Introdução à Texturização

por Rossana B Queiroz

Problema

- Como apresentar superficies com detalhes?
  - Muitos polígonos: alto custo computacional

Introdução

- Solução
  - Utilização de **texturas** para melhorar o **realismo** das cenas renderizadas
- O processo chama-se **mapeamento de textura**
- Iniciou com **Catmull** em 1974
  - Utilizou preenchimento de polígonos com **imagens** para simular **detalhamento** e **realismo visual**

Dr. Ed Catmull

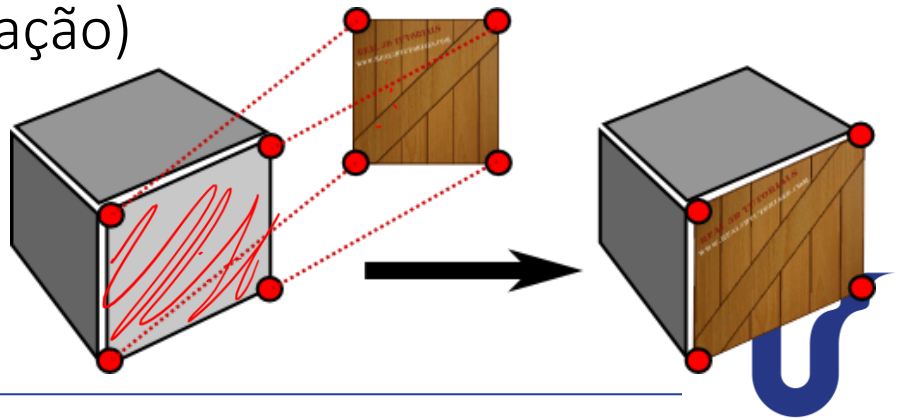
President, Walt Disney and Pixar Animation Studios

Share |

Ed Catmull, Ph.D, is co-founder of Pixar Animation Studios and President of Walt Disney and Pixar Animation Studios. Previously, Catmull was Vice President of the computer division of Lucasfilm Ltd., where he managed development in the areas of computer graphics, video editing, video games and digital audio.

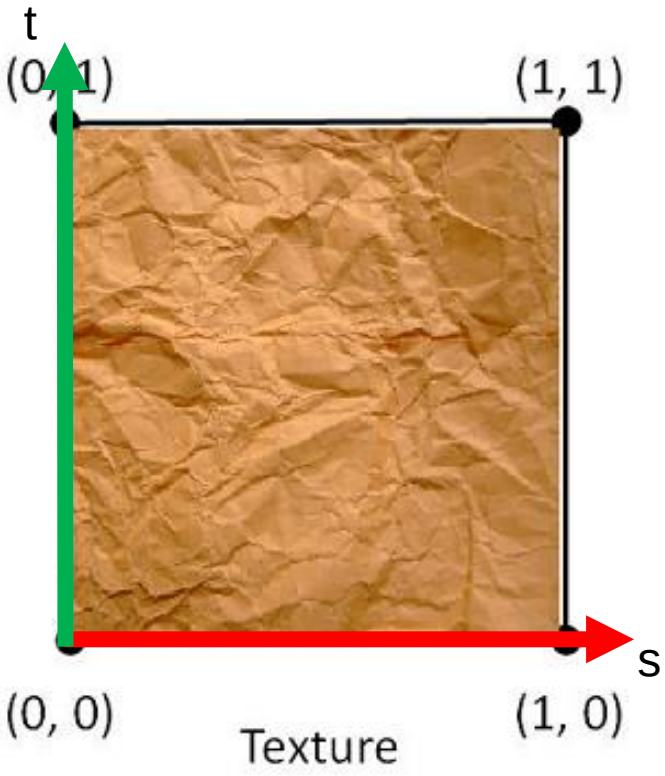
Mapeando Texturas

- Para mapear uma textura num triângulo precisamos informar para cada vértice, a qual parte da textura ele corresponde
- Logo, cada vértice terá uma coordenada de textura
- O fragment shader se encarrega de fazer o preenchimento dos demais pixels (interpolação)



Mapeando Texturas

- As coordenadas variam entre 0 e 1 nos eixos x e y

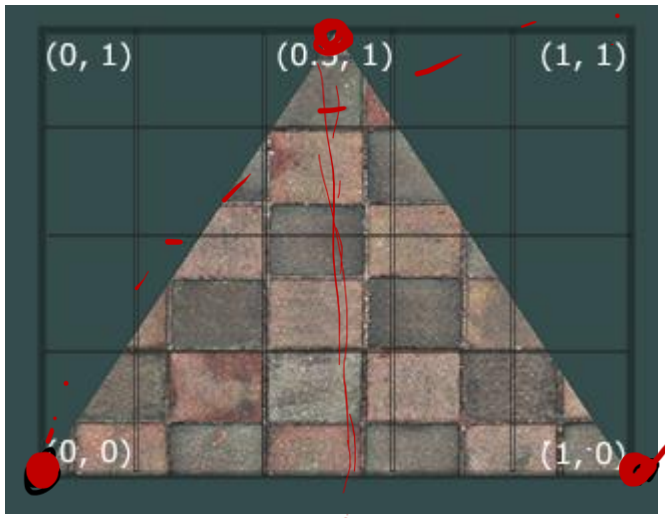


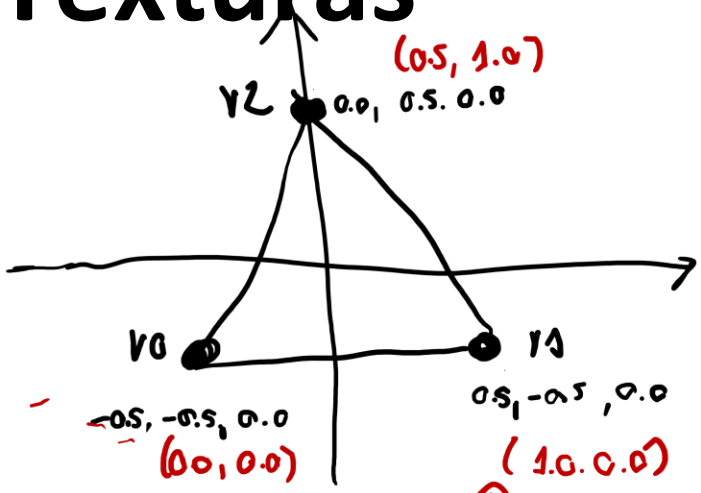
Sampling: processo de recuperar as cores da textura usando as coordenadas de textura

```
GLfloat texCoords[] = {
0.0f, 0.0f, // Inferior esquerdo
1.0f, 0.0f, // Inferior direito
1.0f, 1.0f, // Superior direito
0.0f, 1,0f  // Superior esquerdo
};
```

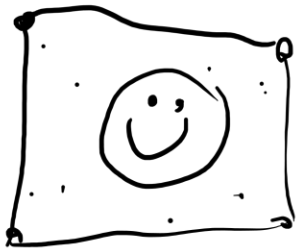
Mapeando Texturas

- Outro exemplo: triângulo






```
float texCoords[] = {
 0.0f, 0.0f, // lower-left corner
 1.0f, 0.0f, // lower-right corner
 0.5f, 1.0f // top-center corner };
```

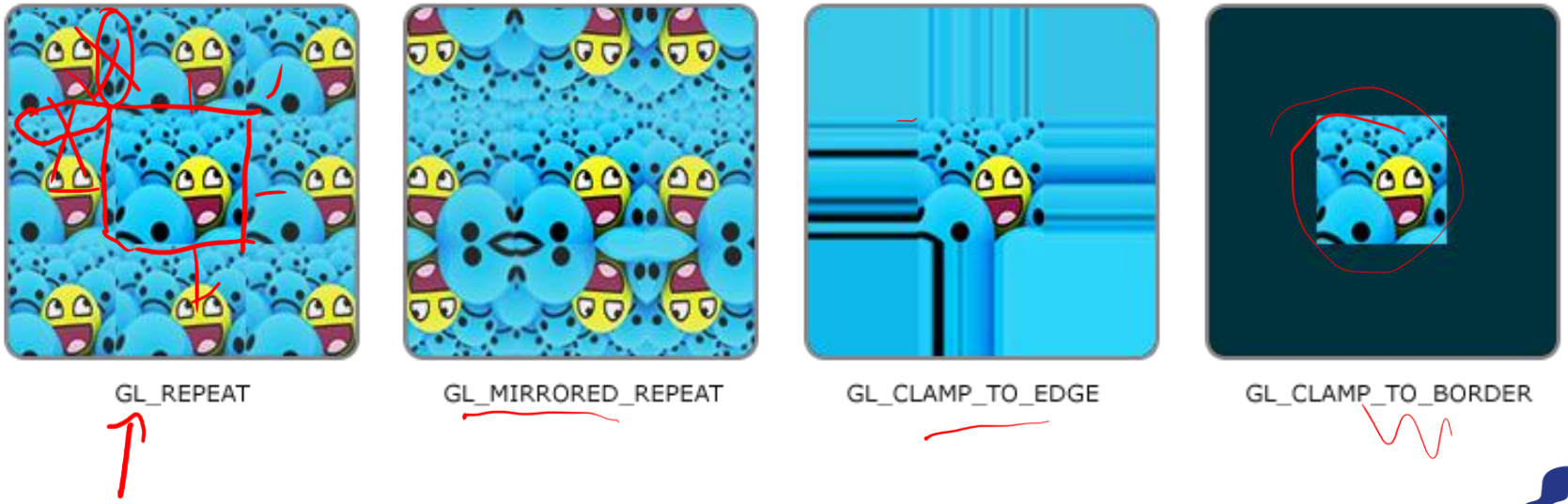


Texture Wrapping

- O que fazer se especificarmos coordenadas de textura fora do intervalo de (0,0) a (1,1)?
  - GL_REPEAT: comportamento padrão, que repete a imagem de textura
  - GL_MIRRORED_REPEAT: mesmo que GL_REPEAT mas espelha a imagem a cada repetição.
  - GL_CLAMP_TO_EDGE: “prende”as coordenadas entre 0 e 1. Os valores mais altos ficam presos às arestas.
  - GL_CLAMP_TO_BORDER: coordenadas for a do intervado recebem uma cor de borda.

Texture Wrapping

- O que fazer se especificarmos coordenadas de textura fora do intervalo de (0,0) a (1,1)?



Texture Wrapping

- Como setar?
  - Coordenadas s, t

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

  - Na opção CLAMP_TO_BORDER, podemos definer uma cor para as bordas

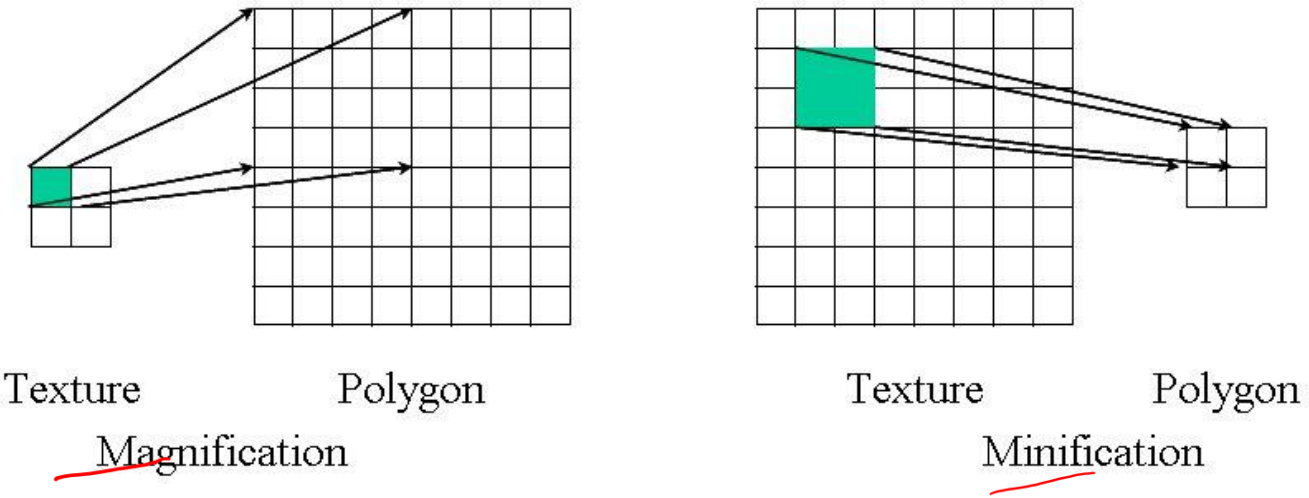
```
float borderColor[] = { 1.0f, 1.0f, 0.0f, 1.0f };
glTexParameterfv(GL_TEXTURE_2D, GL_TEXTURE_BORDER_COLOR,
borderColor);
```

Texture Filtering

- Também conhecido como **Texture smoothing**
- Um problema comum é o que acontece quando a textura é **esticada** ou **comprimida**.
- O processo que trata deste problema é chamado de ***filtering*** *(anti-alias para texturas)*

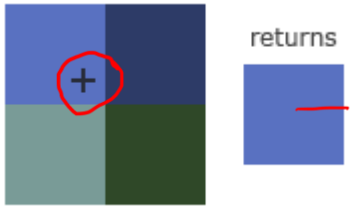
Texture Filtering

- *Ocorre através da* ***magnificação*** *e* minificação dos texels



Texture Filtering

- ***Nearest Neighbor Filtering***
  - Mais rápido e mais simples
  - Quanto falta algum pixel, a cor do **vizinho** mais próximo é utilizada
  - Resultado é uma imagem *"pixelada"*



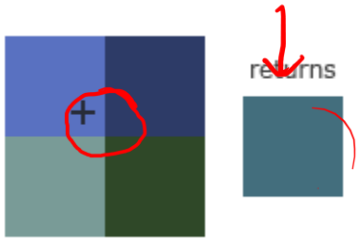

```
glTexParameter(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameter(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
```

Texture Filtering

- ***Linear Filtering***
  - A cor de um texel faltante é calculada pela

média ponderada dos seus vizinhos

  - Resultado é caracterizado com “fuzzy” graphics um resultado mais **suavizado**




```
glTexParameter(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameter(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
```

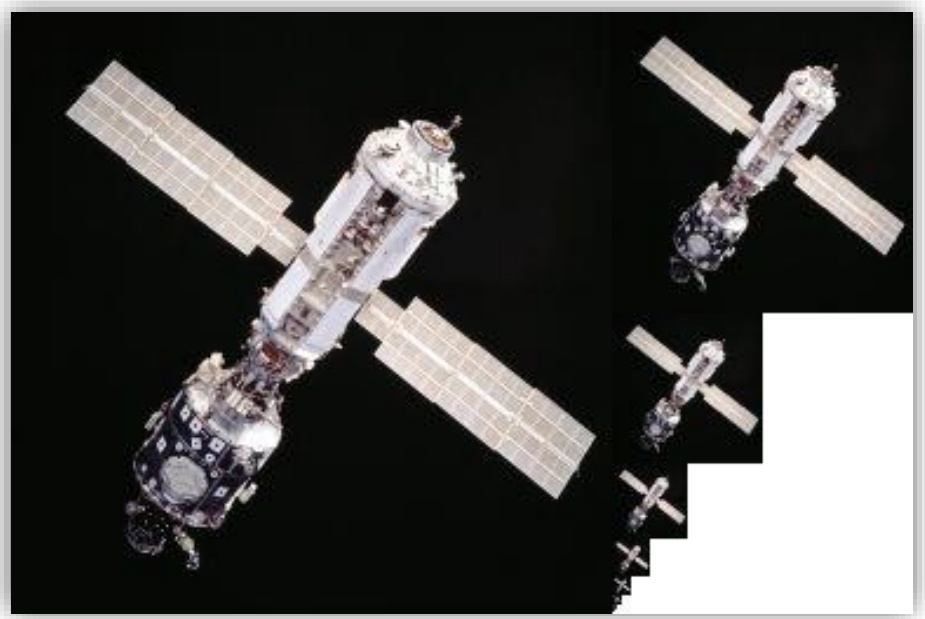
Texture Filtering

GL_NEAREST

GL_LINEAR

Mipmaps

- OpenGL constrói automaticamente diferentes imagens com resoluções menores, a partir da divisão da imagem de textura pela metade sucessivamente



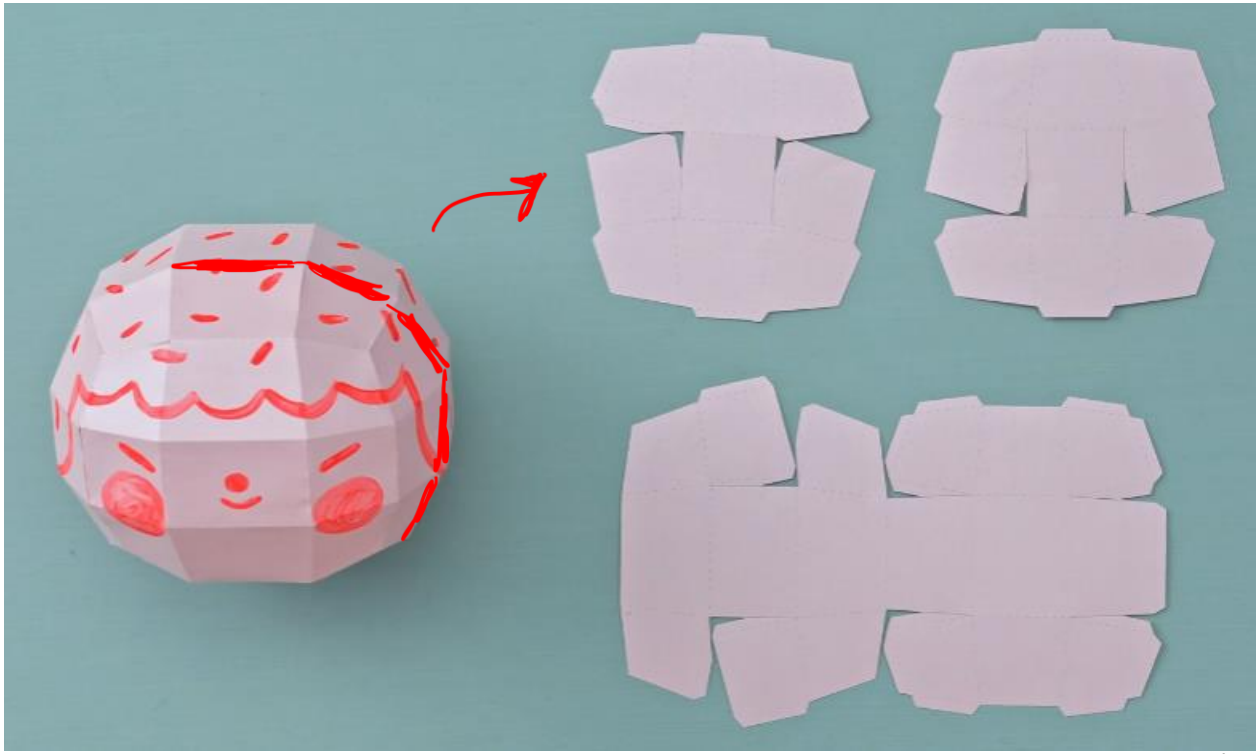
Mipmaps

- Para especificar o método de filtragem entre trocas de mipmaps, podemos setar as opções:
  - GL_NEAREST_MIPMAP_NEAREST: usa o mipmap mais próximo do tamanho do pixel e usa o método de interpolação do vizinho mais próximo para amostrar a textura
  - GL_LINEAR_MIPMAP_NEAREST: usa o mipmap mais próximo do tamanho do pixel e usa o método de interpolação linear para amostrar a textura
  - GL_NEAREST_MIPMAP_LINEAR: faz a interpolação linear dos dois mipmaps mais próximos ao tamanho do pixel e usa o método de interpolação do vizinho mais próximo para amostrar a textura
  - GL_LINEAR_MIPMAP_LINEAR: faz a interpolação linear dos dois mipmaps mais próximos ao tamanho do pixel e usa o método de interpolação linear para amostrar a textura

Mipmaps

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
```

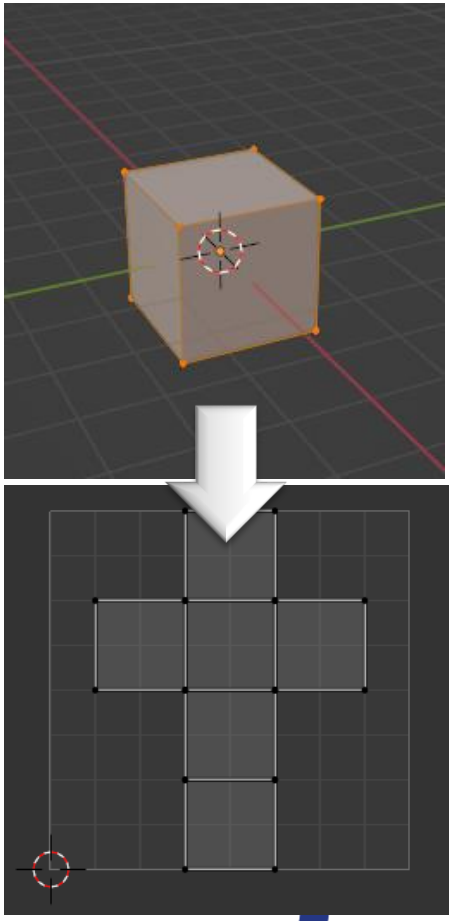


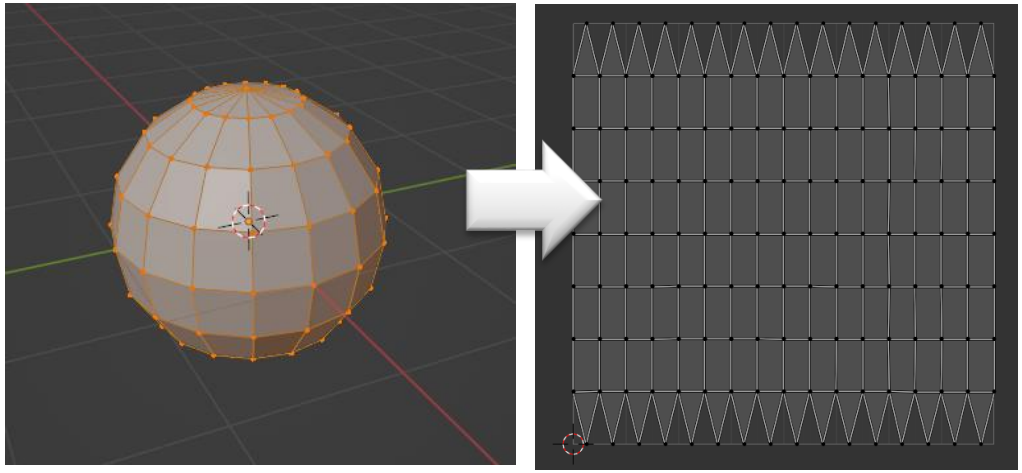
Fonte: Domestika

MAPEAMENTO UV EM MODELOS 3D

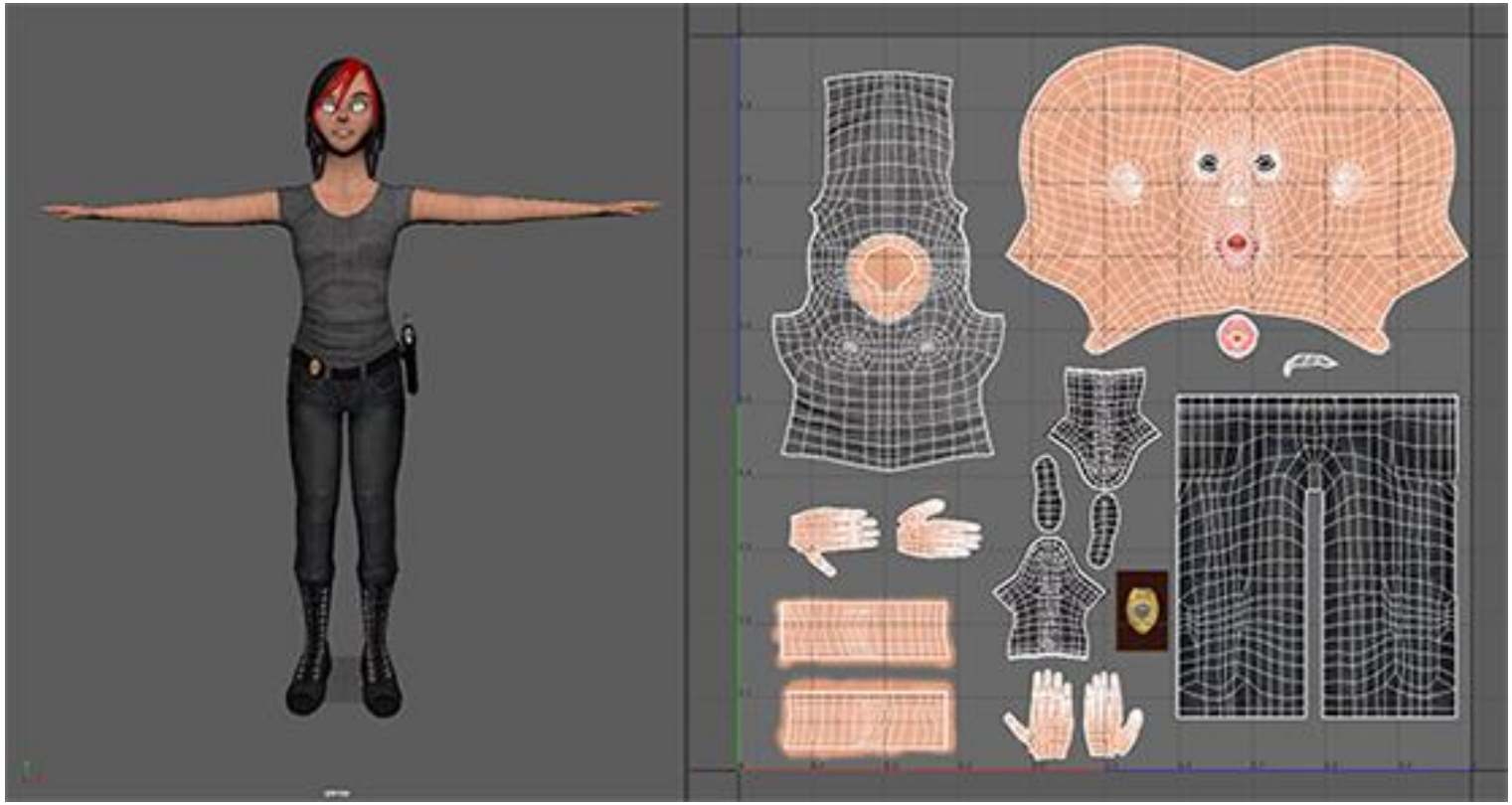
Mapeamento UV em Modelos 3D

- Requer algumas etapas:
  - Recorte da malha, se esta for fechada
  - Abertura a partir dos recortes e projeção

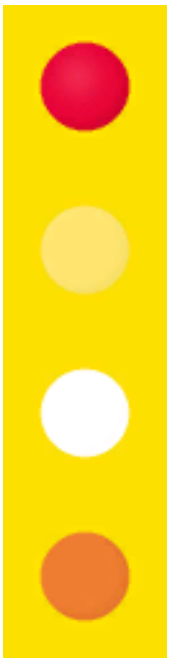


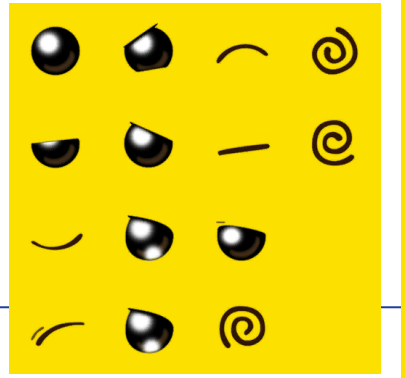


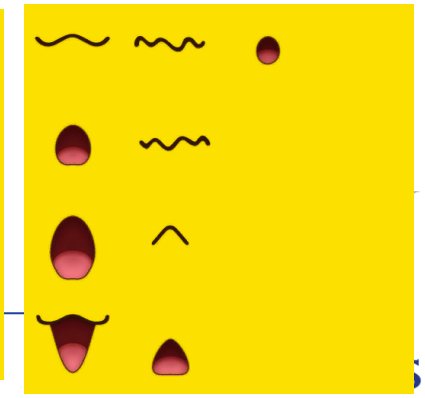
Atlas de Textura



Objetos com mais de um mapa







OPENGL & TEXTURAS - BÁSICO

Iniciando...

- Função ou método que recebe o caminho do arquivo de textura e retorna o ID da textura
  - Primeiro, gera o ID de textura

```
// Gera o identificador da textura na memória
glGenTextures(1, &texID);
glBindTexture(GL_TEXTURE_2D, texID);
```

Iniciando...

– Depois, ajusta-se os parâmetros de wrapping e filtering

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

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

Iniciando...

– Agora fazemos o carregamento do arquivo de imagem, utilizando uma biblioteca auxiliar

```
int width, height, nrChannels;

unsigned char *data = stbi_load(filename.c_str(), &width, &height,
&nrChannels, 0);
```

OBS.: Estamos usando a biblioteca stb_image: https://github.com/nothings/stb
Podemos diretamente adicionar o arquivo .h e .cpp em nosso código

Iniciando...

– Por último, mandamos os dados para OpenGL armazenar a textura na memória

```
if (data)
{
    if (nrChannels == 3) //jpg, bmp
    {
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE,
data);
    }
    else //png
    {
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0, GL_RGBA, GL_UNSIGNED_BYTE,
data);
    }
    glGenerateMipmap(GL_TEXTURE_2D);
}
else
{
    std::cout << "Failed to load texture" << std::endl;
}
```

Iniciando...

- Boas práticas: liberar a memória do buffer e desconectar o ID (“unbind”)

```
stbi_image_free(data);

glBindTexture(GL_TEXTURE_2D, 0);
```

- Não precisa ser dentro do método, mas na inicialização é necessário ativar o buffer de textura da OpenGL:

```
glActiveTexture(GL_TEXTURE0);
```

Iniciando...

- Sobre as unidades de textura: devem ser habilitadas antes de conectá-las (bind) corretamente antes de fazer a chamada de desenho

```
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, textID1);
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, textID2);

glBindVertexArray(VAO);
glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0);
```

E os buffers?

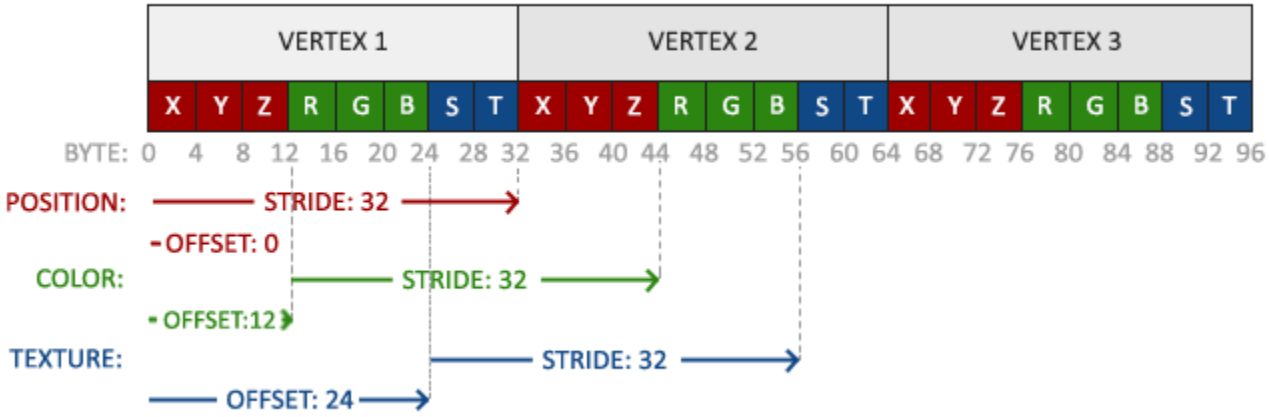
- Temos agora mais um atributo para os vértices!

```
ǧľộắʃ ŵêsʧîçêș
   řộșîçộêș              çộsêș                çộộsđêŋắđắș đê ʧêẏʧụsắ
. _ǧ   . _ǧ  . .ǧ    , .ǧ  . .ǧ  . .ǧ    , .ǧ  , .      ṣưřêsîộs đîsêîʧộ
. _ǧ   . _ǧ  . .ǧ    . .ǧ  , .ǧ  . .ǧ    , .ǧ  . .ǧ     îŋǧêsîộs đîsêîʧộ
 . _ǧ   . _ǧ  . .ǧ   . .ǧ  . .ǧ  , .ǧ    . .ǧ  . .ǧ     îŋǧêsîộs
êṣṛụêsđộ
 . _ǧ   . _ǧ  . .ǧ   , .ǧ  , .ǧ  . .ǧ    . .ǧ  , .      ṣưřêsîộs êṣṛụêsđộ

ụŋṣîĝŋêđ îŋʧ îŋđîçêș
.  ,  ‘      řsîṇêîsộ ʧsîắŋĝụľộ
,  ¸  ‘      ṣêĝụŋđộ ʧsîắŋĝụľộ
```

E os buffers?

- Temos agora mais um atributo para os vértices!



E os buffers?

- Temos agora mais um atributo para os vértices!

```
glGenVertexArrays(1, &VAO); glGenBuffers(1, &VBO); glGenBuffers(1, &EBO);

glBindVertexArray(VAO);

glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);

glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices, GL_STATIC_DRAW);

// position attribute
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)0);
glEnableVertexAttribArray(0);
// color attribute
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(3 * sizeof(float)));
glEnableVertexAttribArray(1);
// texture coord attribute
glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(6 * sizeof(float)));
glEnableVertexAttribArray(2);
```

E no(s) shader(s)?

- Vertex shader

Por que mantivemos o atributo cor? Veremos na aula ;)

```
#version 450 core
layout (location = 0) in vec3 position;
layout (location = 1) in vec3 color;
layout (location = 2) in vec2 tex_coord;

out vec4 vertexColor;
out vec2 texCoord;

uniform mat4 model;
uniform mat4 projection;

void main()
{
    gl_Position = projection * model * vec4(position, 1.0f);
    vertexColor = vec4(color,1.0);
    texCoord = vec2(tex_coord.x, tex_coord.y);
}
```

E no(s) shader(s)?

- Fragment Shader

Por que mantivemos o atributo cor? Veremos na aula ;)

```
#version 450 core
in vec3 vertexColor;
in vec2 texCoord;

out vec4 color;

// pixels da textura
uniform sampler2D tex_buffer;

void main()
{
    color = texture(tex_buffer, texCoord);
}
```

Finalizando...

Nome do método ou função que você criou!

- Não esquecer de chamar o método para carregar a imagem ☺

```
GLuint texID = loadTexture("./textures/mario.png");
```

- Não esquecer de especificar a variável uniform que vai conter os dados da textura no fragment shader

```
glUniform1i(glGetUniformLocation(shader.ID, "tex_buffer"), 0);
```

- Antes da drawcall:

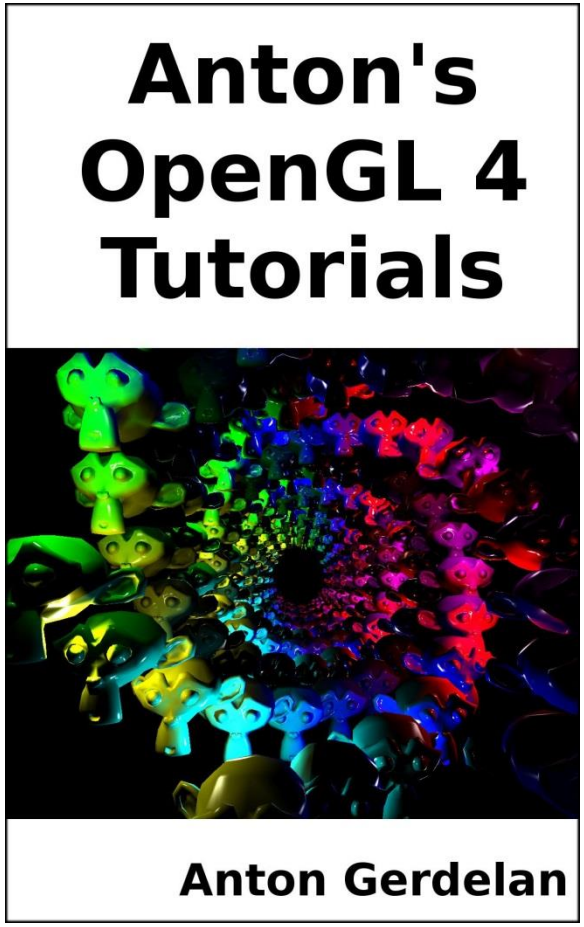
```
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, texID);

glBindVertexArray(VAO);
glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0);
glBindVertexArray(0);
```

Referências

- Slides sobre CG dos professores passados: Christian Hofsetz, Cristiano Franco, Marcelo Walter, Soraia Musse, Leandro Tonietto e Rafael Hocevar
- https://learnopengl.com/Getting-started/Textures

Referências



Ebook para Kindle

Muitos materiais online disponíveis em:

http://antongerdelan.net/opengl/