

# Computação Gráfica

Aula 11: Mapeamento de Texturas



# Introdução teórica

## Superfícies parametrizadas

# Superfícies parametrizadas

Na espetacular disciplina de **Matemática Multivariada**, vocês estudaram as curvas e superfícies parametrizadas. Vamos relembrar um pouco desse assunto, que servirá como base teórica para o estudo do mapeamento de texturas.

# Curva parametrizada

Relacionando cada número real  $t$  a um ponto do espaço definido pela terna ordenada  $(f(t), g(t), h(t))$ , podemos definir uma curva parametrizada. Nesse caso, a variável  $t$  é chamada de **parâmetro**.

Vamos ver um exemplo simples para relembrar.

# Curva parametrizada

Considere a curva  $\gamma$  definida pela equação:

$$\gamma(t) = (2 \cos t, 2 \sin t, 2 + \sin 4t) \quad t \in [0, 2\pi]$$

Você consegue imaginar o formato dessa curva?

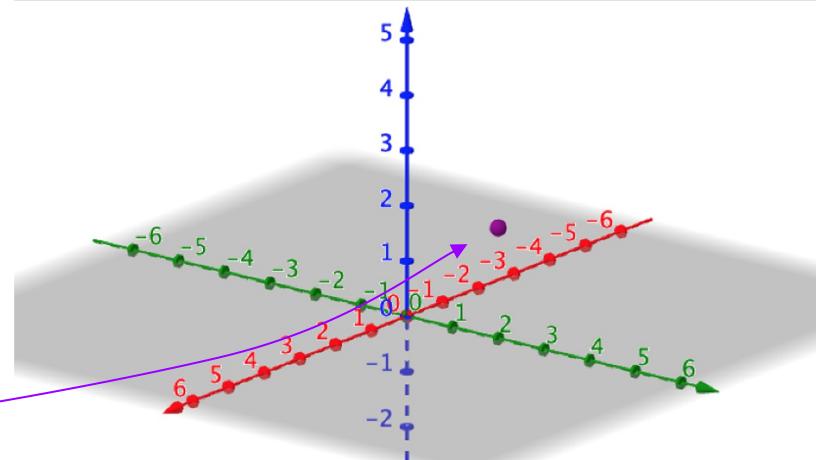
# Curva parametrizada

$$\gamma(t) = (2 \cos t, 2 \sin t, 2 + \sin 4t)$$

$$t \in [0, 2\pi]$$

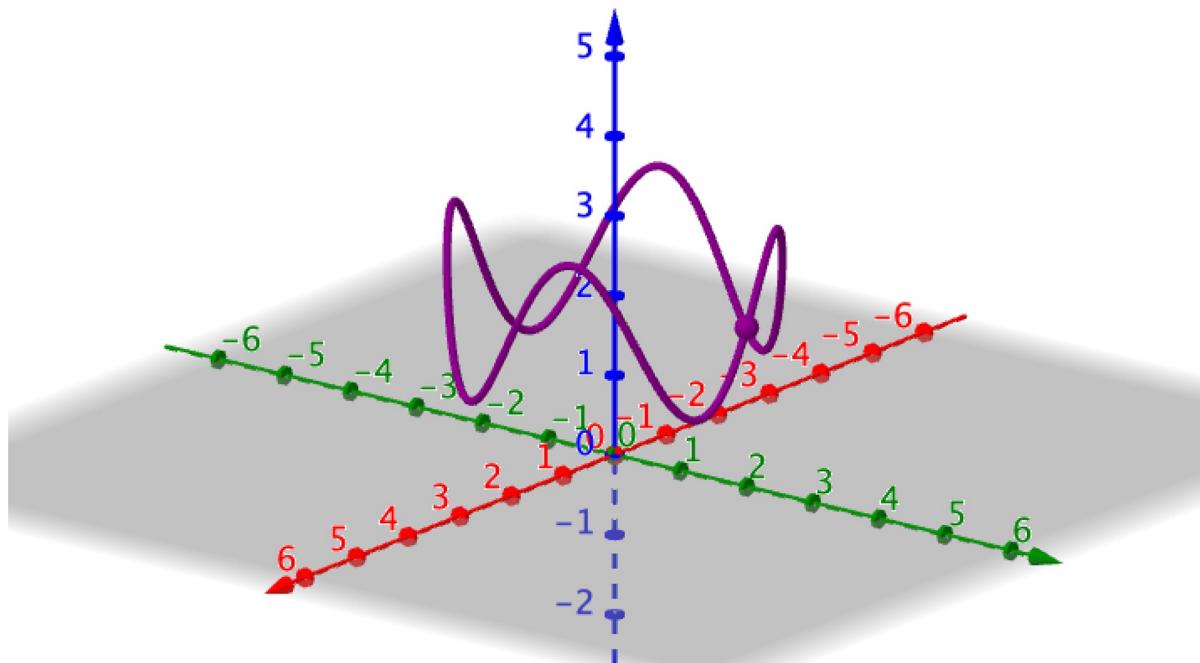


$$\gamma\left(\frac{\pi}{2}\right) = \left(2 \cos \frac{\pi}{2}, 2 \sin \frac{\pi}{2}, 2 + \sin 4 \cdot \frac{\pi}{2}\right) = (0, 2, 2)$$



# Curva parametrizada

$$\gamma(t) = (2 \cos t, 2 \sin t, 2 + \sin 4t) \quad t \in [0, 2\pi]$$



# Superfície parametrizada

Analogamente ao que fizemos para curvas, vamos relacionar cada par ordenado de números reais  $(u, v)$  a um ponto do espaço definido pela terna ordenada  $(f(u, v), g(u, v), h(u, v))$ . Com isso, estamos definindo uma **superfície parametrizada**, sendo  $u$  e  $v$  os parâmetros.

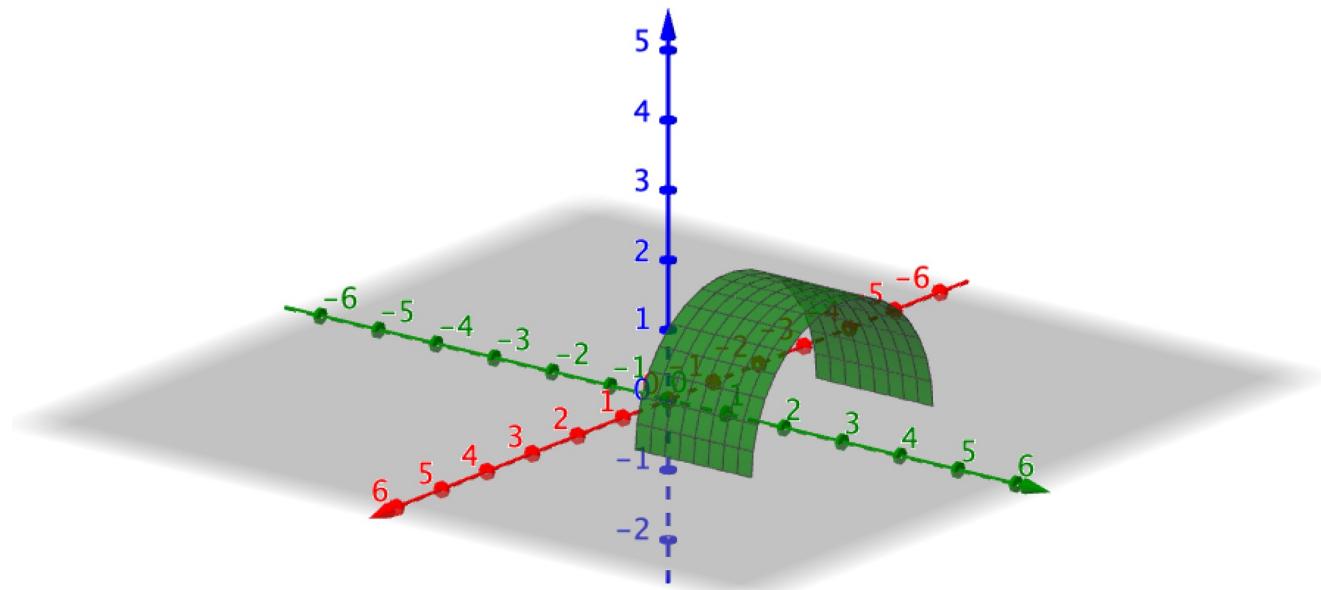
Vamos ver um exemplo.

# Superfície parametrizada

$$r(u, v) = (2 \cos u, v, 2 \sin u)$$

$$u \in [0, \pi]; v \in [1, 3]$$

Tente imaginar como seria essa superfície.



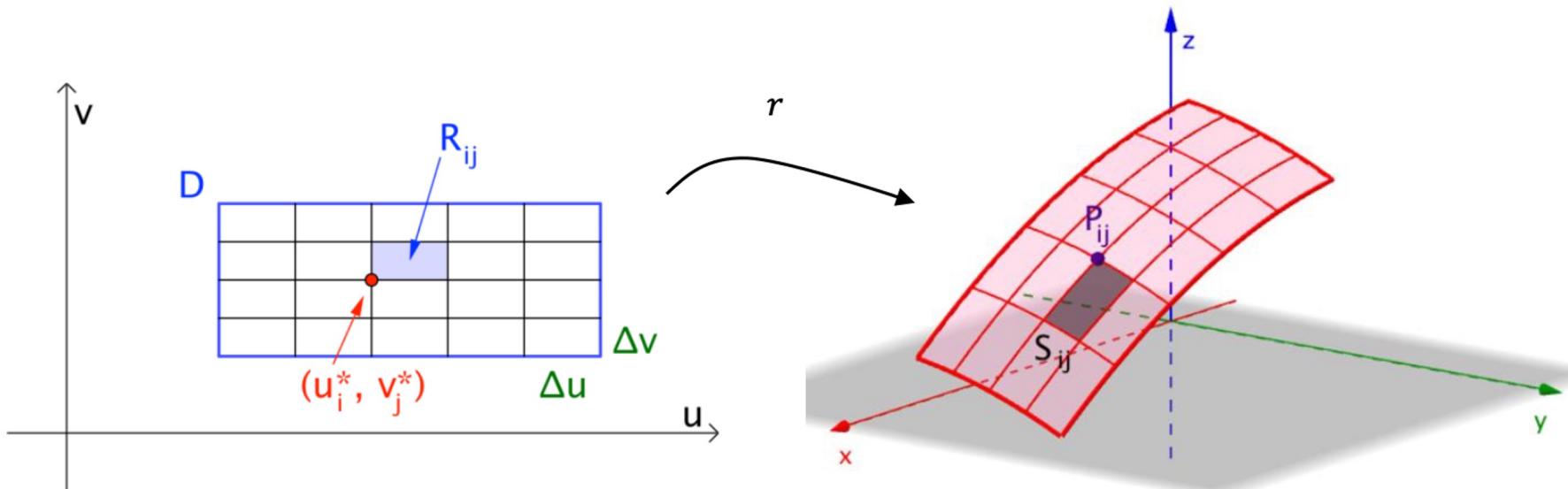
# Superfície parametrizada

Observe que, ao definir uma superfície, estamos associando cada ponto  $(u,v)$  de um plano a um ponto  $(x,y,z)$  do espaço.

Suponha que o plano seja decomposto em um quadriculado (qualquer semelhança com os *pixels* de uma tela NÃO é mera coincidência).

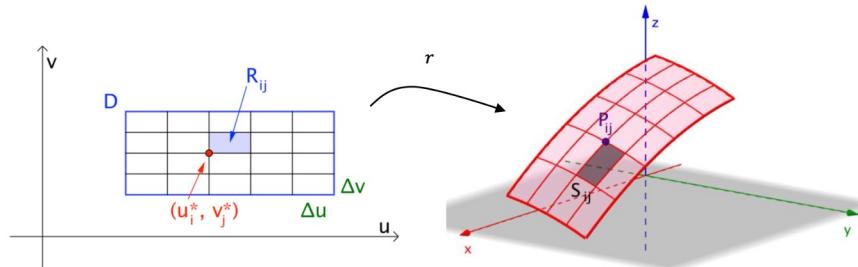
Como esse quadriculado seria visto na superfície?

# Superfície parametrizada



Vamos chamar esses "quadrados" que aparecem sobre a superfície de **retalhos**.

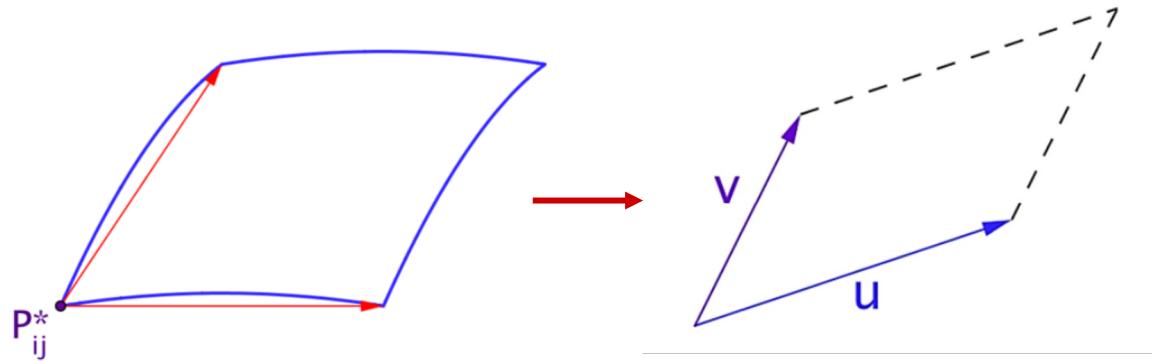
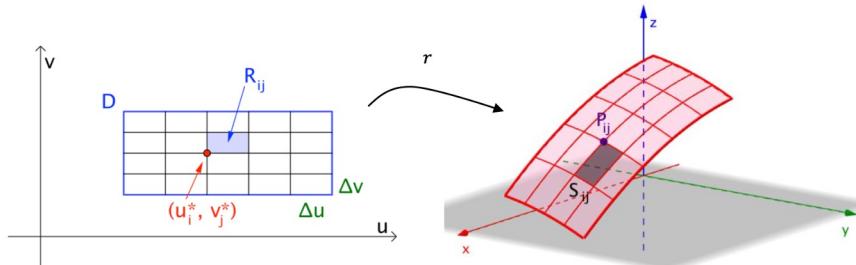
# Superfície parametrizada



Precisaremos determinar algumas características geométricas dos retalhos a partir dos parâmetros  $u$  e  $v$ .

Apesar dos retalhos serem superfícies curvas, podemos aproxima-los por paralelogramos (quanto menores forem os quadrados da nossa malha, melhor será essa aproximação).

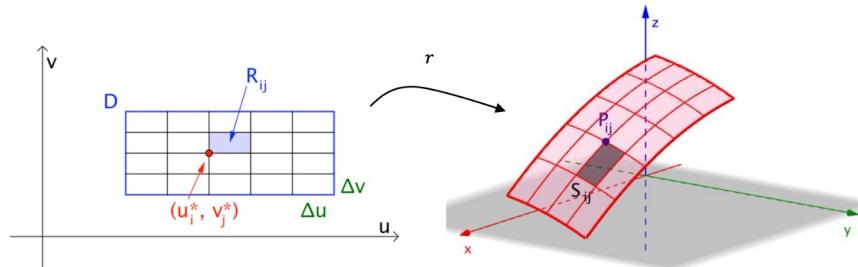
# Superfície parametrizada



Área do retalho:  $A_R \approx \|u \times v\|$

Comprimento dos lados do retalho:  $L_R \approx \|u\|$  e  $\|v\|$

# Superfície parametrizada



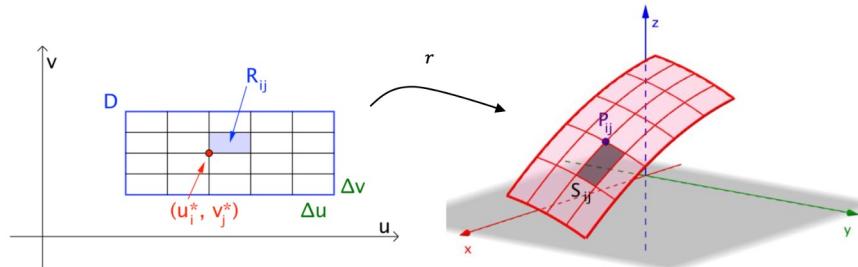
E como determinar os vetores  $u$  e  $v$  que formam o paralelogramo representativo de cada retalho?

Lembrando das propriedades das derivadas, temos que, para  $\Delta x$  e  $\Delta y$  pequenos, podemos usar as aproximações:

$$\Delta f \approx \frac{\partial f}{\partial x} \cdot \Delta x$$

$$\Delta f \approx \frac{\partial f}{\partial y} \cdot \Delta y$$

# Superfície parametrizada

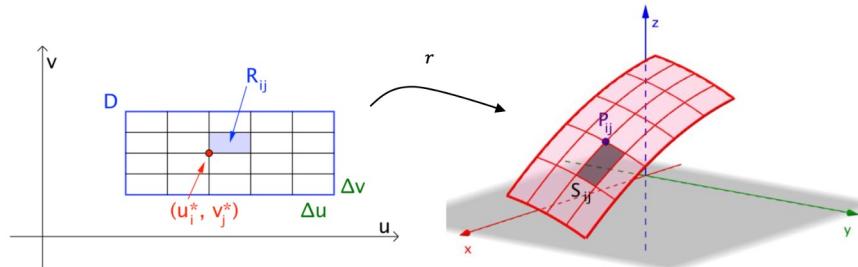


Dessa forma, para construir nosso paralelogramo "aproximador" dos retalhos, tomamos dois vetores tangentes à superfície no ponto  $P_{ij}$ , dados por:

$$r_u^* = \frac{\partial r}{\partial u}(u_i^*, v_j^*)$$

$$r_v^* = \frac{\partial r}{\partial v}(u_i^*, v_j^*)$$

# Superfície parametrizada

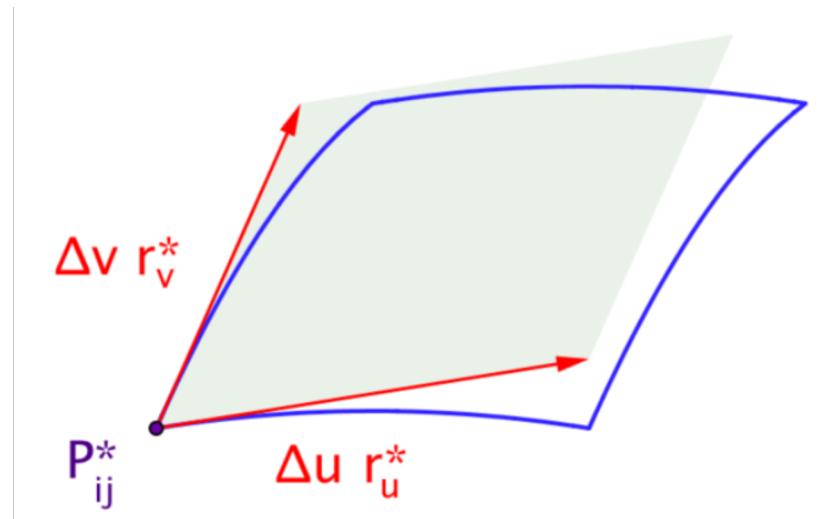


Multiplicando esses vetores por  $\Delta u$  e  $\Delta v$ , respectivamente, temos o paralelogramo que aproxima o retalho.

$$r_u^* = \frac{\partial r}{\partial u}(u_i^*, v_j^*)$$

---

$$r_v^* = \frac{\partial r}{\partial v}(u_i^*, v_j^*)$$





# Aplicando a teoria: O mapeamento de texturas

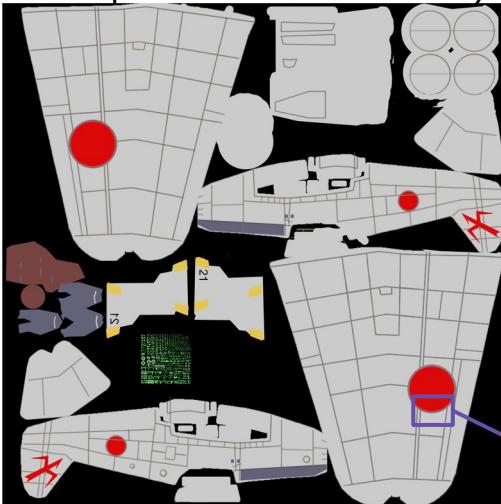
# Texel

Um **texel** (texture element) é a unidade fundamental de uma textura.

No processo de renderização existe um mapeamento do **texel** para o pixel apropriado na imagem final.

# O que é texturizar

Texture (originária por exemplo de um JPG ou PNG)

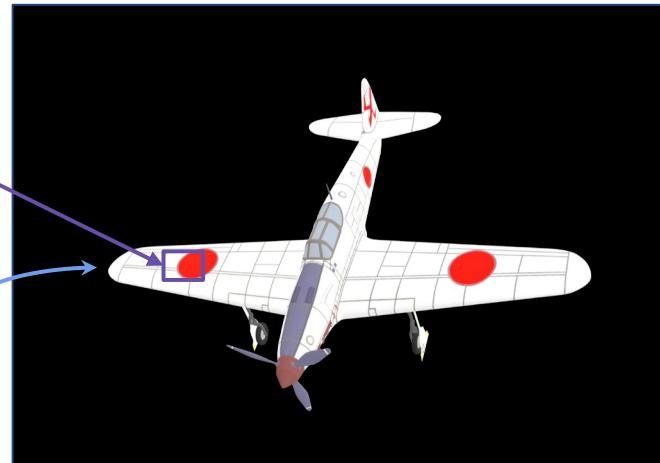


Na renderização, para definir a cor do pixel de um objeto texturizado é necessário identificar e calcular a cor pela textura

Geometria (composta por triângulos)

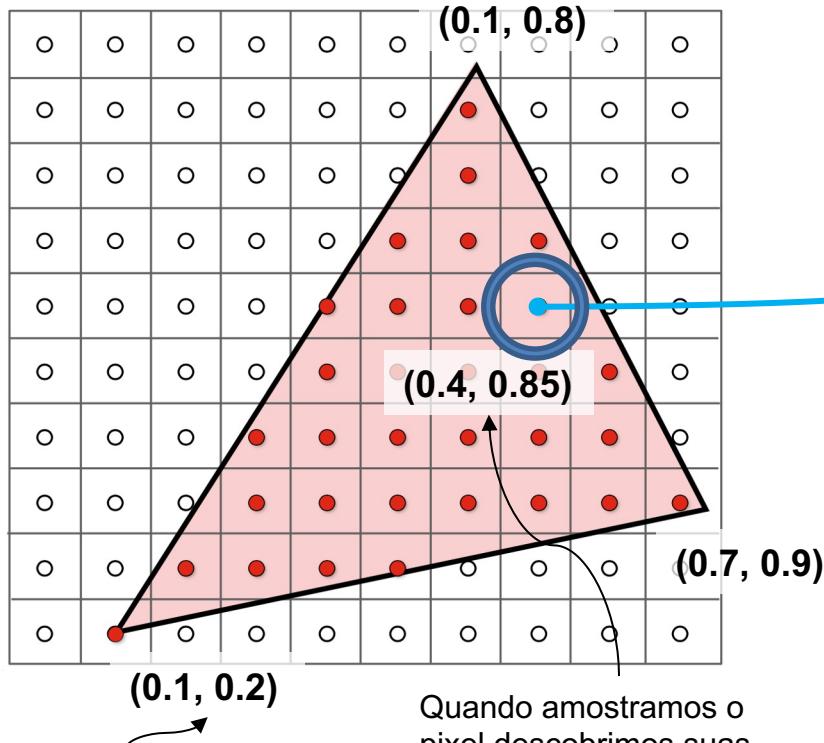


vértices possuem suas coordenadas no espaço, mas também coordenadas de mapeamento de textura



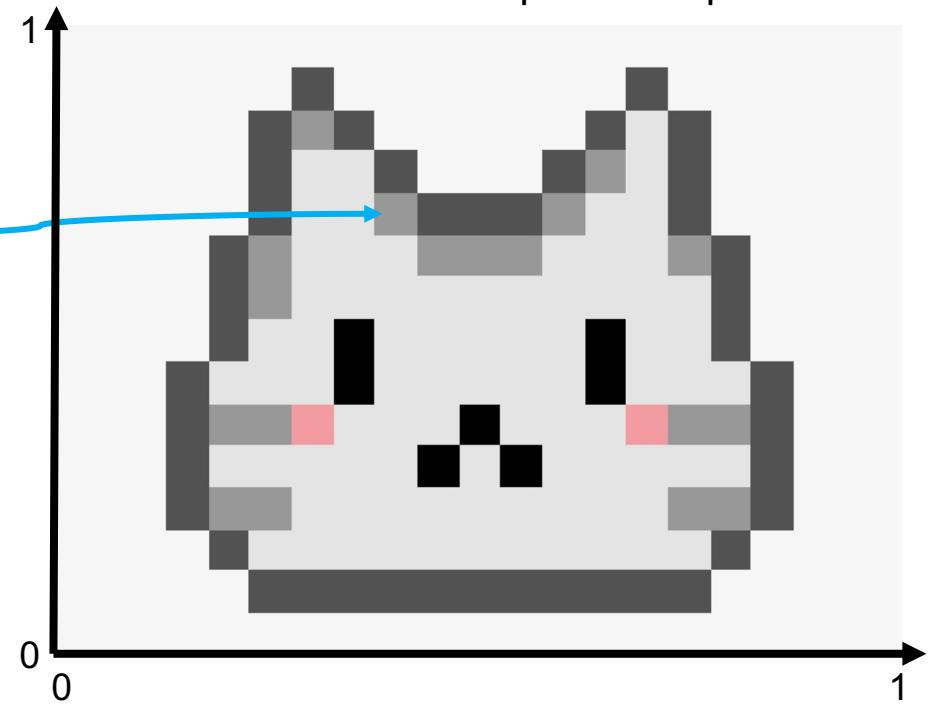
# Amostrando Texturas (pontuais)

Na rasterização amostramos o pixel

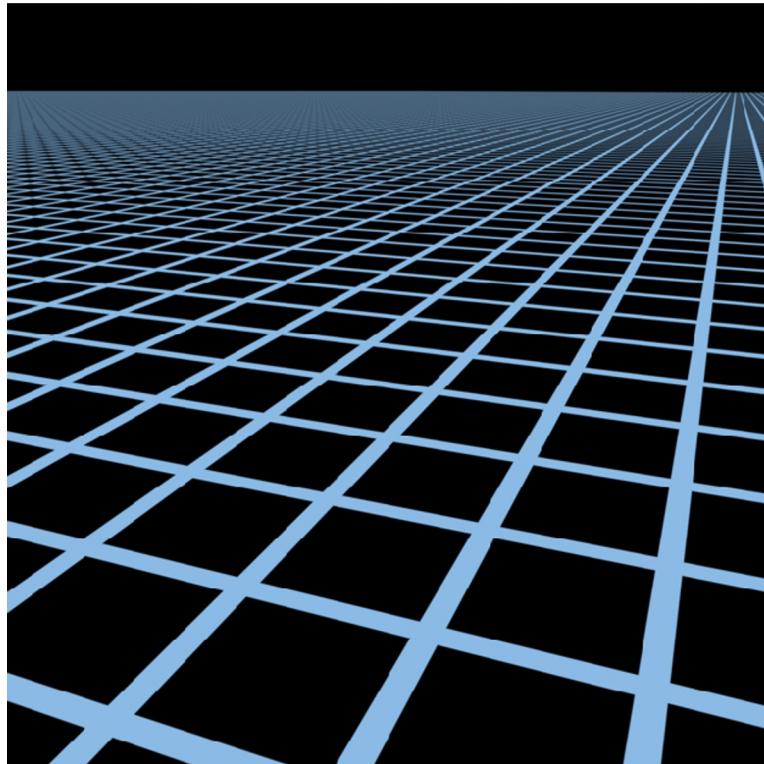


Quando amostramos o pixel descobrimos suas coordenadas baricêtricas

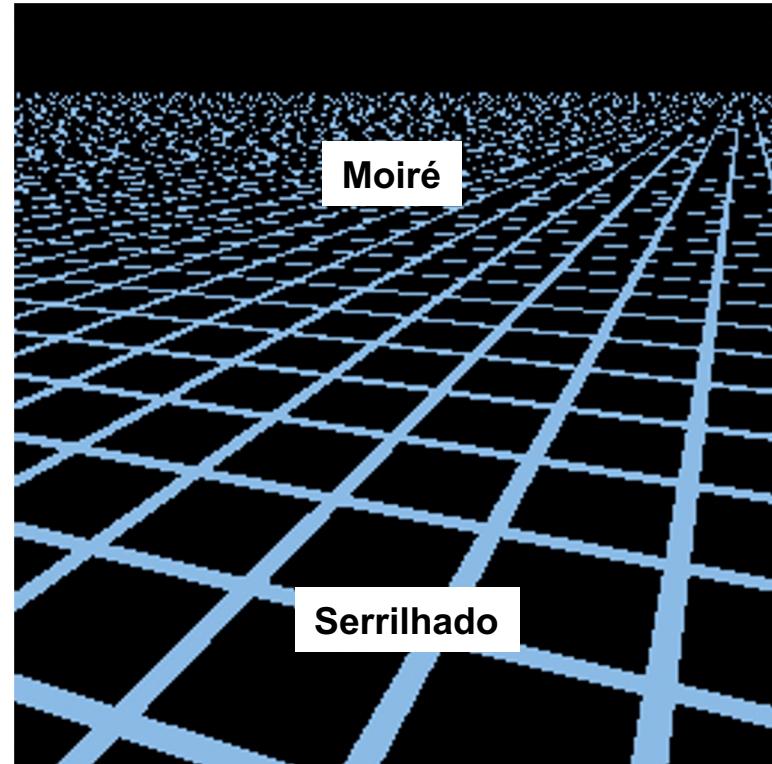
Podemos então descobrir as coordenadas de textura ( $u, v$ ), selecionar a cor e aplicar no pixel



# Amostrando Texturas (pontuais)

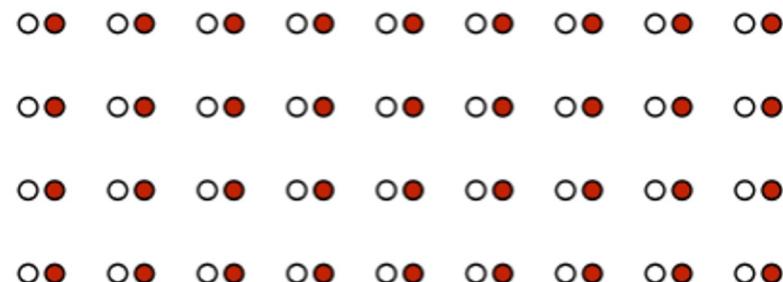
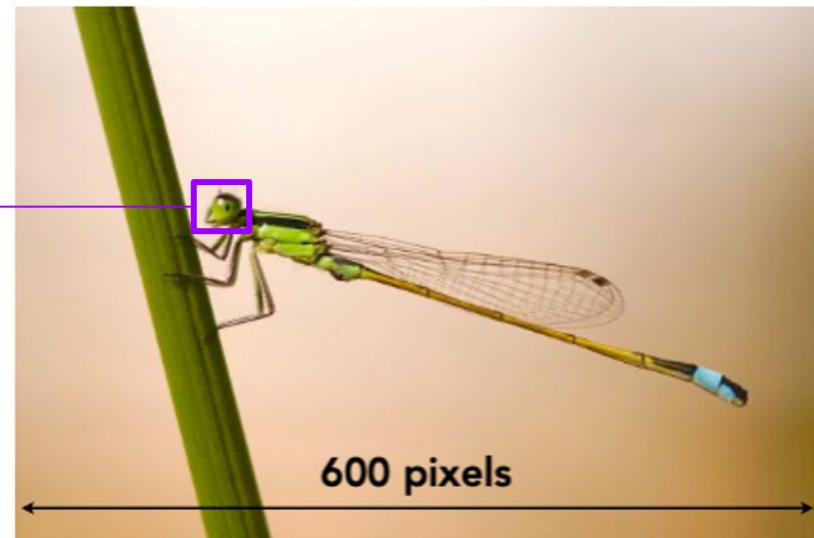
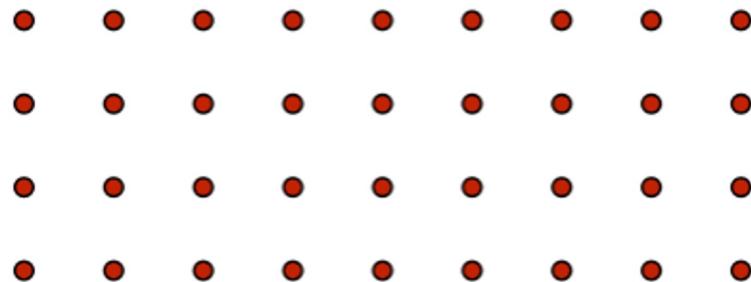
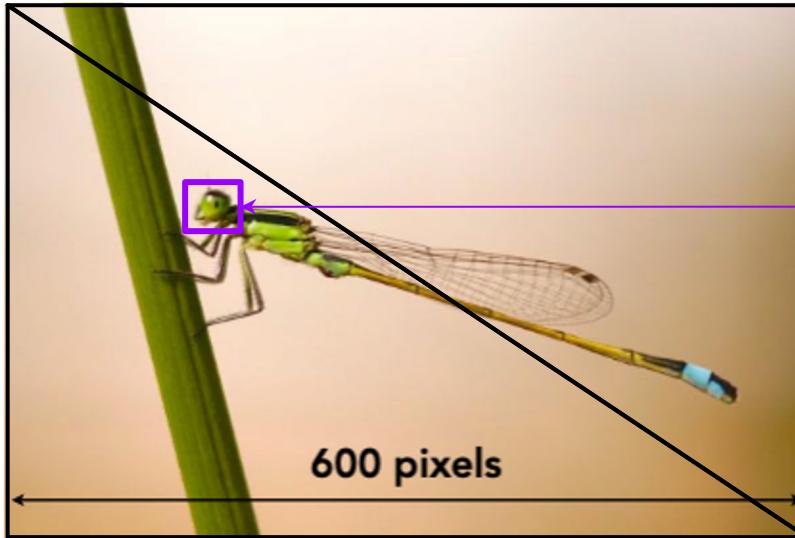


Referência em Alta-resolução  
imagem original em 1280x1280 pixels



Por amostragem (pontuais)  
256x256 pixels

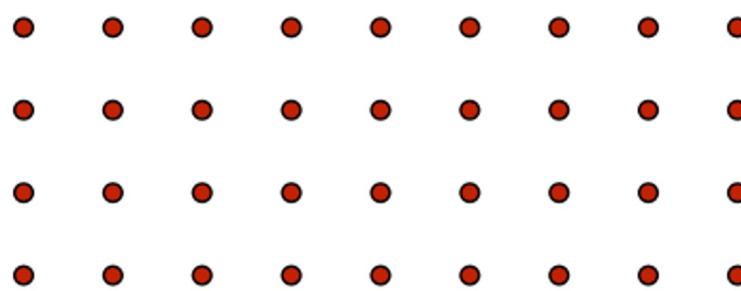
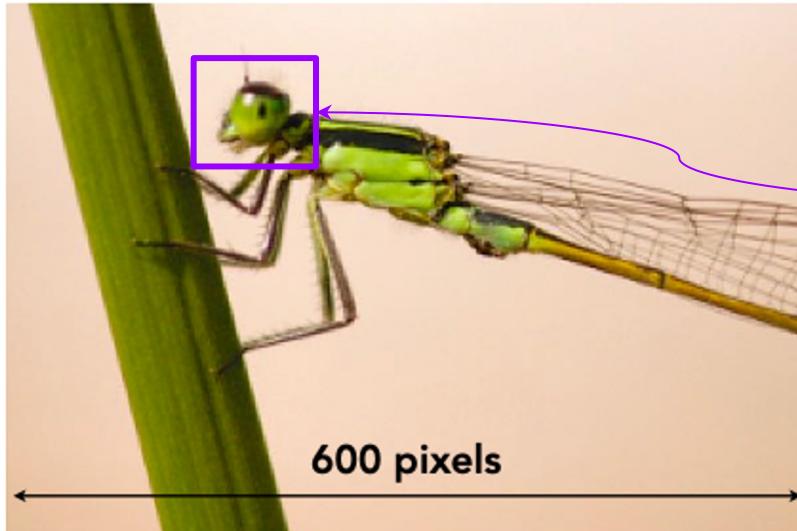
# Taxa de Amostras na Tela x na Textura



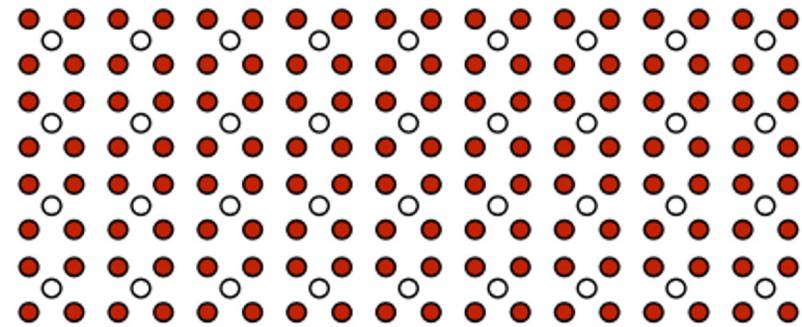
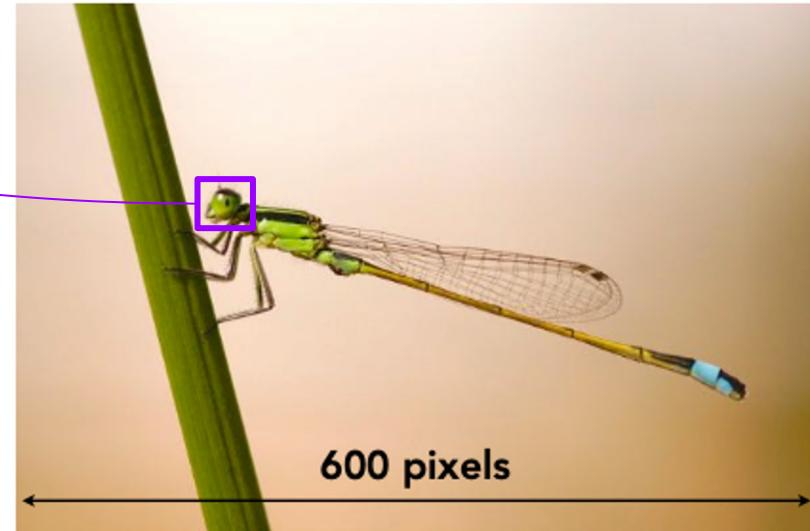
Mapeamento 1:1

Pontos vermelhos = amostras necessárias para renderizar  
Pontos brancos = amostras existentes no mapa de textura

# Taxa de Amostras na Tela x na Textura



Espaço da Tela (x,y)

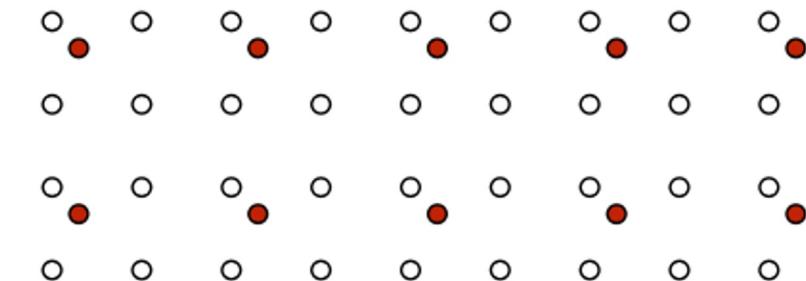
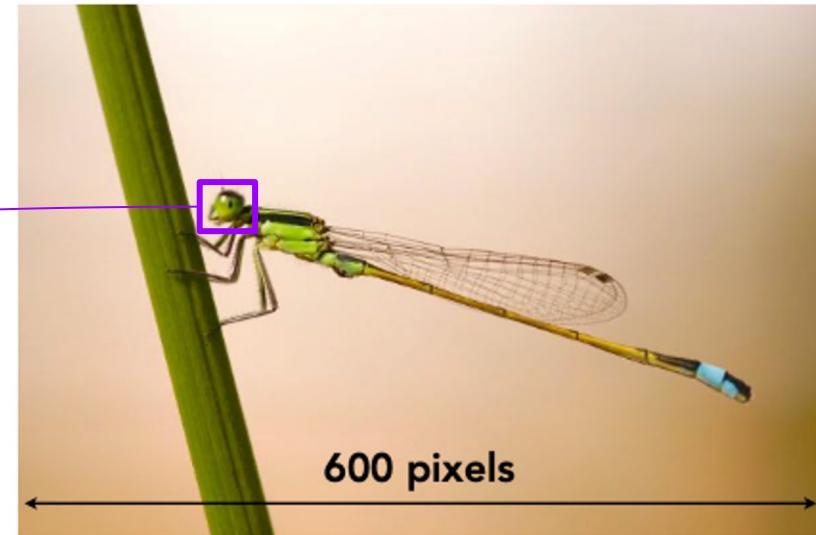
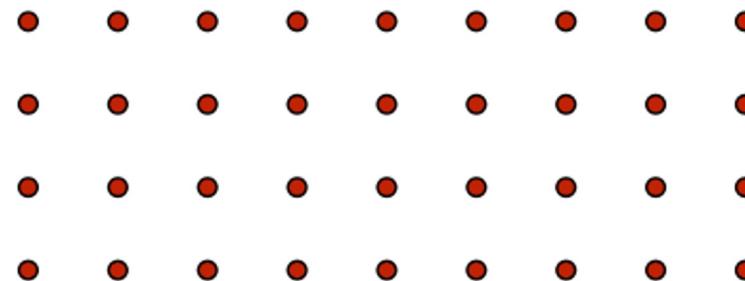
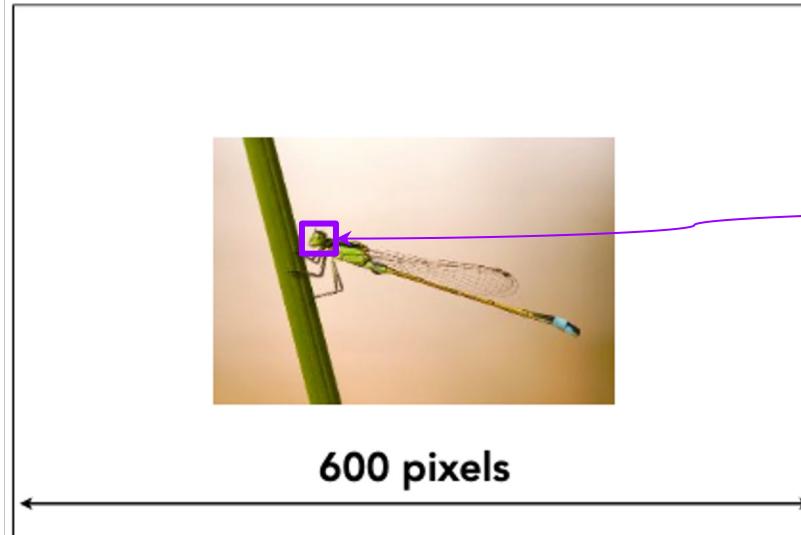


Espaço da Textura (u,v)

Pontos vermelhos = amostras necessárias para renderizar  
Pontos brancos = amostras existentes no mapa de textura

**Ampliado (magnified)**

# Taxa de Amostras na Tela x na Textura



**Reduzido (minified)**

Pontos vermelhos = amostras necessárias para renderizar  
Pontos brancos = amostras existentes no mapa de textura

# Taxa de Amostragem das Texturas

A frequência de amostragem no espaço da tela se traduz em uma frequência de amostragem no espaço de textura, determinado pela função de mapeamento.

Em geral, essa frequência varia ao longo da cena dependendo das transformações geométricas, das transformações de visualização e da função de coordenada de textura.

# Área do Pixel na Tela x Área do Texel

Tamanho de visualização ideal:

- Mapeamento 1:1 entre taxa de amostragem do pixel e taxa de amostragem do texel

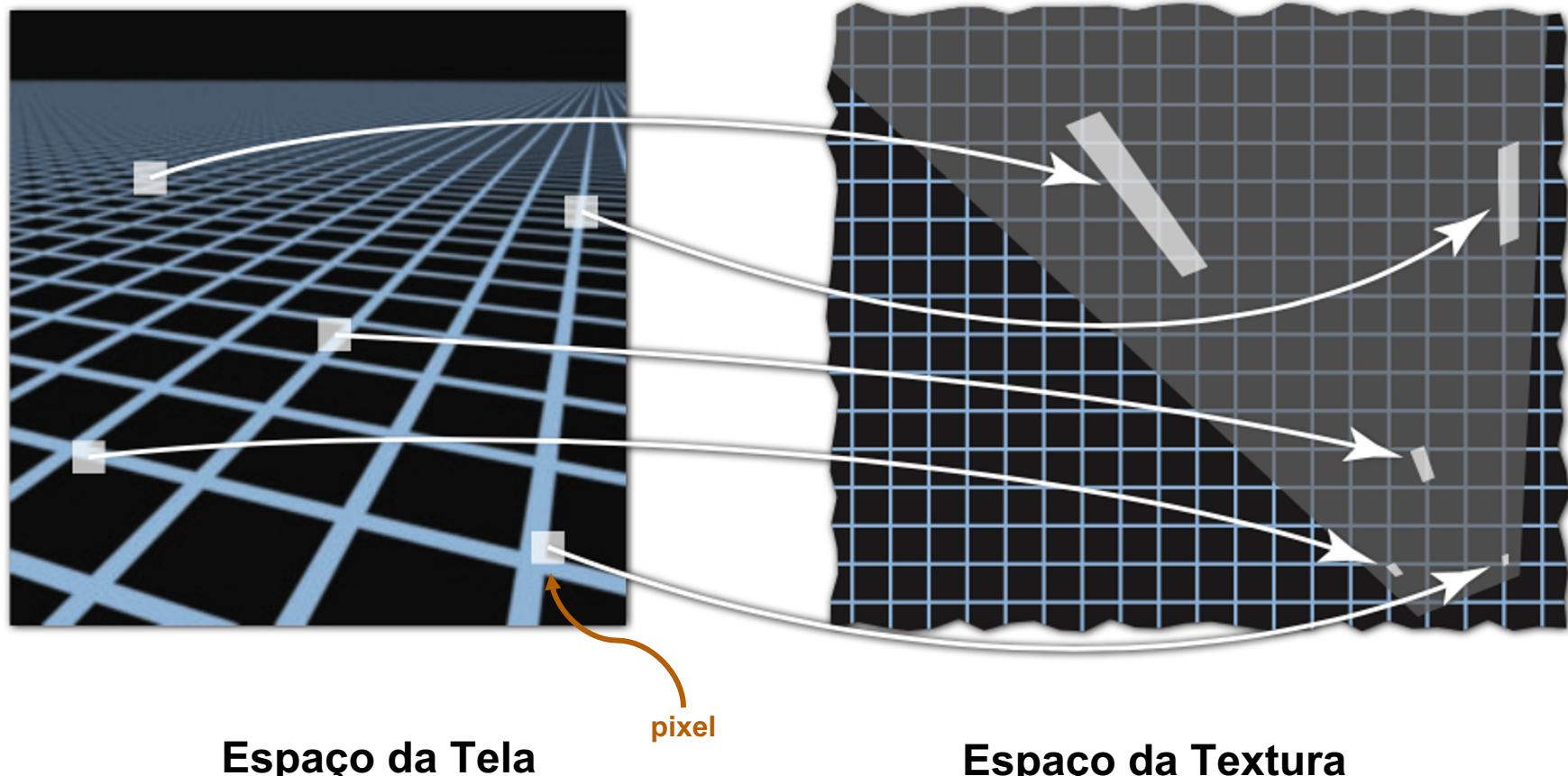
Quando ampliado (magnification)

- O mesmo texel vai influenciar vários pixels na tela

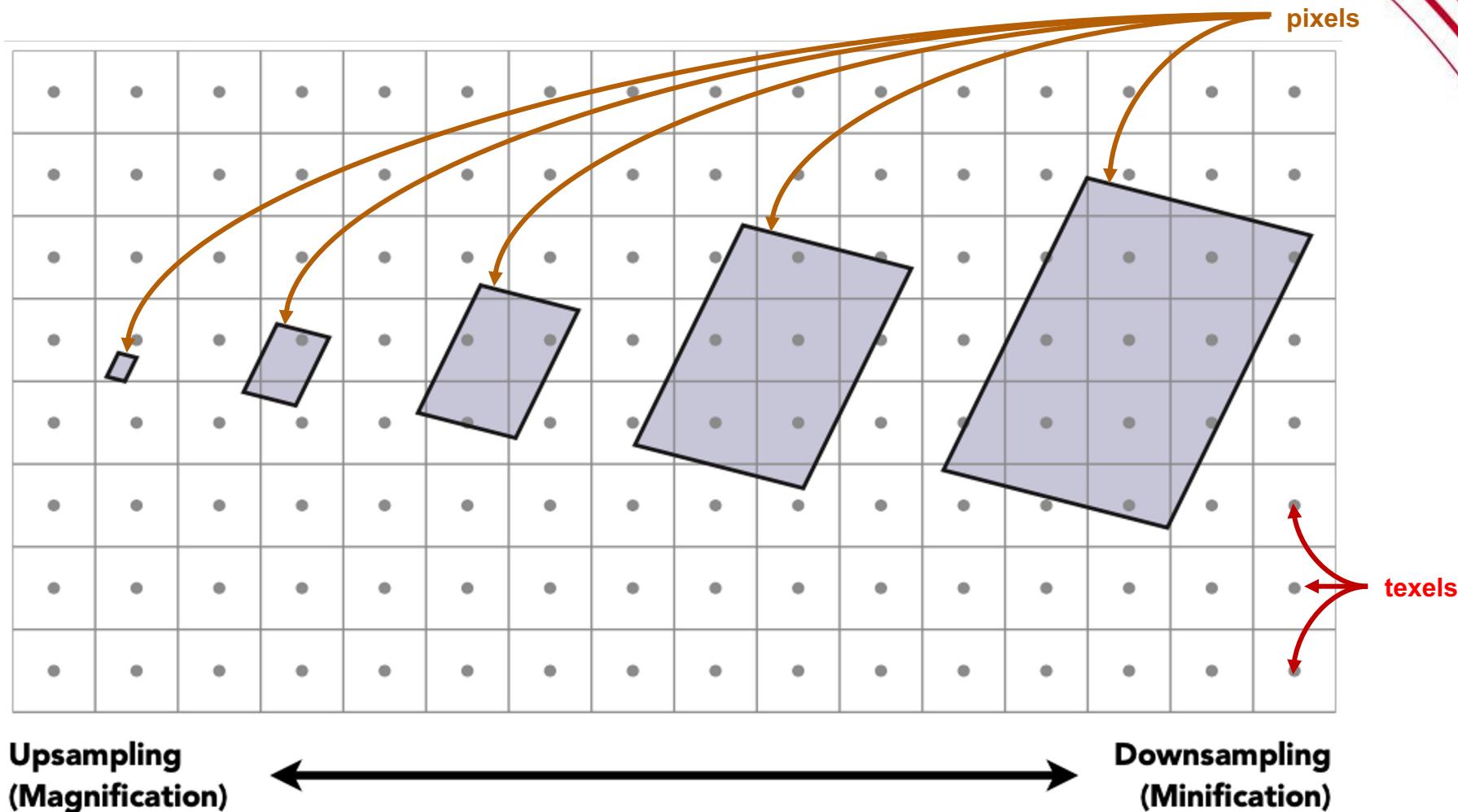
Quando menor (minificação)

- Possibilidades de amostras de texels para cada pixel

# Região (footprint) do Pixel na Textura



# Região (footprint) do Pixel na Textura



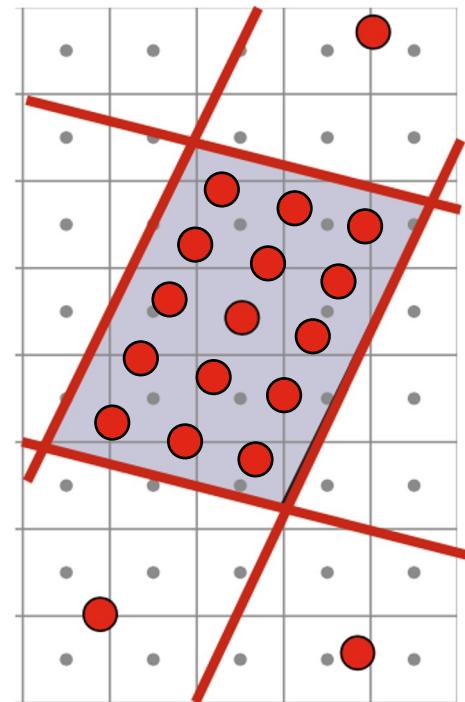
# Casos podem ficar complicados

Desafio:

- Muitos texels podem contribuir para o pixel
- Apenas a amostragem de um deles pode resultar em aliasing
- A forma da "pegada" do pixel pode ser complexa

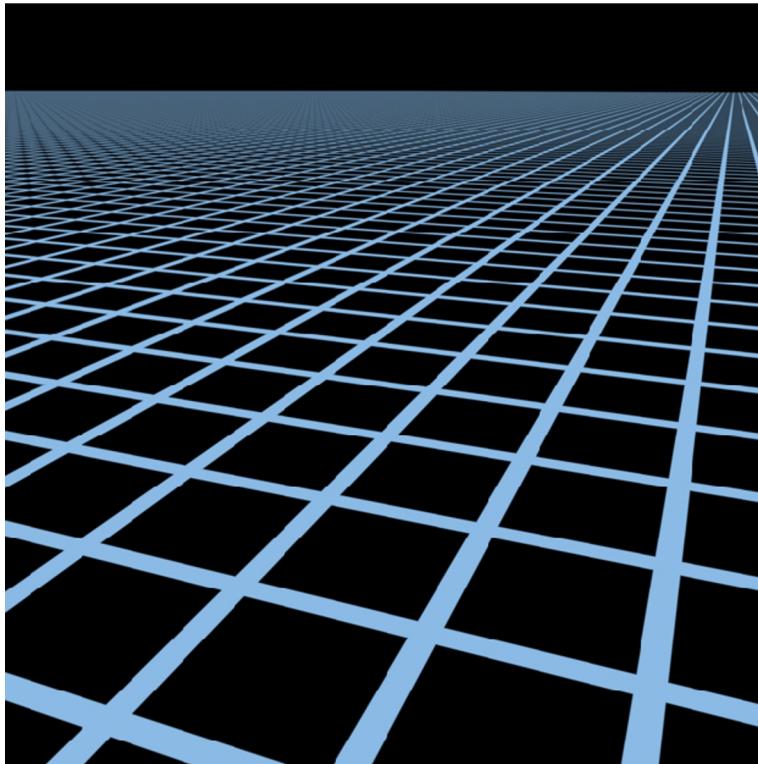
Uma solução:

- Fazendo a média de muitas amostras de textura por pixel.



Linhas vermelhas = limites dos pixels da tela  
Pontos vermelhos = ponto de amostragem do pixel  
Pontos cinza = centro o textel

# Superamostragem removerá os artefatos?



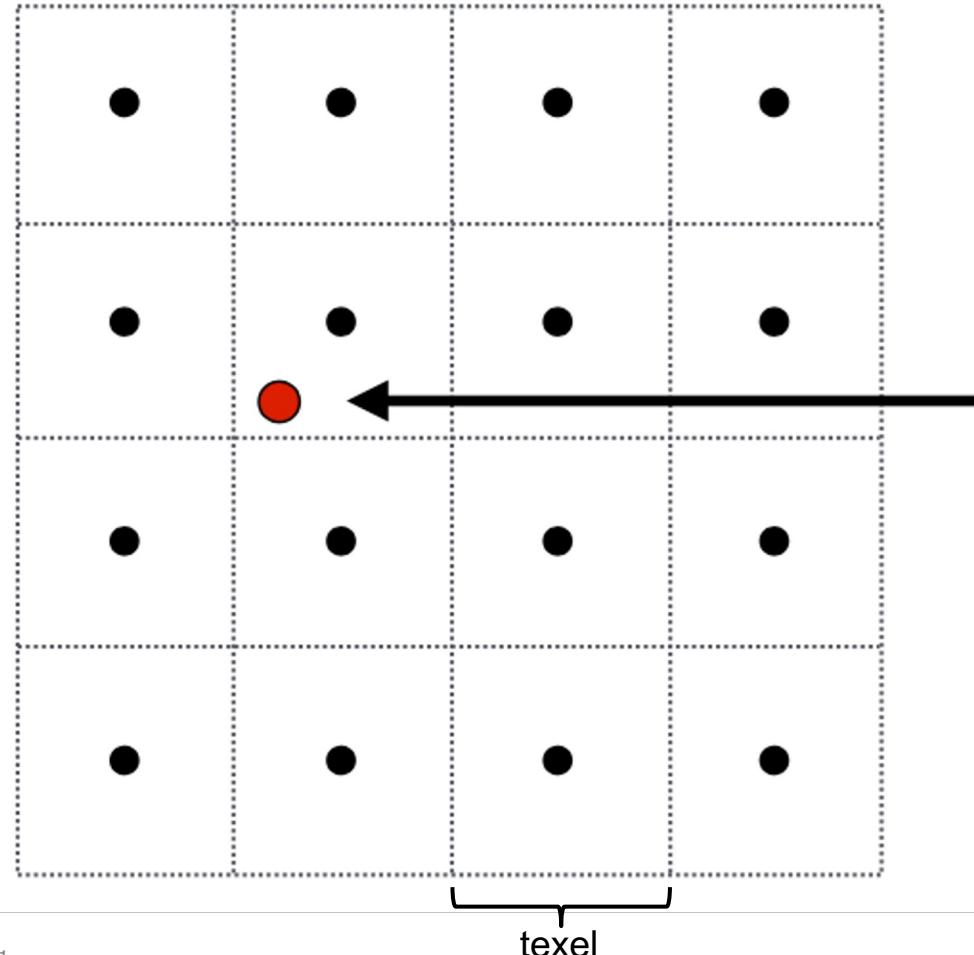
Referência em Alta-resolução



Superamostragem 512x

Alta qualidade, mas custoso

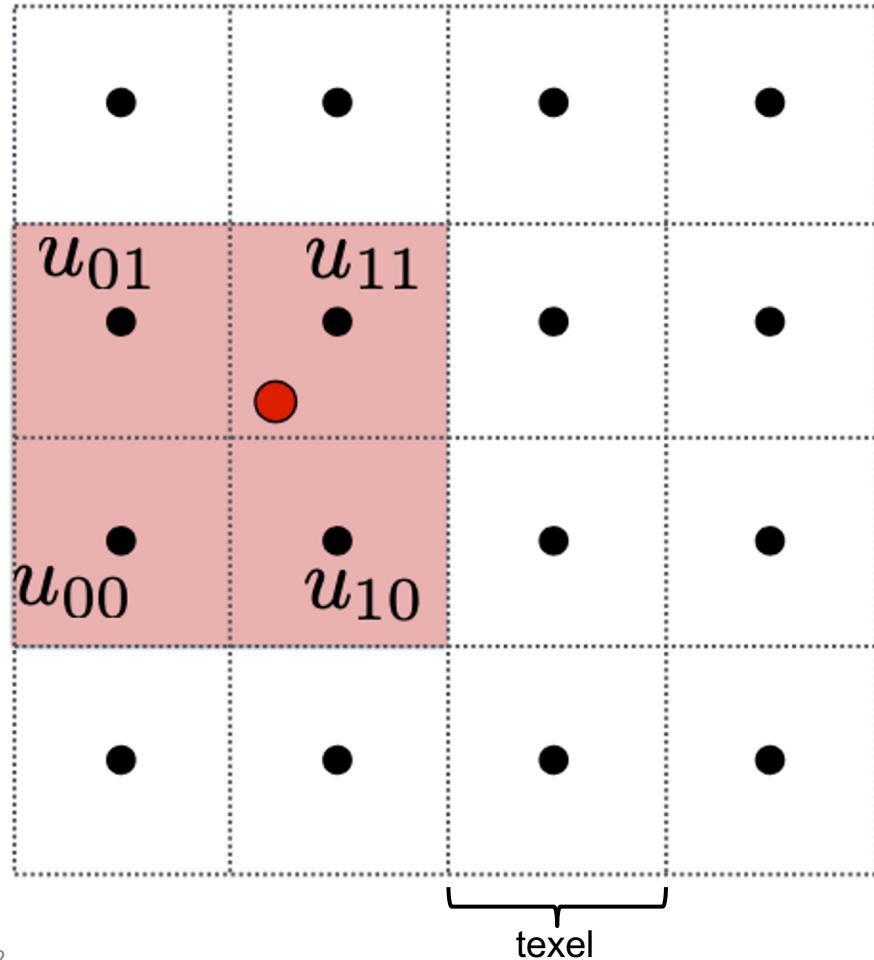
# Filtro Bilinear



Se deseja obter uma amostra do valor de textura  $f(x, y)$  no ponto vermelho

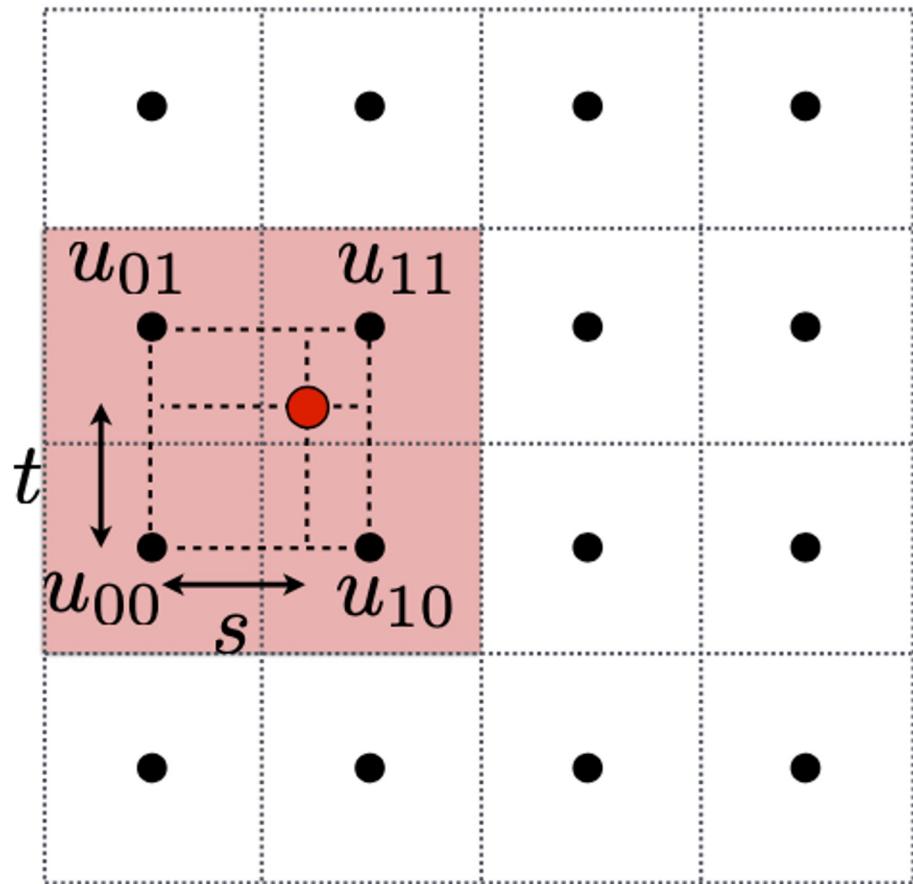
Pontos pretos indicam locais de amostra de textura

# Filtro Bilinear



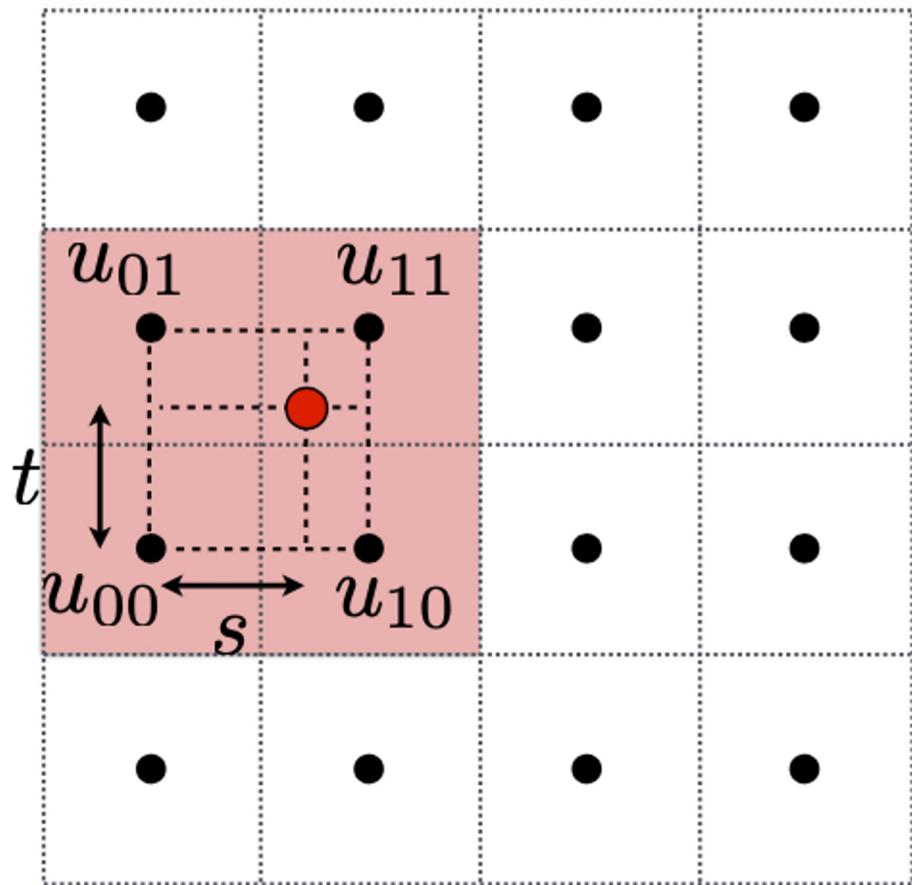
Pegue os 4 locais de amostra mais próximos, com valores de textura conforme rotulados.

# Filtro Bilinear



pegue os deslocamentos fracionários (s,t)

# Filtro Bilinear



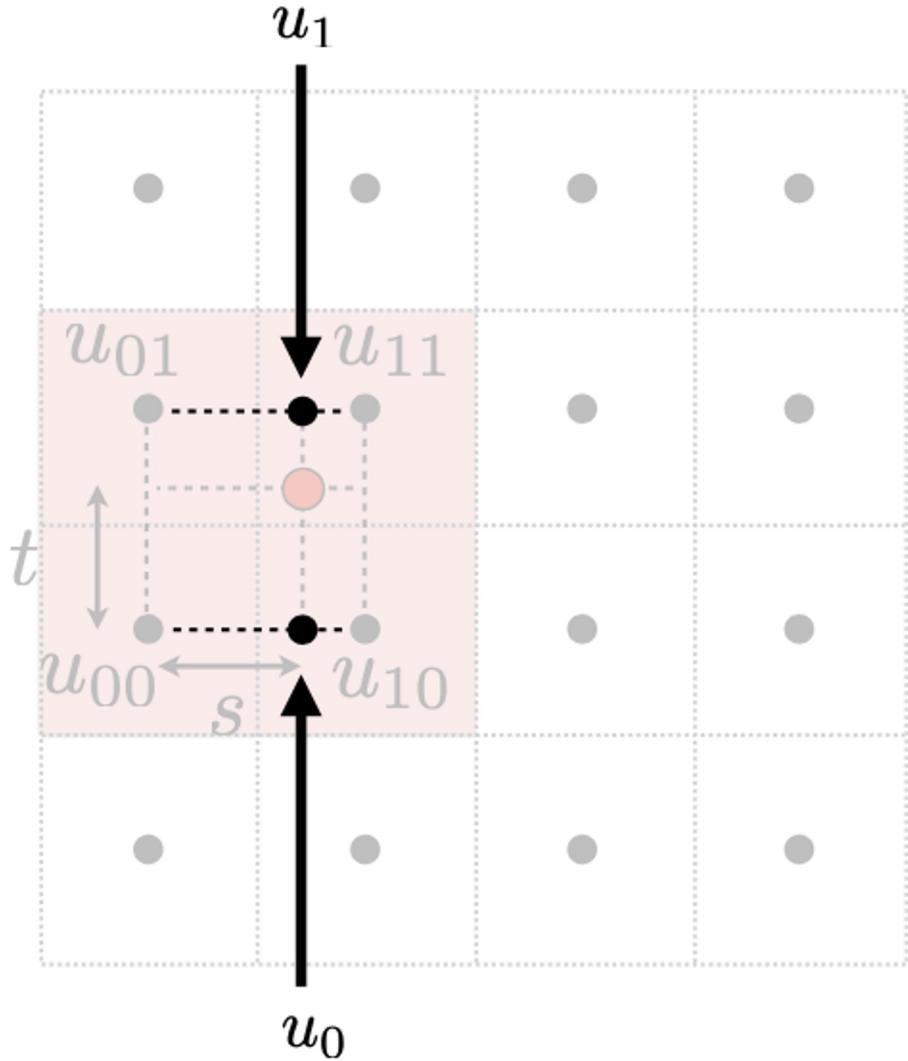
Exemplo: Interpolação Linear (1D)

$$\text{lerp}(x, v_0, v_1) = v_0 + x(v_1 - v_0)$$



$x$  varia de 0 a 1

# Filtro Bilinear



Interpolação Linear (1D)

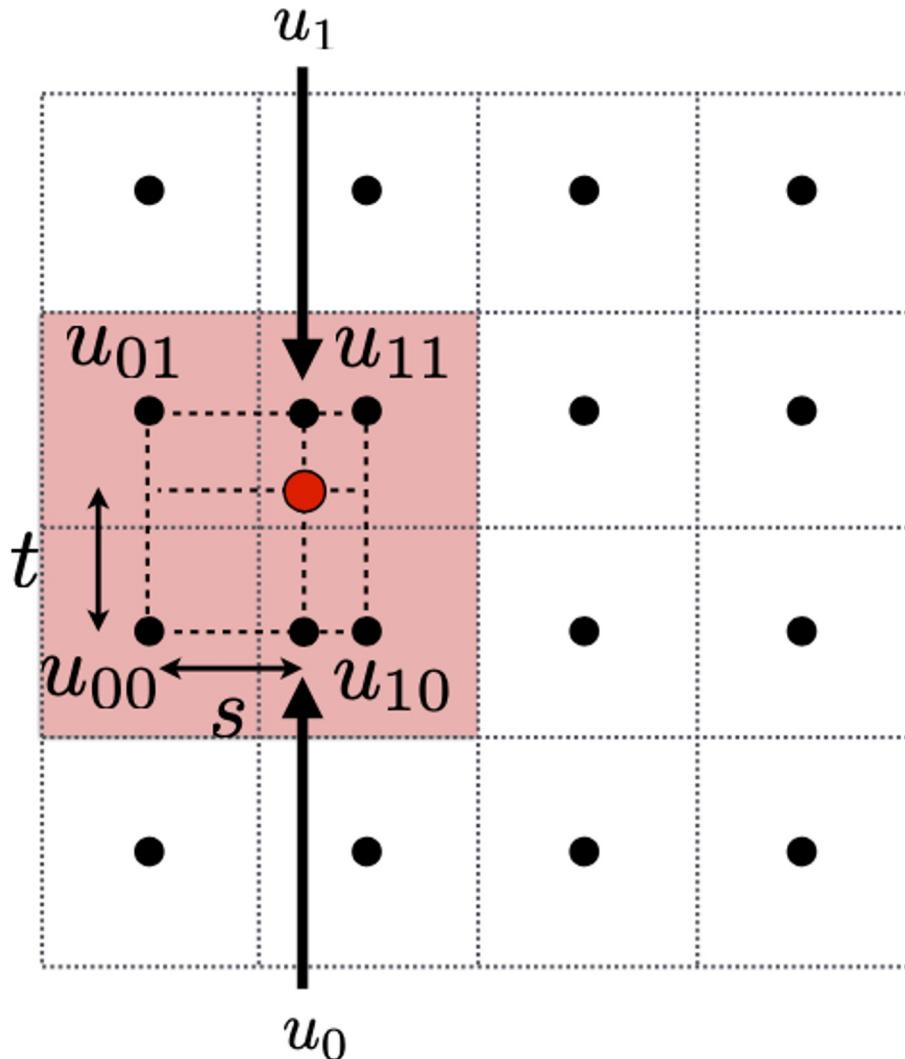
$$\text{lerp}(x, v_0, v_1) = v_0 + x(v_1 - v_0)$$

Interpolações Intermediárias (horizontal)

$$u_0 = \text{lerp}(s, u_{00}, u_{10})$$

$$u_1 = \text{lerp}(s, u_{01}, u_{11})$$

# Filtro Bilinear



Interpolação Linear (1D)

$$\text{lerp}(x, v_0, v_1) = v_0 + x(v_1 - v_0)$$

Interpolações Intermediárias (horizontal)

$$u_0 = \text{lerp}(s, u_{00}, u_{10})$$

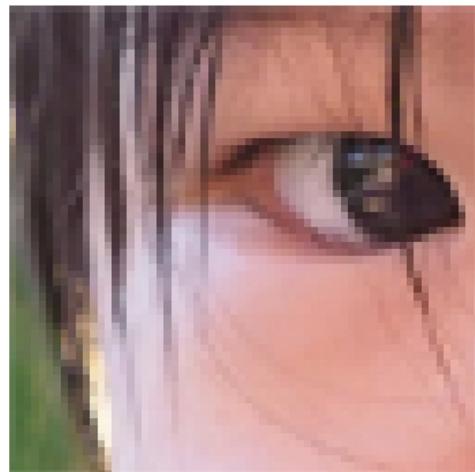
$$u_1 = \text{lerp}(s, u_{01}, u_{11})$$

Interpolação Final (vertical)

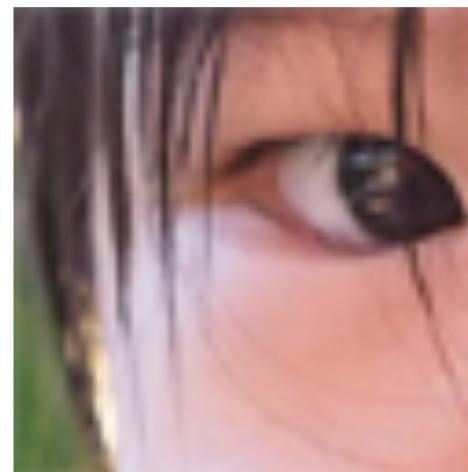
$$f(x, y) = \text{lerp}(t, u_0, u_1)$$

# Ampliando Texturas – Caso mais simples

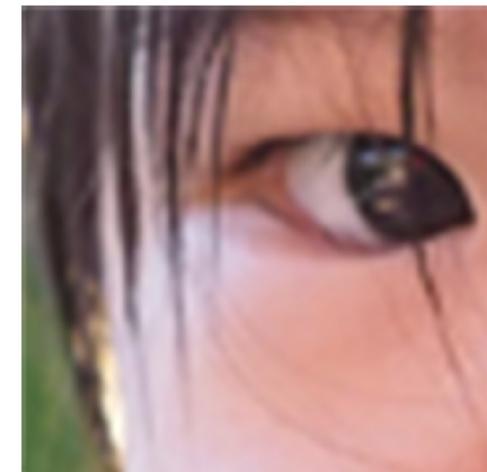
Usando interpolação das imagens



**Nearest**



**Bilinear**  
4 texels ( $2 \times 2$ )



**Bicubic**  
16 texels ( $4 \times 4$ )

# Redução de texturas em vários níveis

## Mais Desafiador

- Muitos texels podem contribuir para a área de um pixels
- A forma da área do pixel pode ser complexa.

## Ideia:

- Filtro passa-baixa e sub-amostrar a textura
- Usar uma resolução que corresponda a resolução da tela

# Mipmap (L. Williams 83)



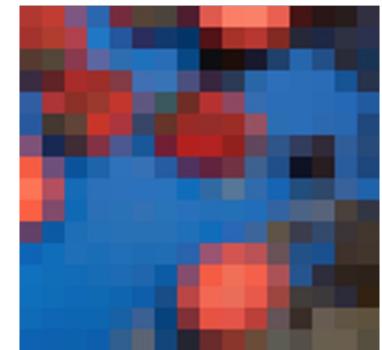
Nível 0 = 128x128



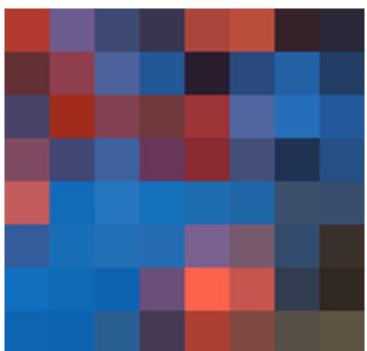
Nível 1 = 64x64



Nível 2 = 32x32



Nível 3 = 16x16



Nível 4 = 8x8



Nível 5 = 4x4



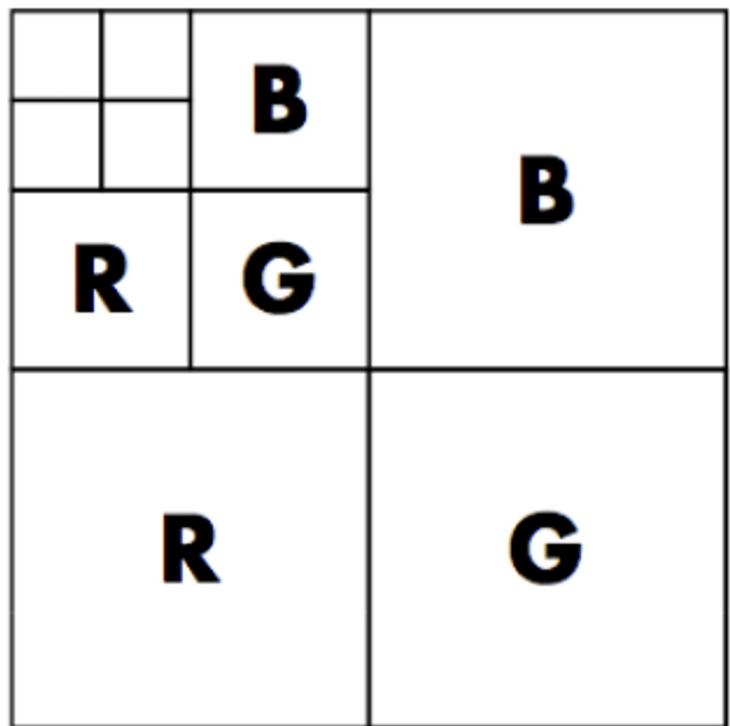
Nível 6 = 2x2



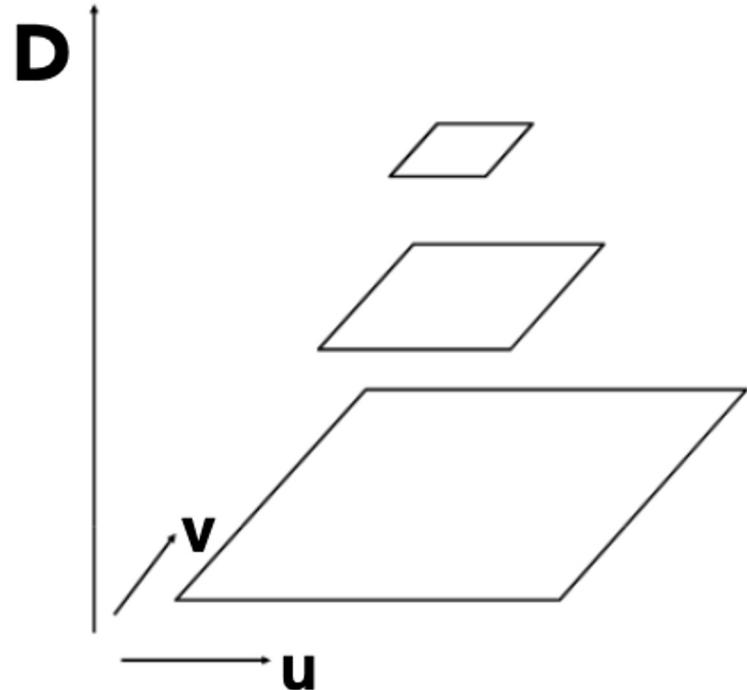
Nível 7 = 1x1

“Mip” vem do latim “multum in parvo”, que significa uma multidão em um pequeno espaço

# Mipmap (L. Williams 83)



Layout original do L. Williams  
para os Mipmaps



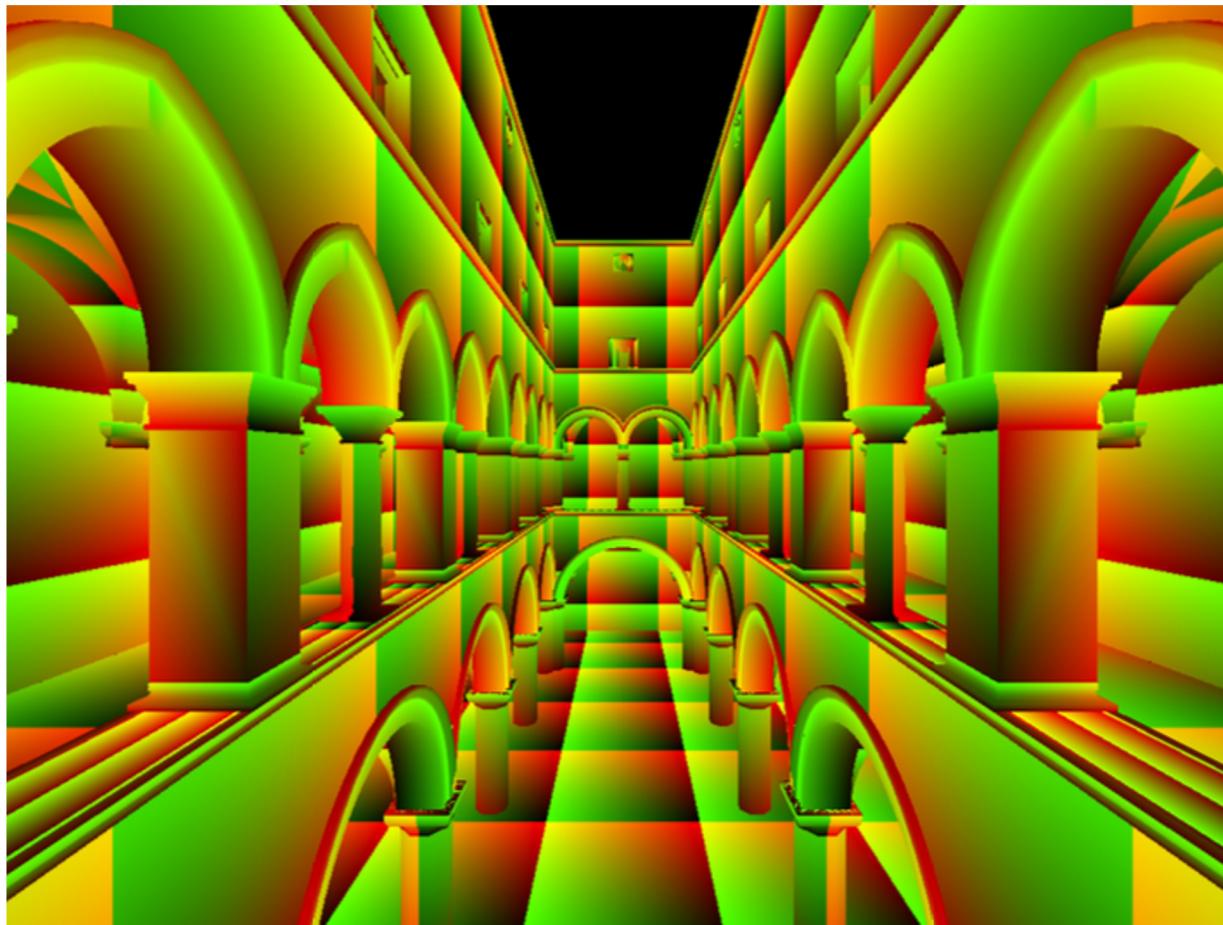
Hierarquia dos Mipmaps  
D = nível

# Modelo do Palácio de Sponza



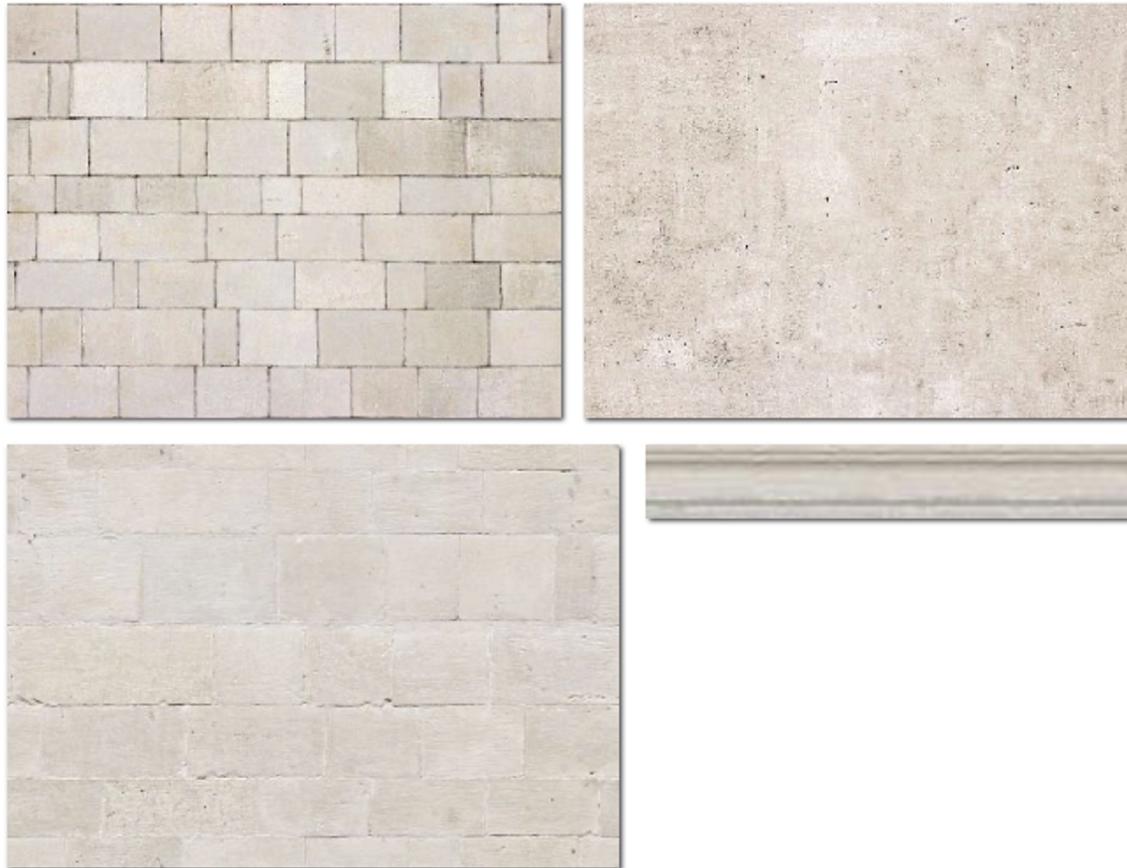
Texturas Aplicadas nas Superfícies

# Modelo do Palácio de Sponza



Visualização das Coordenadas de Texturas

# Modelo do Palácio de Sponza



Exemplo de Texturas Usadas

# Mipmap nível 0 – Resolução Máxima das Texturas



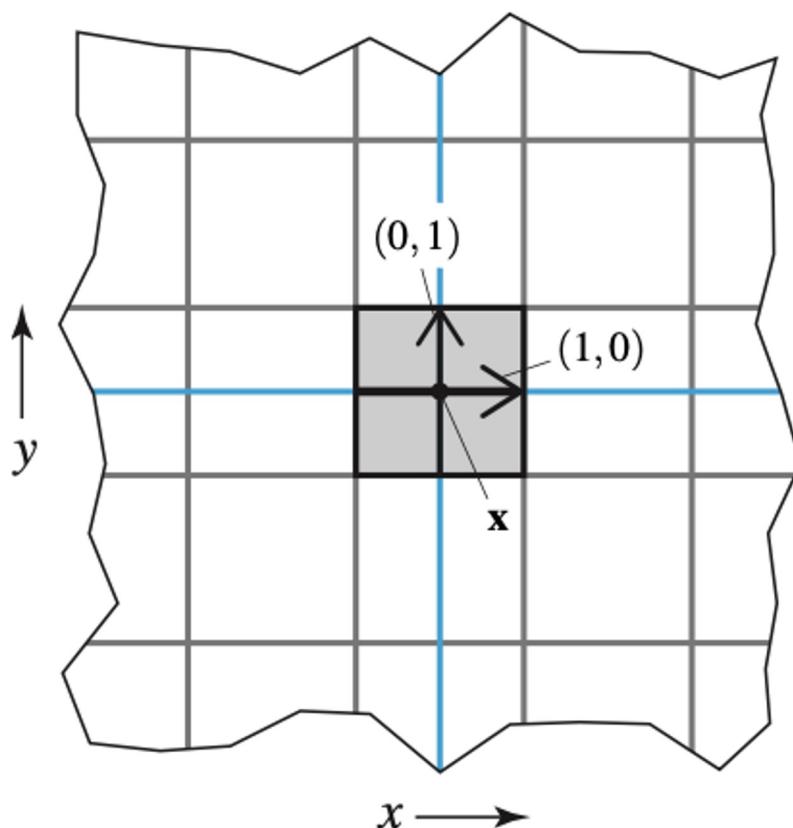
# Mipmap nível 2 – Sub-amostragem 4x4



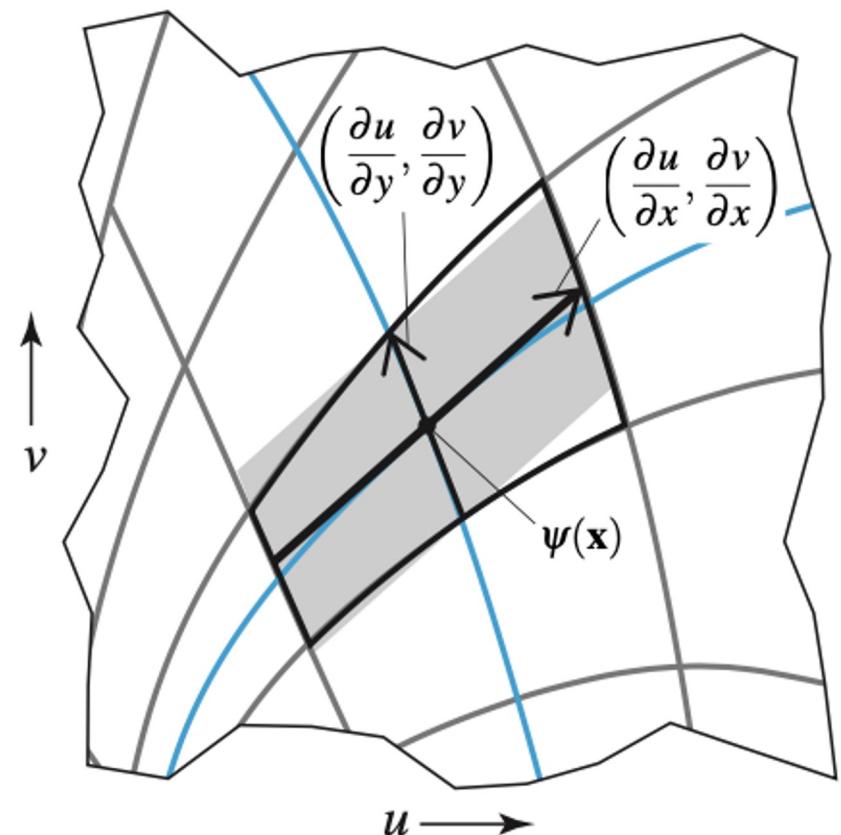
# Mipmap nível 4 – Sub-amostragem 16x16



# Estimativa da Área do Pixel com Jacobiano



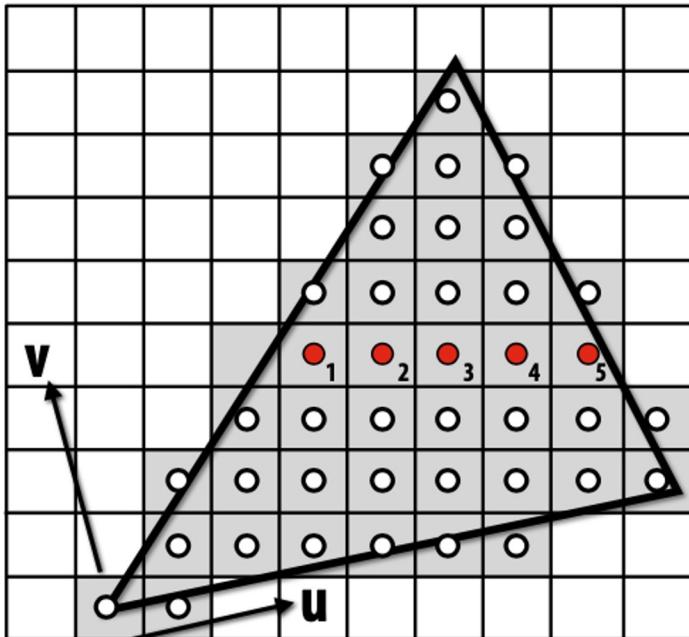
Espaço da Tela



Espaço da Textura

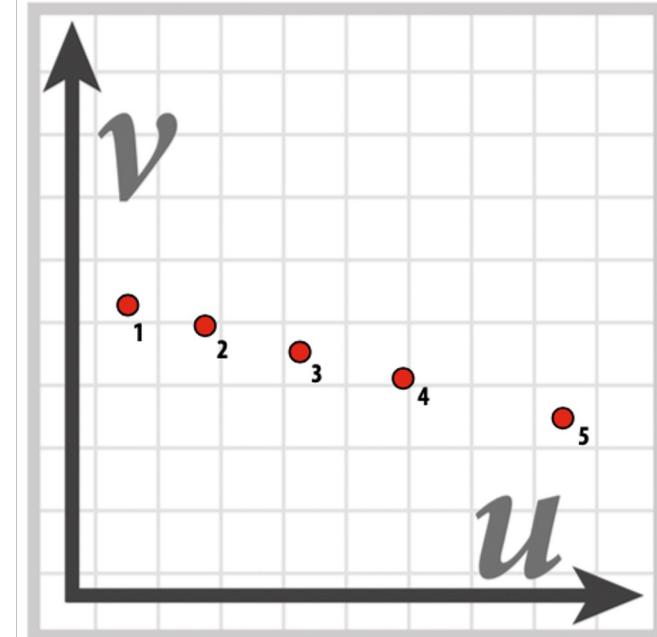
# Algumas dificuldades

Amostras no espaço da tela ( $x, y$ )



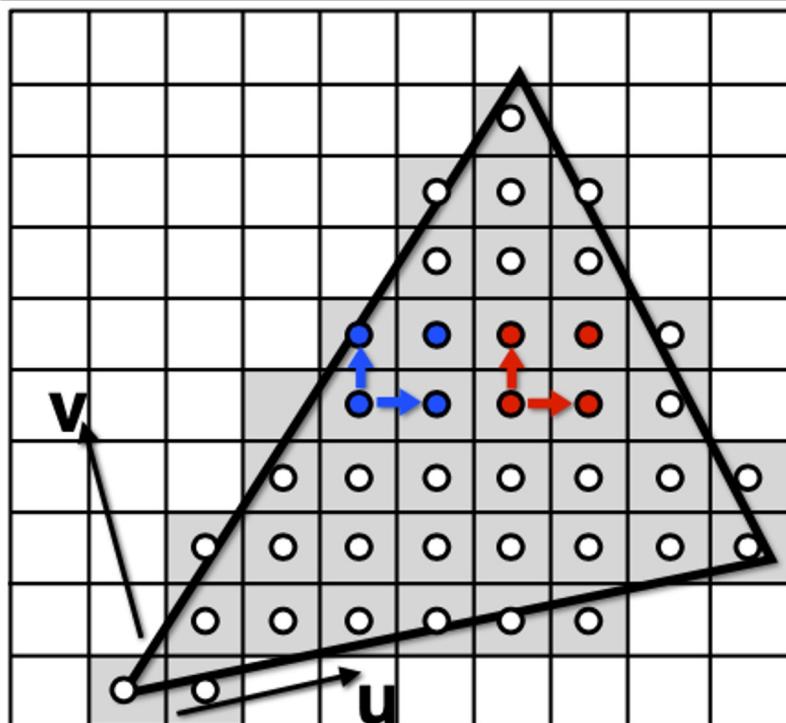
As posições de amostra são distribuídas uniformemente no espaço da tela (rasterizador mostra a aparência do triângulo nesses locais)

Amostras no espaço da textura ( $u, v$ )

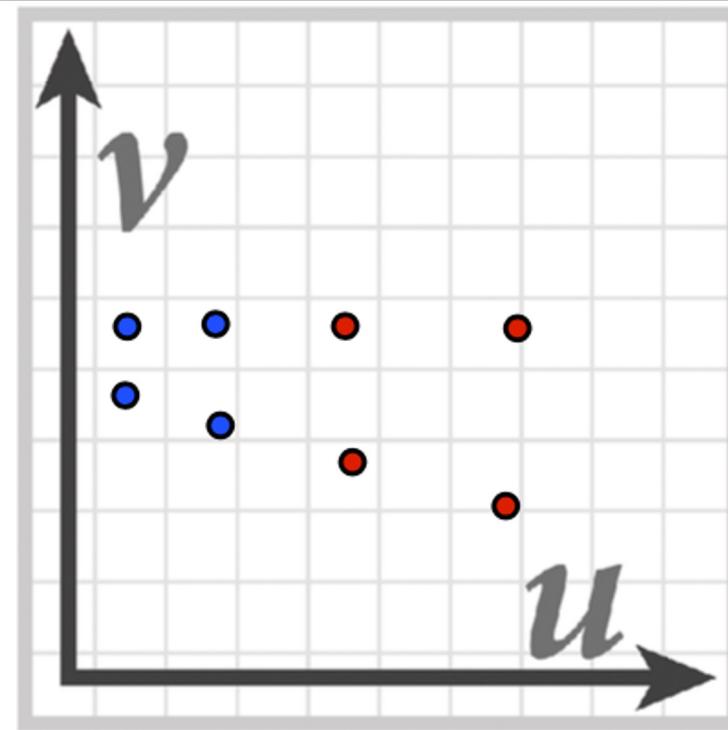


Posições de amostra de textura no espaço de textura (a função de textura é amostrada nesses locais)

# Calculando os níveis de Mipmaps



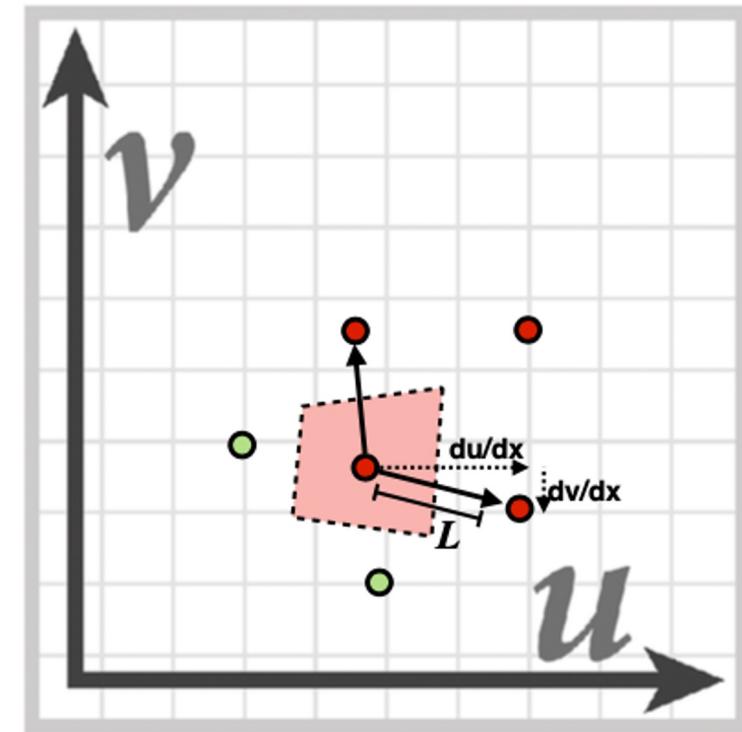
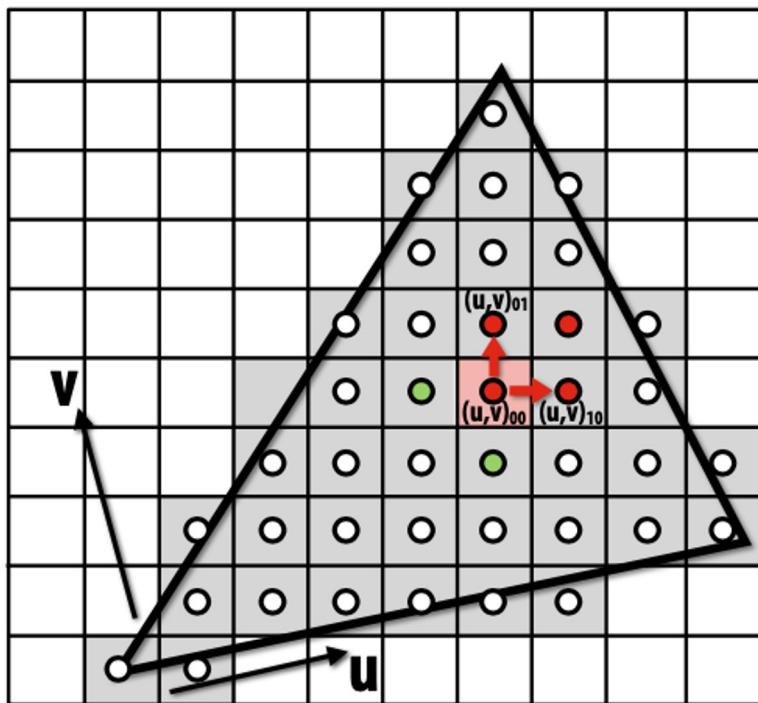
Espaço da Tela ( $x,y$ )



Espaço da Textura ( $u,v$ )

Vamos estimar a área da textura usando as coordenadas das amostras vizinhas.

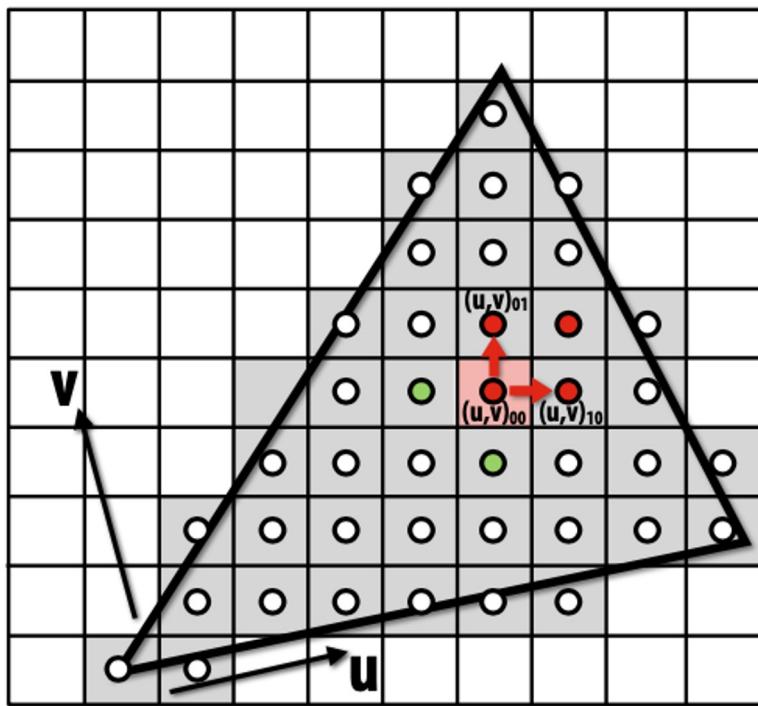
# Calculando os níveis de Mipmaps



$$D = \log_2 L$$
$$L = \max \left( \sqrt{\left( \frac{du}{dx} \right)^2 + \left( \frac{dv}{dx} \right)^2}, \sqrt{\left( \frac{du}{dy} \right)^2 + \left( \frac{dv}{dy} \right)^2} \right)$$

Cuidado:  $du$  e  $dv$  são definidos pela quantidade de texels.

# Calculando os níveis de Mipmaps

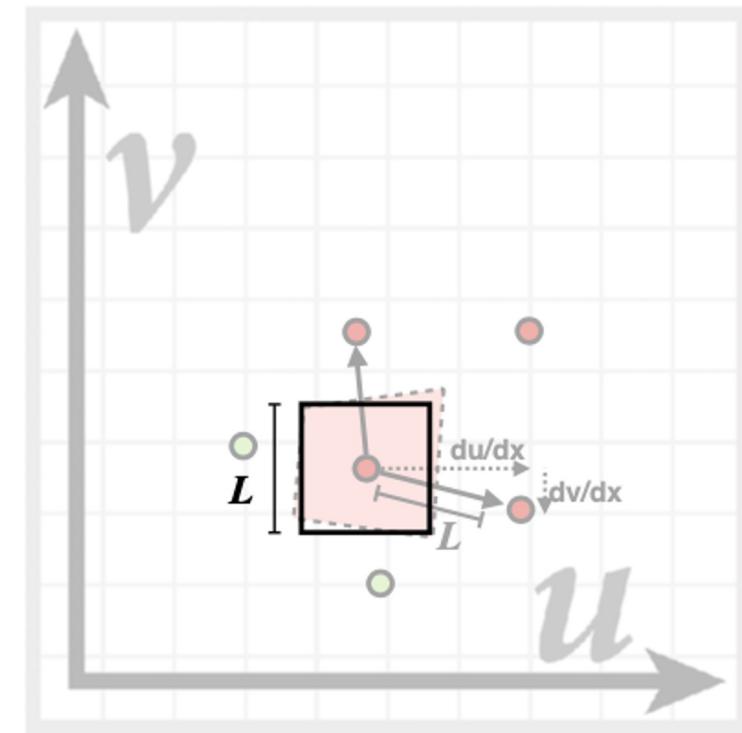


$$\frac{\partial u}{\partial x} = \frac{u_{10} - u_{00}}{1}$$

$$\frac{\partial v}{\partial x} = \frac{v_{10} - v_{00}}{1}$$

$$\frac{\partial u}{\partial y} = \frac{u_{01} - u_{00}}{1}$$

$$\frac{\partial v}{\partial y} = \frac{v_{01} - v_{00}}{1}$$

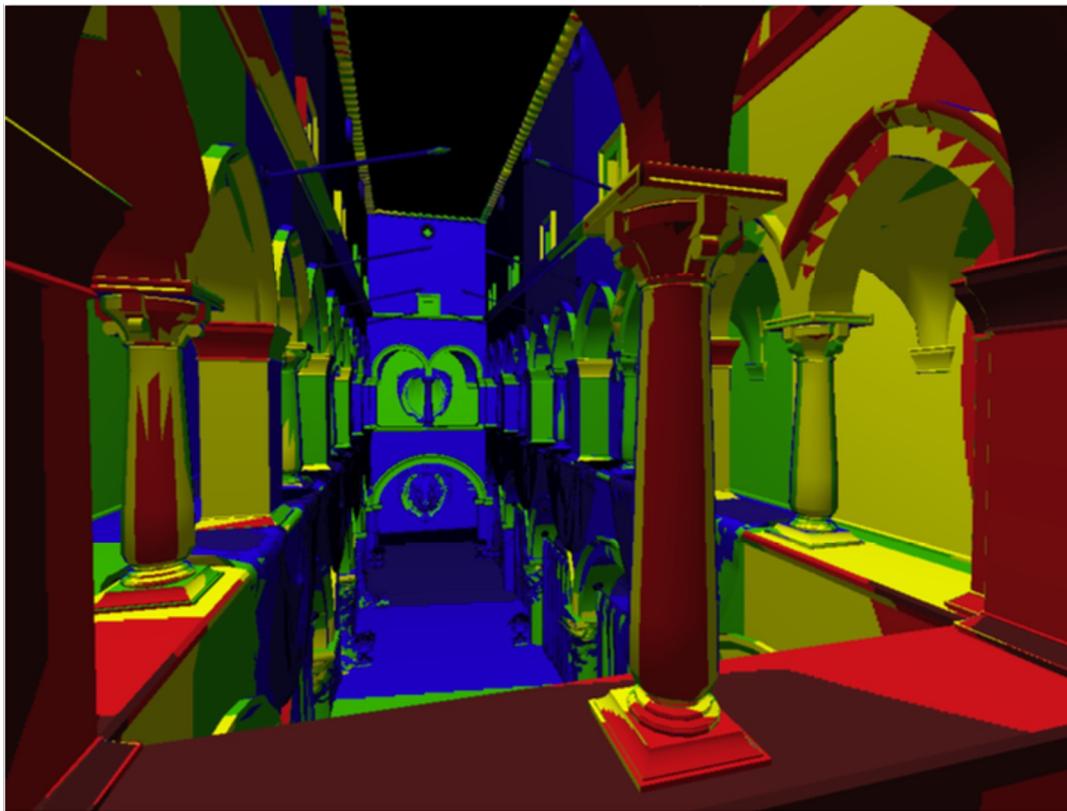


$$D = \log_2 L$$

$$L = \max \left( \sqrt{\left( \frac{du}{dx} \right)^2 + \left( \frac{dv}{dx} \right)^2}, \sqrt{\left( \frac{du}{dy} \right)^2 + \left( \frac{dv}{dy} \right)^2} \right)$$

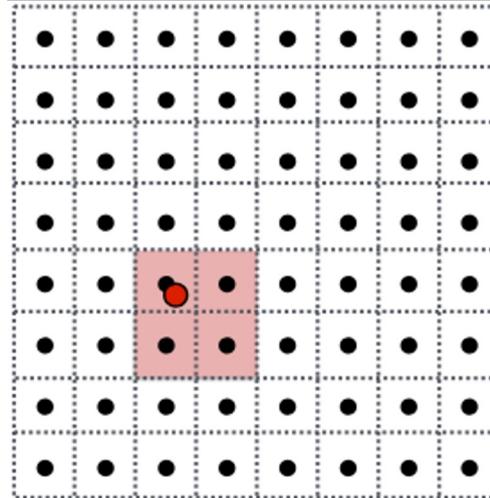
Cuidado: du e dv são definidos pela quantidade de texels.

# Visualizando com os níveis de Mipmap

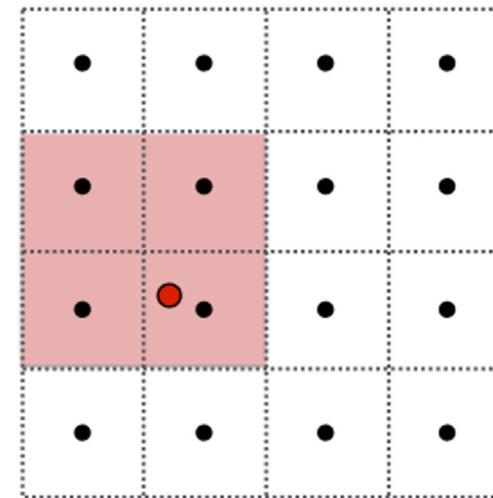


D sendo cortado no nível mais próximo encontrado

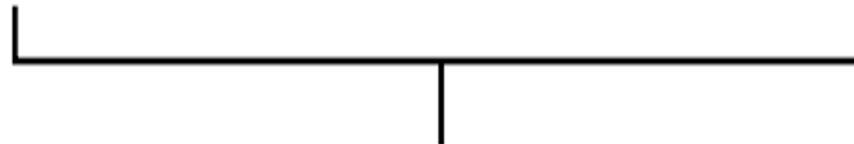
# Filtragem Trilinear



Mipmap nível D

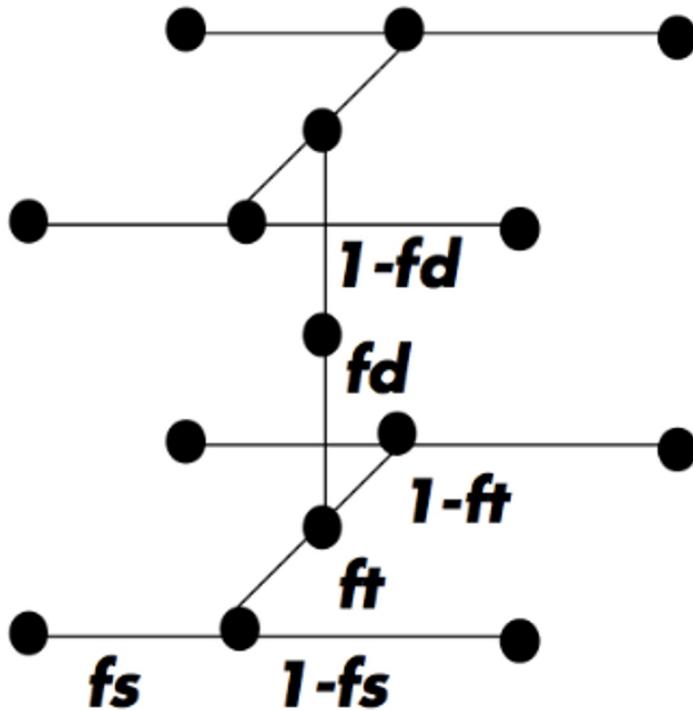


Mipmap nível D+1

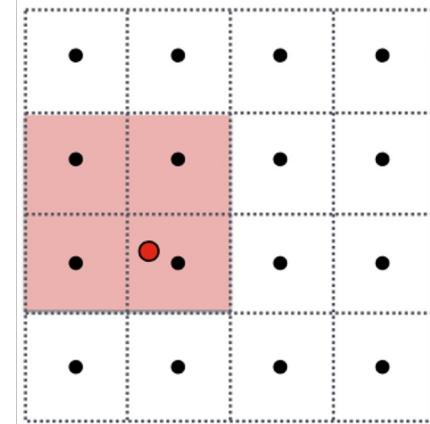


Interpolação Linear baseada em um valor de D contínuo.

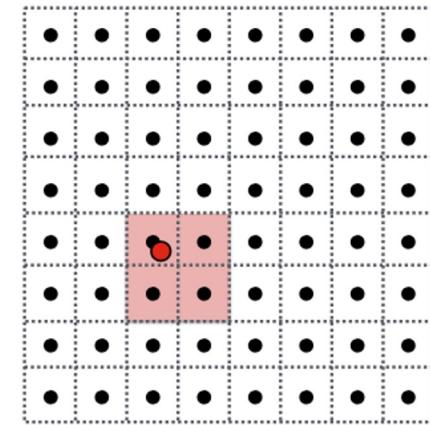
# Filtro Trilinear



$$lerp(t, v_1, v_2) = v_1 + t(v_2 - v_1)$$

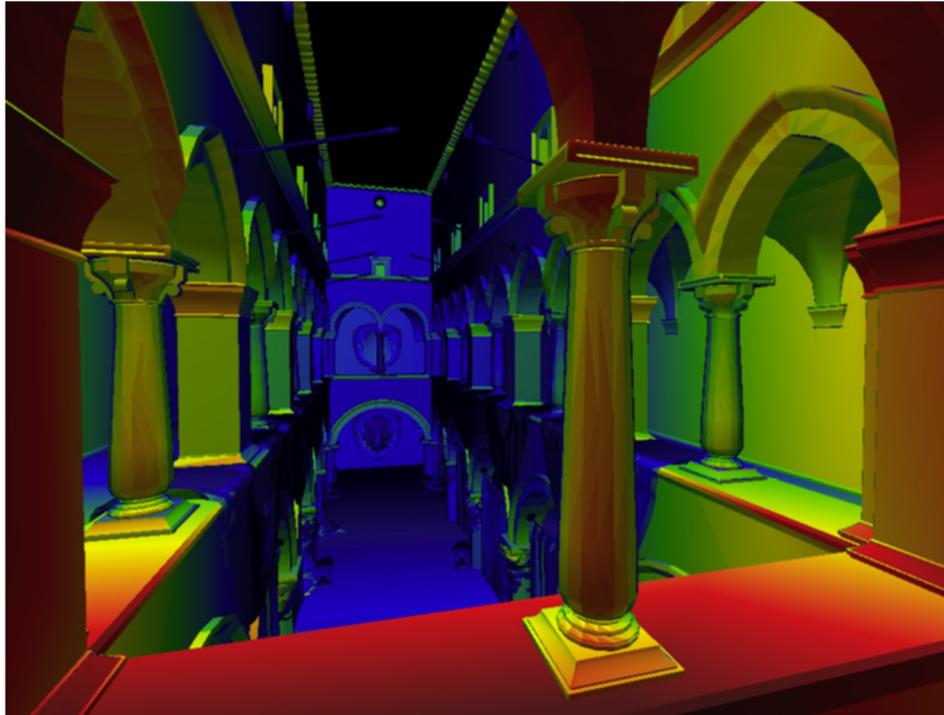


mip-map texels: nível  $d+1$



mip-map texels: nível  $d$

# Visualizando níveis de Mipmap contínuos



Filtragem Trilinear: visualizando D de forma contínua

# Custo da Filtragem Bilinear x Trilinear

## Amostragem Bilinear

- 4 leituras de texels
- 3 interpolações lineares ( 3 multiplicações + 6 somas)

## Amostragem Trilinear

- 8 leituras de texels
- 7 interpolações lineares ( 7 multiplicações + 14 somas)

# Color e TextureCoordinate

Os nós **Color** e **TextureCoordinate** define um conjunto de pontos que podem ser usados para armazenar cores e coordenadas de textura respectivamente.

```
Color : X3DColorNode {  
    MFColor [in,out] color  [NULL] [0,1]  
    SFNode [in,out] metadata NULL  [X3DMetadataObject]  
}
```

```
TextureCoordinate : X3DTextureCoordinateNode {  
    SFNode [in,out] metadata NULL [X3DMetadataObject]  
    MFVec2f [in,out] point  []  (-∞,∞)  
}
```

# ImageTexture

O nó **ImageTexture** serve para ler uma imagem, o atributo **url** é usado para ter uma lista de pontos de acesso da imagem.

```
ImageTexture : X3DTexture2DNode, X3DUrlObject {  
    SFNode [in,out] metadata      NULL [X3DMetadataObject]  
    MFString [in,out] url        [] [URI]  
    SFBool [] repeatS           TRUE  
    SFBool [] repeatT           TRUE  
    SFNode [] textureProperties NULL [TextureProperties]  
}
```

# Computação Gráfica

Luciano Soares  
[<lpsoares@insper.edu.br>](mailto:lpsoares@insper.edu.br)

Fabio Orfali  
[<fabioo1@insper.edu.br>](mailto:fabioo1@insper.edu.br)