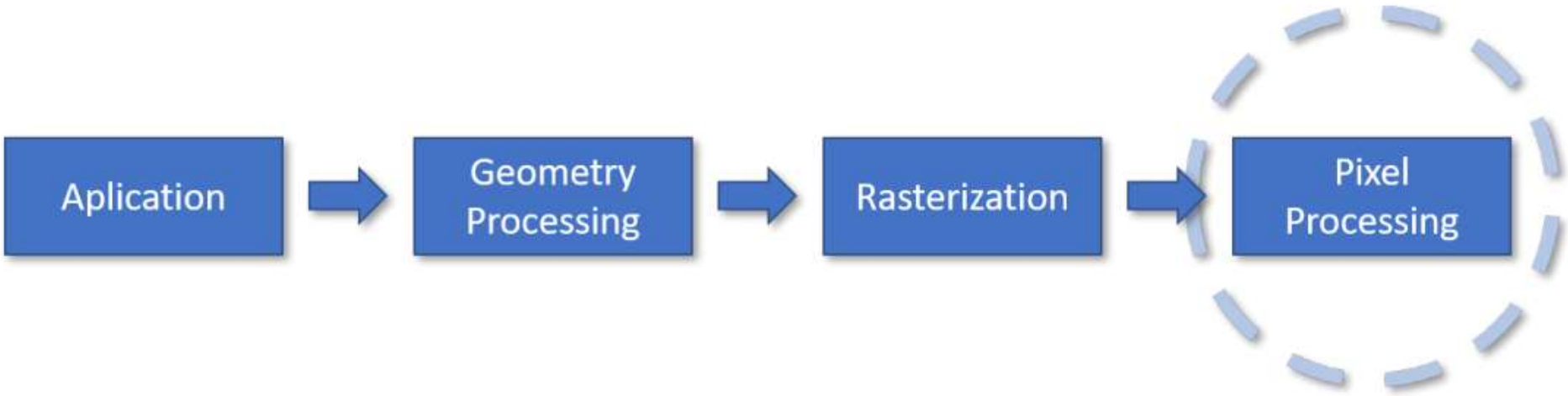
Texturas

Dr. Ivan Sipiran

Graphics Rendering Pipeline



Pixel Processing

- Aquí se ejecutan las operaciones por cada píxel dentro de las primitivas
- Dos sub-etapas funcionales
  - Pixel Shading
  - Merging

Pixel Processing

Pixel Shading

- Usando los valores de data interpolada como entrada, se genera el color asociado al píxel.
- Ejecutado por núcleos programables de la GPU
- Dicho programa es llamado Fragment Shader
- Diferentes técnicas, la más común es texturado

Merging

- La información de cada píxel es almacenada en el color buffer , que es un arreglo rectangular de colores.
- Se mezclan los distintos colores de fragmentos, resolviendo la visibilidad: algoritmo z-buffer
- También se conoce como Raster Operations Pipeline
- No es completamente programable, pero si configurable.

Texture Mapping

Problema

- Se quiere añadir más detalle a una superficie
- Dos formas:

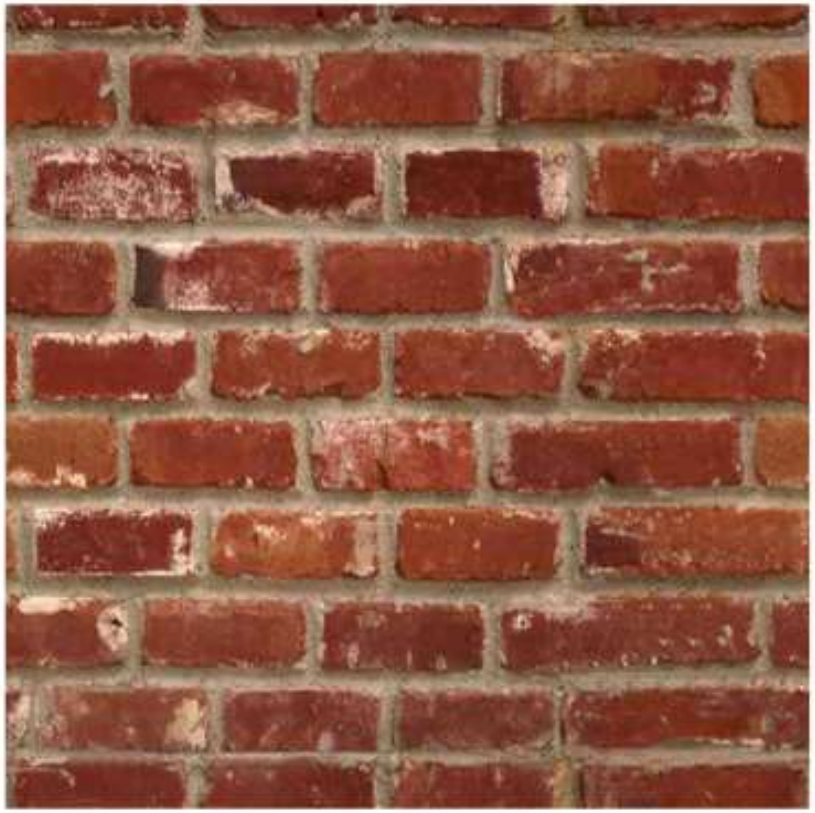
Más polígonos:

- Ralentiza la velocidad de rendering
- Es difícil de modelar detalles

Texturas:

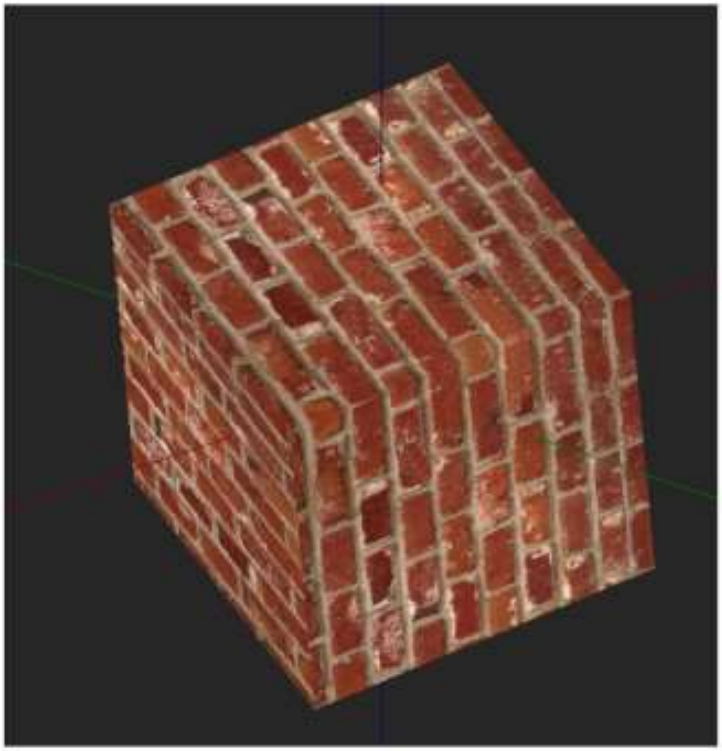
- La complejidad de la textura no afecta la complejidad de procesamiento
- Soporte eficiente en hardware

Texture mapping



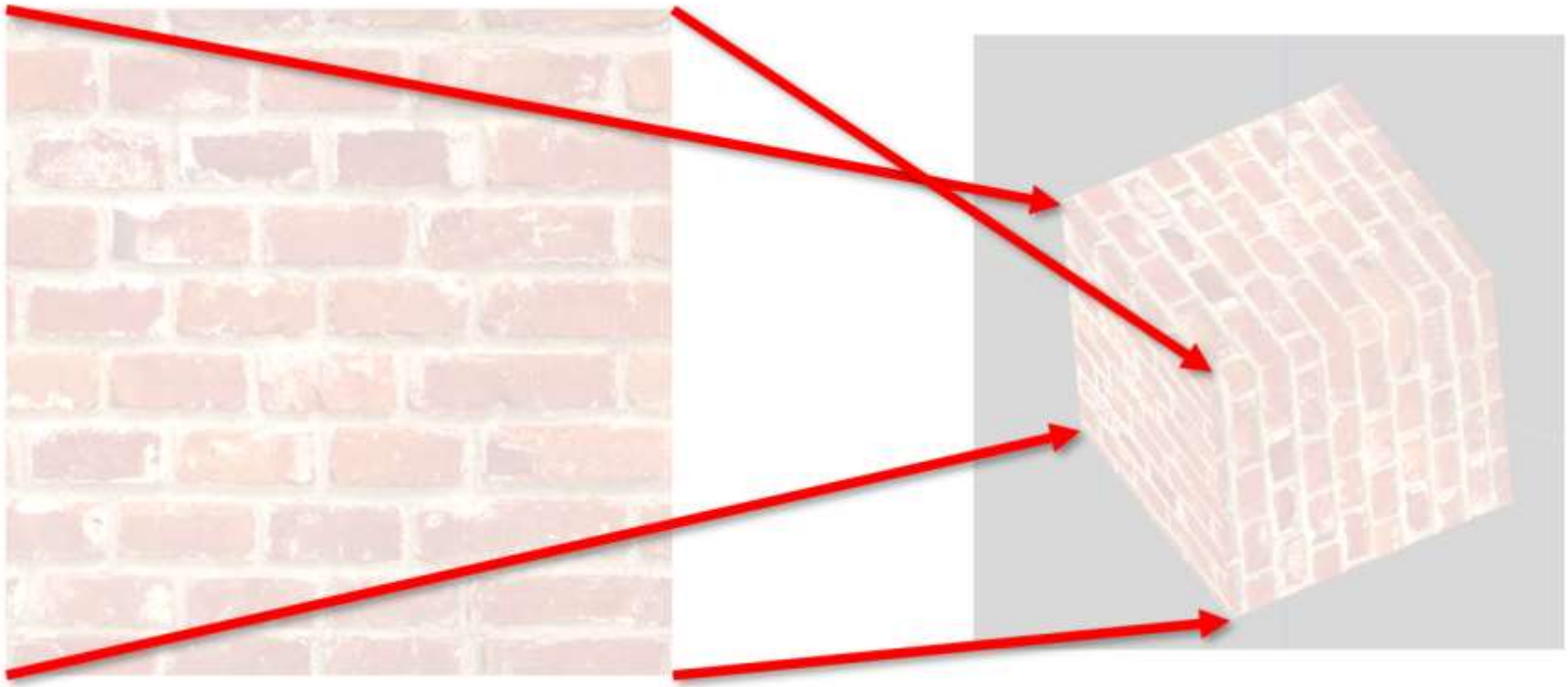
Textura

?

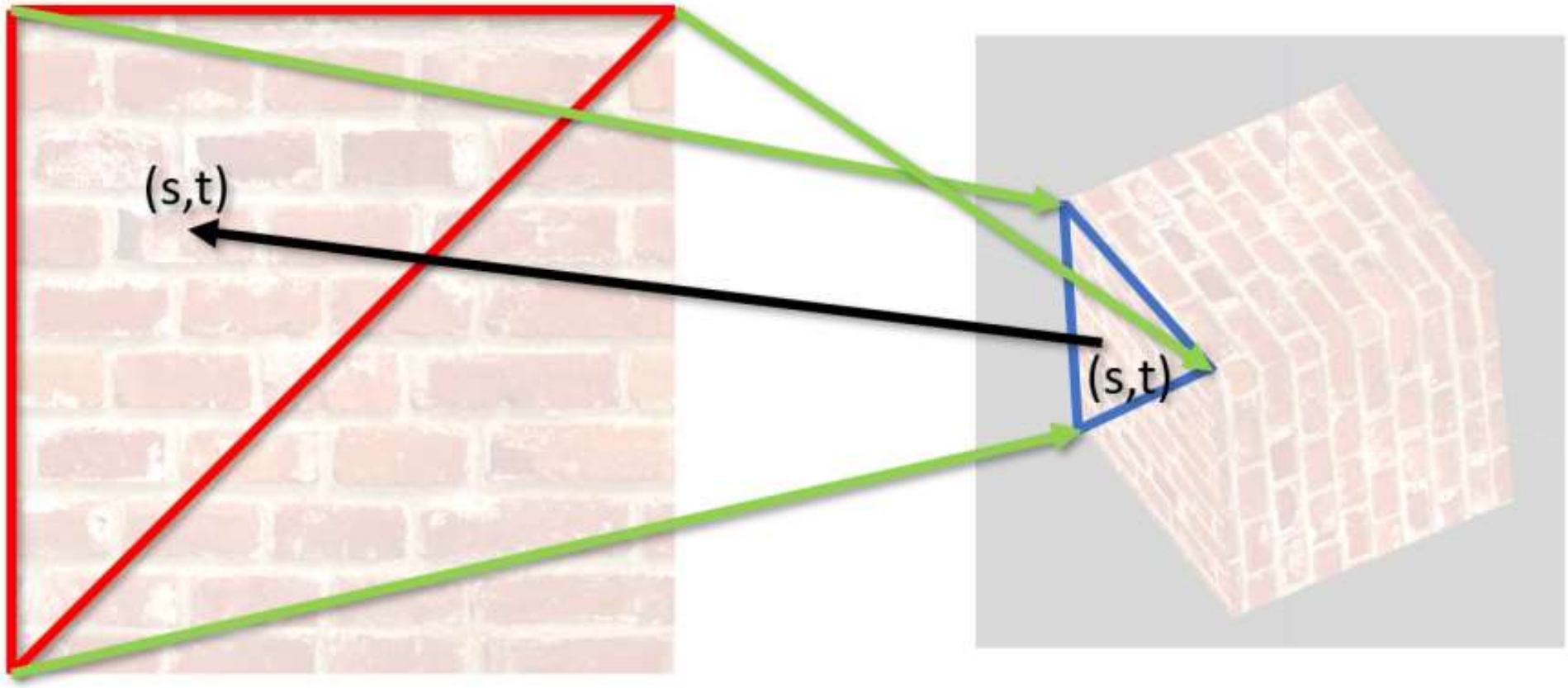


Polígonos en 3D

Texture mapping

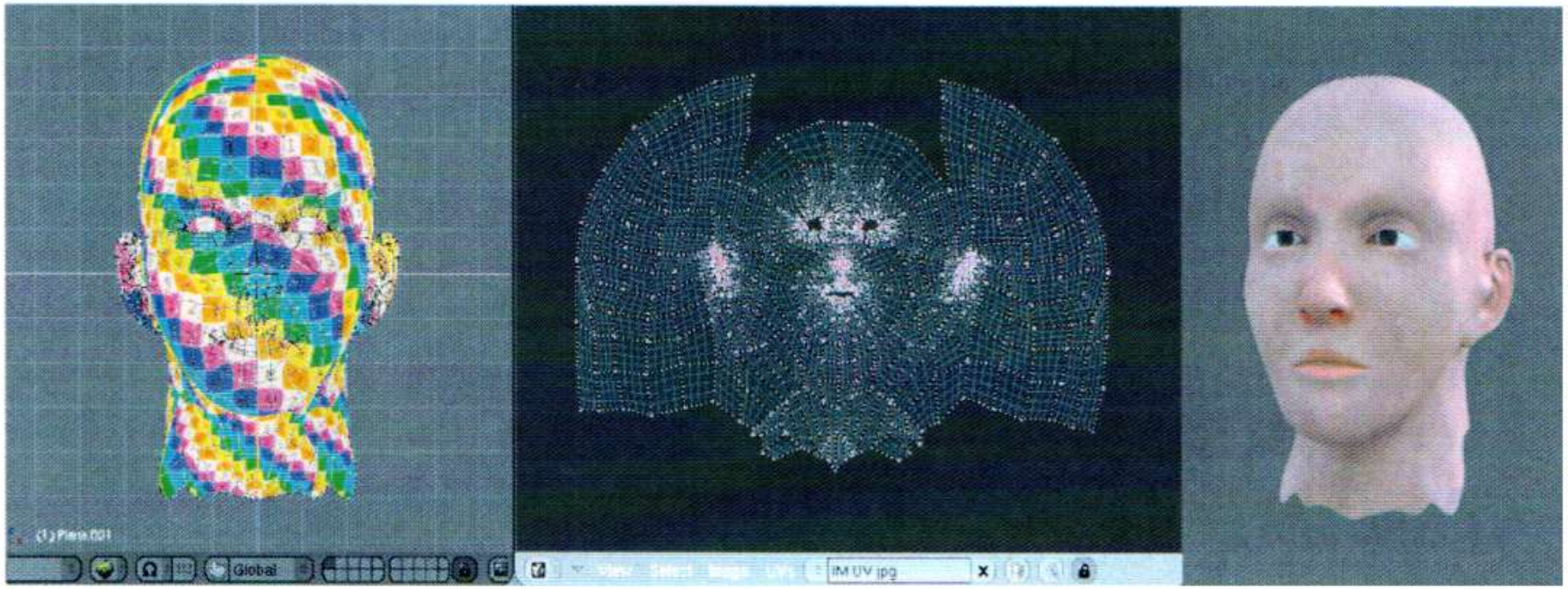


Texture mapping



Parametrización de texturas y modelos

- Para un modelo general, compuesto de varios polígonos simples, se debe establecer una parametrización



Texture mapping

- Una textura es una imagen
  - Se necesita una librería externa para cargar la imagen en memoria
  - Nota: En Python podemos usar Pillow
  - El mismo programa puede generar una imagen
  - La textura es enviada al GPU para ser controlada por OpenGL
- Píxels en una textura se llaman texels
- Cada textura tiene su propio espacio de coordenadas llamado texel coordinates, el cual escala la imagen al rango [0,1]
- Este espacio también es llamado UV o ST

Texturas en OpenGL

- Coordenadas de texturas se escriben como atributos de cada vértice
- Valores intermedios son interpolados
- Podemos reemplazar color por coordenadas de texturas
- Luego
  - Vertex Shader: Simplemente transfiere las coordenadas de textura
  - Fragment Shader: Recibe las coordenadas de textura interpoladas
- En vez de interpolar color, se interpola la textura

```
vertices = [
#   positions          texCoords
    -0.5, -0.5, 0.0,   0, 0,
     0.5, -0.5, 0.0,   1, 0,
     0.5,  0.5, 0.0,   1, 1,
    -0.5,  0.5, 0.0,   0, 1]

indices = [
     0, 1, 2,
     2, 3, 0]
```

Texturas en OpenGL

- Vertex Shader

```
uniform mat4 transform;

in vec3 position;
in vec2 texCoords;

out vec2 outTexCoords;

void main()
{
    gl_Position = transform * vec4(position, 1.0f);
    outTexCoords = texCoords;
}
```

Texturas en OpenGL

- Fragment Shader

```
in vec2 outTexCoords;

out vec4 outColor;

uniform sampler2D samplerTex; // interpolador!

void main()
{
    outColor = texture(samplerTex, outTexCoords);
}
```

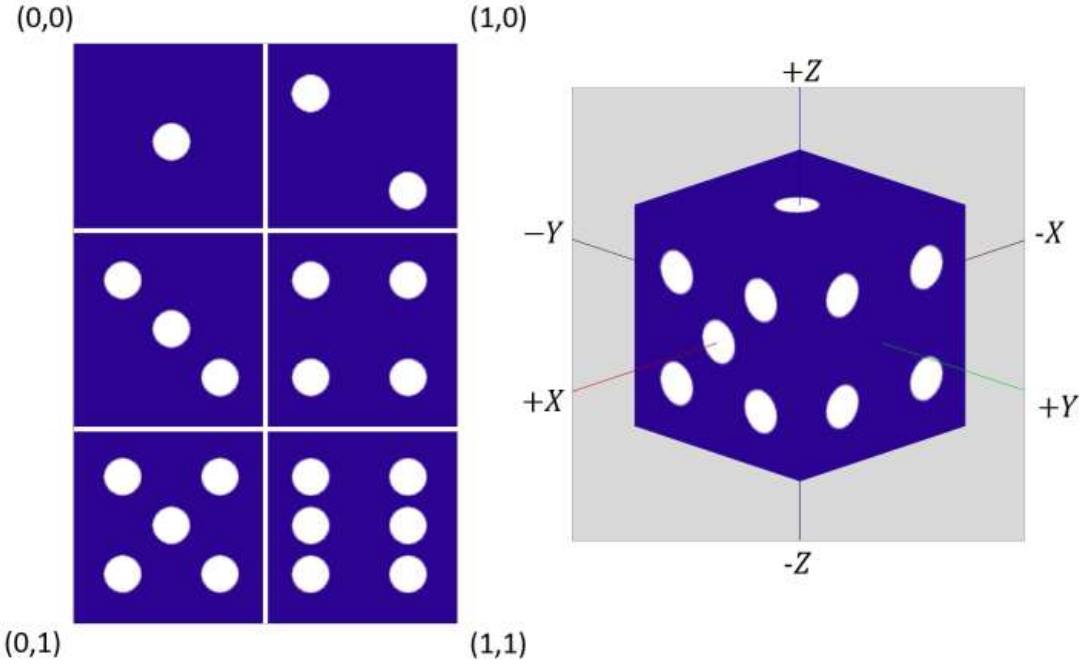
Texturas en OpenGL

- Cargado de textura

```
texture = glGenTextures(1)
glBindTexture(GL_TEXTURE_2D, texture)

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_NEAREST)
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR)

image = Image.open(imgName)
img_data = numpy.array(image, np.uint8)

if image.mode != "RGBA":
    print("Image mode not supported.")
    raise Exception()

glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA,
    image.size[0], image.size[1], 0, GL_RGBA, GL_UNSIGNED_BYTE, img_data)
```

Texturas en OpenGL

- Sistema de referencia imágenes

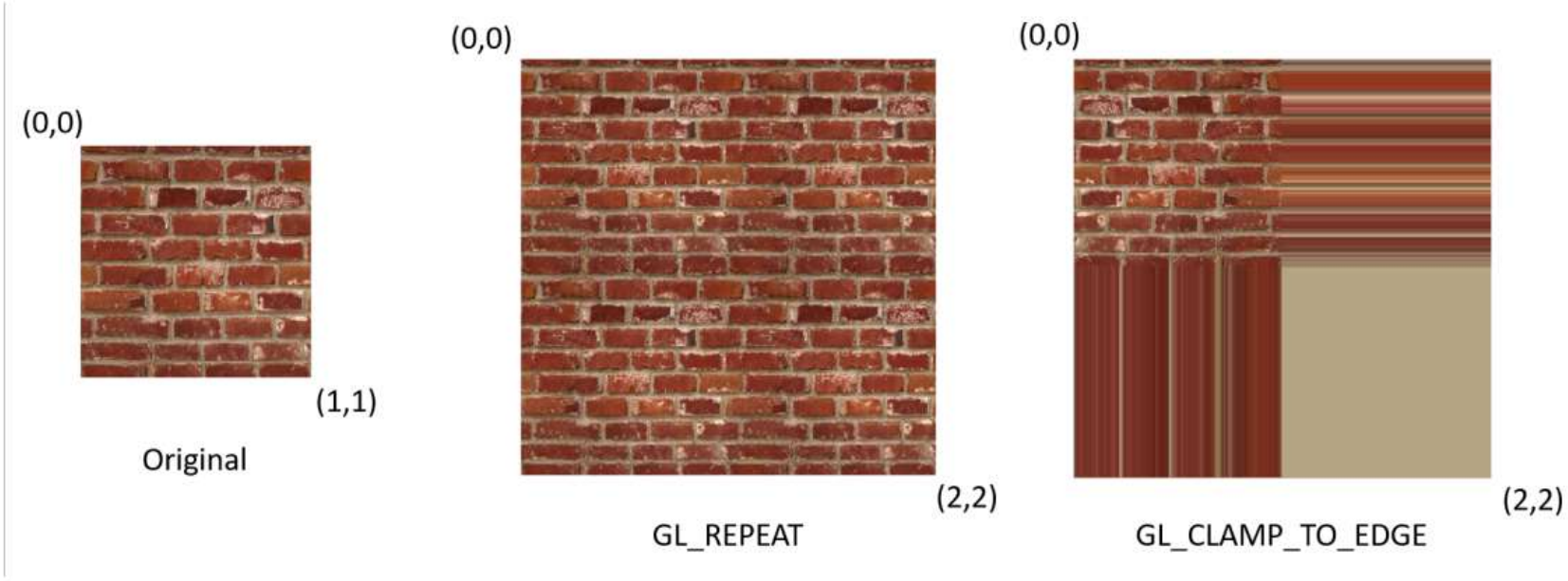


Número 4 queda definido por:

```
vertices = [
...
# positions        # tex coords
-0.5,  0.5, -0.5, 1/2, 2/3,
 0.5,  0.5, -0.5,   1, 2/3,
 0.5,  0.5,  0.5,   1, 1/3,
-0.5,  0.5,  0.5, 1/2, 1/3,
...
]
```

Texturas en OpenGL

- Qué sucede si usamos valores fuera del rango [0,1]?




```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT|GL_CLAMP_TO_EDGE)
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT|GL_CLAMP_TO_EDGE)
```

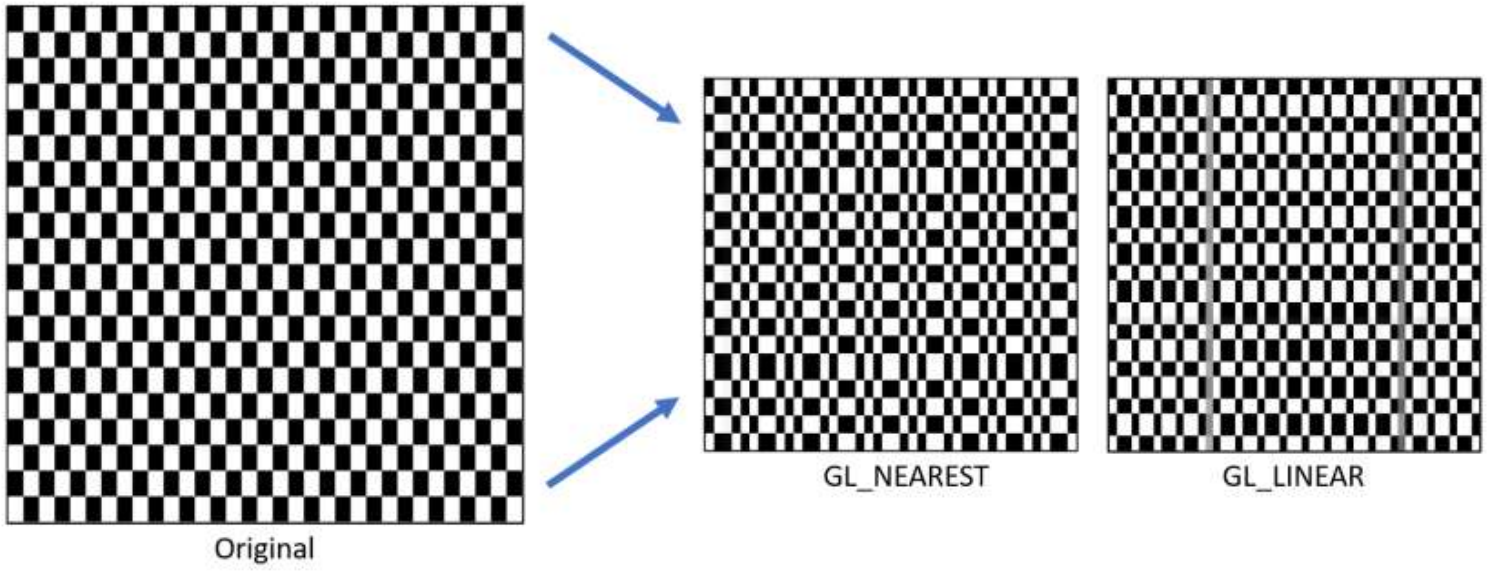
Texturas en OpenGL

- Modo de interpolación – Magnificación
  - Los colores consultados raramente se encontrarán en la textura, por lo que deben ser interpolados, en el caso en que la textura es más pequeña que los pixels que cubre, se habla de filtro de magnificación.


```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR|GL_NEAREST)
```

Texturas en OpenGL

- Modo de interpolación – Minimización
  - Si la textura es más grande que los píxeles que cubre, necesitamos una forma de asociar el color de todas formas. Aquí se habla de filtro de minimización

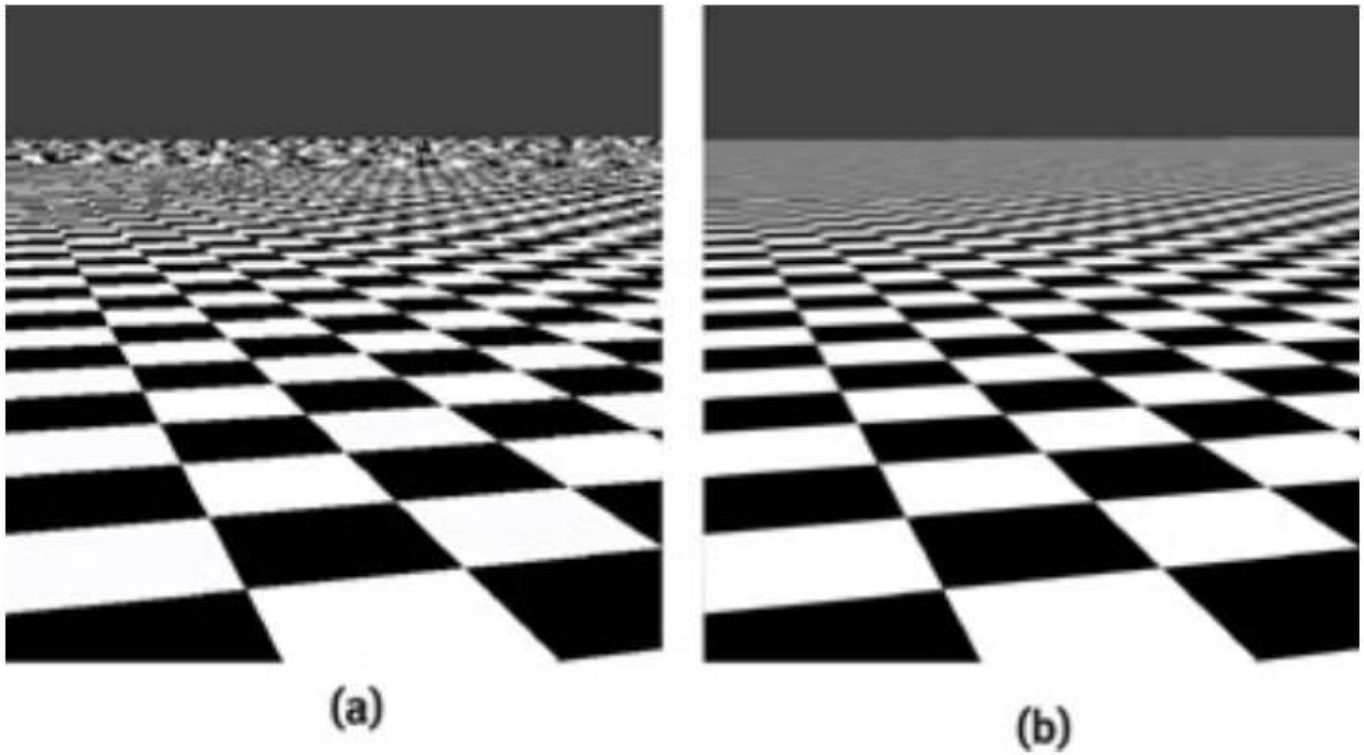



```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST)
```

Configuración del VAO

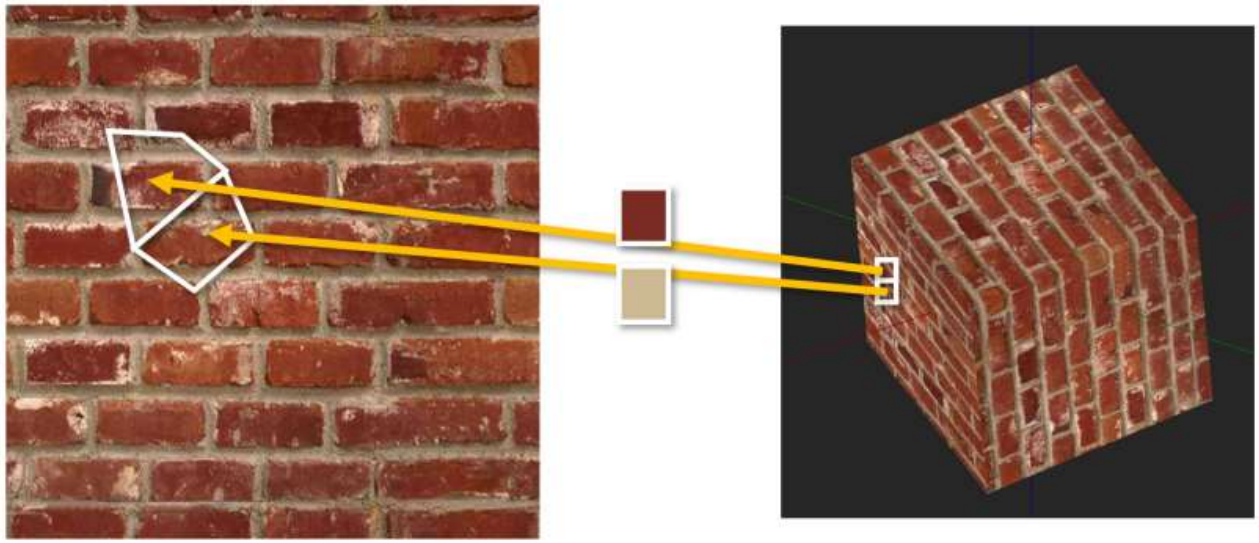
```
# Binding the proper buffers
glBindVertexArray(vao)
glBindBuffer(GL_ARRAY_BUFFER, vbo)
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, ebo)

# 3d vertices + 2d texture coordinates => 3*4 + 2*4 = 20 bytes
position = glGetAttribLocation(shaderProgram, "position")
glVertexAttribPointer(position, 3, GL_FLOAT, GL_FALSE, 20, ctypes.c_void_p(0))
glEnableVertexAttribArray(position)

texCoords = glGetAttribLocation(shaderProgram, "texCoords")
glVertexAttribPointer(texCoords, 2, GL_FLOAT, GL_FALSE, 20, ctypes.c_void_p(12))
glEnableVertexAttribArray(texCoords)
```

Problema!



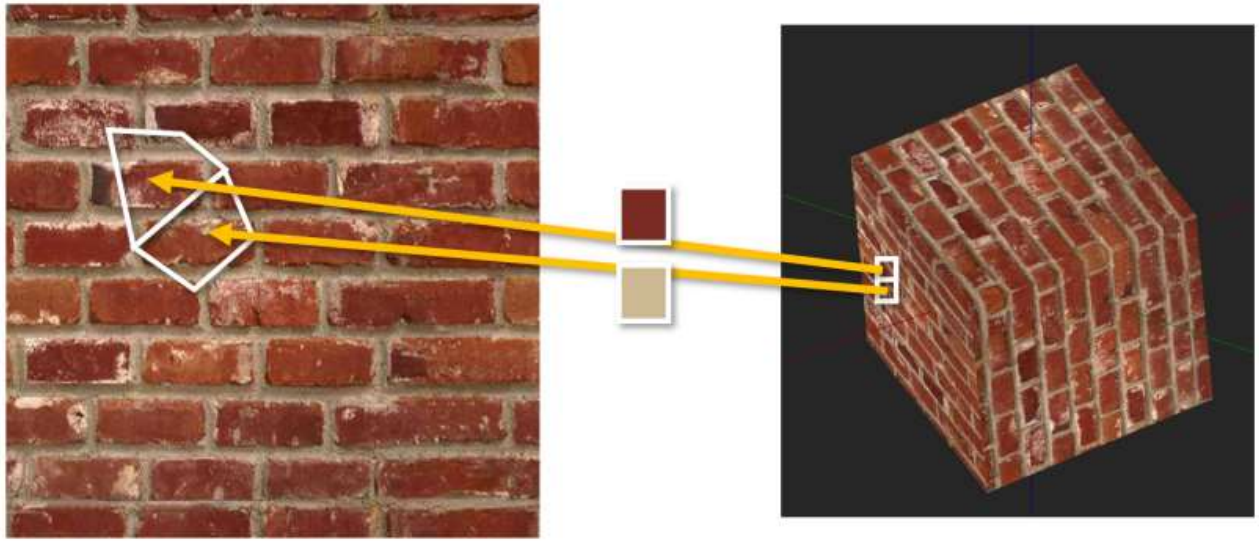
- Para patrones complejos, se genera aliasing.

Aliasing en texturas



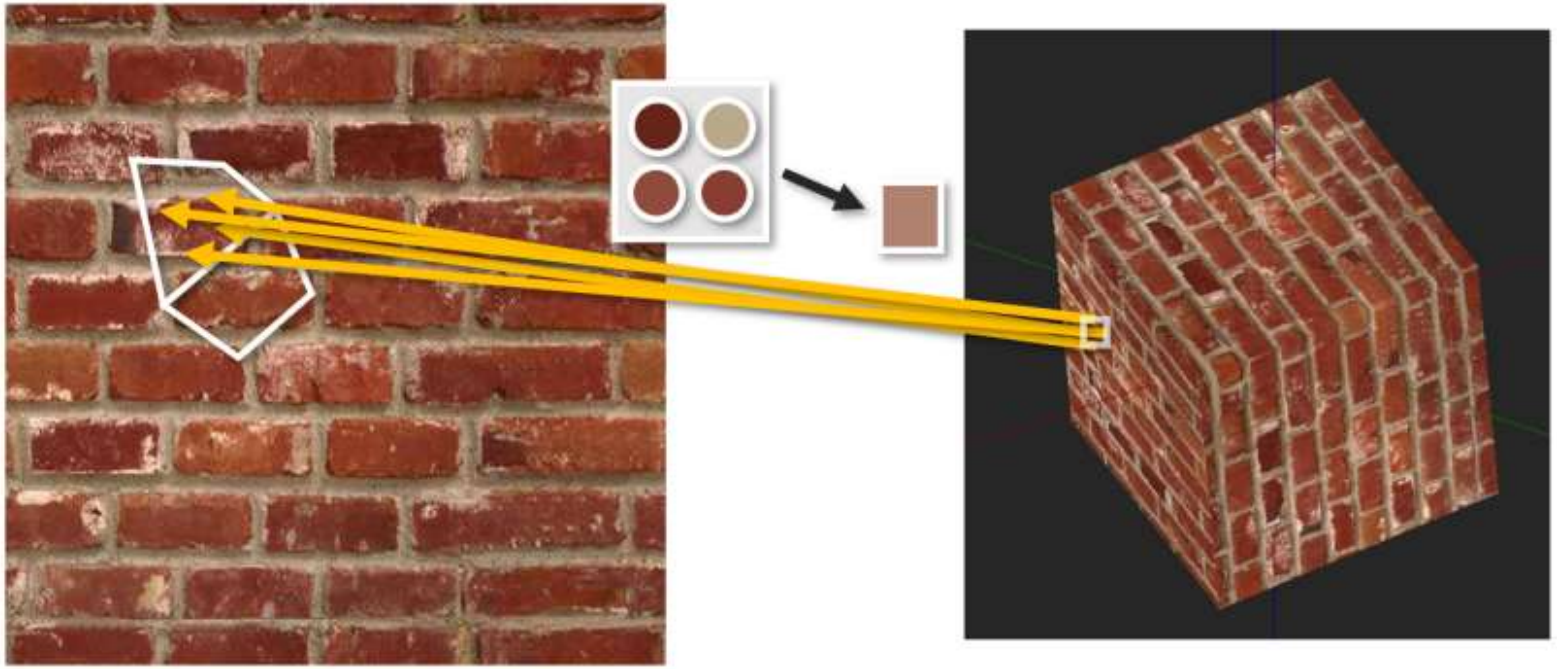
- Si objetos lejanos poseen texturas grandes, cada pixel tiene asociado muchos texels.
- Considerando un único texel por pixel, pixeles vecinos tendrán colores distintos
- Esta transición brusca es la raíz del problema del aliasing.

Aliasing en texturas



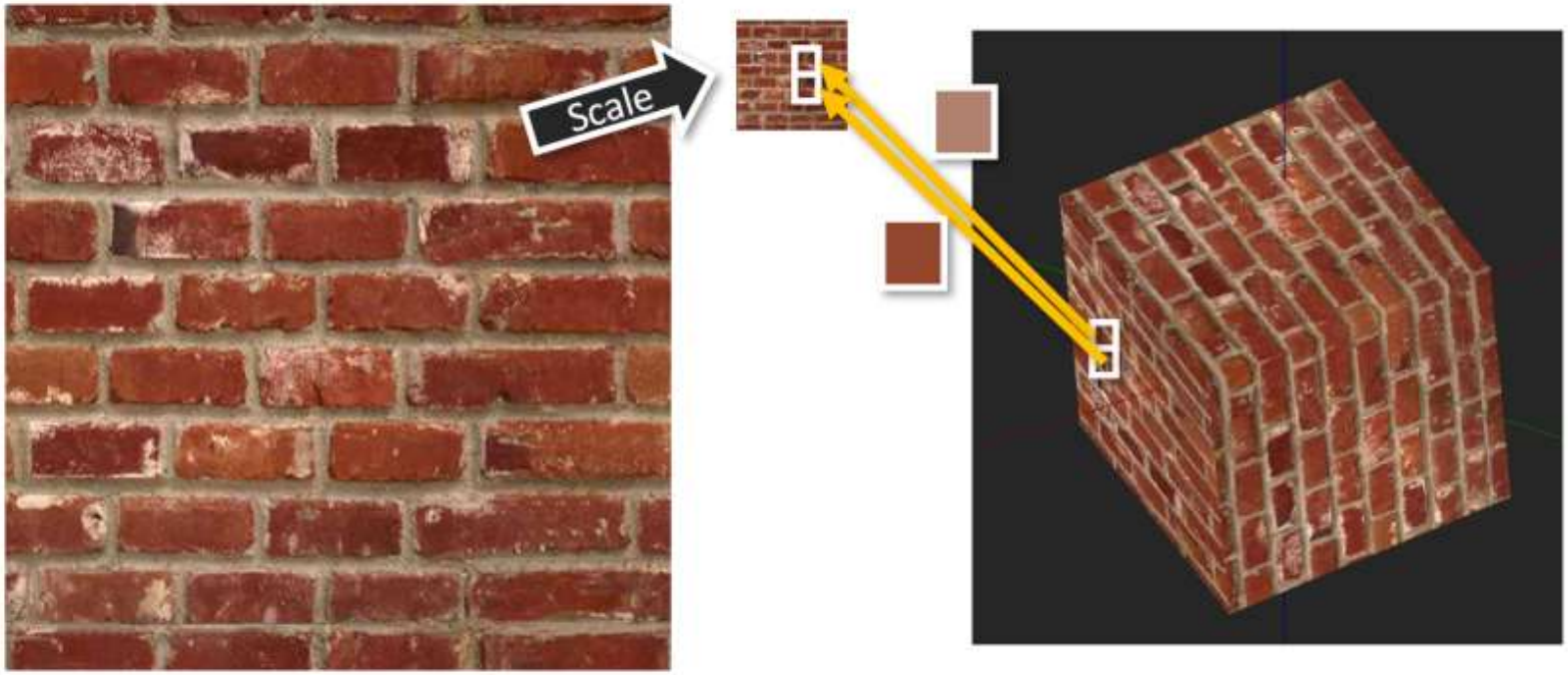
- Un remedio paliativo es considerar el color promedio en cada pixel.
- Por supuesto esta idea es costosa e inviable para tiempo real.

Aliasing en texturas



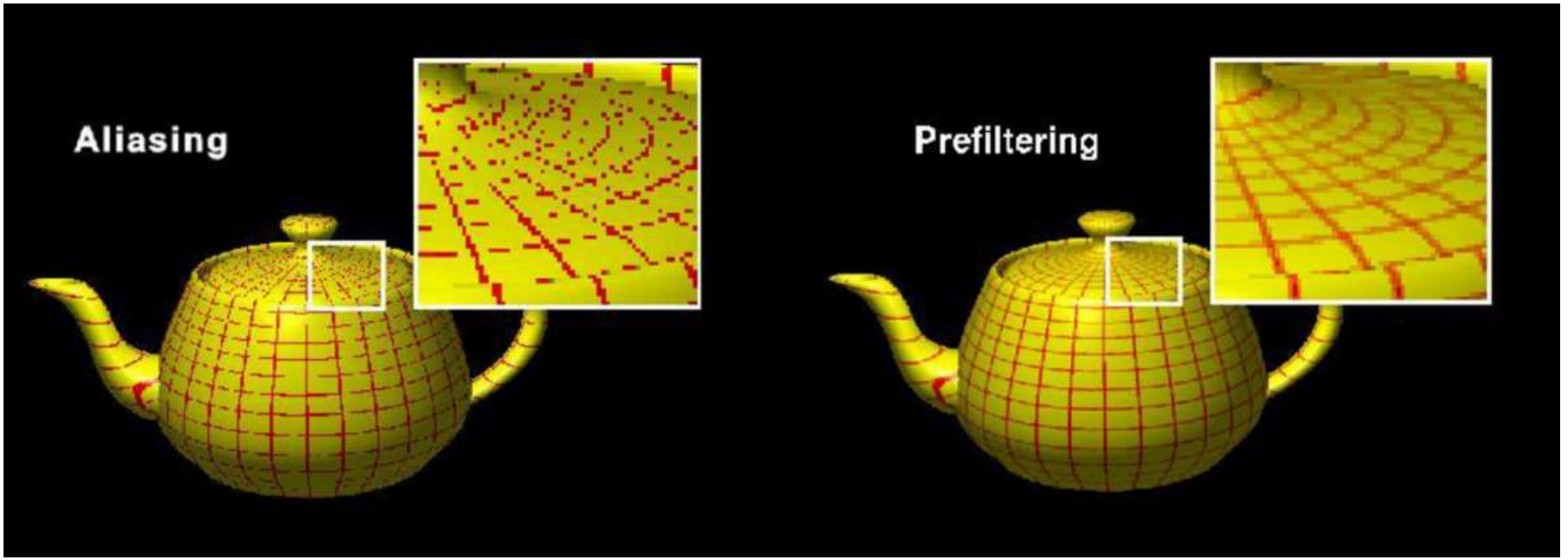
- En aproximaciones Monte-Carlo, podemos generar varias muestras asociadas a un píxel y con ellas generar una mejor aproximación del color promedio asociado al pixel.

Aliasing en texturas

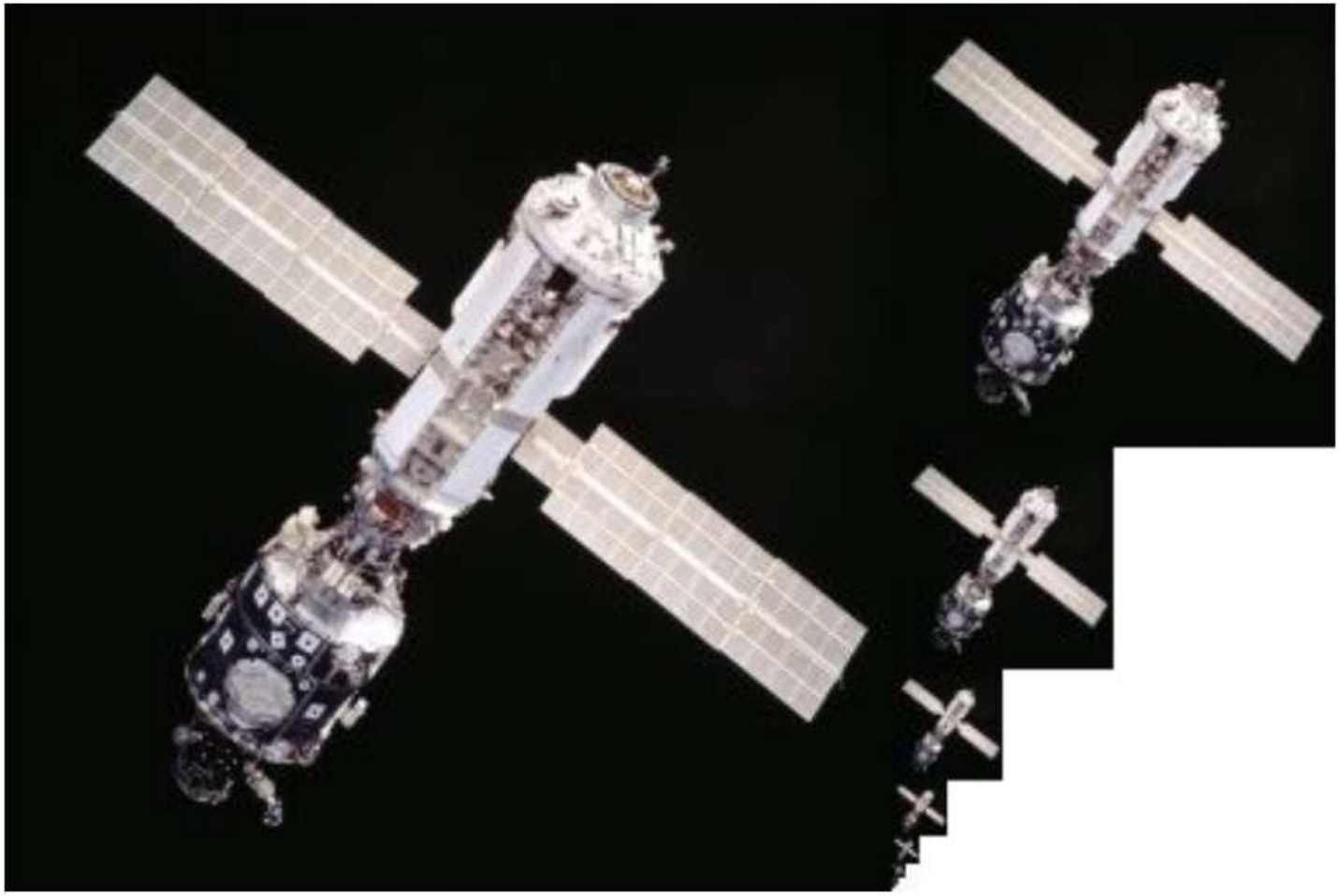


- Otra idea es precalcular imágenes promediadas mas pequeñas, de manera que cada píxel tenga pocos píxeles de cuales escoger, disminuyendo el problema del aliasing.

Aliasing en texturas



Mip-Map



Mip-maps en OpenGL

```
texture = glGenTextures(1)
glBindTexture(GL_TEXTURE_2D, texture)

# texture wrapping params
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE)
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE)

# texture filtering params
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_NEAREST)
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR)

glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, image.size[0], image.size[1],
    0, GL_RGB, GL_UNSIGNED_BYTE, img_data)
glGenerateMipmap(GL_TEXTURE_2D)
```

Mip-Maps en OpenGL

- GL_NEAREST_MIPMAP_NEAREST: Utiliza el mipmap mas cercano al tamaño del pixel y utiliza sobre él una interpolación del vecino más cercano.
- GL_LINEAR_MIPMAP_NEAREST: Utiliza el mipmap más cercano al tamaño del pixel y utiliza una interpolación lineal para escoger color.
- GL_NEAREST_MIPMAP_LINEAR: Interpola linealmente los dos mipmaps más cercanos al tamaño del pixel, e interpola este nivel usando el vecino más cercano.
- GL_LINEAR_MIPMAP_LINEAR: Interpola linealmente los dos mipmaps más cercanos al tamaño del pixel e interpola linealmente este nuevo nivel para determinar el color del pixel.