UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Texturas

Disciplina: Computação Gráfica (BCC35F)
Curso: Ciência da Computação

Prof. Walter T. Nakamura
waltertakashi@utfpr.edu.br

Campo Mourão - PR

Baseado no livro *Computer Graphics Through OpenGL: From Theory to Experiments*, de Sumanta Guha, CRC Press, 3ª edição.

Sumário

Sumário

Introdução

- Uma **textura** é uma imagem que é aplicada em um polígono ou uma malha
  - **Texels:** pixels na textura que armazena o valor das cores, similar aos pixels no frame buffer
- A textura pode ser uma **imagem externa** importada para o OpenGL ou **gerada internamente** pelo programa
  - Uma vez carregada, não há diferença entre as duas formas

Introdução

- **Texturizar** uma superfície é basicamente pintar uma imagem sobre ela
- Benefícios para a CG:
  - **Autenticidade:** para representar os objetos de uma forma realista é necessário que as superfícies do objeto sejam pintadas
  - **Ilusão de detalhe geométrico:** ao invés de tentar recriar um objeto com primitivas gráficas, fazer uma pintura dele na cena pode obter resultados mais realistas em uma fração de custo de inúmeros triângulos

Introdução

Introdução

Sumário

Comandos OpenGL

- Define a quantidade de texturas e um array com as texturas

```
void glGenTextures(GLsizei n, GLuint * textures)
```

Exemplo: `glGenTextures(2, texture);`

- Cria um novo objeto de textura ou ativa um objeto existente:

```
void glBindTexture(GLenum target, GLuint texture);
```

Exemplo: `glBindTexture(GL_TEXTURE_2D, texture[0]);`

  - **Parâmetros**
    - **target:** `GL_TEXTURE_1D, GL_TEXTURE_2D, GL_TEXTURE_3D, GL_TEXTURE_1D_ARRAY…`
    - **texture:** ID da textura definida no `glGenTextures`

Comandos OpenGL

- Especifica a imagem da textura para o objeto de textura atual:

```
void glTexImage2D(GLenum target, GLint level, Glint internalformat,
GLsizei width, GLsizei height, Glint border, GLenum format, GLenum
type, const void * data);
```

Exemplo:
```
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, image[0]->width,
image[0]->height, 0, GL_RGBA, GL_UNSIGNED_BYTE, image[0]->data);
```

- Especificando os parâmetros do ambiente de textura:

```
void glTexEnvf( GLenum target, GLenum pname, GLfloat param);
```

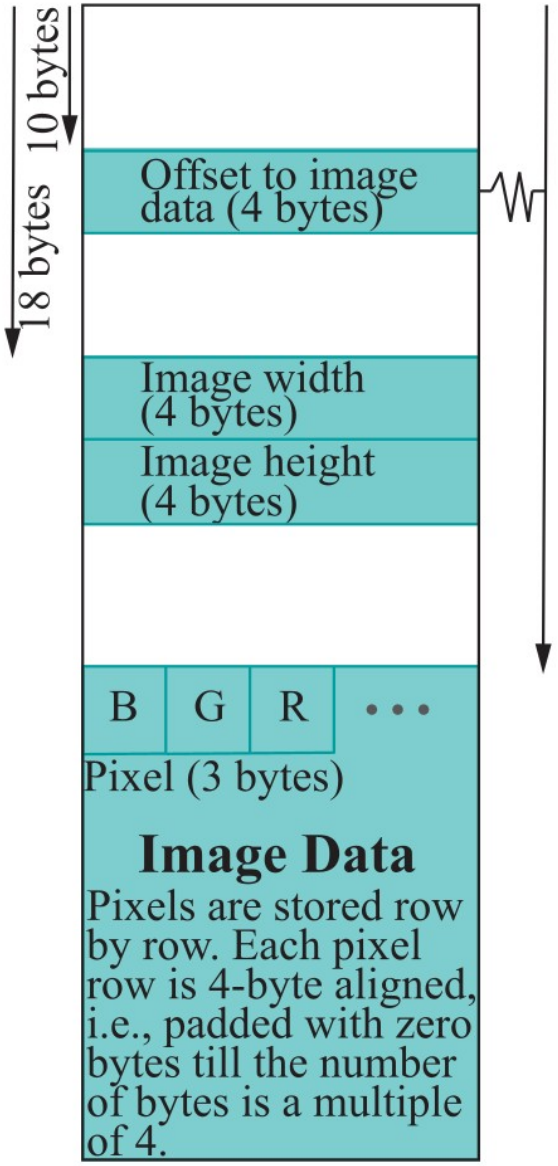
Exemplo:
```
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
glEnable(GL_TEXTURE_2D);
```

Exercício 1

1) Abra o arquivo `texturedSquare.cpp` e execute para ver o seu funcionamento
2) Substitua a imagem com outros baixados da Web
3) Crie a função `createStripedBoard()` que gera uma imagem de um quadrado listrado em uma array RGBA de 64 x 64 x 4
   - O programa deve poder alternar para esta textura além das 2 já existentes

Função getBMP()

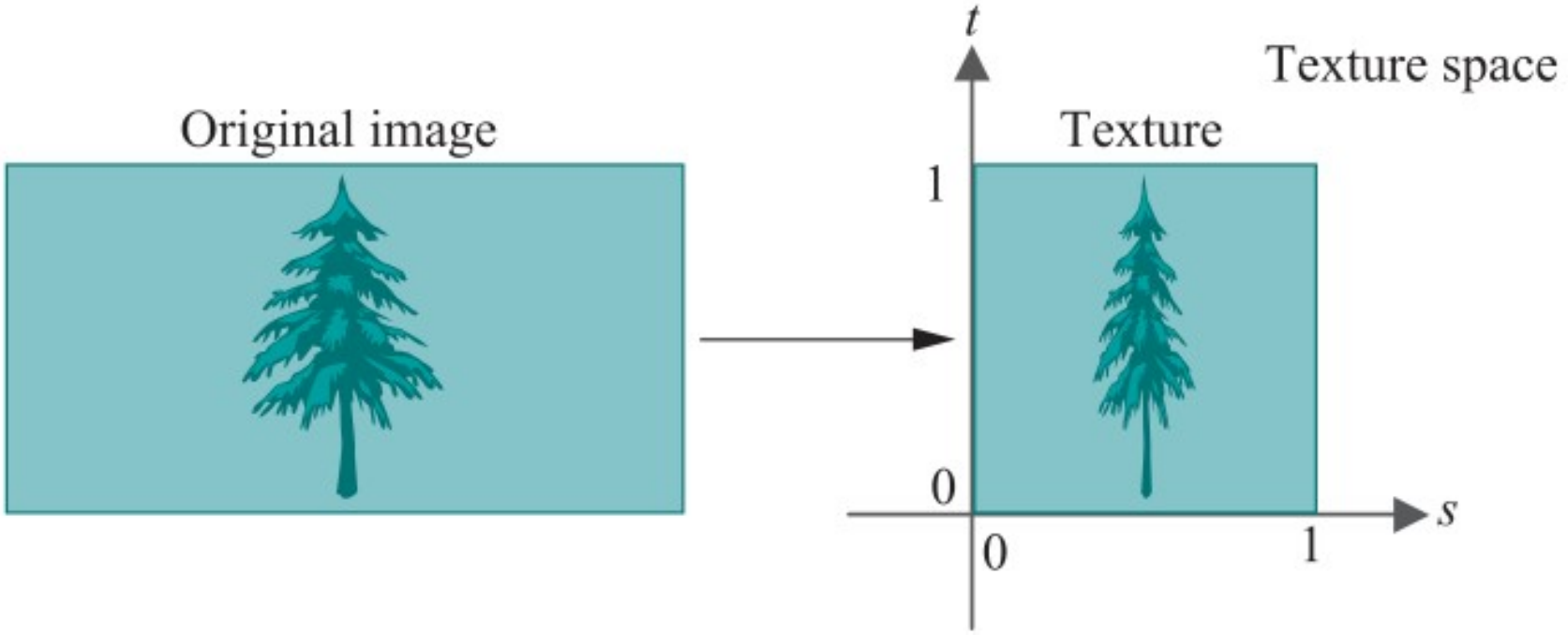
- Uma imagem BMP possui a seguinte estrutura:
  - Após os **10 primeiros bytes** há um campo de **4 bytes** de offset
  - Após **18 bytes** há a largura e a altura da imagem em 2 campos de **4 bytes**
  - Em seguida, há os **dados da imagem**
    - Os dados da imagem são armazenados no **formato BGR**
      - 1 byte por cor → **3 bytes** por pixel
    - Os dados são armazenados em linhas de pixels, cada linha alinhado em **4 bytes**
      - Se a linha **não** for múltiplo de 4, ela é preenchida com zero-bytes até ser múltiplo de 4



Sumário

Espaço da Textura, Coordenadas e Mapa

- Uma vez carregada, a textura ocupa um **quadrado** com bordas nos vértices (0,0), (1,0), (1,1) e (0,1) de um **plano virtual** chamado espaço da textura
- Os eixos são denotados como `s` e `t`



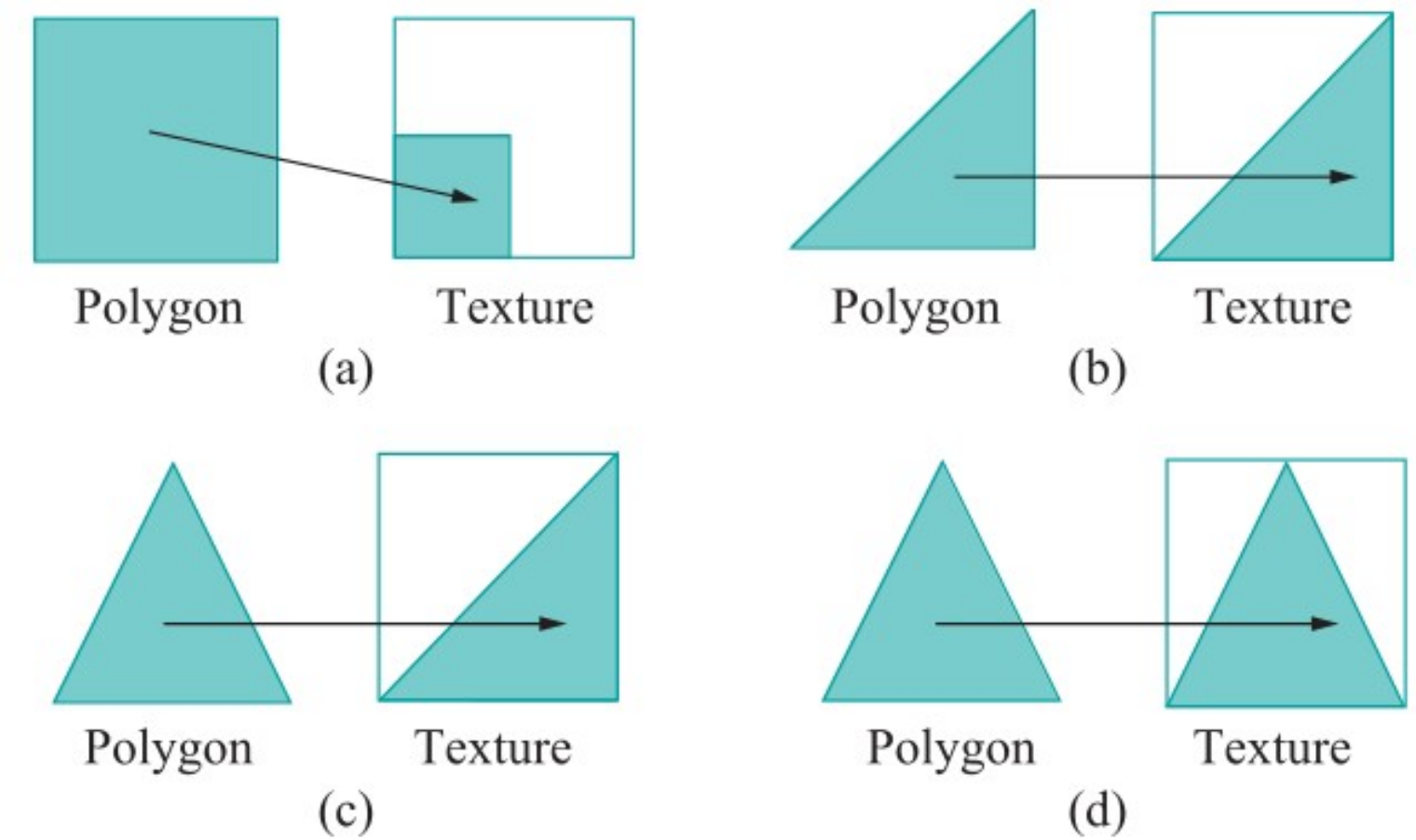
Espaço da Textura, Coordenadas e Mapa

- No código abaixo, **cada vértice** do polígono é mapeado para um **ponto** no espaço da textura
  - As coordenadas mapeadas no espaço da textura são chamadas de **coordenadas da textura** do vértice

```
glBegin(GL_POLYGON);
    glTexCoord2f(0.0, 0.0); glVertex3f(-10.0, -10.0, 0.0);
    glTexCoord2f(1.0, 0.0); glVertex3f(10.0, -10.0, 0.0);
    glTexCoord2f(1.0, 1.0); glVertex3f(10.0, 10.0, 0.0);
    glTexCoord2f(0.0, 1.0); glVertex3f(-10.0, 10.0, 0.0);
glEnd();
```

- O mapeamento dos vértices do polígono para o espaço da textura são **interpolados** para obter o **mapa da textura**
- Cada ponto do polígono é desenhado com o valor RGB da imagem no **mapa da textura**

Espaço da Textura, Coordenadas e Mapa



Espaço da Textura, Coordenadas e Mapa

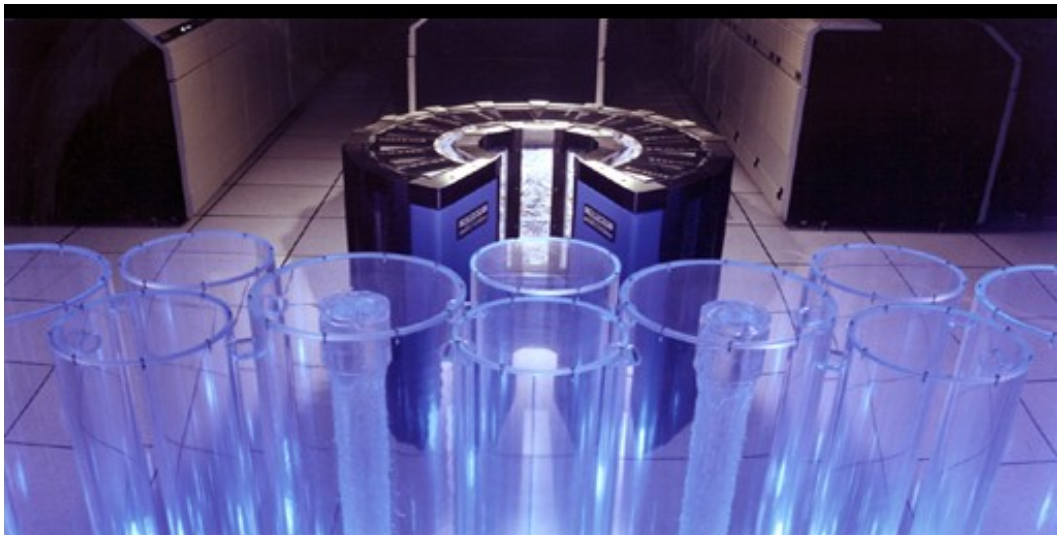
- A textura pode acabar aparecendo de forma distorcida caso as proporções entre o objeto e a textura não sejam equivalentes

Exercício 2

- Quais são as coordenadas das texturas dos seguintes pontos nas coordenadas do mundo?
  - a) (0.0, 0.0, 0.0)
  - b) (5.0, 5.0, 0.0)
  - c) (10.0, 0.0, 0.0)

Exercício 3

1) Carregue a imagem `cray2.bmp` e compare o resultado da aplicação da textura com a imagem original
2) Ajuste as especificações do polígono para que a imagem não apareça distorcida



Exercício 4

- Altere as especificações do polígono para mapear um polígono de 5 pontas nas coordenadas do mundo para um polígono de 5 pontas no espaço da textura:

```
glBegin(GL_POLYGON);
   glTexCoord2f(0.0, 0.0); glVertex3f(-10.0, -10.0, 0.0);
   glTexCoord2f(1.0, 0.0); glVertex3f(10.0, -10.0, 0.0);
   glTexCoord2f(1.0, 0.5); glVertex3f(20.0, 0.0, 0.0);
   glTexCoord2f(0.5, 1.0); glVertex3f(0.0, 10.0, 0.0);
   glTexCoord2f(0.0, 1.0); glVertex3f(-10.0, 0.0, 0.0);
glEnd();
```

Sumário

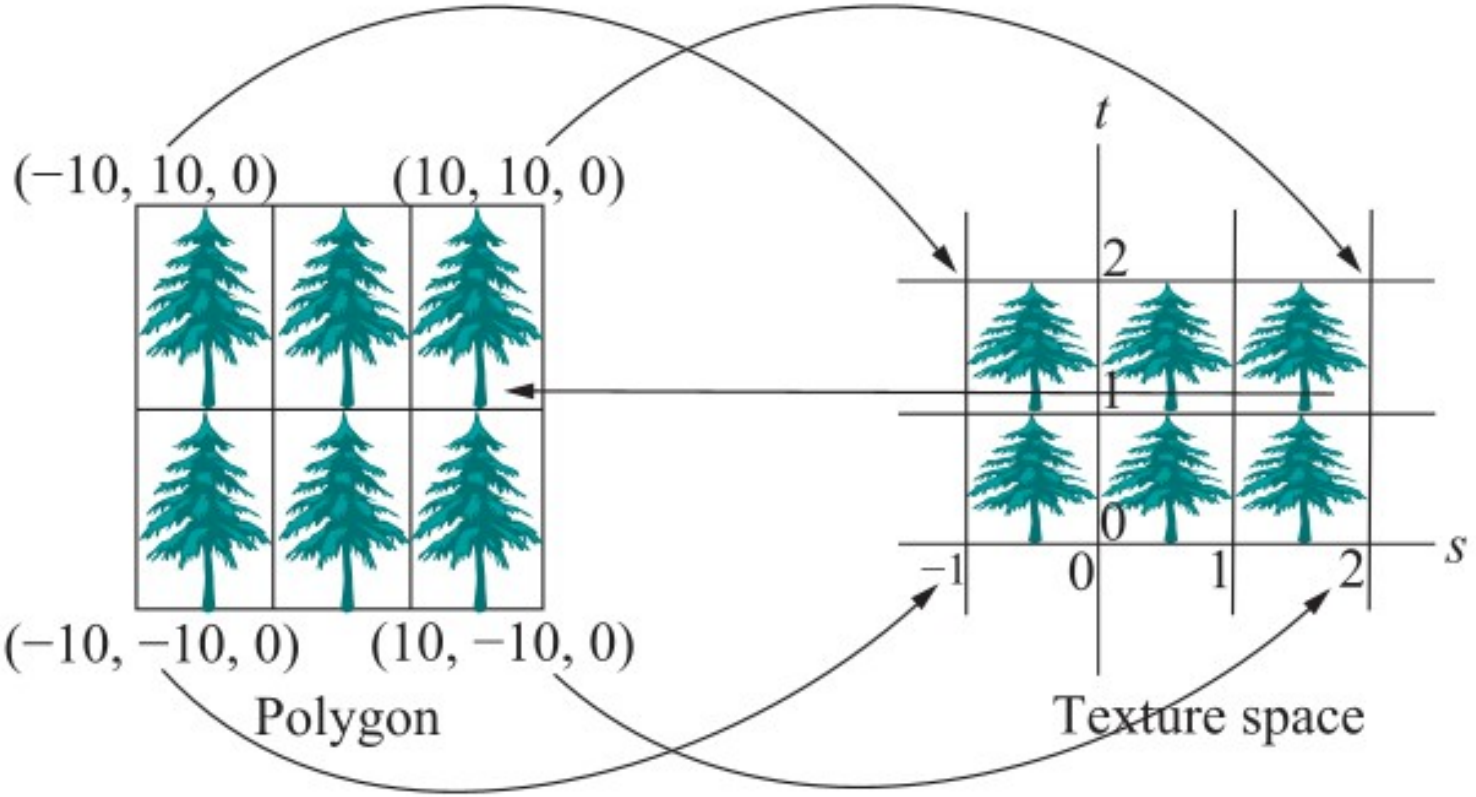
Repetindo e Apertando

- Até o momento mantivemos as coordenadas da textura no intervalo de [0, 1] nos eixos `s` e `t`
- O que aconteceria se definíssemos **valores** fora desse intervalo?
  - Experimente definindo valores fora do intervalo [0, 1] nas coordenadas da textura e veja o resultado

```
glBegin(GL_POLYGON);
   glTexCoord2f(-1.0, 0.0); glVertex3f(-10.0, -10.0, 0.0);
   glTexCoord2f(2.0, 0.0); glVertex3f(10.0, -10.0, 0.0);
   glTexCoord2f(2.0, 2.0); glVertex3f(10.0, 10.0, 0.0);
   glTexCoord2f(-1.0, 2.0); glVertex3f(-10.0, 10.0, 0.0);
glEnd();
```

Repetindo e Apertando

- A textura está sendo **repetida** em cada quadrado unitário do espaço da textura
- Como o polígono é mapeado para um retângulo 3 x 2 no espaço da textura, ela é pintada com **6 cópias da textura**, cada uma redimensionada para uma razão de aspecto de 2:3



Repetindo e Apertando

- Isso ocorre por conta do parâmetro `GL_REPEAT` definido na função `glTexParameteri()` programa:

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
```

- Esse parâmetro nas duas definições especifica que o modo de envelopamento (Wrapping Mode) é **repetir a textura** em ambos os eixos `s` e `t`

Repetindo e Apertando

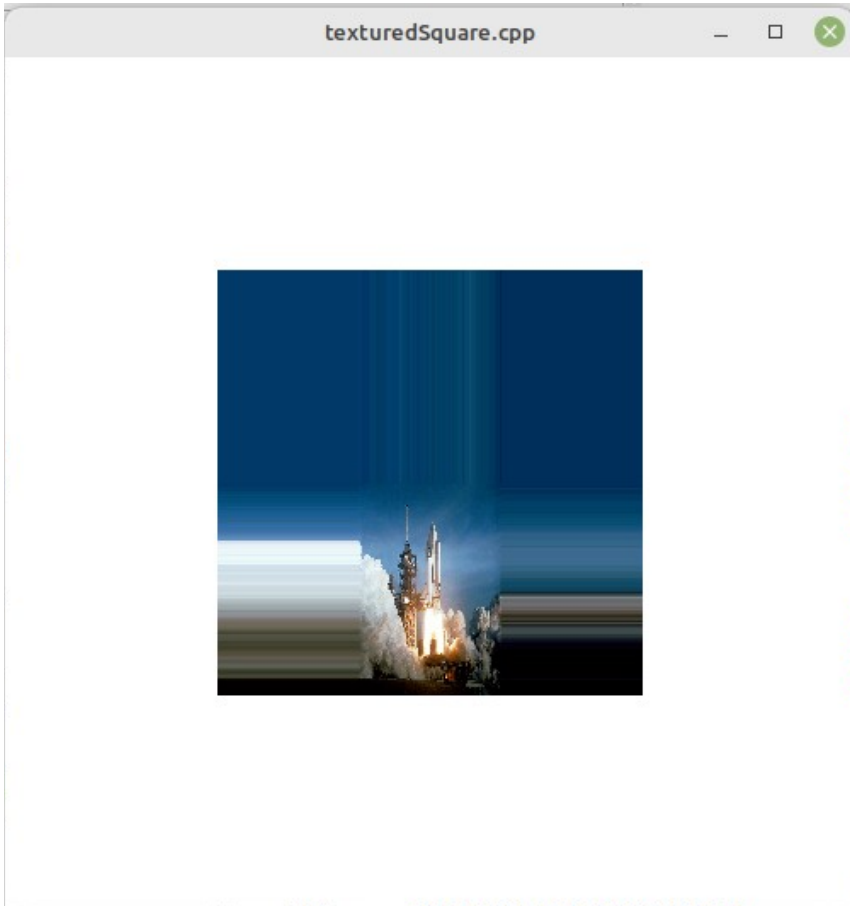
- **Experimento:**
  - Restaure a versão original do arquivo texturedSquare.cpp e mude as coordenadas da textura conforme abaixo:

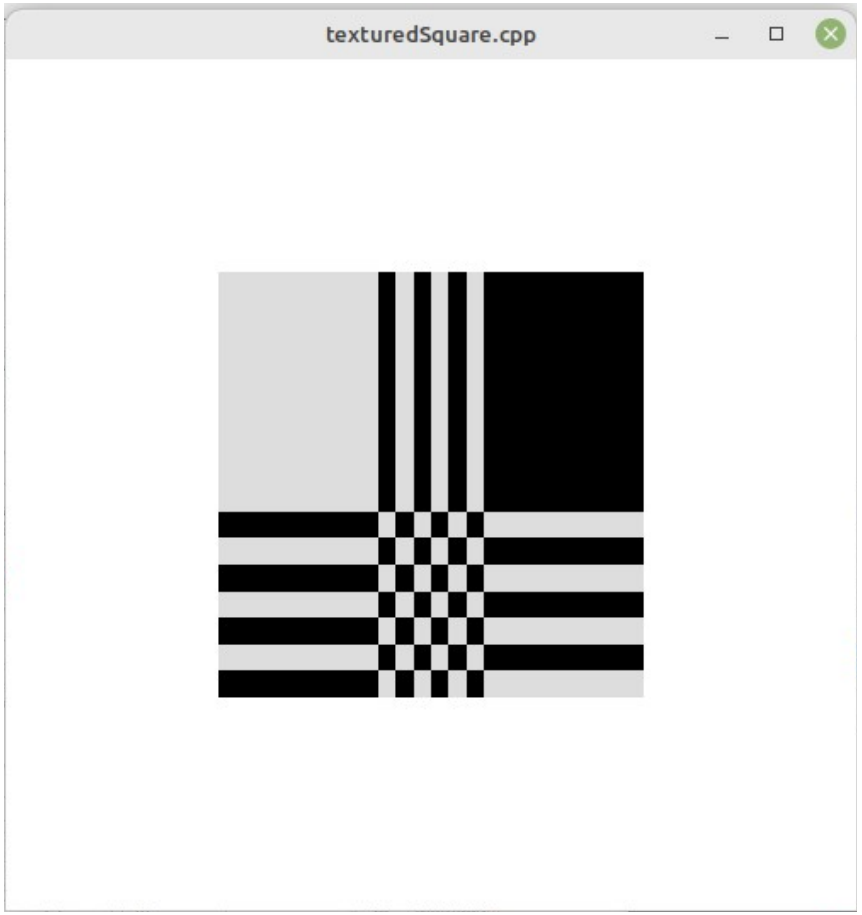
```
glBegin(GL_POLYGON);
    glTexCoord2f(-1.0, 0.0); glVertex3f(-10.0, -10.0, 0.0);
    glTexCoord2f(2.0, 0.0); glVertex3f(10.0, -10.0, 0.0);
    glTexCoord2f(2.0, 2.0); glVertex3f(10.0, 10.0, 0.0);
    glTexCoord2f(-1.0, 2.0); glVertex3f(-10.0, 10.0, 0.0);
glEnd();
```

  - Em seguida, altere o parâmetro `GL_REPEAT` para `GL_CLAMP_TO_EDGE` nas funções `loadTextures()` e `loadChessBoardTexture()` somente para o eixo `s` e veja o resultado
  - Repita o procedimento, agora alterando o parâmetro também para o eixo `t`

Repetindo e Apertando

- O parâmetro `GL_CLAMP_TO_EDGE` define que, ao longo do eixo especificado, a última cor da textura é repetida até o final do objeto mapeado





Repetindo e Apertando

- A opção de **repetição (`GL_REPEAT`)** é adequado para preencher a superfície de um objeto com um padrão específico
  - **Exemplos:** uma parede com padrão de tijolos, uma mesa com um padrão de madeira, um solo coberto com grama, etc.
- A opção de **clamping (`GL_CLAMP_TO_EDGE`)** é adequado para pintar uma única cópia de uma textura, de forma que a textura fique alinhada com as bordas

Exercício 5

- Selecione alguma(s) textura(s) na internet e aplique sobre um cubo. Permita que o usuário rotacione esse cubo utilizando o mouse.

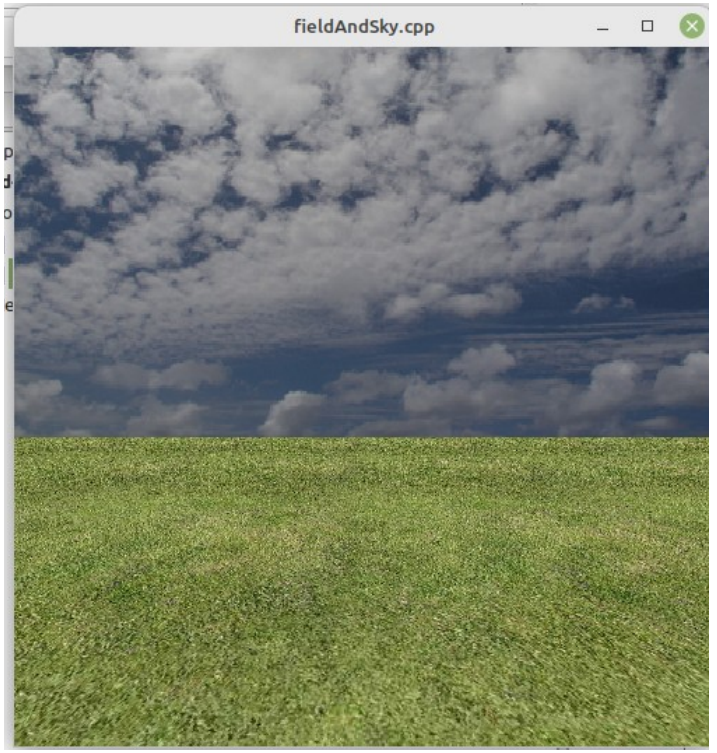
Sumário

Filtragem

- Abra o arquivo fieldAndSky.cpp e execute-o
- Movimente a câmera pelo cenário utilizando as setas direcionais para cima e para baixo
- O que você percebeu? Algo "estranho"?

Efeito Cintilante

- À medida que avançamos ou recuamos a posição da câmera, o gramado "**brilha**" em um efeito "**cintilante**"
- Esse **efeito** é causado por conta de um problema comum na CG que é o *aliasing*
- Isso se deve à:
  - Forma como o OpenGL aplica a textura
  - Forma como as cores são exibidas na tela

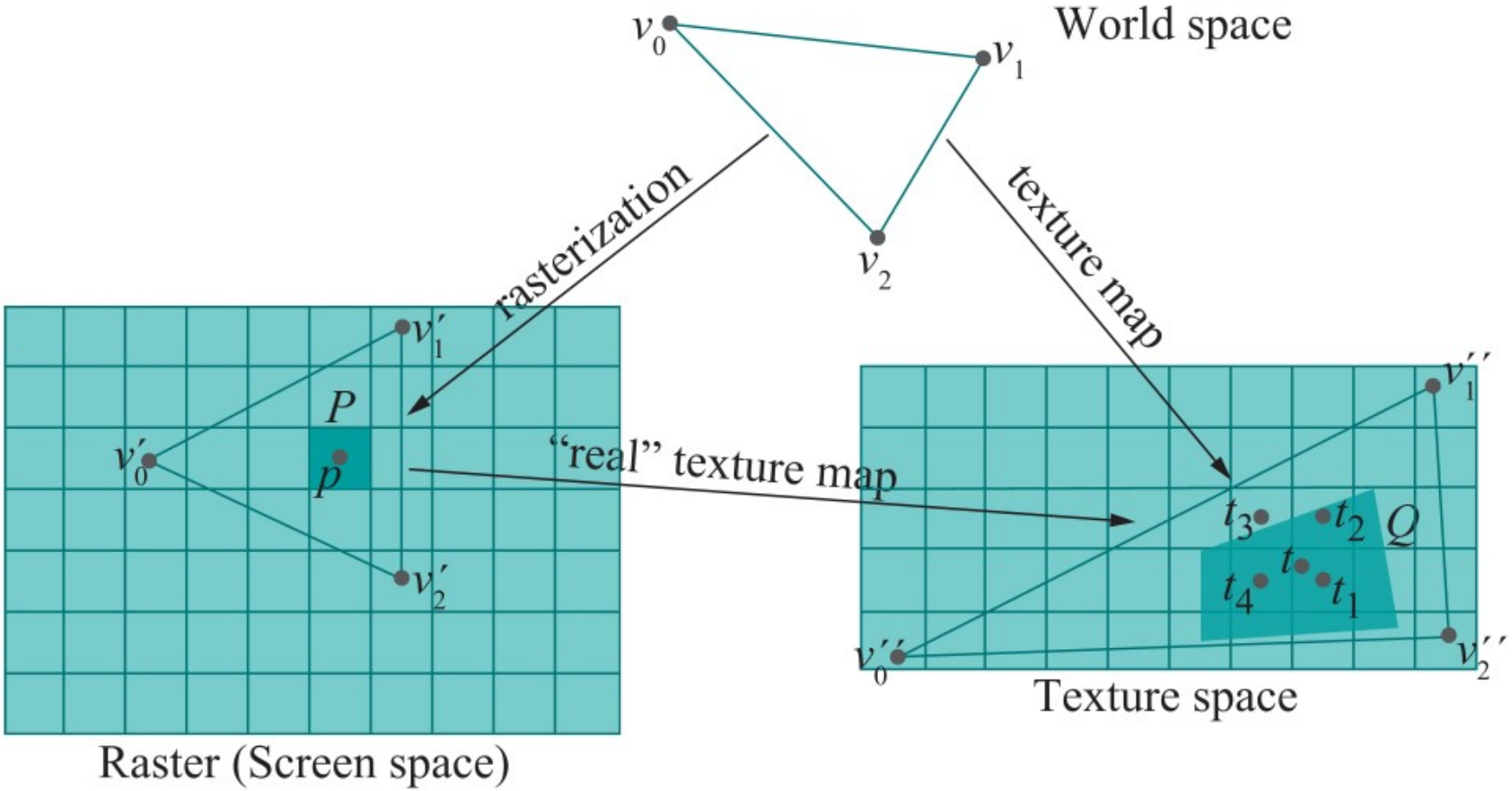


Efeito Cintilante

- **Relembrando:**
  - Aplicação de texturas no OpenGL:
    - O mapa de textura é obtido interpolando em cada objeto a coordenada da textura especificada nos vértices
    - Cada ponto do objeto é colorido com a cor do ponto na textura mapeada
  - Rasterização:
    - O valor das cores (RGB / RGBA) é associada ao pixel na tela e não a um ponto

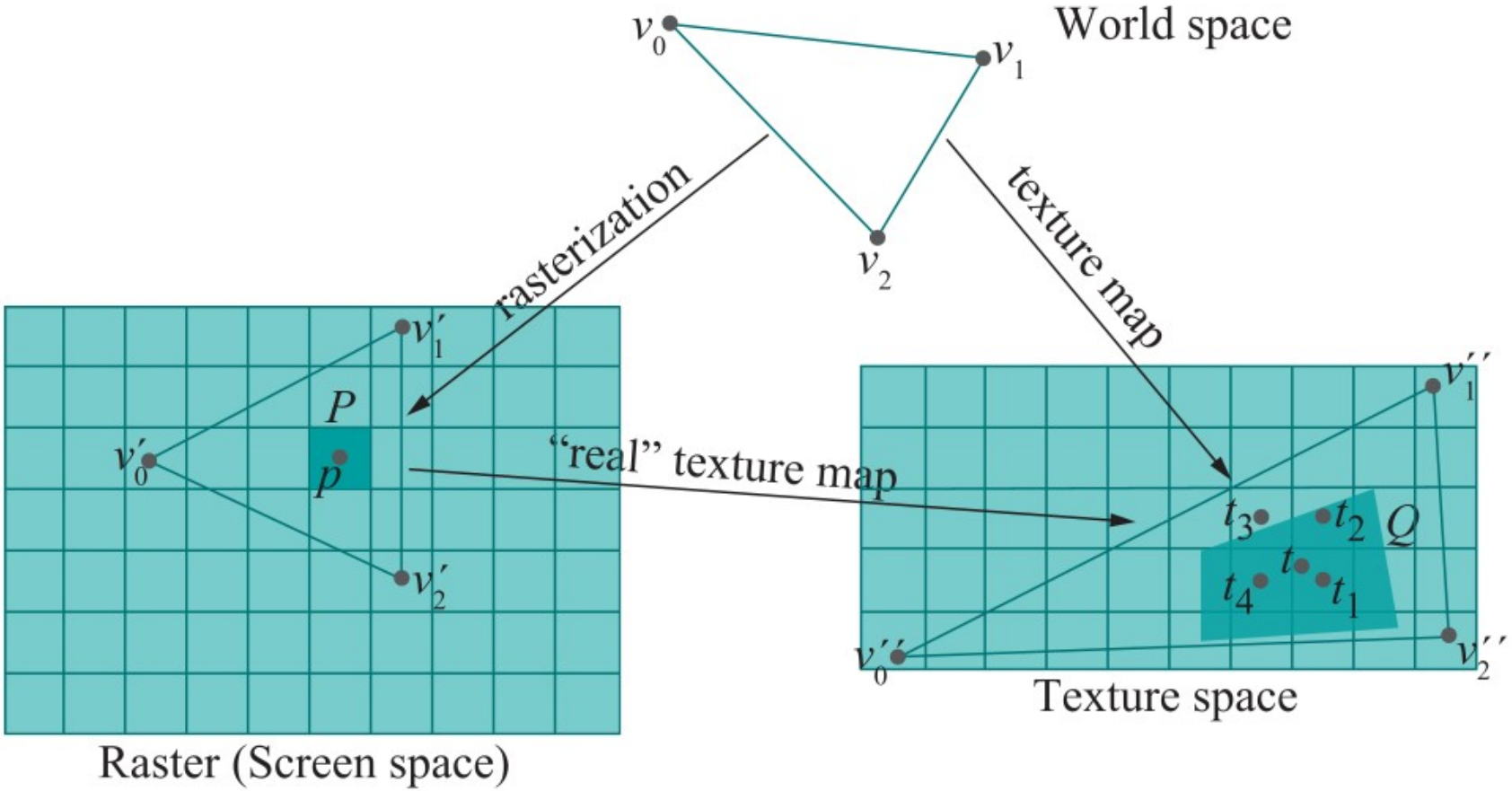
Efeito Cintilante - Causas

- Considere o mapeamento do objeto para a tela. O pixel `P` é mapeado para o quadrilátero `Q` no espaço da textura.
- O mapeamento nem sempre é 1 x 1
  - Como o OpenGL deve selecionar o valor das cores para o pixel `P`?



Efeito Cintilante - Causas

- **Possível solução:**
  - Identificar o texel cujo centro está mais próximo do ponto `t` (nesse caso t_1)
  - Aplicar a cor correspondente a esse texel ao pixel `P`



Efeito Cintilante - Causas

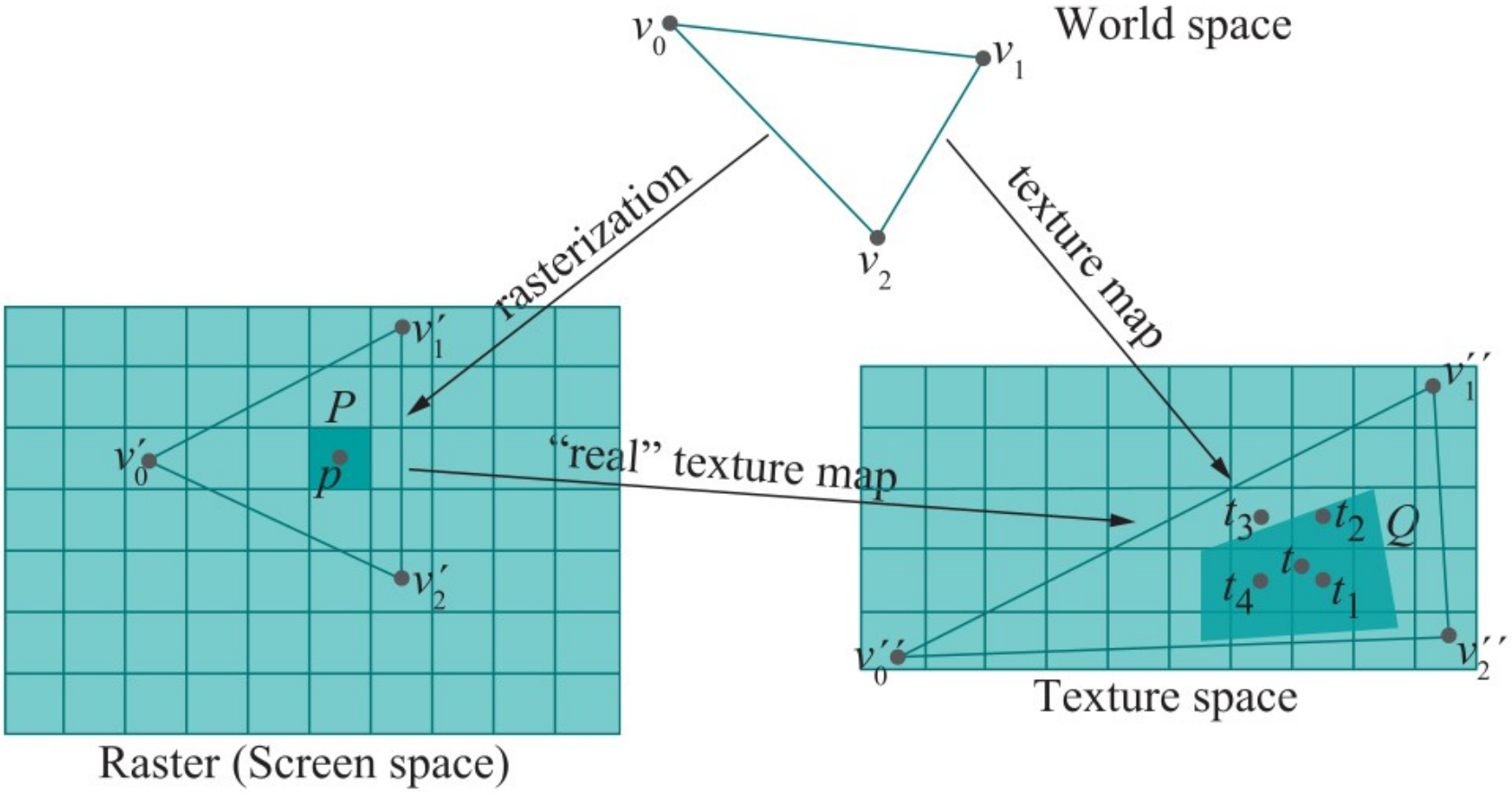
- Essa solução é utilizada aplicando a opção de filtragem com o parâmetro GL_NEAREST

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
```

- Qual o problema dessa solução?

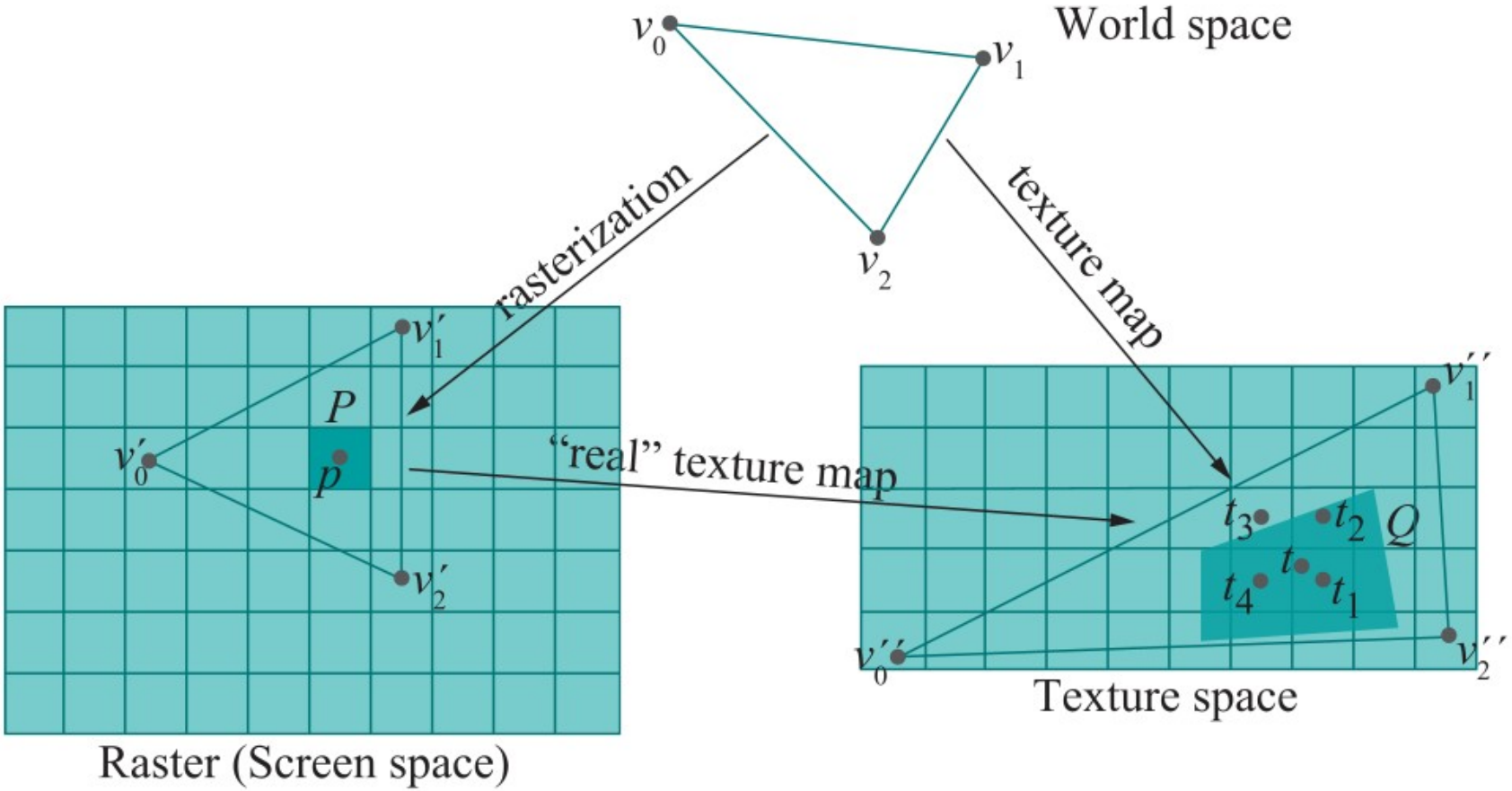
Efeito Cintilante - Causas

- Ao mover a posição de um objeto, o ponto `t` pode estar, agora, mais próximo do texel t_4, levando a uma **mudança de cores** durante a movimentação
  - Como poderíamos amenizar esse efeito?



Efeito Cintilante – Amenizando o efeito

- Uma possível solução é realizar a **amostragem dos texels** ao redor do ponto `t` e calcular a média para **suavizar** o efeito de transição
- Isso pode ser feito definindo a filtragem `GL_LINEAR` nos parâmetros de especificação da textura

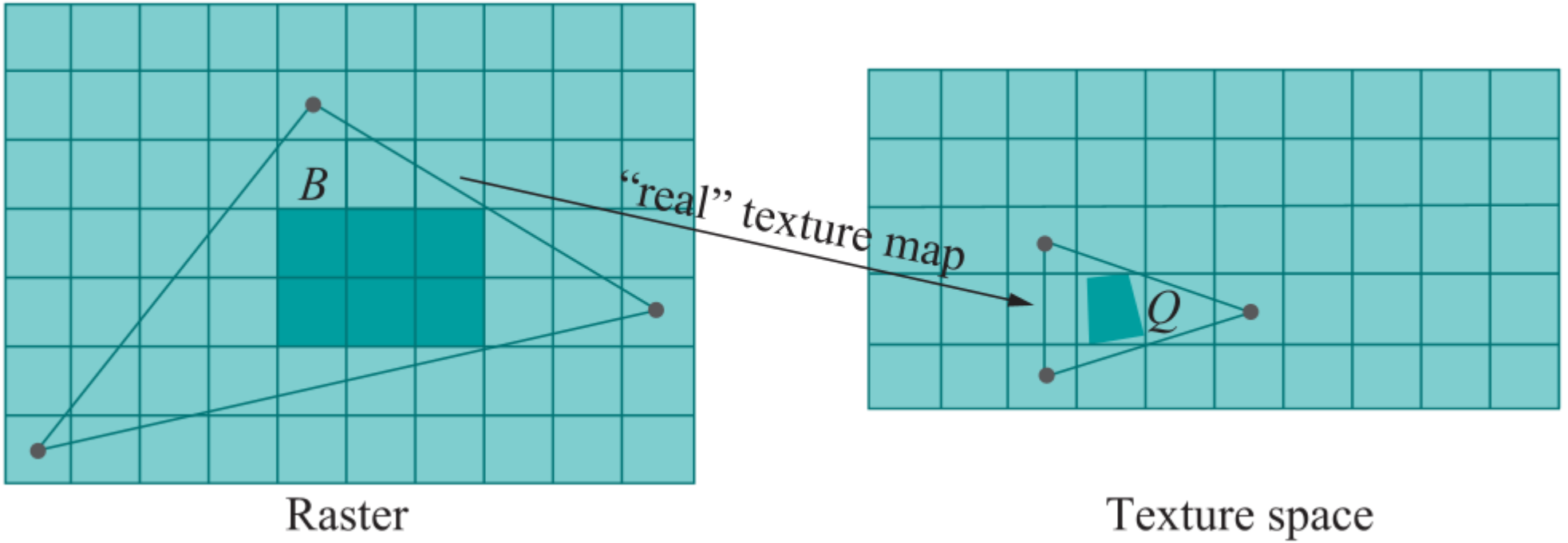


Filtragem

- Processo de amostragem de valores de cores para os pixels
- OpenGL oferece algumas opções de filtragem
  - GL_NEAREST
  - GL_LINEAR
- O OpenGL reconhece **duas formas** distintas de mapear as cores
  - Redução (Minification)
  - Ampliação (Magnification)

Filtragem – Minification e Magnification

- **Minification:** ccorre quando um **pixel** é mapeado para **múltiplos texels**
- **Magnification:** ocorre quando vários **pixels** são mapeados para um **único texel**



Filtragem – Minification e Magnification

- Especificando a filtragem em OpenGL

```
glTexParameteri(GL_TEXTURE_2D, case, filter);
```

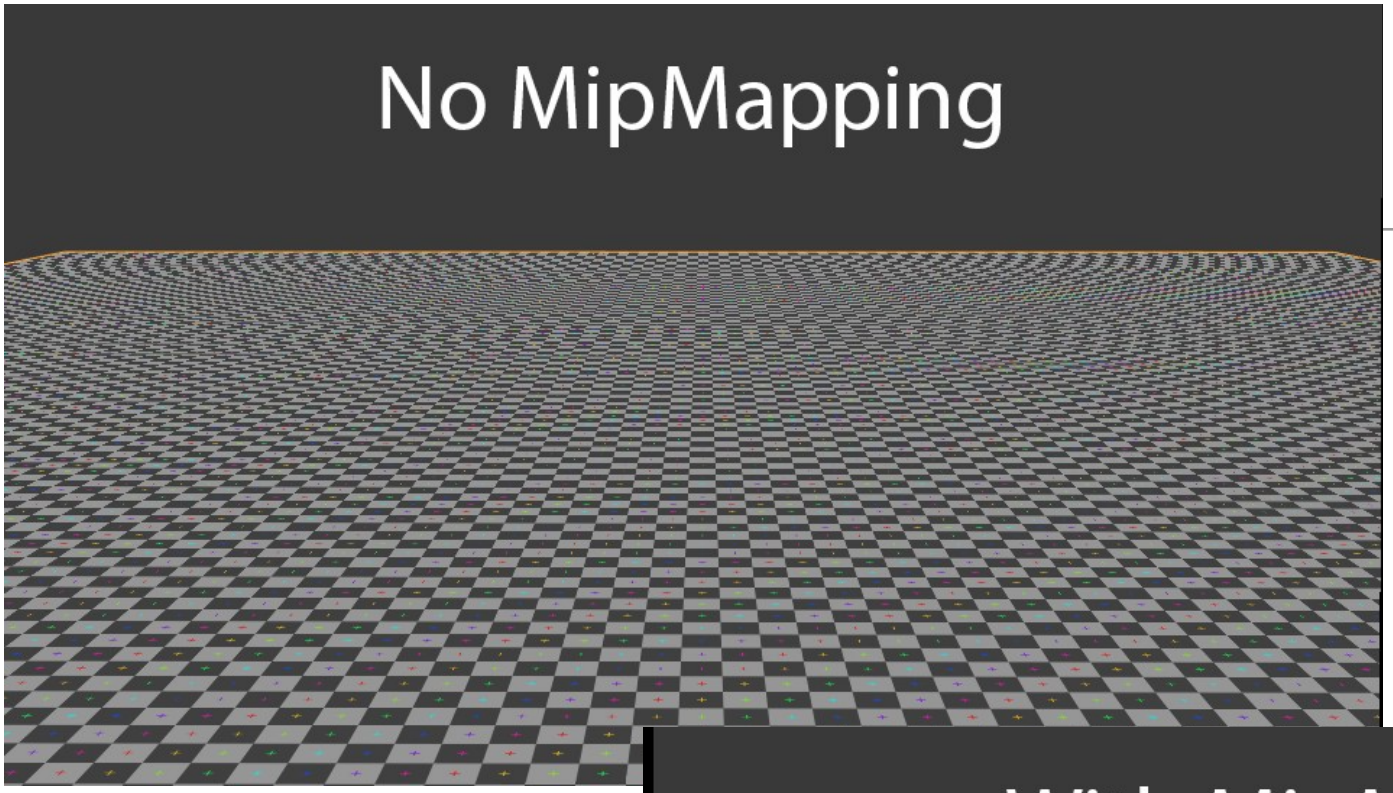
- No exemplo `fieldAndSky.cpp`, o programa especifica que em ambos os casos (minification ou magnification), a filtragem `GL_NEAREST` deve ser aplicada

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
```

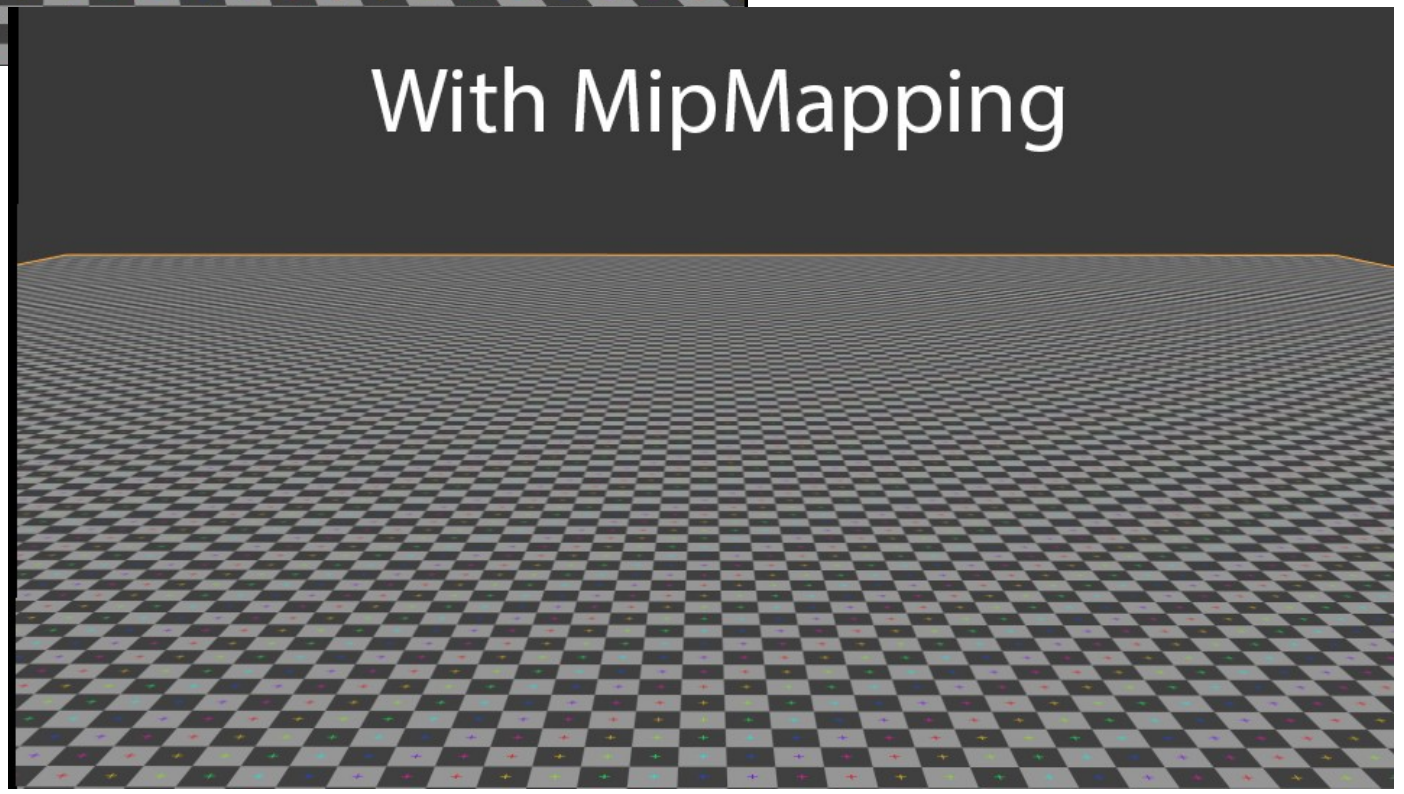
Filtragem - Mipmaps

- O OpenGL permite a pré-definição de um **conjunto de texturas** para ser usado em diferentes níveis de minification
- A partir da textura original, um **conjunto de texturas** de resoluções menores, chamadas de `mipmaps`, são geradas
  - MIP é um acrônimo para o termo em latim "*multum in parvo*" que significa muito em pouco
- À medida que o objeto ocupa um espaço menor na tela, o OpenGL mapeia a textura desse objeto para o `mipmap` que permite uma correspondência próxima de 1x1
  - Reduz a carga computacional para aplicação da filtragem
  - Garante uma maior qualidade das texturas

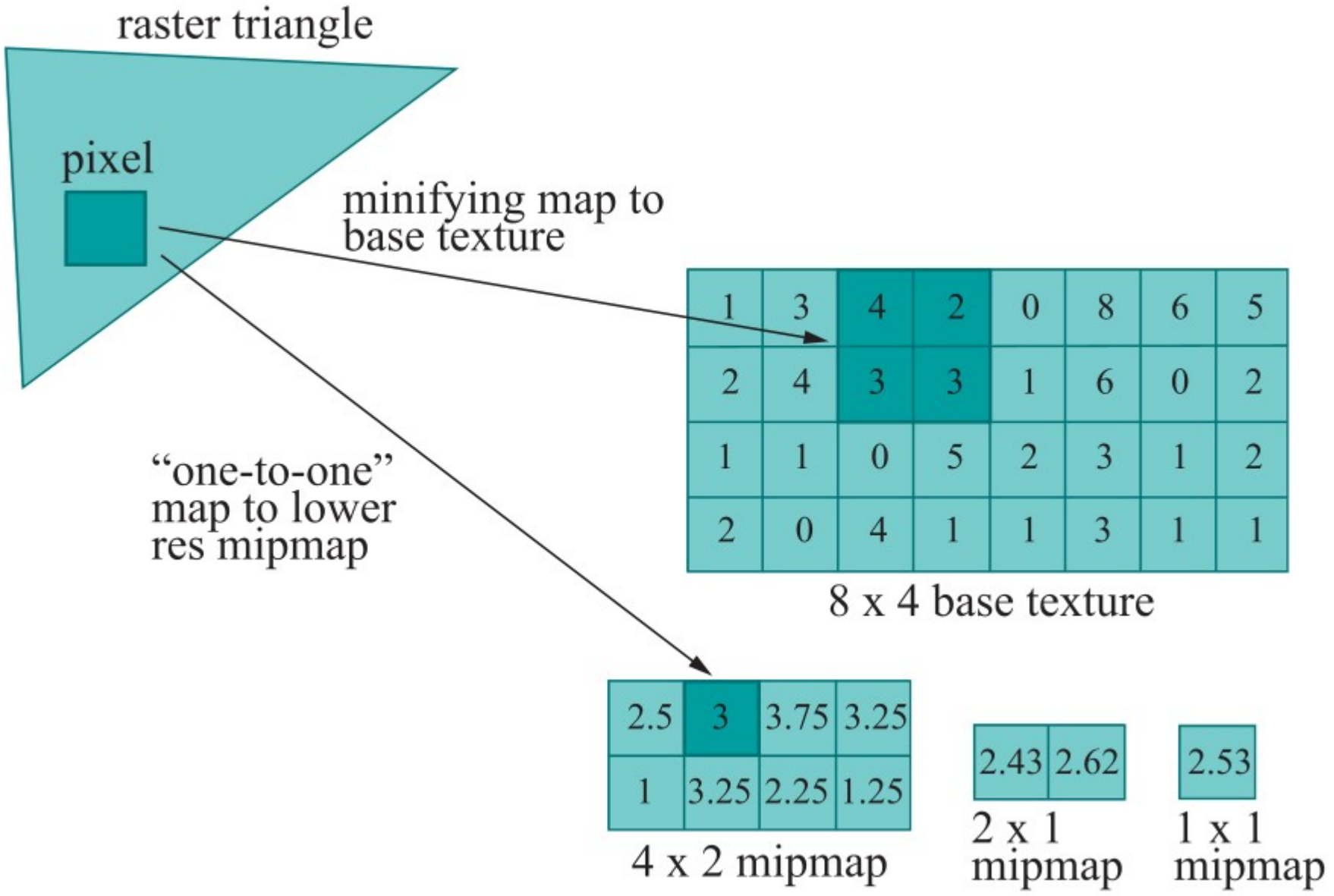
Filtragem - Mipmaps



UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ



Filtragem - Mipmaps



Filtragem - Mipmaps

- Além das opções `GL_NEAREST` e `GL_LINEAR` (aplicáveis para a minification e magnification), podem ser usadas outras quatro opções de filtragem para a minification:
  - `GL_NEAREST_MIPMAP_NEAREST`
  - `GL_LINEAR_MIPMAP_NEAREST`
  - `GL_NEAREST_MIPMAP_LINEAR`
  - `GL_LINEAR_MIPMAP_LINEAR`

Exercício 6

- Dado a textura 4 x 8 ao lado com os valores das cores de cada texel, identifique todos os mipmaps resultantes até a menor resolução possível.

1	0	4	2
3	2	1	5
0	1	2	6
8	2	7	7
2	3	1	2
6	4	3	8
7	3	6	1
3	5	0	2