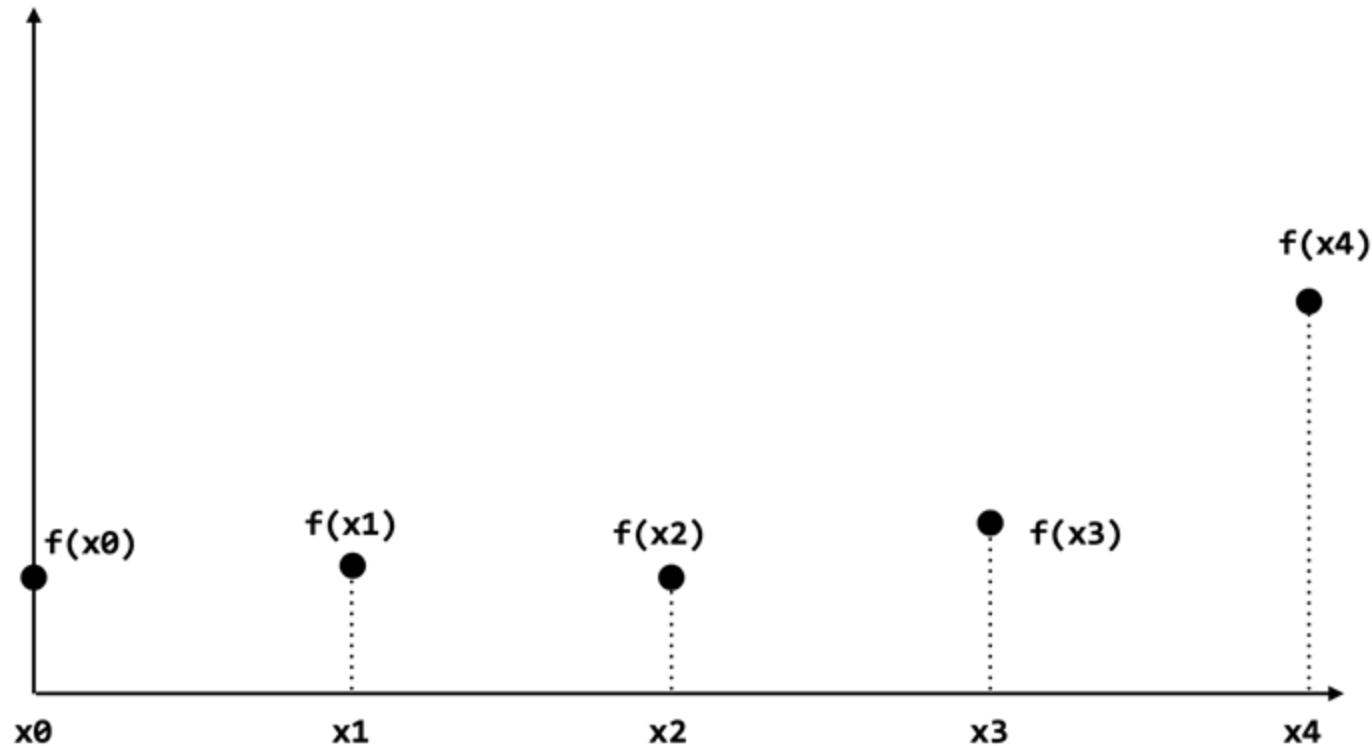
# Amostragem (Sampling)

Pegue medidas de um sinal (amostras)



# Reconstrução

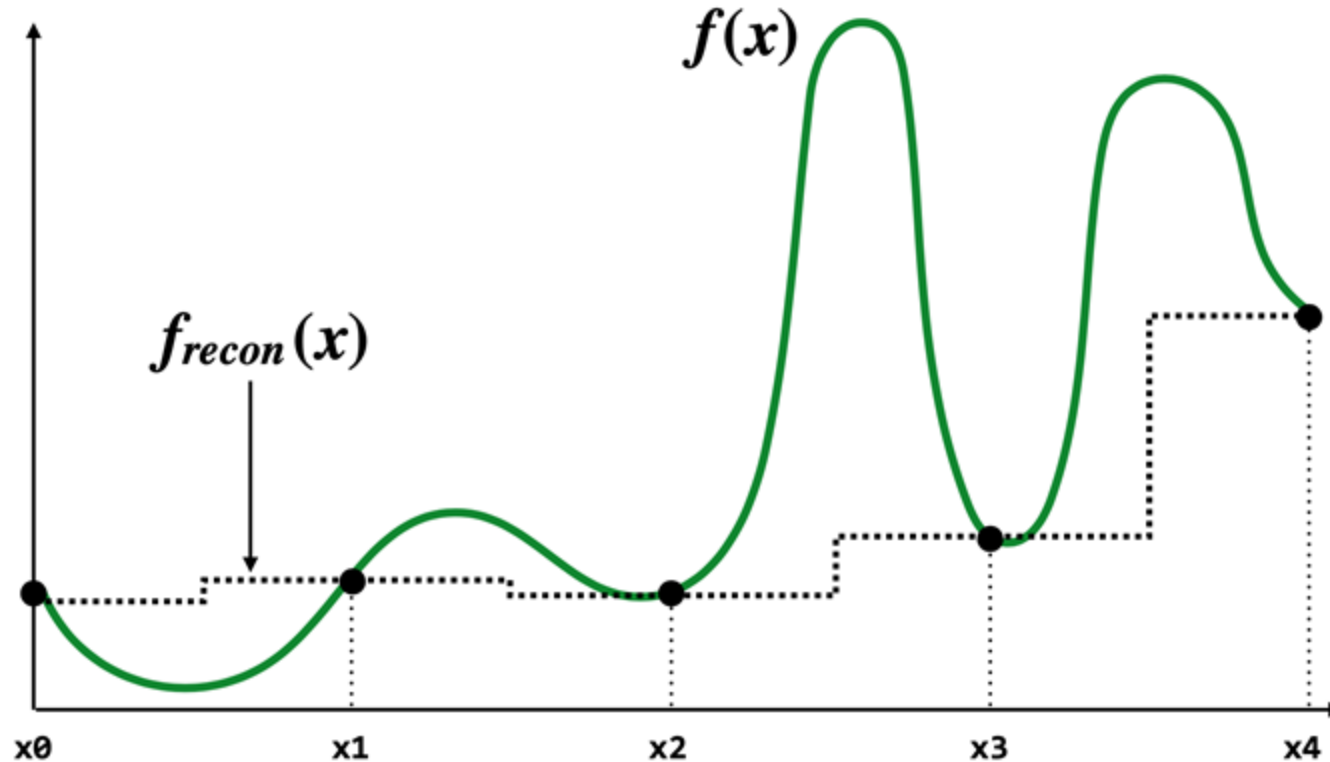
Dado um conjunto de amostras, como podemos tentar reconstruir o sinal original  $f(x)$ ?



# Reconstrução constante por partes

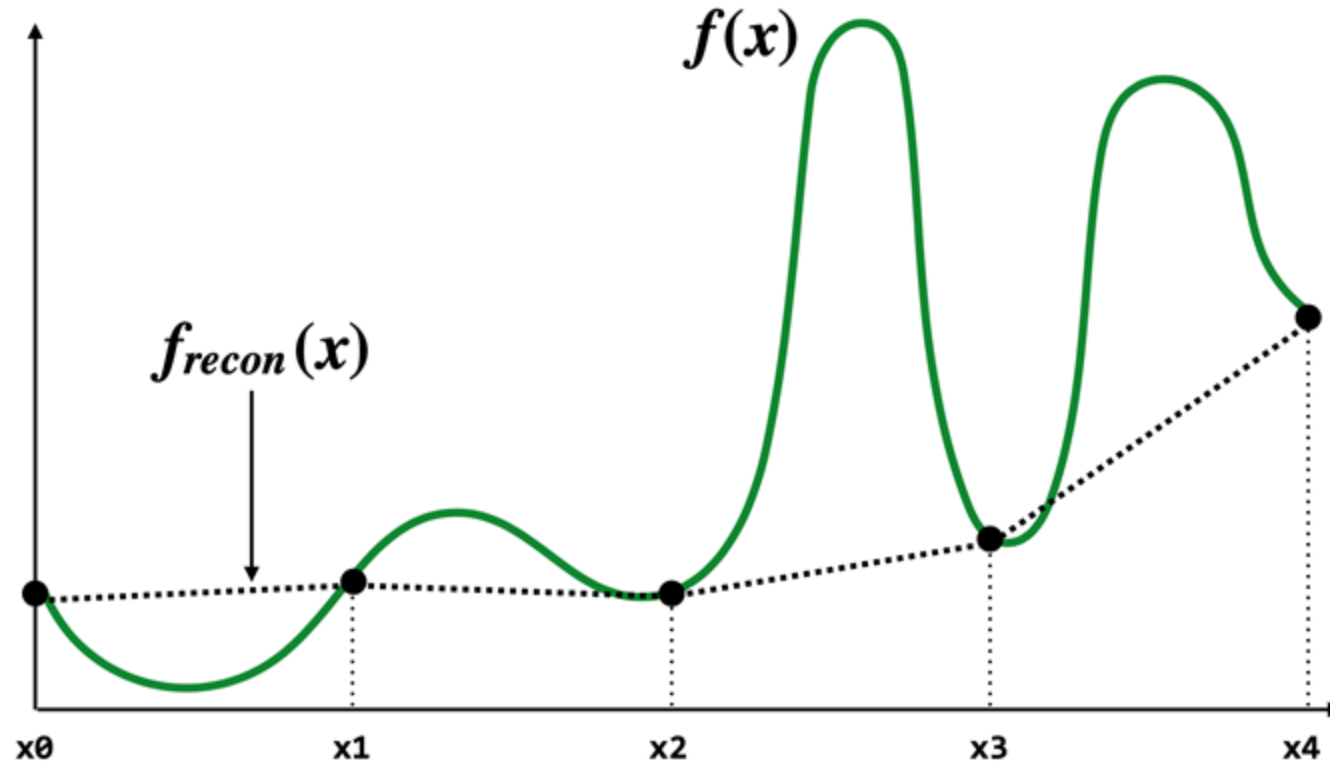
$f_{recon}(x)$  = valor mais próximo da amostra de  $X$

$f_{recon}(x)$  aproxima  $f(x)$

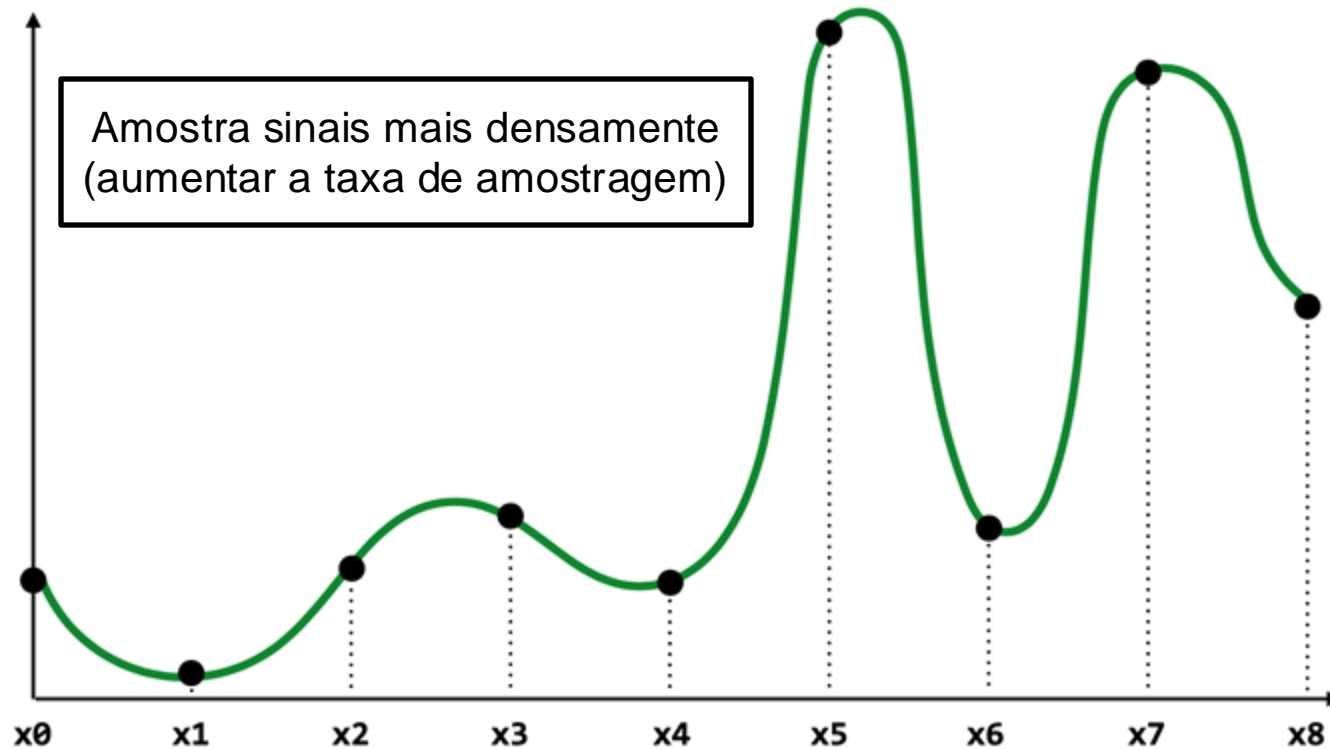


# Reconstrução linear por partes

$f_{recon}(x)$  = interpolação linear entre os valores das duas amostras mais próximas a  $x$

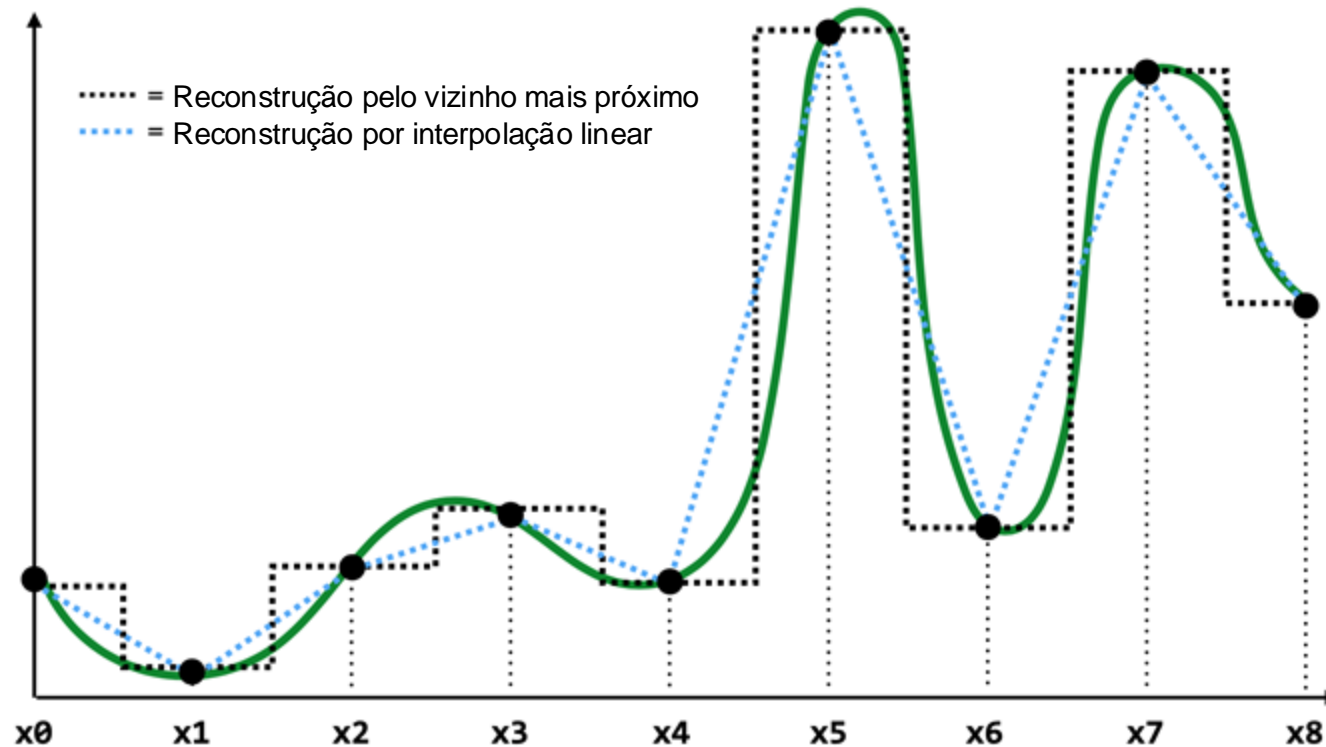


# Reconstrução com mais amostras



# Reconstrução com mais amostras

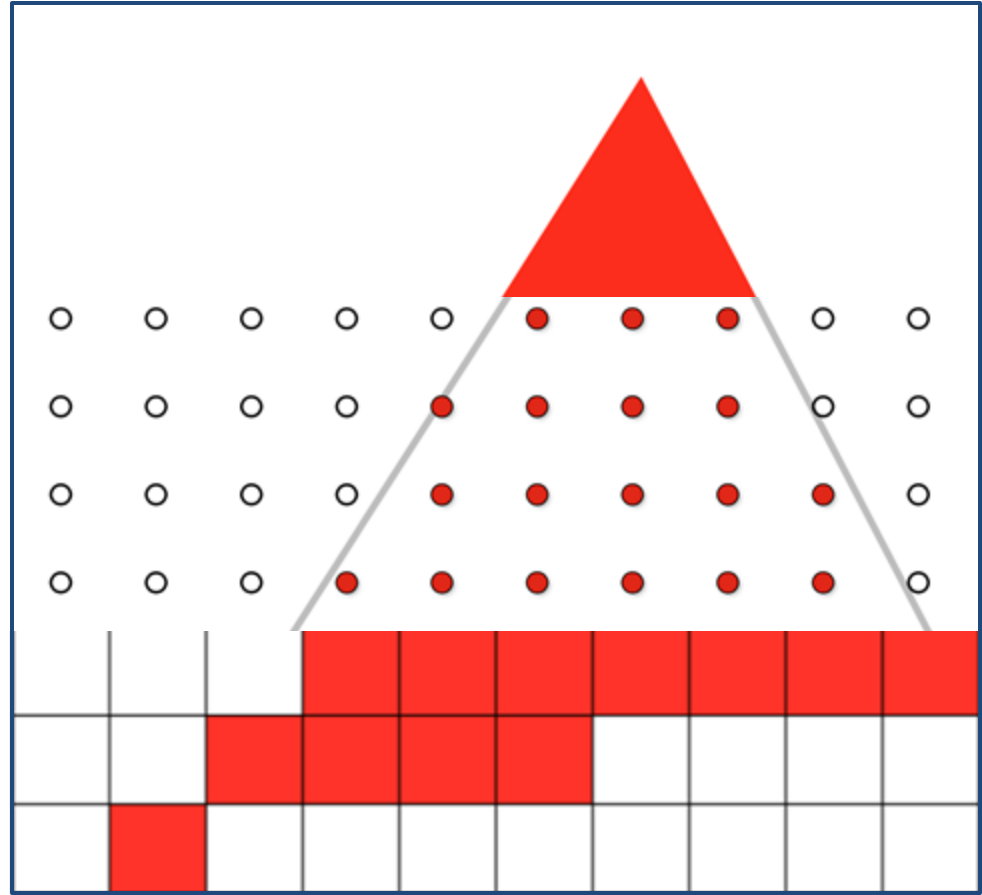
Reconstruções mais precisas resultam de amostragens mais densas





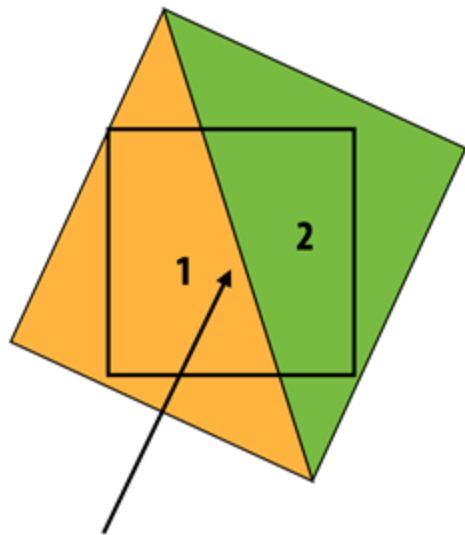
# Perguntas:

- Qual valor o pixel deveria ter?
- Por que o serrilhamento parece "errado"?
- Ideias para uma fórmula de pixel de "qualidade superior"?
- O que há de certo/errado sobre a amostragem pontual?

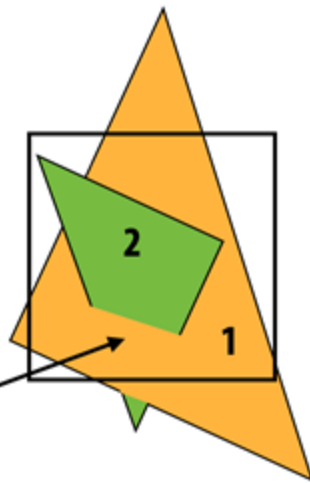


# Pintando os pixels

A análise da cobertura dos triângulos fica mais complicada ao considerar a oclusão de um triângulo por outro.



Metade do pixel coberto pelo triângulo 1 e a outra metade coberta pelo triângulo 2

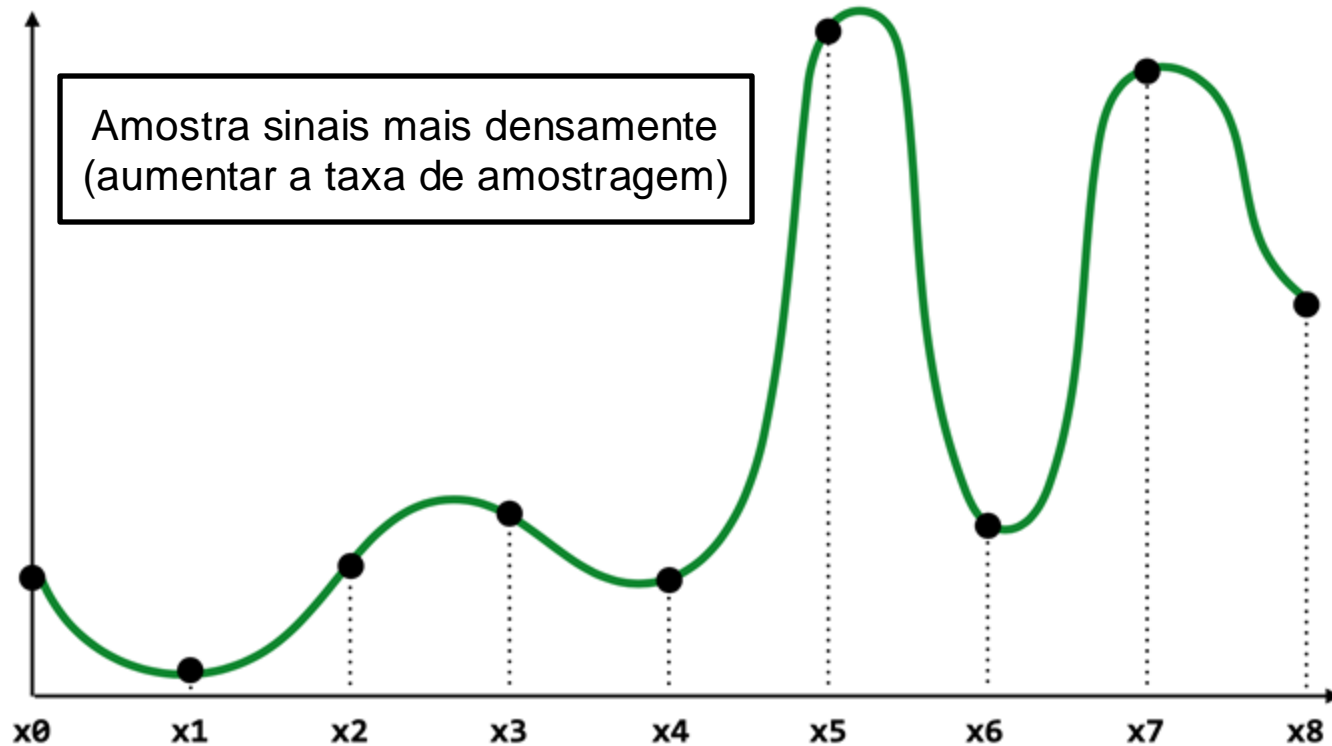


Interpenetração de triângulos: ainda mais complicado

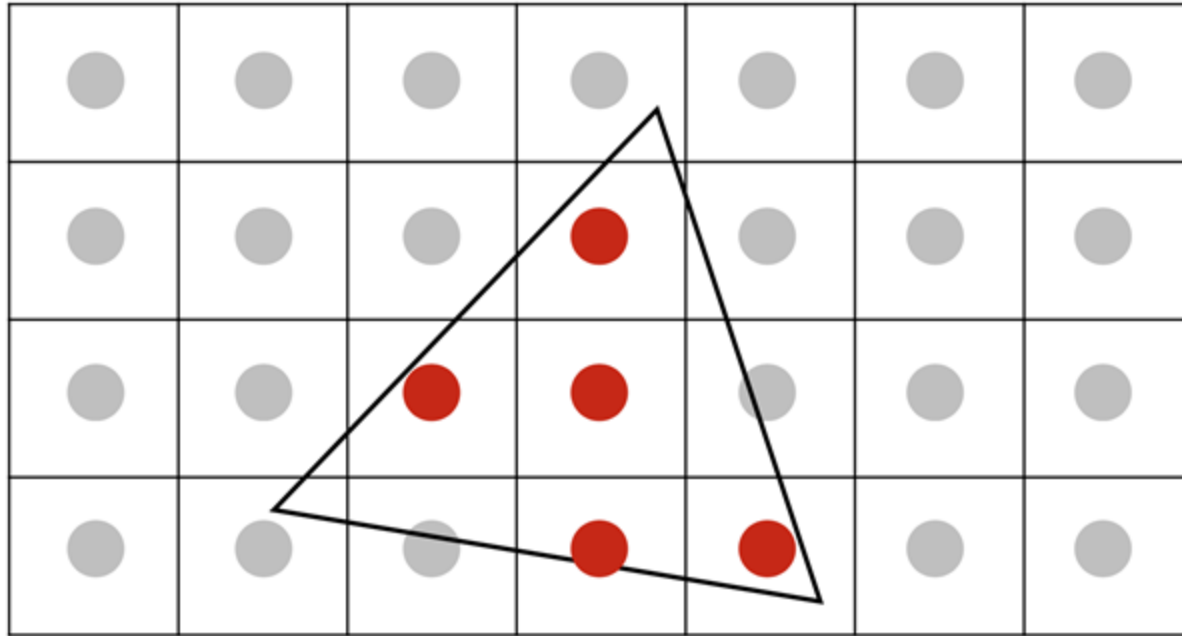


Duas regiões do triângulo 1 contribuem para o pixel. Uma dessas regiões nem mesmo é convexa.

# Representar um sinal de forma mais precisa

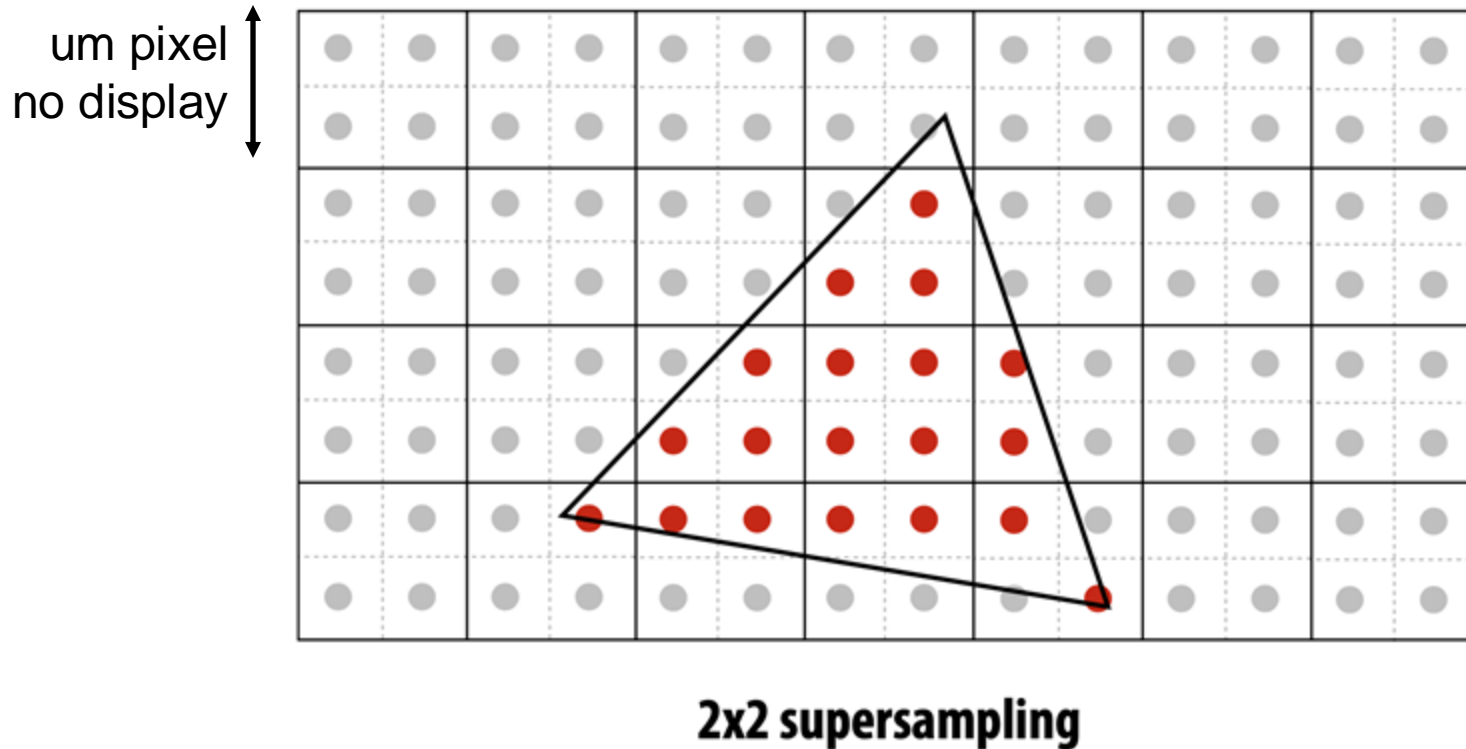


# Amostrando um ponto por pixel



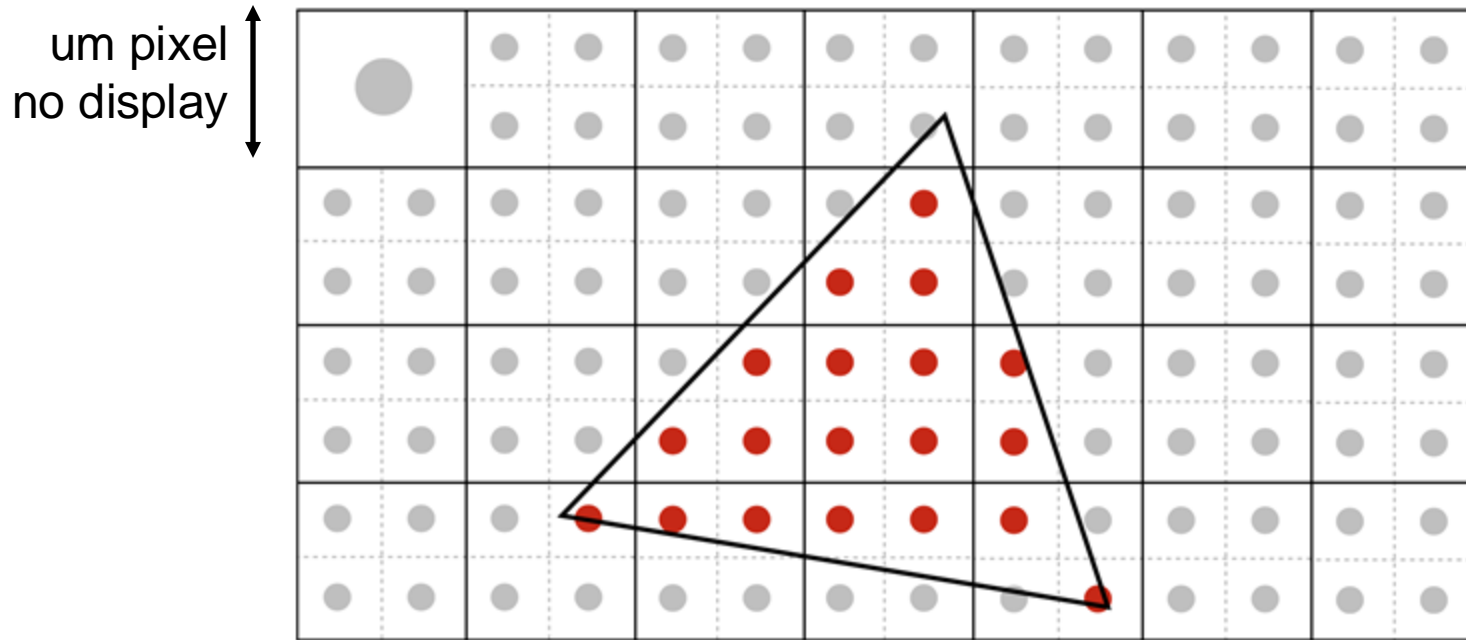
# Supersampling

Pegue  $N \times N$  amostras para cada pixel



# Supersampling

Faça a média com as amostras de cada pixel

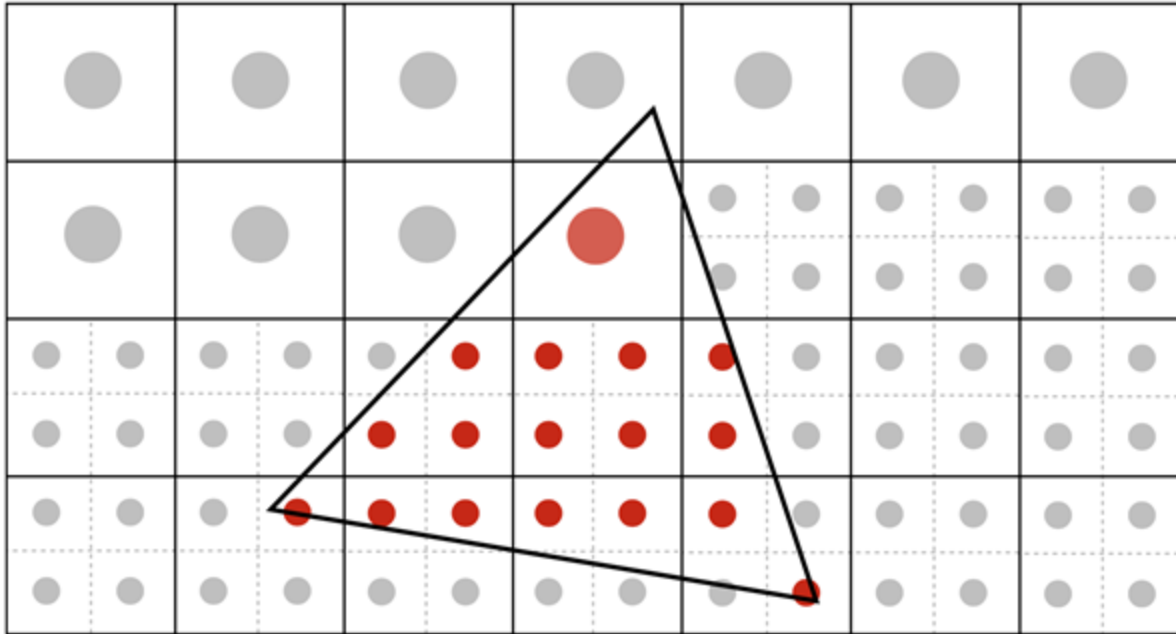


**Calculando a Média**

# Supersampling

Faça a média com as amostras de cada pixel

um pixel  
no display

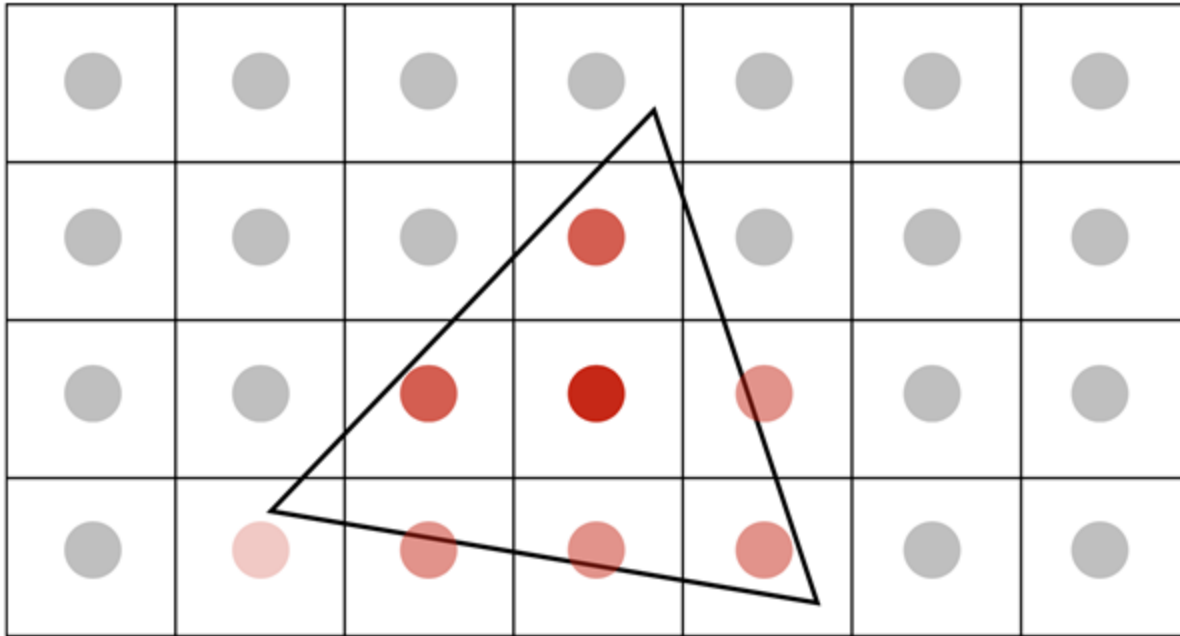


**Calculando a Média**

# Supersampling

Faça a média com as amostras de cada pixel

um pixel  
no display



**Calculando a Média**

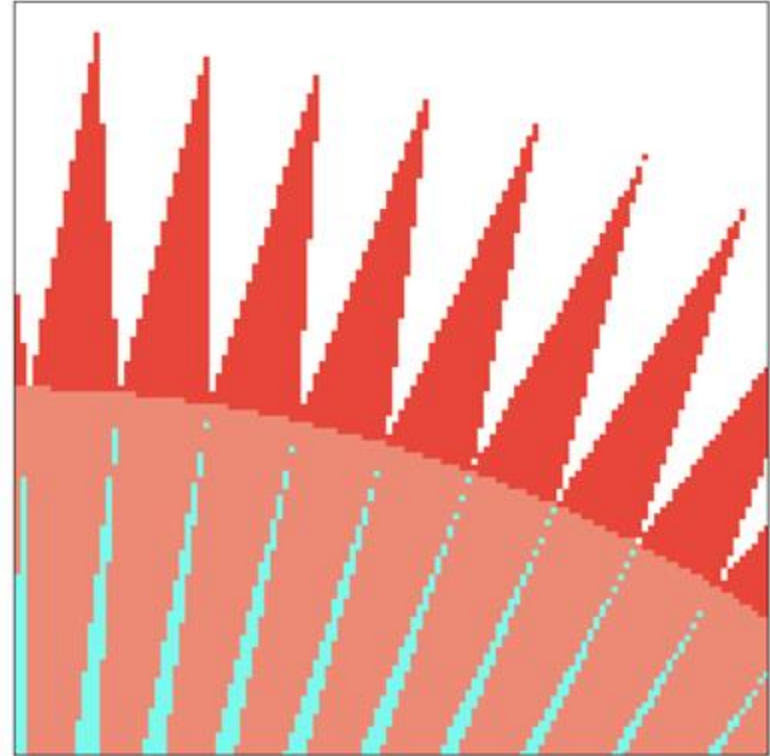
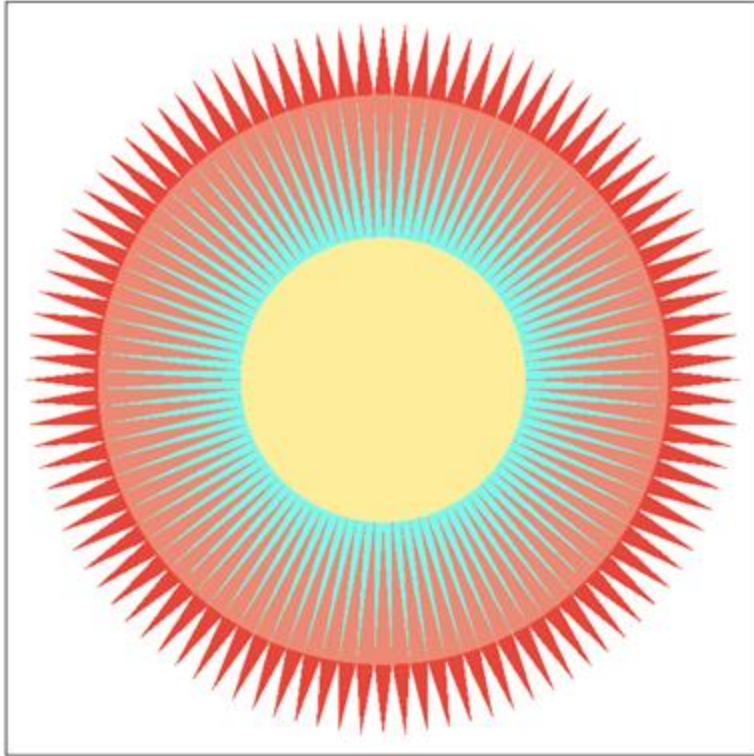


# Resultado do Supersampling

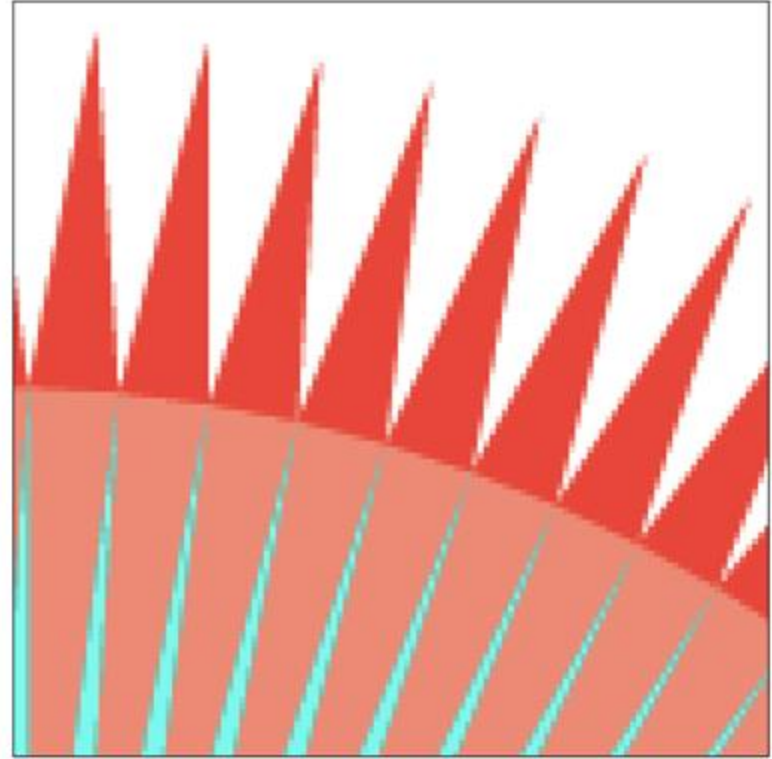
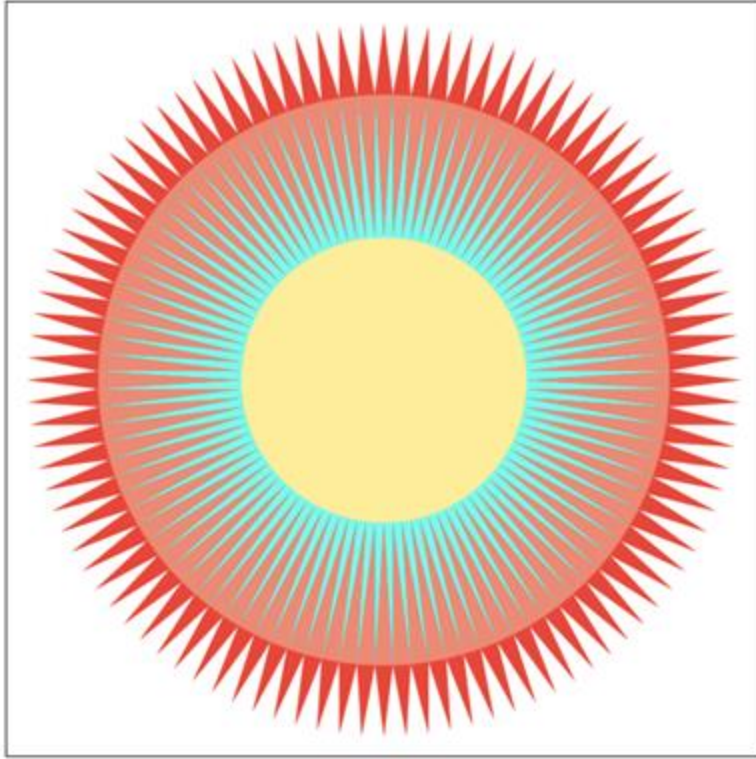
Valores a serem emitidos por pixel no display.

			75%			
		100%	100%	50%		
	25%	50%	50%	50%		

# Uma amostra por pixel



# 4x4 supersampling + downsampling



# Resultado

Se tem uma maior suavização dos cantos das imagens

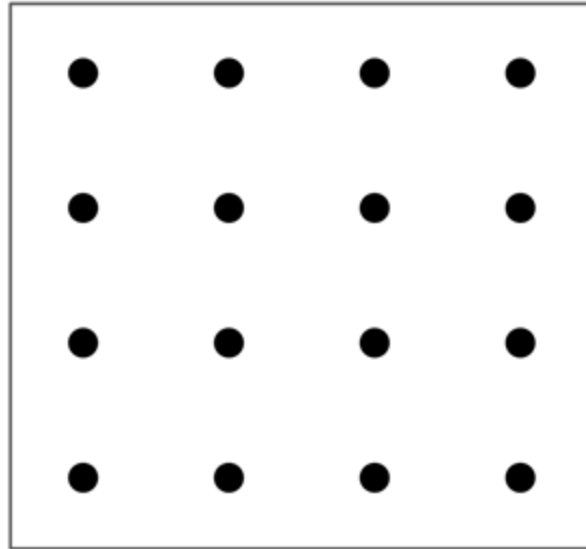
Aliased



Anti-Aliased

# Supersampling

Podemos aproximar cálculo do valor do pixel amostrando vários locais dentro dele e calculando a média de seus valores.

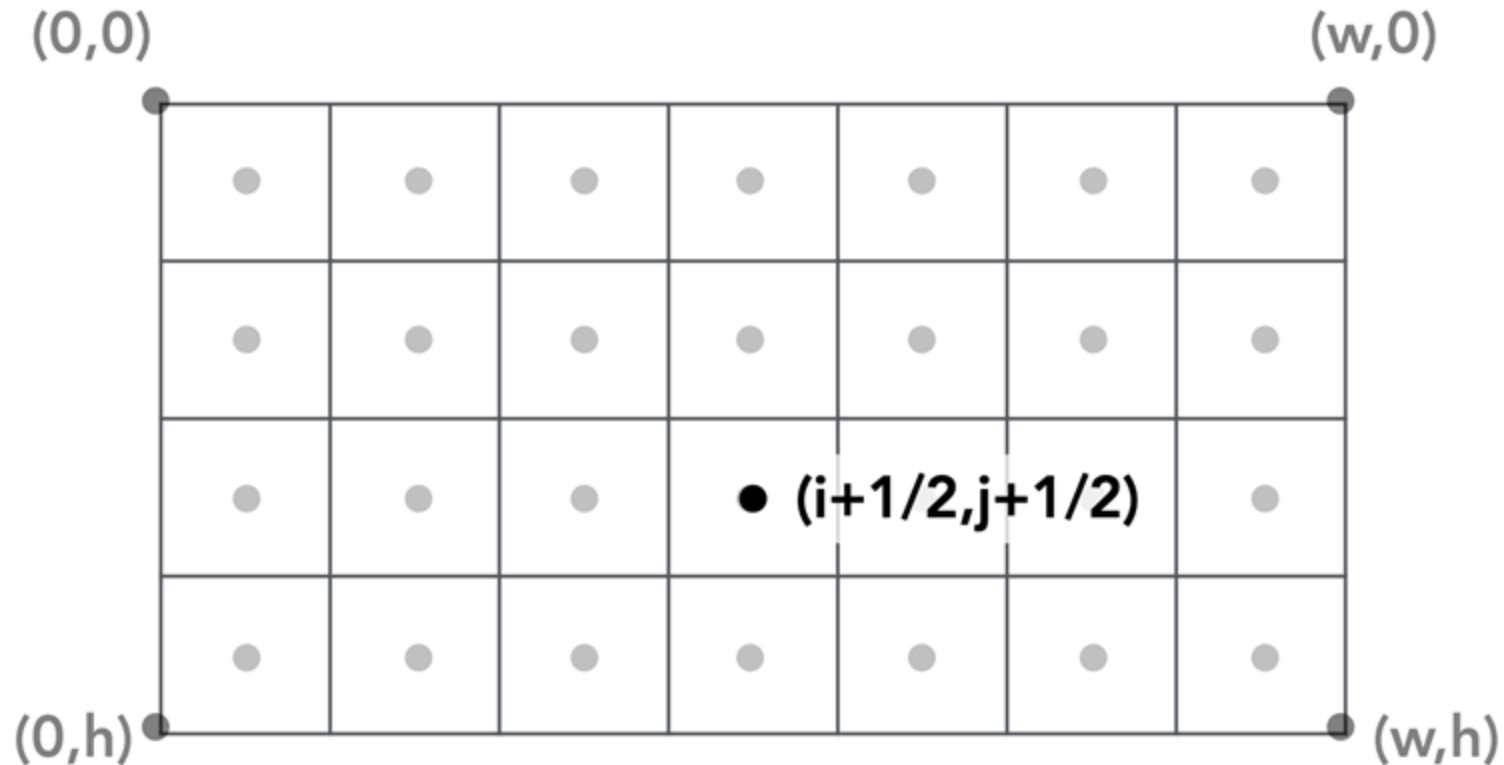


**4x4 supersampling**

# Técnicas de Supersampling

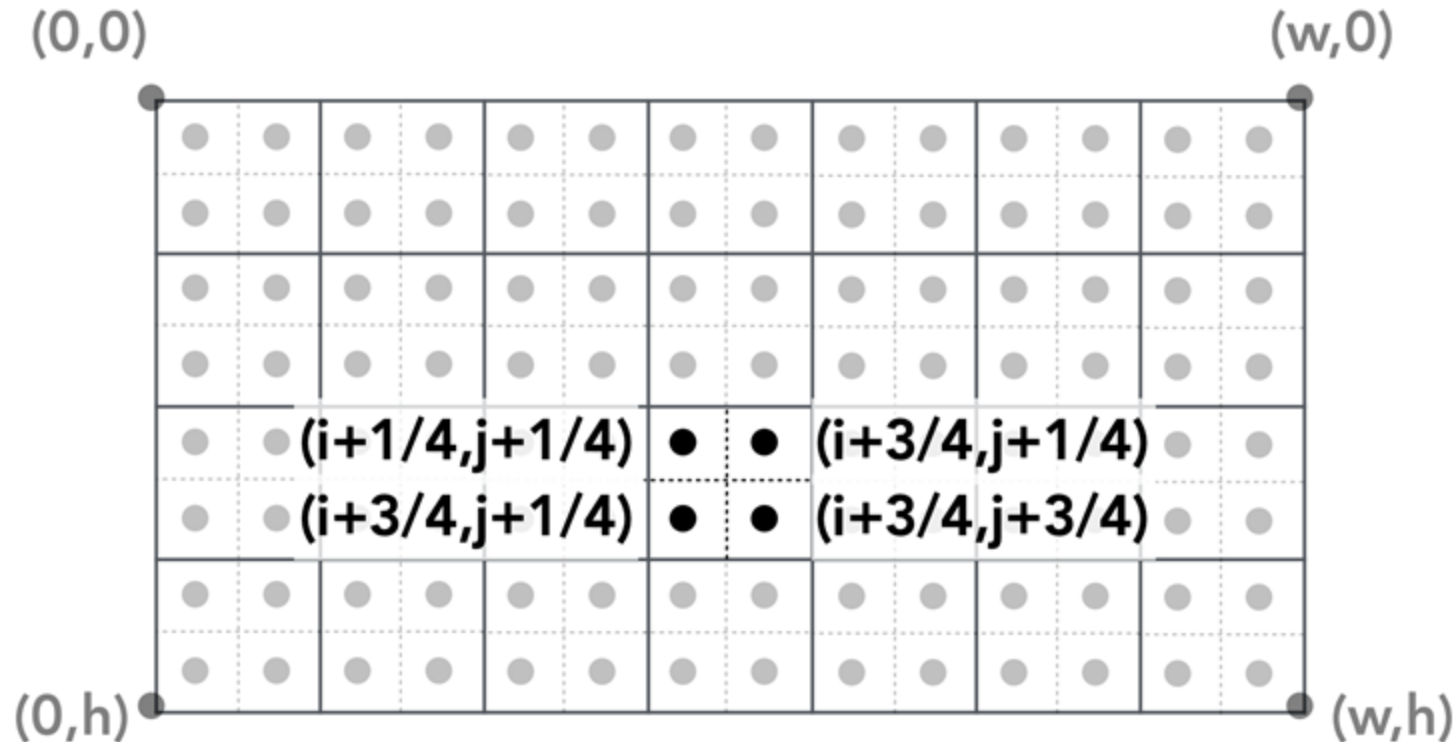
O local de fazer as amostras pode mudar, por exemplo:

## Amostragem Regular



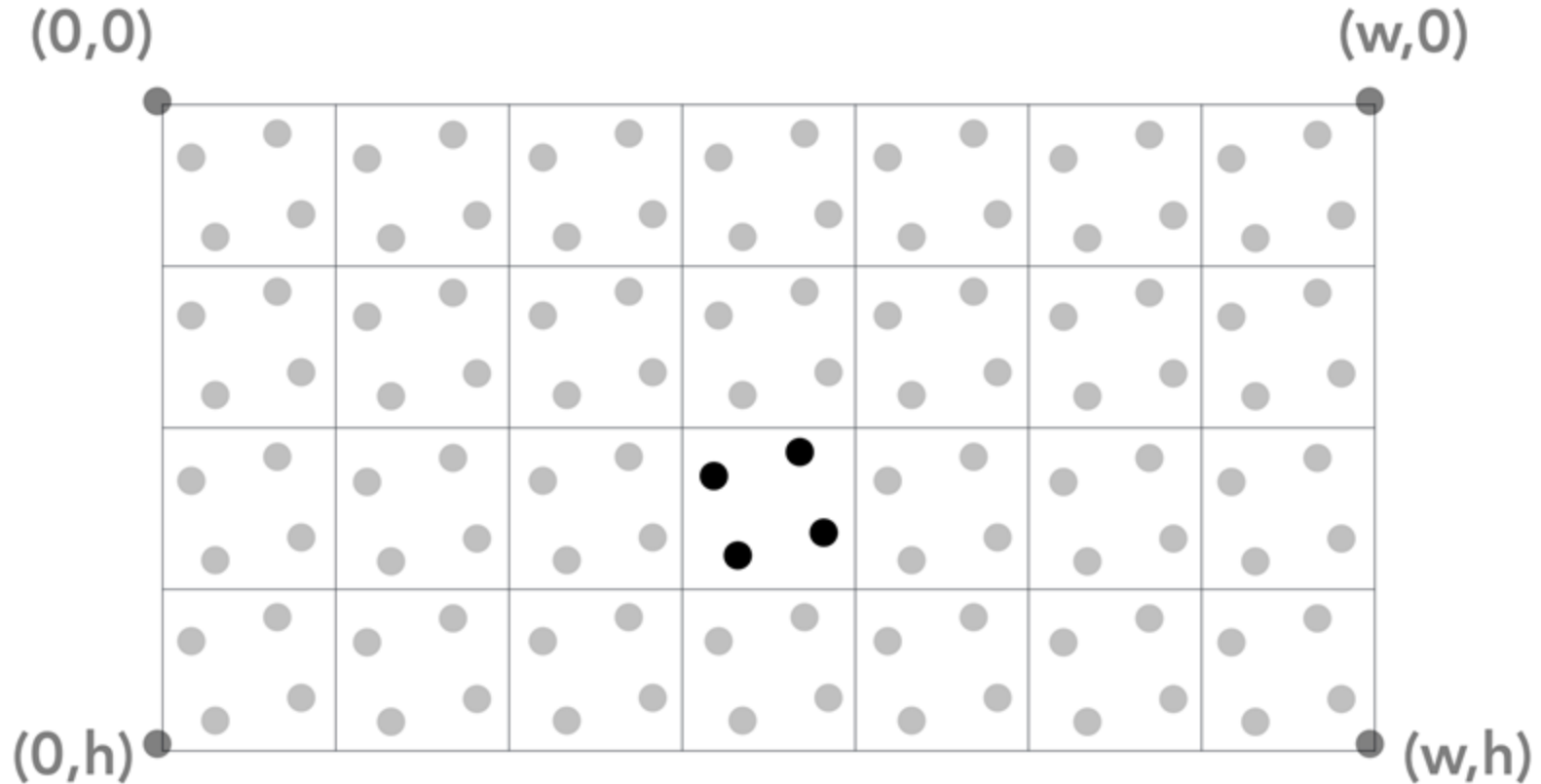
# Técnicas de Supersampling

## 2x2 Supersampling: amonstragem para pixel (i,j)



# Técnicas de Supersampling

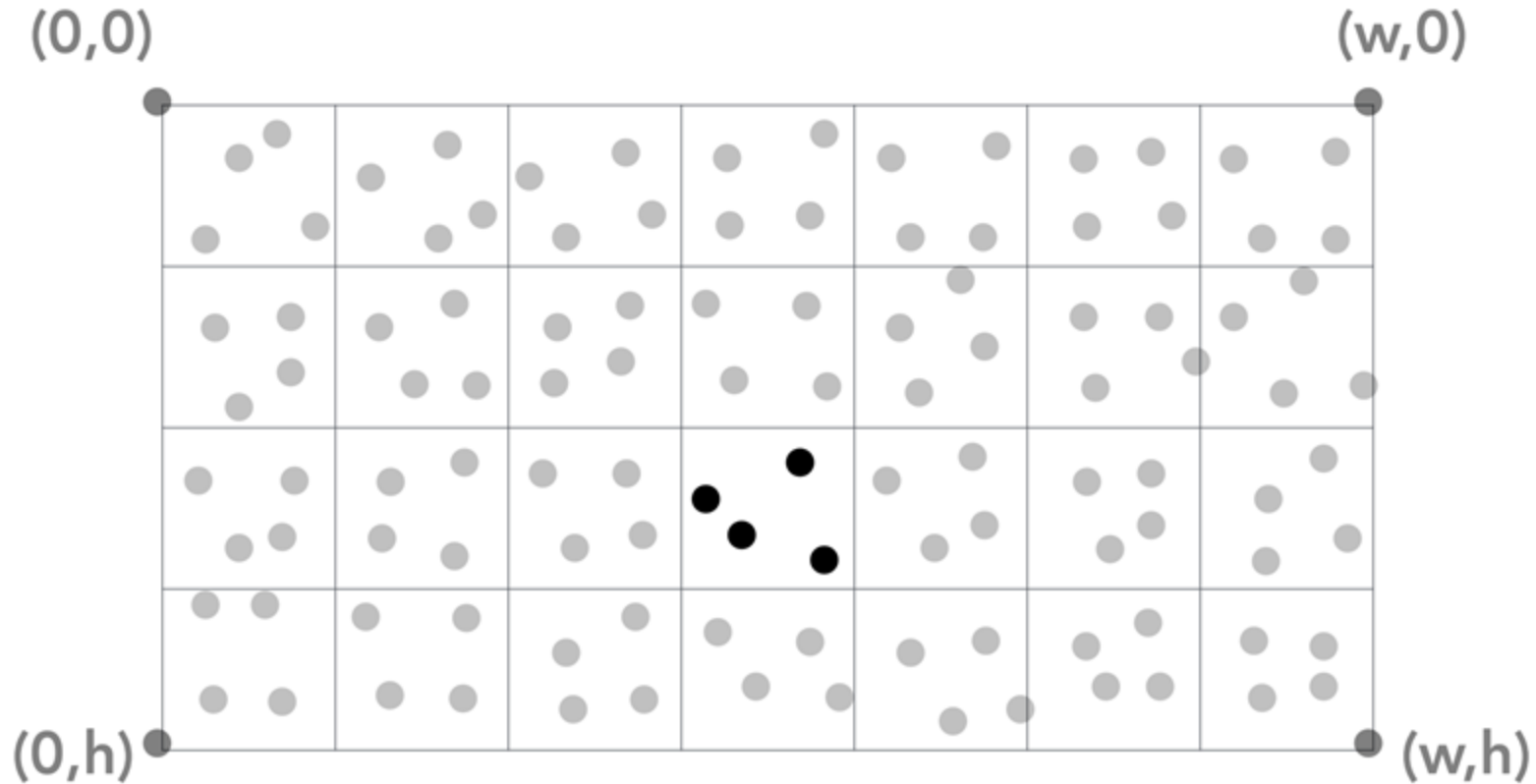
## Off-Grid Sampling





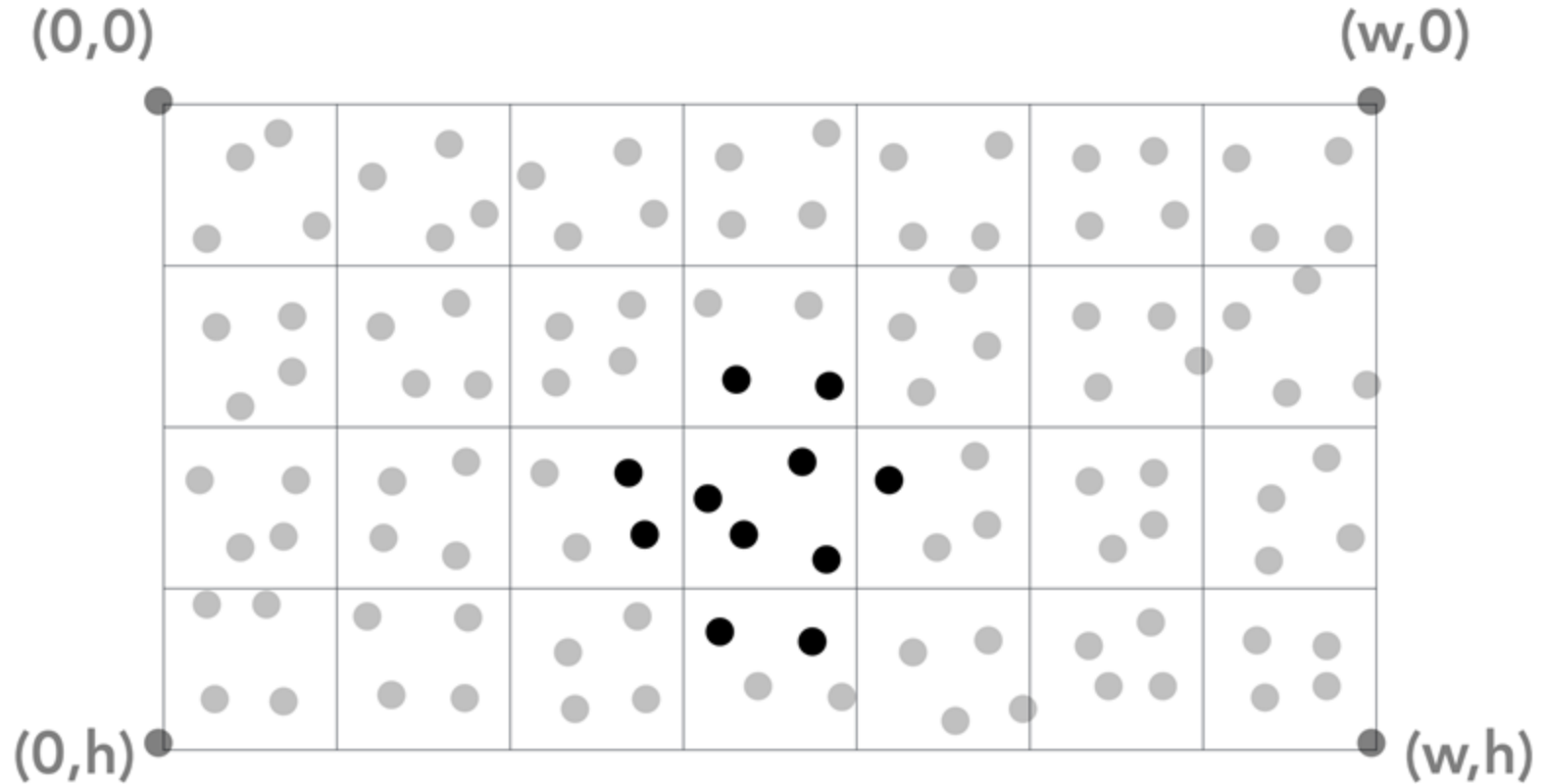
# Técnicas de Supersampling

## Random Sampling



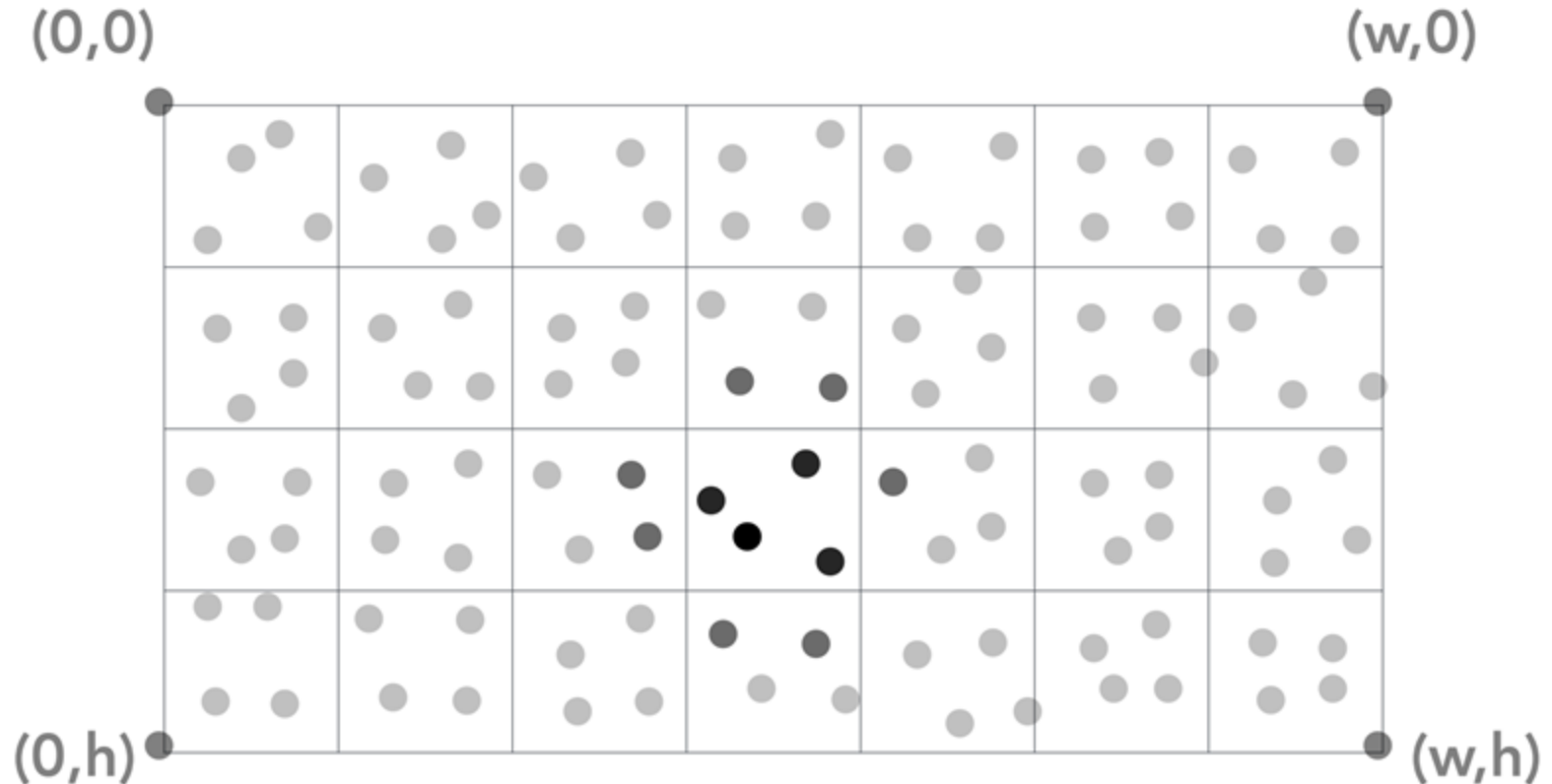
# Técnicas de Supersampling

## Amostras "fora" do pixel

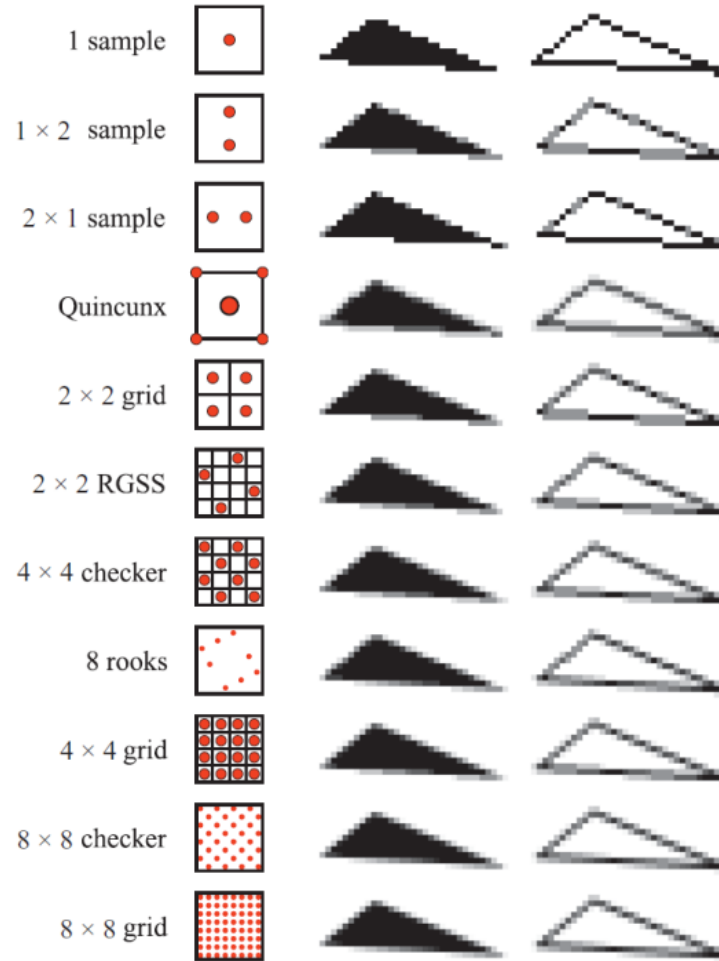


# Técnicas de Supersampling

## Peso de Amostras Não-Uniforme



# Supersampling



# Resultado SSAA (Super-Sampling Anti-Aliasing)



# Existem outras técnicas?

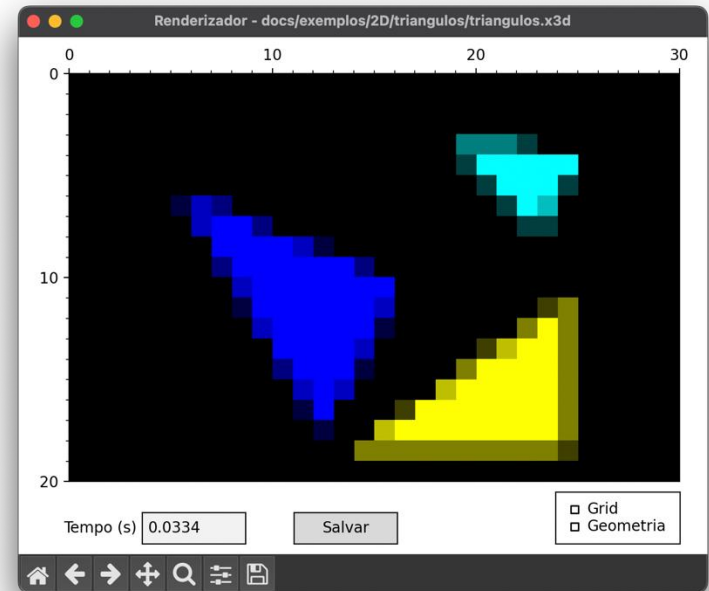
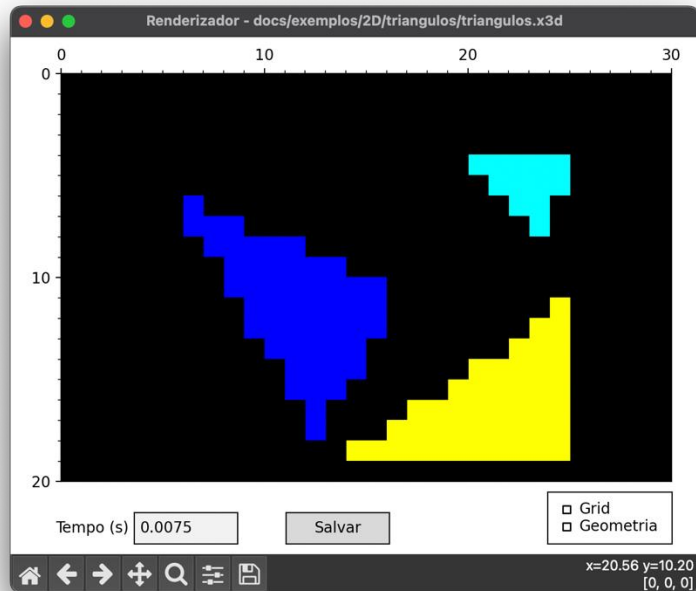
## SIM

- SSAA (Super Sampling Anti-Aliasing) \*
  - FSAA (Full Scene/Screen/Sample Anti-Aliasing)
- MSAA (MultiSample Anti-Aliasing)
- FXAA (Fast Approximate Anti-Aliasing)
- TAA (Temporal Anti-Aliasing)
- TXAA (Temporal Anti-Aliasing Nvidia)
- MLAA (Morphological Anti-Aliasing)
- SMAA (Sub-pixel Morphological Anti-Aliasing)
- DLAA (Deep learning anti-aliasing)

Isso é uma sopa de letras e alguns são especializações de outros.  
Veremos mais sobre isso nas próximas aulas.

# Onde ver:

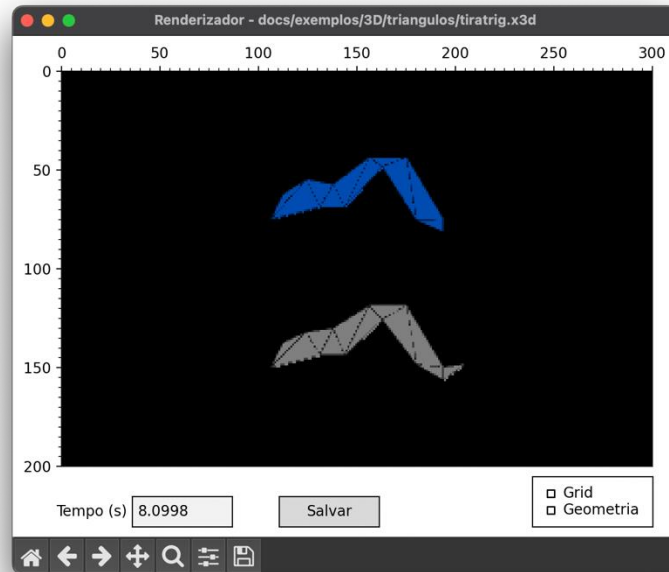
## Rode novamente os exemplos 2D



# Dica

Não tente fazer a amostra na hora de desenhar os triângulos.  
Crie um novo Framebuffer maior, depois faça o downsampling.

Problema típico de fazer as amostras no desenho e não no buffer!

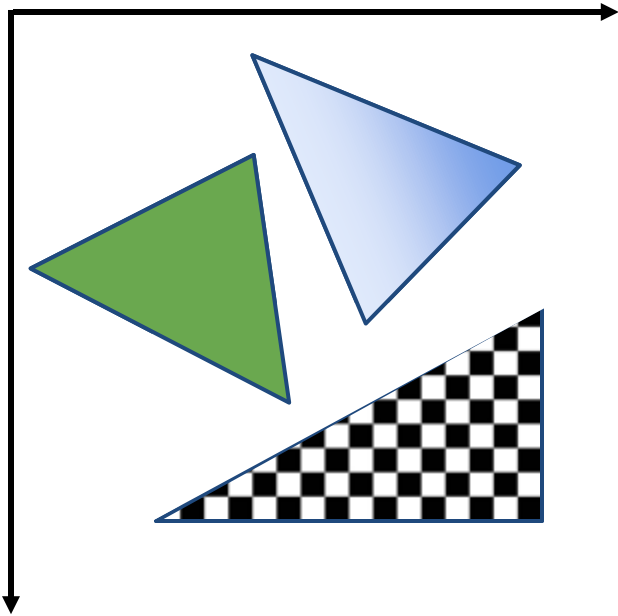


Quando não fica pior!

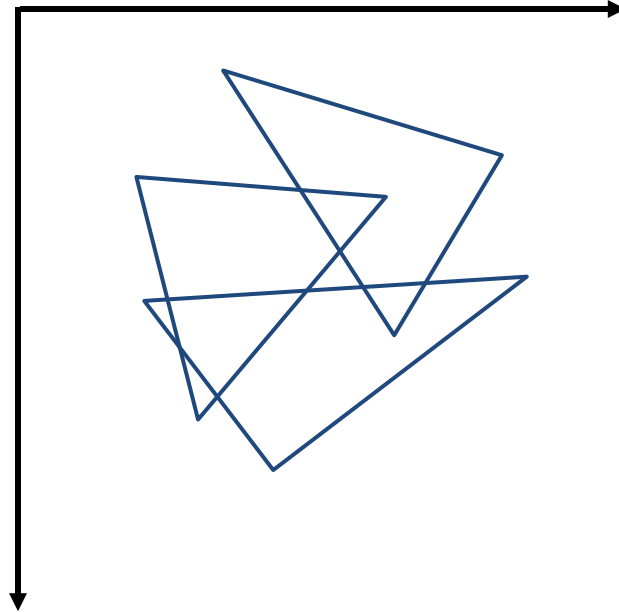


# Visibilidade

Primeiro. Como saber quais pixels pintar se houver muitos polígonos.



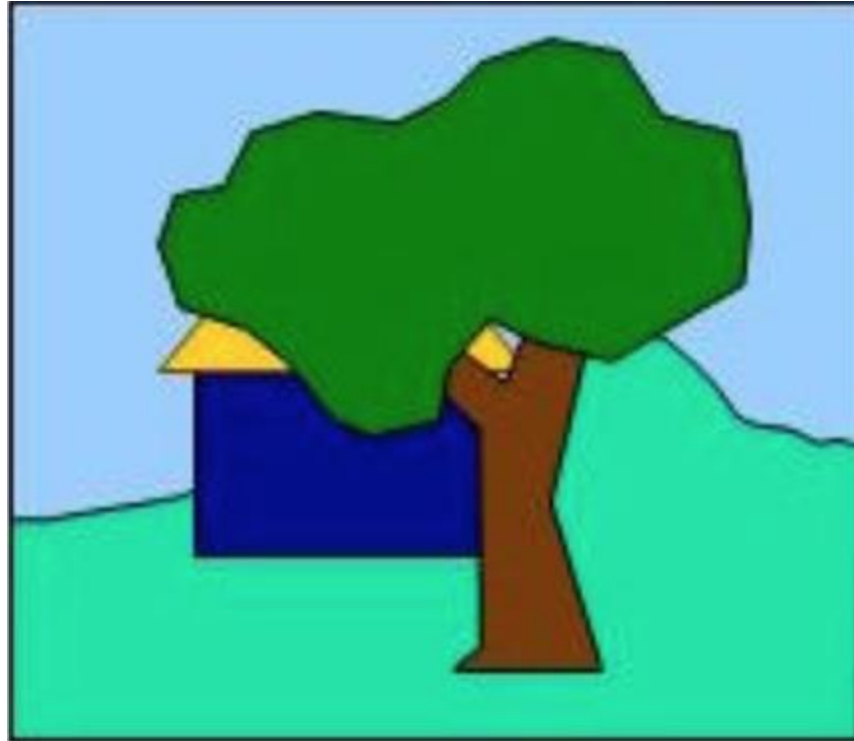
E se eles estiverem se intersectando?




# Algoritmo do Pintor



Inspirado em como pintores pintam


Pinte de trás para a frente, sobrescrevendo o framebuffer



# PS1: Depth Ordering Table (OT)

 Graphics Processing Unit (GPU)

  Search

 GitHub  
☆ 57 🗨 28

**PlayStation Specifications - psx-spx**

- Home
- Memory Map
- I/O Map
- Graphics Processing Unit (GPU)**
- Geometry Transformation Engine (GTE)
- Macroblock Decoder (MDEC)
- Sound Processing Unit (SPU)
- Interrupts
- DMA Channels
- Timers
- CDROM Drive
- Controllers and Memory Cards
- Pocketstation
- Serial Port (SIO)
- Expansion Port (PIO)
- Memory Control
- Unpredictable Things
- CPU Specifications
- Kernel (BIOS)
- Arcade Cabinets
- Konami System 573

## GPU Depth Ordering

### Absent Depth Buffer

The PlayStation's GPU stores only RGB colors in the framebuffer (ie. unlike modern 3D processors, it's NOT buffering Depth values; leaving apart the Mask bit, which could be considered as a tiny 1bit "Depth" or "Priority" value). In fact, the GPU supports only X,Y coordinates, and it's totally unaware of Z coordinates. So, when rendering a polygon, the hardware CANNOT determine which of the new pixels are in front/behind of the old pixels in the buffer.

### Simple Ordering

The rendering simply takes place in the ordering as the data is sent to the GPU (ie. the most distant objects should be sent first). For 2D graphics, it's fairly easy follow that order (eg. even multi-layer 2D graphics can be using DMA2-continous mode).

### Depth Ordering Table (OT)

For 3D graphics, the ordering of the polygons may change more or less randomly (eg. when rotating/moving the camera). To solve that problem, the whole rendering data is usually first stored in a Depth Ordering Table (OT) in Main RAM, and, once when all polygons have been stored in the OT, the OT is sent to the GPU via "DMA2-linked-list" mode.

### Initializing an empty OT (via DMA6)

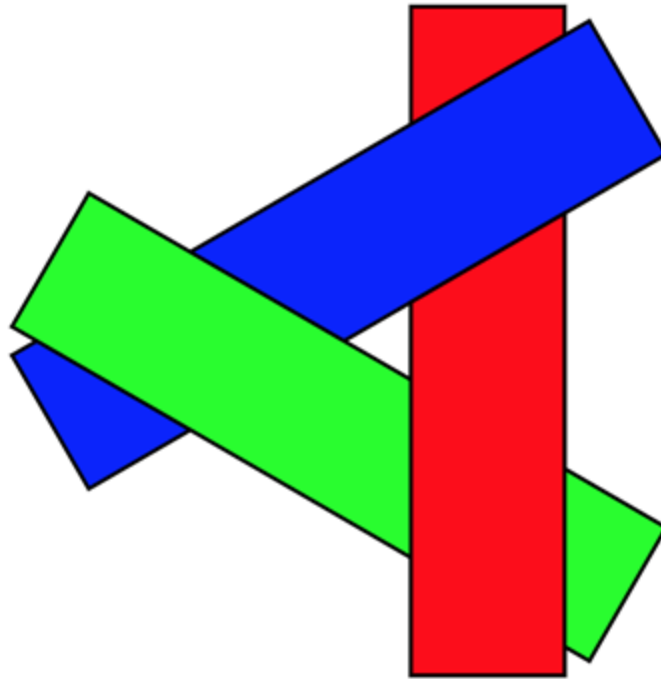
DMA channel 6 can be used to set up an empty linked list, in which each entry points to the previous:

### Table of contents

- GPU I/O Ports, DMA Channels, Commands, VRAM
  - GPU I/O Ports (1F801810h and 1F801814h in Read/Write Directions)
- GPU Timers / Synchronization
- GPU-related DMA Channels (DMA2 and DMA6)
- GPU Command Summary
- Clear Cache
- Quick Rectangle Fill
- VRAM Overview / VRAM Addressing
- GPU Render Polygon Commands
  - Notes
- GPU Render Line Commands
  - Note
- Wire-Frame
- GPU Render Rectangle Commands
  - Texture Origin and X/Y-Flip
  - Note
- GPU Rendering Attributes
  - Vertex (Parameter for Polygon, Line, Rectangle commands)

# Algoritmo do Pintor

Requer uma ordenação em profundidade  $O(n \log n)$  para  $n$  triângulos  
Porém, podem ocorrer situações de ordem não resolvíveis



# Z-Buffer

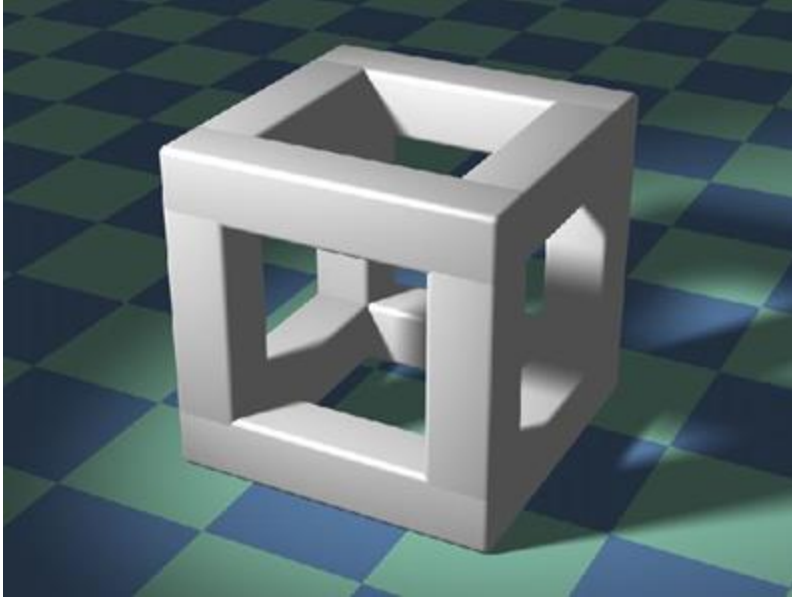
Este é o algoritmo de remoção de superfície oculta usado atualmente em sistema gráficos.



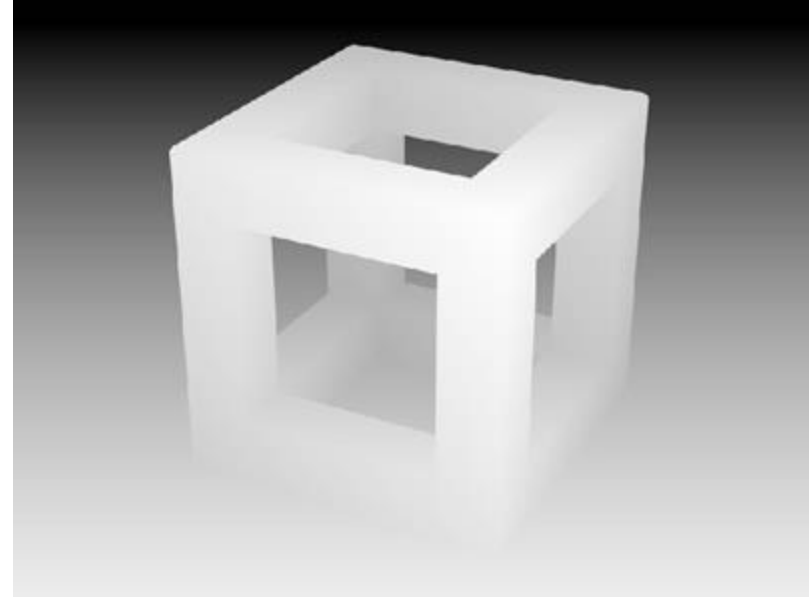
## Princípio:

- Armazene o valor da profundidade (coordenada  $z$ ) para cada posição de amostrada
- Necessário mais uma região de memória para os valores de profundidade
  - O framebuffer armazena valores de cores RGB
  - O depth buffer (z-buffer) armazena profundidade (16 a 32 bits)
  - Tradicionalmente são armazenados em ponto fixo (inteiros)

# Exemplos de Z-Buffer



Buffer de Cores



Z Buffer

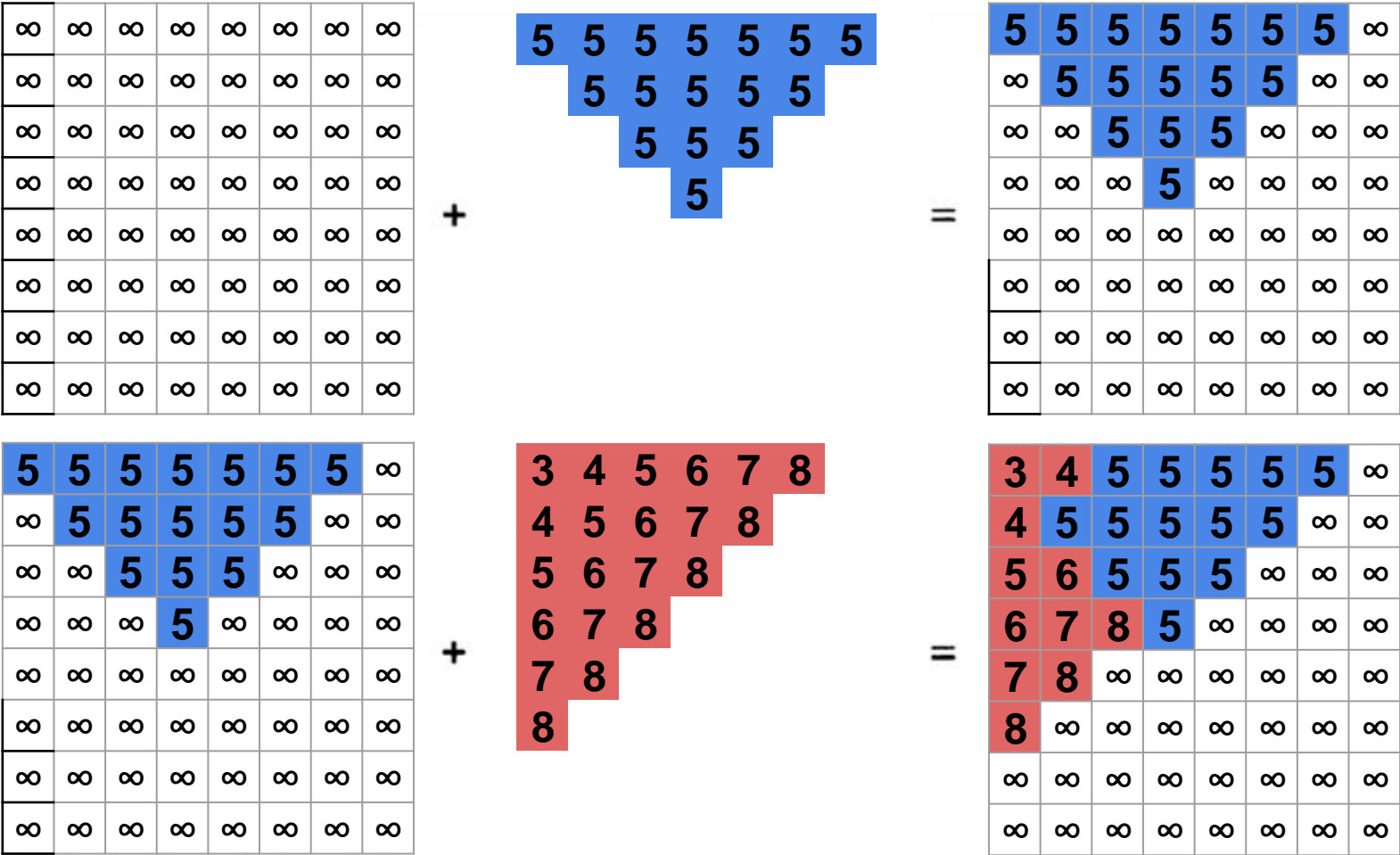
# Algoritmo do Z-Buffer

Inicialize o buffer com valores no infinito.

Durante a rasterização:

```
for (cada triângulo T)
  for (cada amostra (x,y,z) em T)
    if (z < zbuffer[x,y]) # amostra mais perto até o momento
      zbuffer[x,y] = z # atualiza a distância z
      framebuffer[x,y] = rgb # atualize a cor
    else
      pass      # não faz nada
```

# Algoritmo do Z-Buffer





# Complexidade do Z-Buffer

Complexidade

- $O(n)$  para  $n$  triângulos

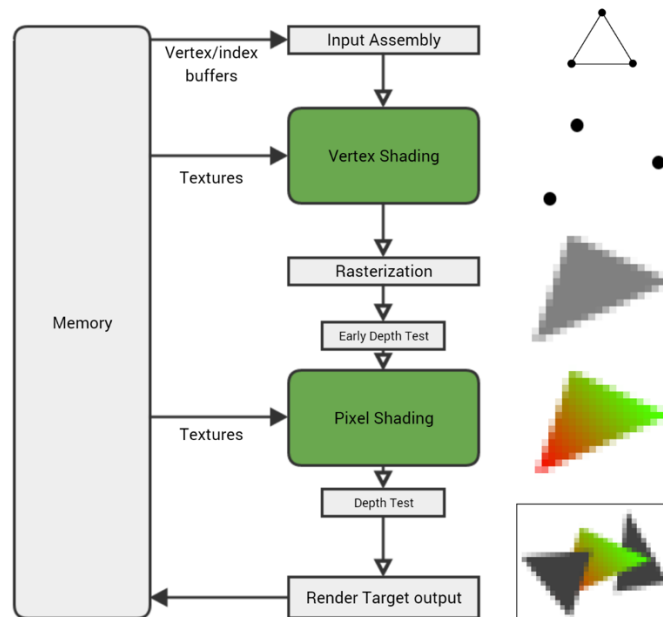
Algoritmo de visibilidade mais usado

- Implementado em hardware para todas as GPUs
- Usado por OpenGL, Vulkan, Direct3D, ...

# Dica

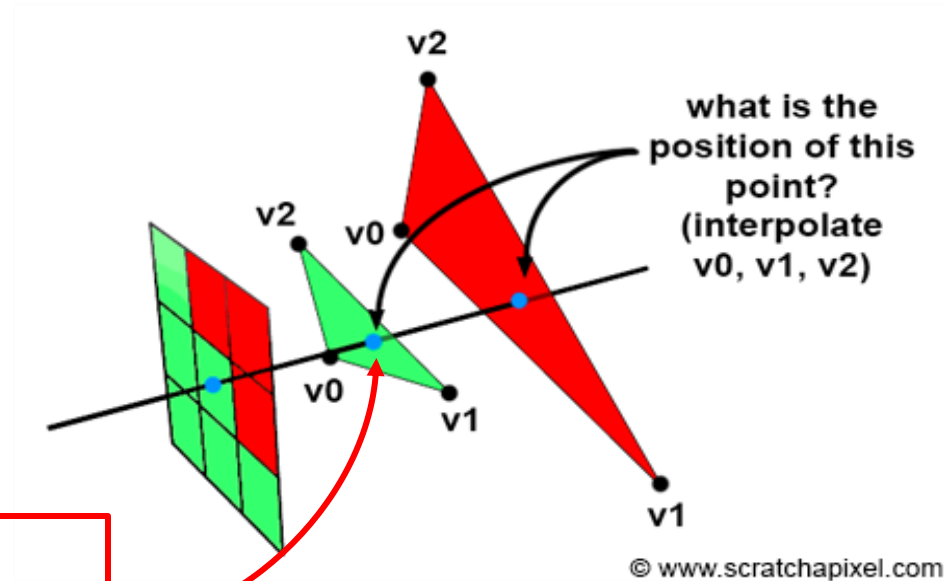
Quando fazer o teste de profundidade?  
Antes ou depois de verificar a cor do vértice?  
**Antes (mas nem sempre é possível)**

A maioria das GPUs oferece suporte ao Early Depth Test, esse teste permite executar o teste de profundidade antes da execução do fragment shader. Sempre que estiver claro que um fragmento não será visível (está atrás de outros objetos), podemos descartar o fragmento prematuramente.



# Identificando a posição do Z

Temos de identificar o valor do Z para cada pixel usando as fórmulas já estudadas.



$$Z = \frac{1}{\alpha \frac{1}{Z_0} + \beta \frac{1}{Z_1} + \gamma \frac{1}{Z_2}}$$

# Precisão do Z-Buffer

O Z-Buffer tem uma maior precisão dos pontos próximos ao Near

$$P = \begin{bmatrix} \frac{near}{right} & 0 & 0 & 0 \\ 0 & \frac{near}{top} & 0 & 0 \\ 0 & 0 & -\frac{far+near}{far-near} & -\frac{2far*near}{far-near} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Exemplo:

Near = 10

Far = 1000

Lembrando que o NDC vai de -1 a 1.

=> Normalizando para 0 a 1 e depois convertido para inteiro para o espaço do Z-buffer

$$\begin{bmatrix} 0 \\ 0 \\ -10 \\ 1 \end{bmatrix} \quad \begin{array}{l} Z = -1 \\ Z\text{-Normalizado} = 0 \\ Z\text{-Buffer}^{32b} = 0 \end{array}$$

$$\begin{bmatrix} 0 \\ 0 \\ -20 \\ 1 \end{bmatrix} \quad \begin{array}{l} Z = \sim 0 \\ Z\text{-Normalizado} = \sim 0,5 \\ Z\text{-Buffer}^{32b} = 2.169.175.402 \end{array}$$

$$\begin{bmatrix} 0 \\ 0 \\ -100 \\ 1 \end{bmatrix} \quad \begin{array}{l} Z = \sim 0,8 \\ Z\text{-Normalizado} = \sim 0,9 \\ Z\text{-Buffer}^{32b} = 3.904.515.723 \end{array}$$

$$\begin{bmatrix} 0 \\ 0 \\ -1000 \\ 1 \end{bmatrix} \quad \begin{array}{l} Z = 1 \\ Z\text{-Normalizado} = 1 \\ Z\text{-Buffer}^{32b} = 4.294.967.295 \end{array}$$

$$\begin{bmatrix} 0 \\ 0 \\ -40 \\ 1 \end{bmatrix} \quad \begin{array}{l} Z = \sim 0,5 \\ Z\text{-Normalizado} = \sim 0,75 \\ Z\text{-Buffer}^{32b} = 3.253.763.102 \end{array}$$

$$\begin{bmatrix} 0 \\ 0 \\ -500 \\ 1 \end{bmatrix} \quad \begin{array}{l} Z = \sim 0,98 \\ Z\text{-Normalizado} = \sim 0,99 \\ Z\text{-Buffer}^{32b} = 4.251.583.787 \end{array}$$

# Precisão do Z-Buffer

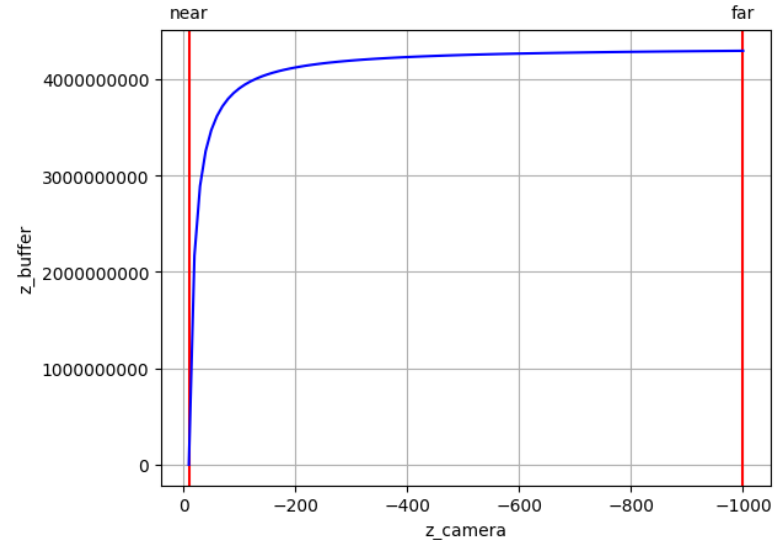
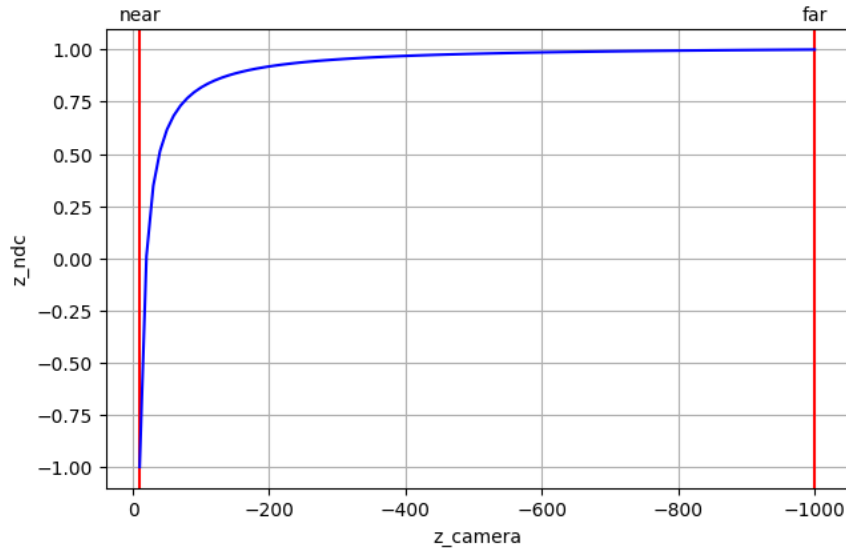
O Z-Buffer tem uma maior precisão dos pontos próximos ao Near

$$P = \begin{bmatrix} \frac{near}{right} & 0 & 0 & 0 \\ 0 & \frac{near}{top} & 0 & 0 \\ 0 & 0 & -\frac{far+near}{far-near} & -\frac{2far*near}{far-near} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Exemplo:

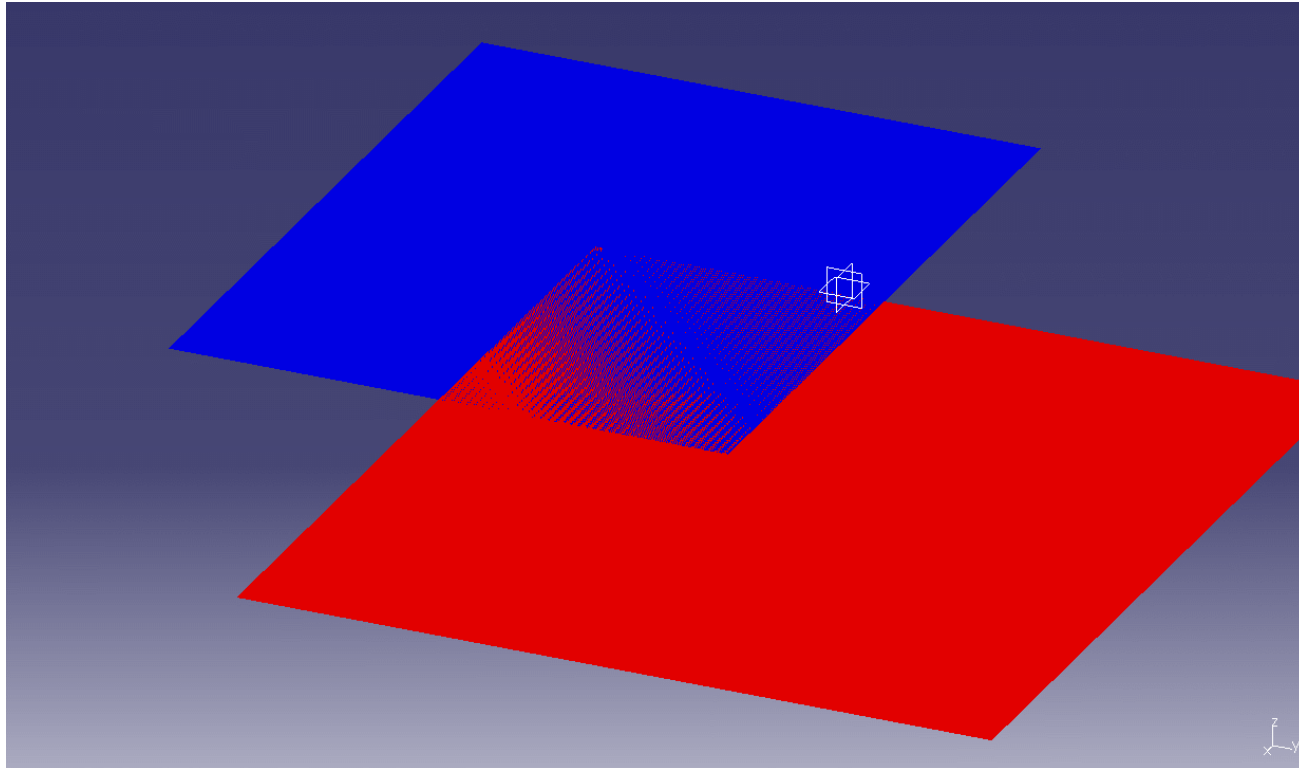
Near = 10

Far = 1000



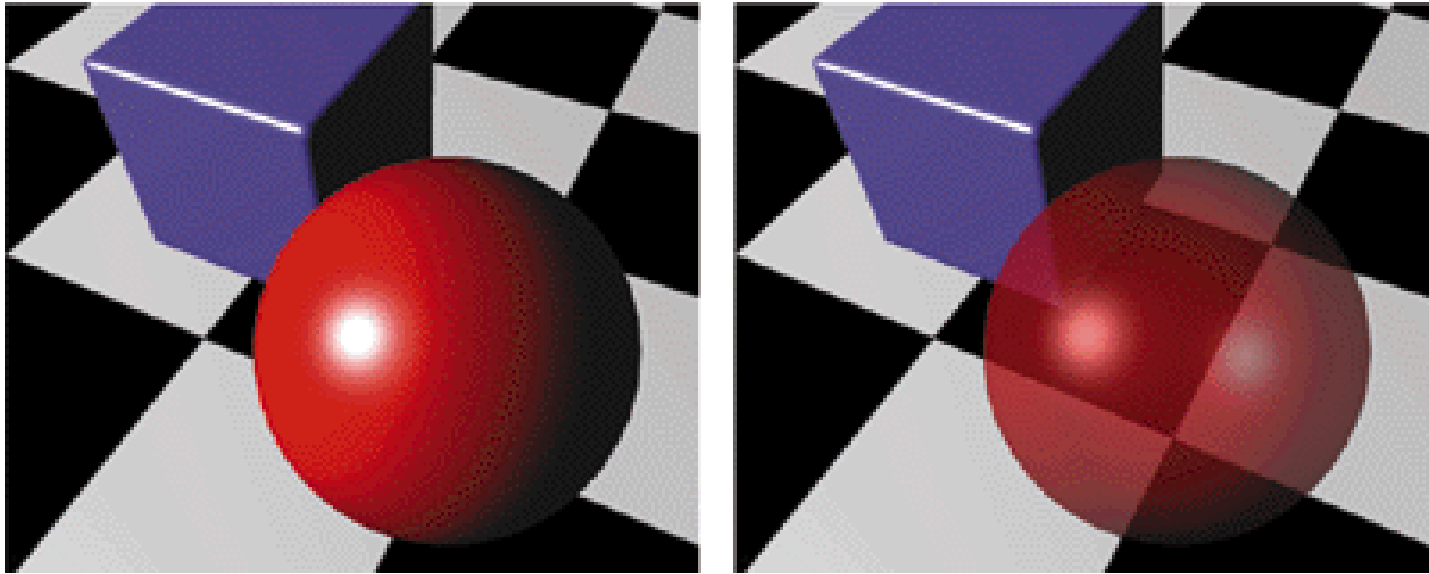
# Z-fighting

Problema clássico que ocorre quando não há como identificar claramente que superfície está a frente da outra.



# Transparência

Objetos podem ter transparências, isso permite que vejamos, o que está atrás do objeto em função de um valor de transparência.



# Transparência no X3D

O campo de *transparency* (**transparência**) especifica o quão "translúcido" é um objeto, com 1.0 sendo totalmente transparente e 0.0 totalmente opaco.

```
Material : X3DMaterialNode {  
  SFFloat   [in,out] ambientIntensity  0.2          [0,1]  
  SFColor  [in,out] diffuseColor      0.8 0.8 0.8 [0,1]  
  SFColor   [in,out] emissiveColor      0 0 0        [0,1]  
  SFNode    [in,out] metadata           NULL         [X3DMetadataObject]  
  SFFloat   [in,out] shininess          0.2          [0,1]  
  SFColor   [in,out] specularColor      0 0 0        [0,1]  
  SFFloat  [in,out] transparency      0          [0,1]  
}
```



# Cálculo da Transparência

Você vai precisar combinar a cor anterior com a cor do objeto.

```
for (cada triângulo T)
  for (cada amostra (x,y,z) em T)
    if (z < zbuffer[x,y]) # amostra mais perto até o momento
      zbuffer[x,y] = z # atualiza a distância z
      cor_anterior = framebuffer[x,y] * transparência
      cor_nova = rgb * (1 - transparência)
      framebuffer[x,y] = cor_anterior + cor_nova
    else
      pass      # não faz nada
```

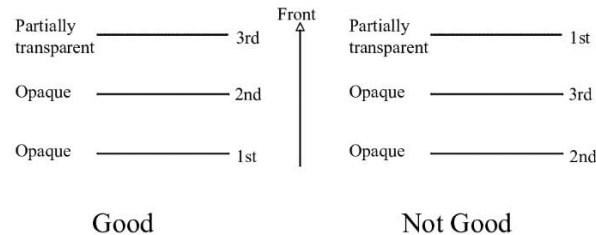
# Estratégias para resolver

1. desenhe objetos opacos primeiro usando o z-buffer
2. desenhar objetos transparentes do mais distante para o mais próximo

Não há necessidade de ordenar para a entrega do projeto

## Z-Buffer and Transparency

Transparency requires partial sorting



Another solution:

- Linked list of RGB-Z- $\alpha$  at each pixel (Alpha Buffer)

# Renderizador em Python

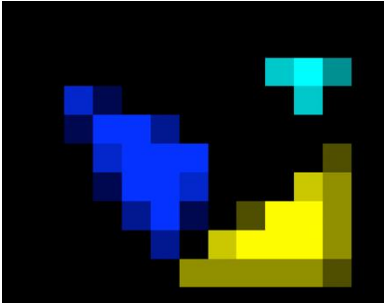
No Renderizador você poderá criar um buffer para o z-buffer:

```
gpu.GPU.framebuffer_storage(  
    self.framebuffers["FRONT"],  
    gpu.GPU.DEPTH_ATTACHMENT,  
    gpu.GPU.DEPTH_COMPONENT32F,  
    self.width,  
    self.height  
)
```

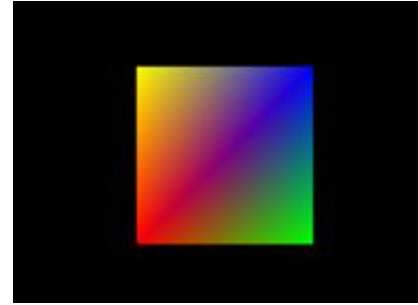
E depois usar as seguintes instruções:

```
gpu.GPU.read_pixel([x, y], gpu.GPU.DEPTH_COMPONENT32F)  
gpu.GPU.read_pixel([x, y], gpu.GPU.RGB8)
```

# Quarta parte do projeto 1



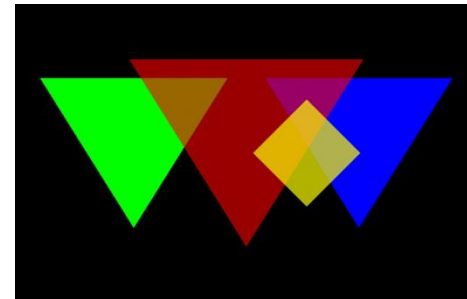
triangulos.x3d  
(anti-aliasing)



cores.x3d



retangulos.x3d

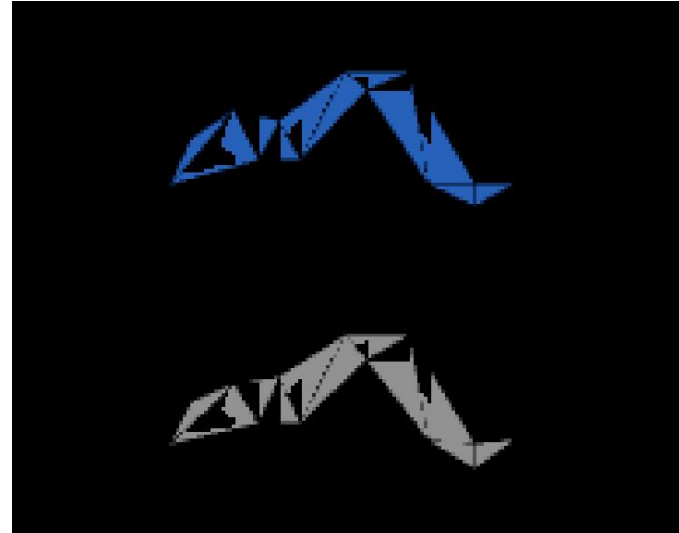
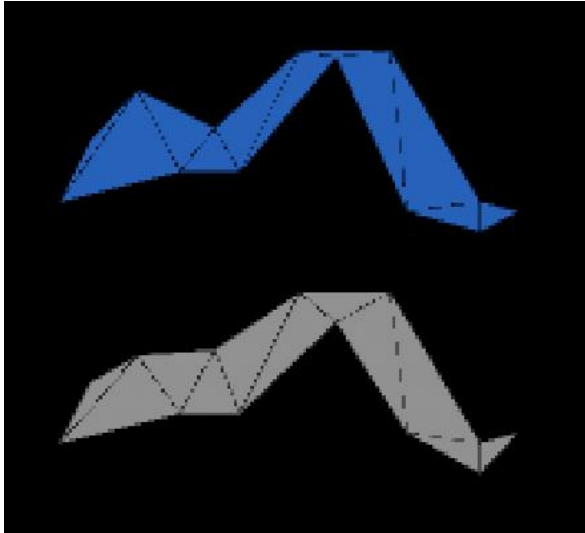


transparencia.x3d

<https://lpsoares.github.io/Renderizador/>

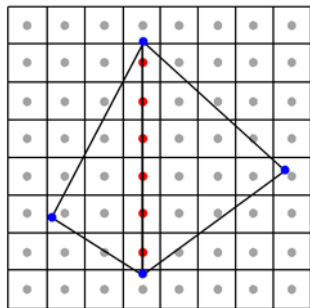
# Problemas que podem acontecer

Não vai funcionar se você tentar resolver dentro do *inside* test do pixel (só desenho do triângulo) e não usar um framebuffer adicional maior para o supersampling.

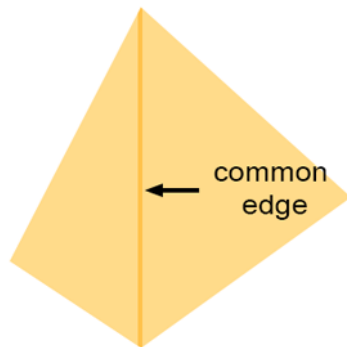


# Problemas que podem acontecer

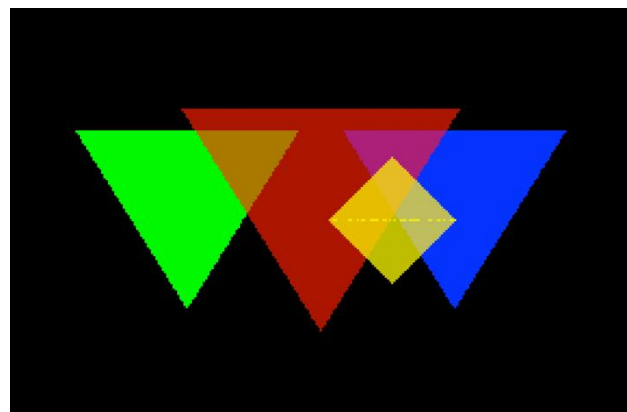
Devido a transparência, uma borda pode aparecer na junção dos polígonos. Isso acontece que os pixels amostram os dois triângulos.



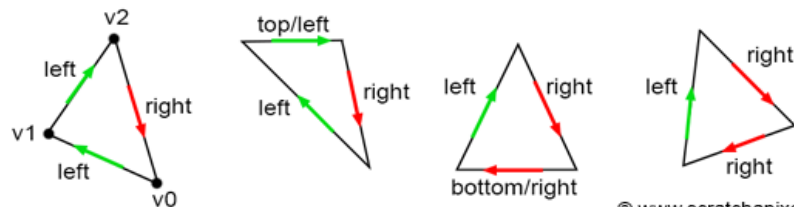
© www.scratchapixel.com



© www.scratchapixel.com



Para resolver isso seu renderizador precisaria alguma estratégia como: “**top-left rule**”



© www.scratchapixel.com

# Computação Gráfica

Luciano Soares

<lpsoares@insper.edu.br>

Fabio Orfali

<fabioO1@insper.edu.br>

## Referência interessante:

<https://www.scratchapixel.com/lessons/3d-basic-rendering/rasterization-practical-implementation/visibility-problem-depth-buffer-depth-interpolation.html>