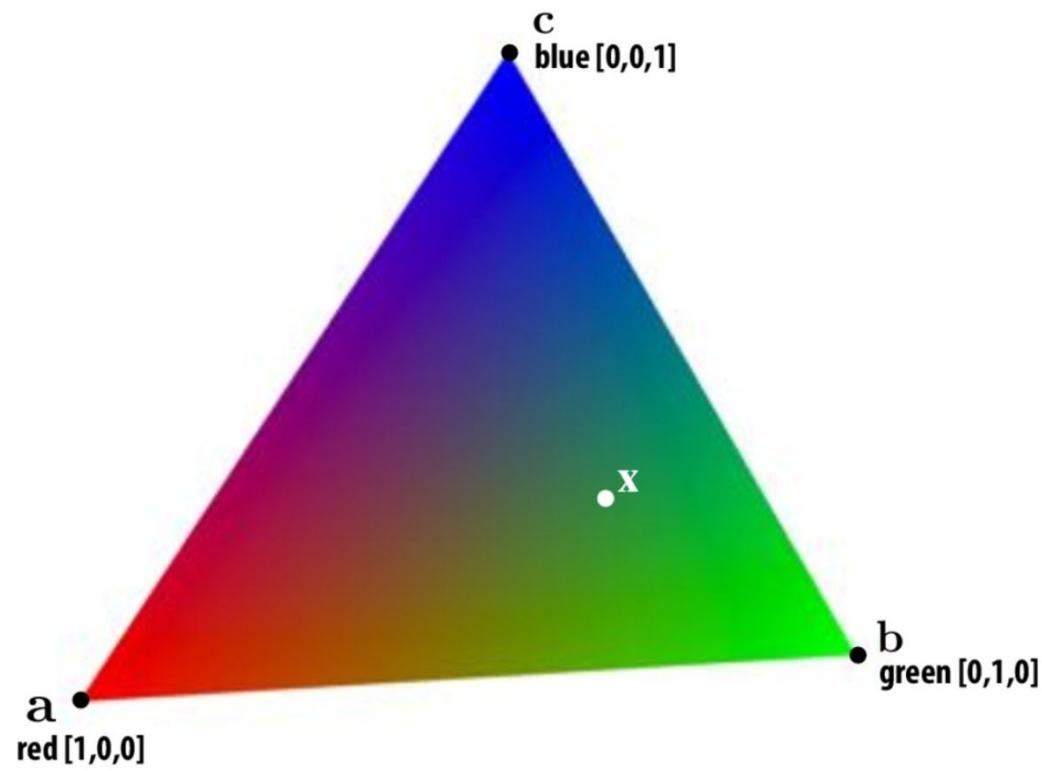


CC3501

Texturas

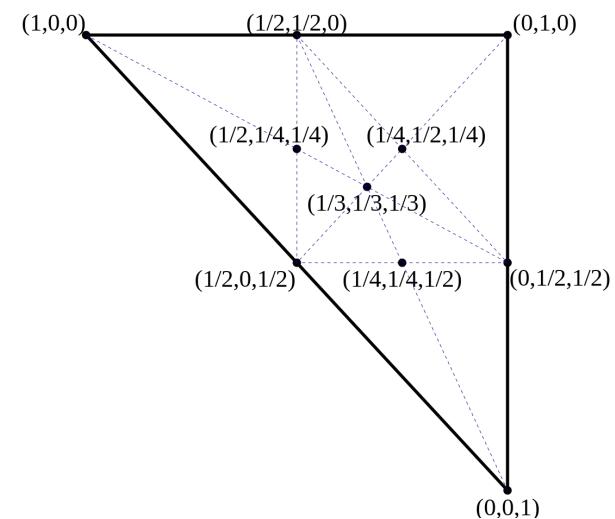
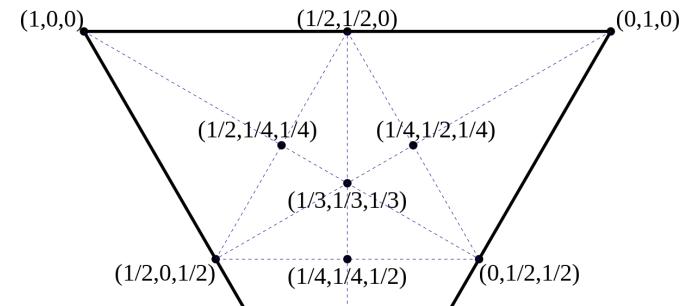
Eduardo Graells-Garrido
Otoño 2023



¿Cómo se **calcula** el color del punto x?

$$\text{color}(x) = \text{color}(x_i)\phi_i + \text{color}(x_j)\phi_j + \text{color}(x_k)\phi_k$$

Usando **coordenadas baricéntricas** podemos interpolar el color de los vértices al color del punto x (y por tanto, a los píxeles correspondientes).



¿Cómo graficar una superficie detallada?



¿Cómo hacer que la pelota tenga esa apariencia?

¿Cómo hacer que la madera tenga esos patrones típicos?

Mejorar resolución

¡Podemos usar modelos 3D con más triángulos! Sin embargo, **agregar más triángulos no necesariamente mejora la calidad del modelo** (ver imagen de ejemplo). El trabajo artístico requerido también es costoso. Y, sobre todo, **ralentiza** el rendering.



60
triangles



600
triangles



6000
triangles



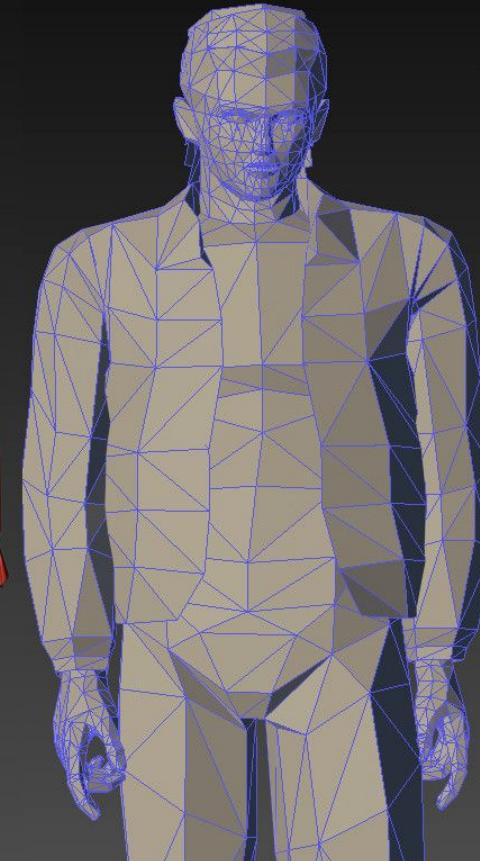
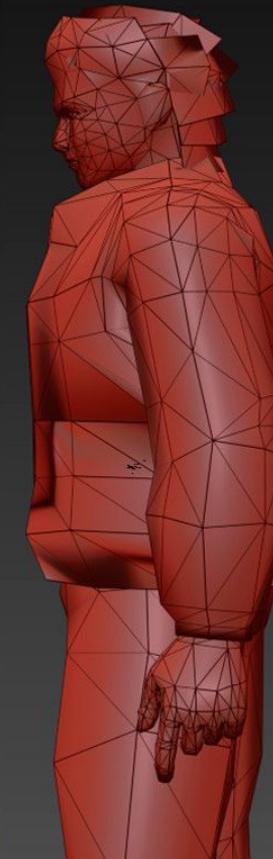
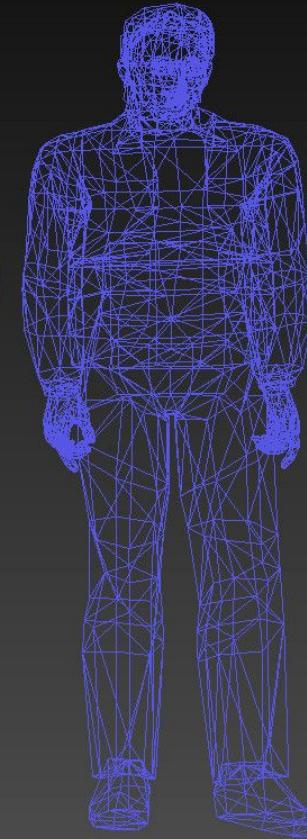
60000
triangles

Diminishing returns - 15 years ago even doubling the amount of triangles resulted in a much better mesh. Now, multiplying the amount by 10 hardly does.

[+] [Perspective] [Realistic]

Total

Polys: 3.013



Utilizando **texturas** es posible graficar detalles en las superficies sin incrementar la cantidad de polígonos. (¿Quién es este personaje?)

¿Qué significa *textura*?

[**https://es.wiktionary.org/wiki/textura**](https://es.wiktionary.org/wiki/textura)

1

Propiedades visuales y en especial táctiles de una superficie, así como las sensaciones que producen.

2

Estructura característica del tejido de los hilos que forman una tela.

3

Estructura física característica de un objeto o un material, dado por el tamaño, forma u organización de las partes que lo componen.

4

Sensación de la tela basada en la pintura y su método de aplicación.

Texture Mapping

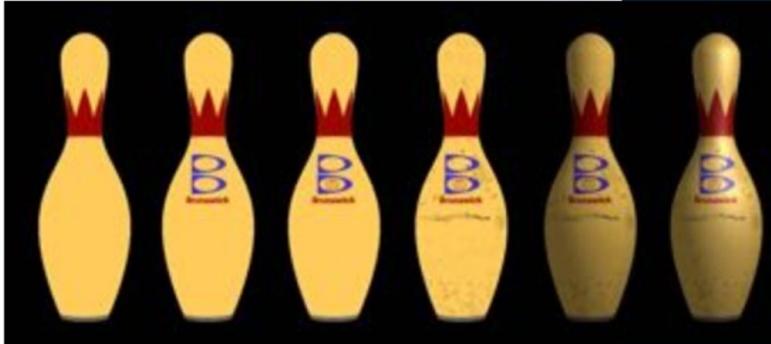
¿Por qué usar texturas?

¡Soporte eficiente en hardware!

La complejidad de la textura (imagen) no afecta la complejidad de procesamiento.

Hoy nos enfocaremos en el uso de texturas para apariencia visual.





Multiple layers of texture maps for color, logos, scratches, etc.



La textura puede contener información visual (color, logos, etc) y (**spoiler**) también información sobre la superficie.

normal mapping

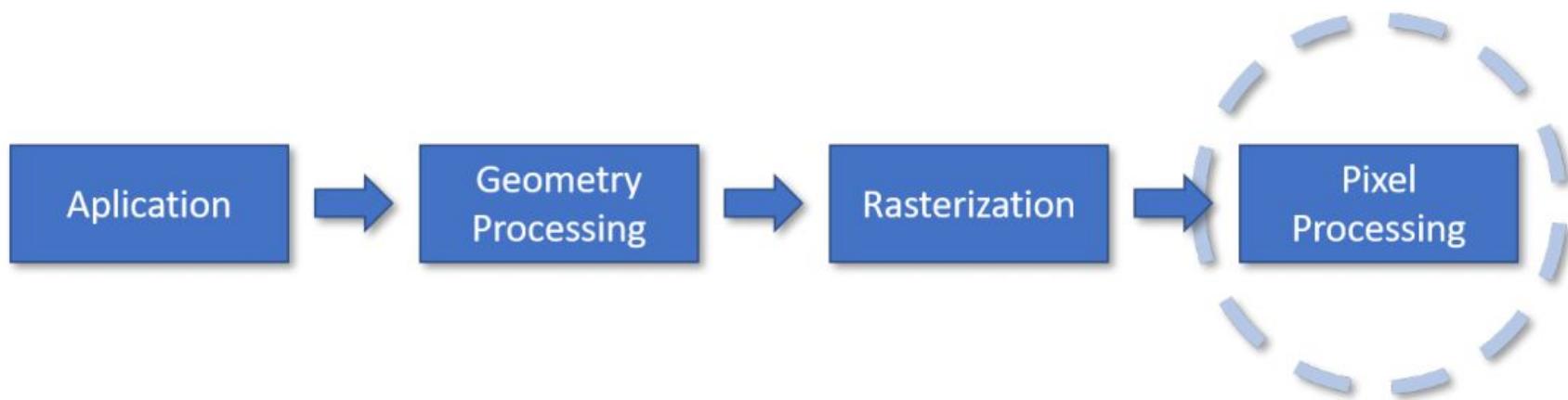


displacement mapping



Spoiler: más adelante (en unidad de iluminación) trabajaremos con las normales de la superficie. Podemos usar texturas como **mapas de normales** (normal maps).

¿Dónde se aplican las texturas?



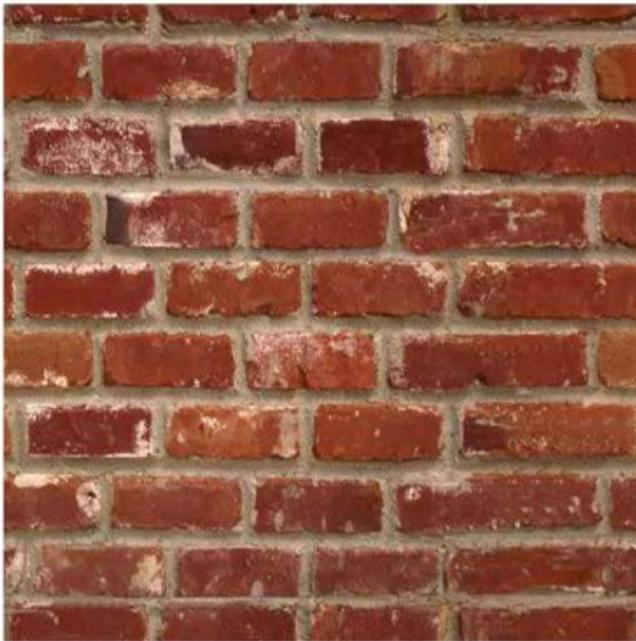
Pixel Shading

Usando los valores de **data interpolada** como entrada, se genera el **color asociado** al píxel.

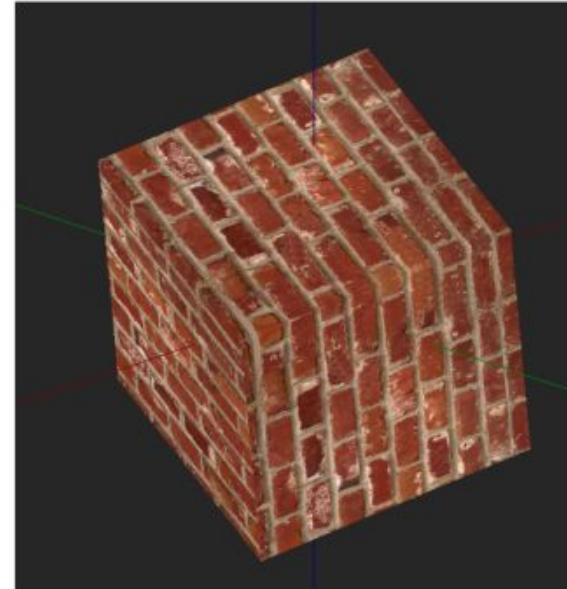
Ejecutado por núcleos programables de la GPU: el programa es llamado **Fragment Shader**

Se aplican múltiples técnicas para determinar el color, la más común es **texturado**.

Texture Mapping

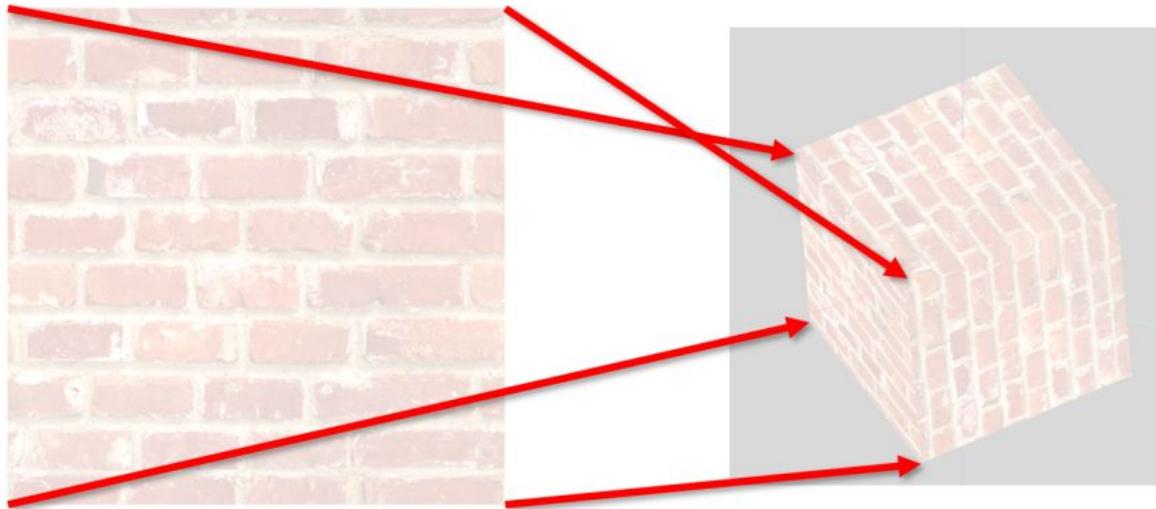


Textura

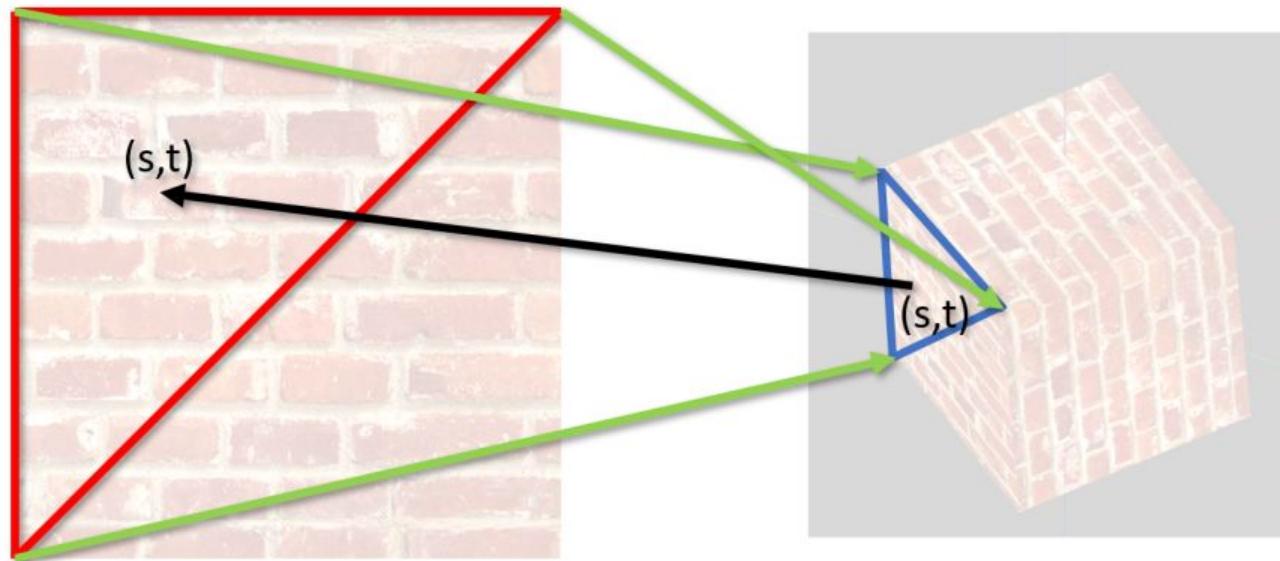


Polígonos en 3D

Texture mapping

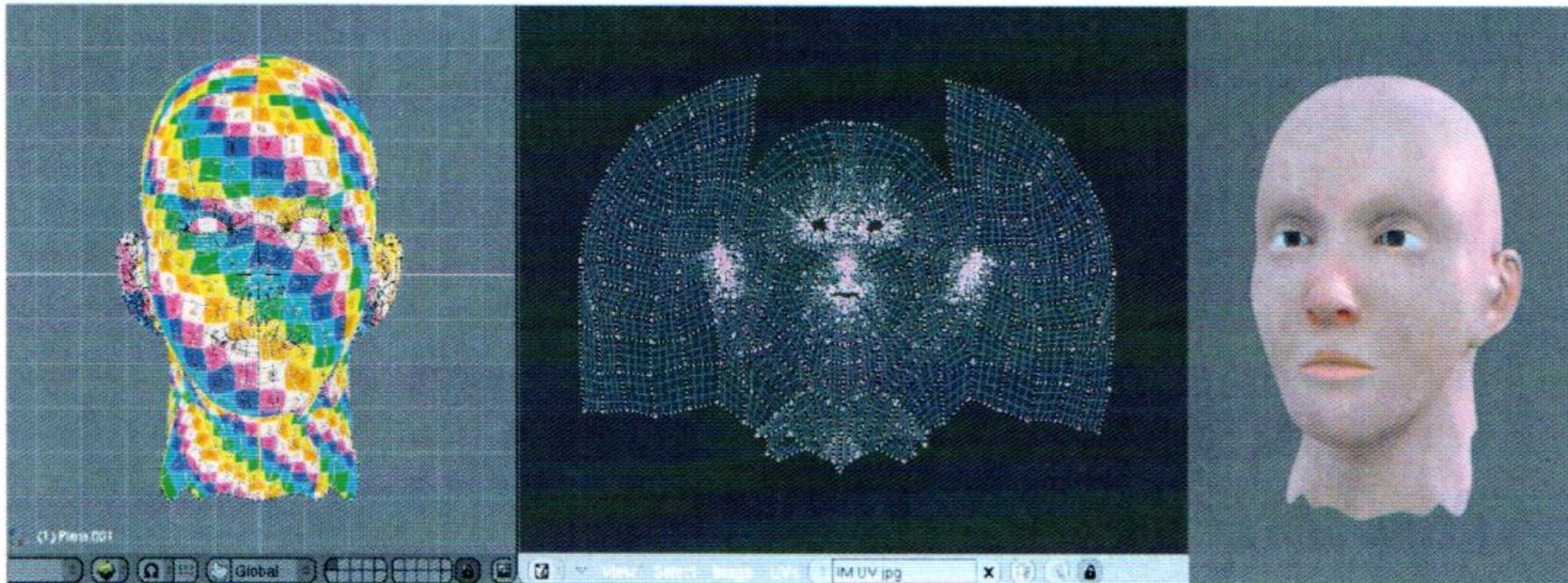


Texture mapping



Parametrización de texturas y modelos

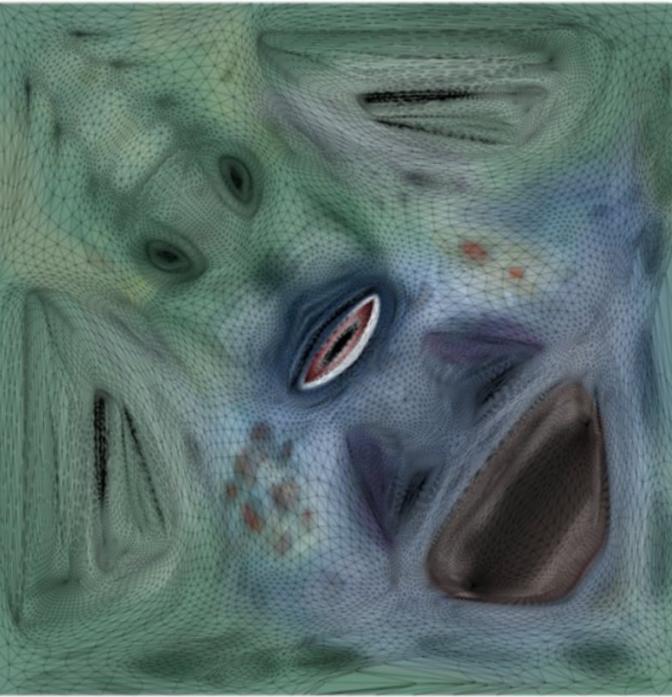
Para un modelo general, compuesto de varios polígonos simples, se debe establecer una parametrización



Rendered result



Triangle vertices in texture space



La parametrización puede ocupar todo el espacio disponible, o no. Podría ser una única malla, o bien, puede tener múltiples partes. Hay flexibilidad.

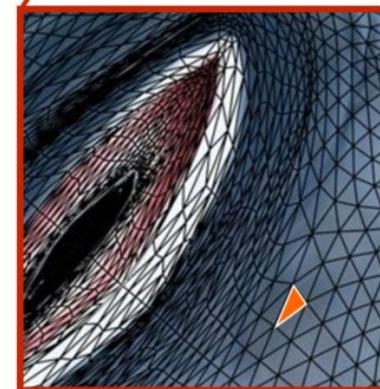
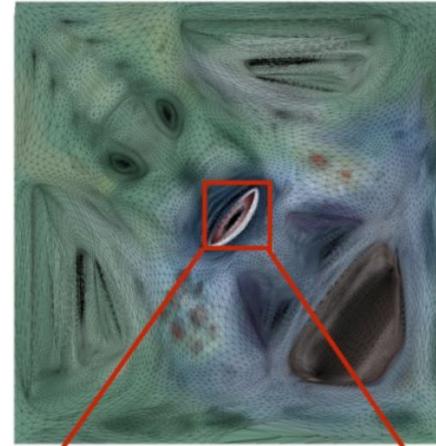
rendering without texture



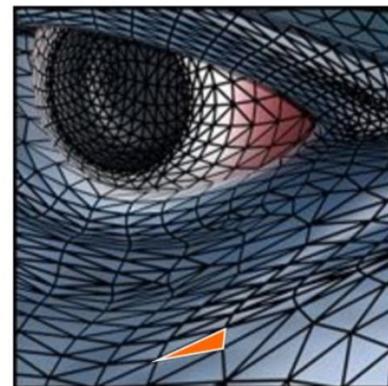
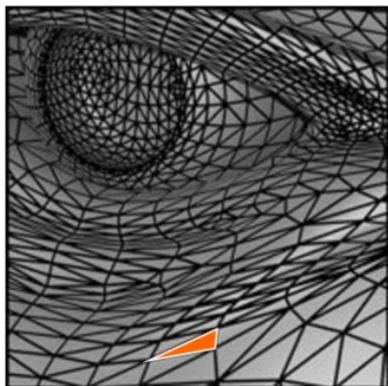
rendering with texture



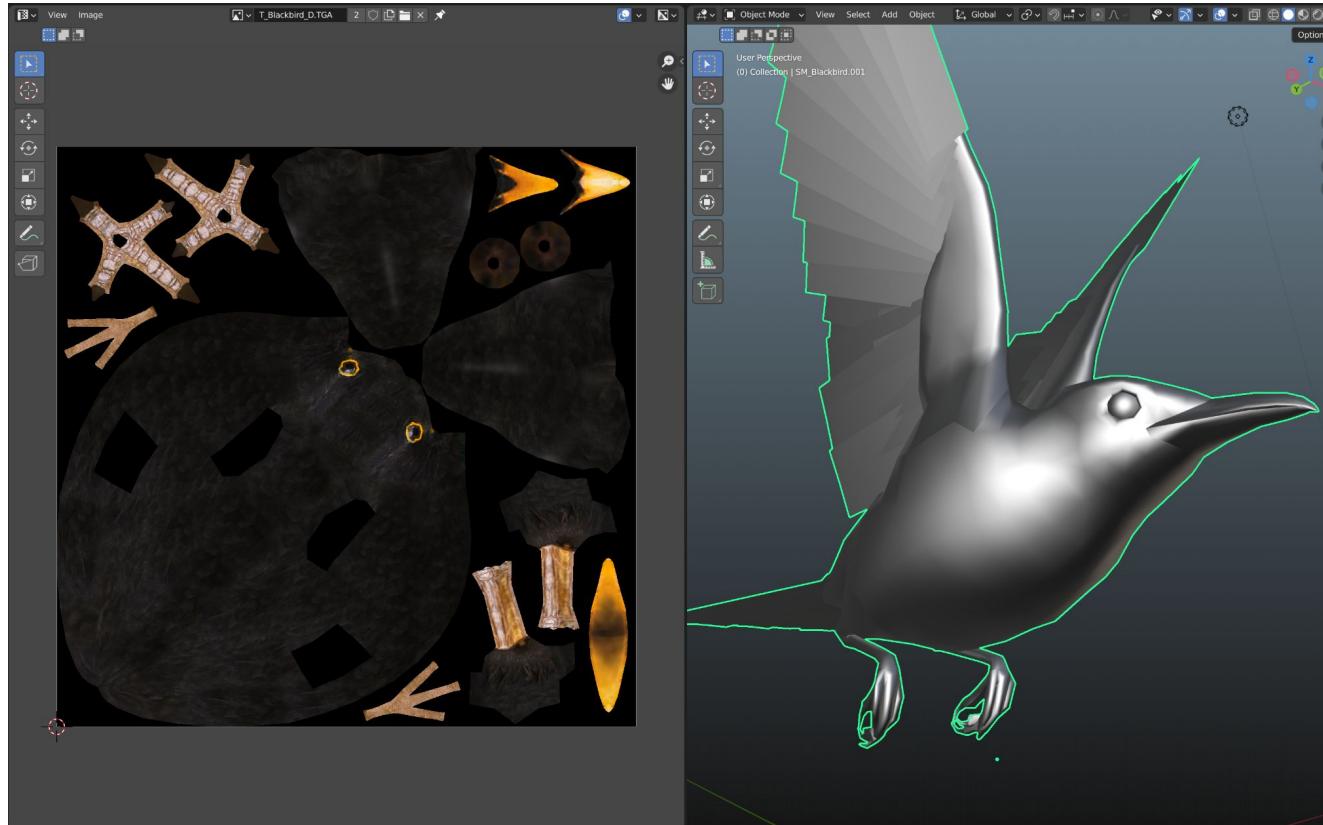
texture image



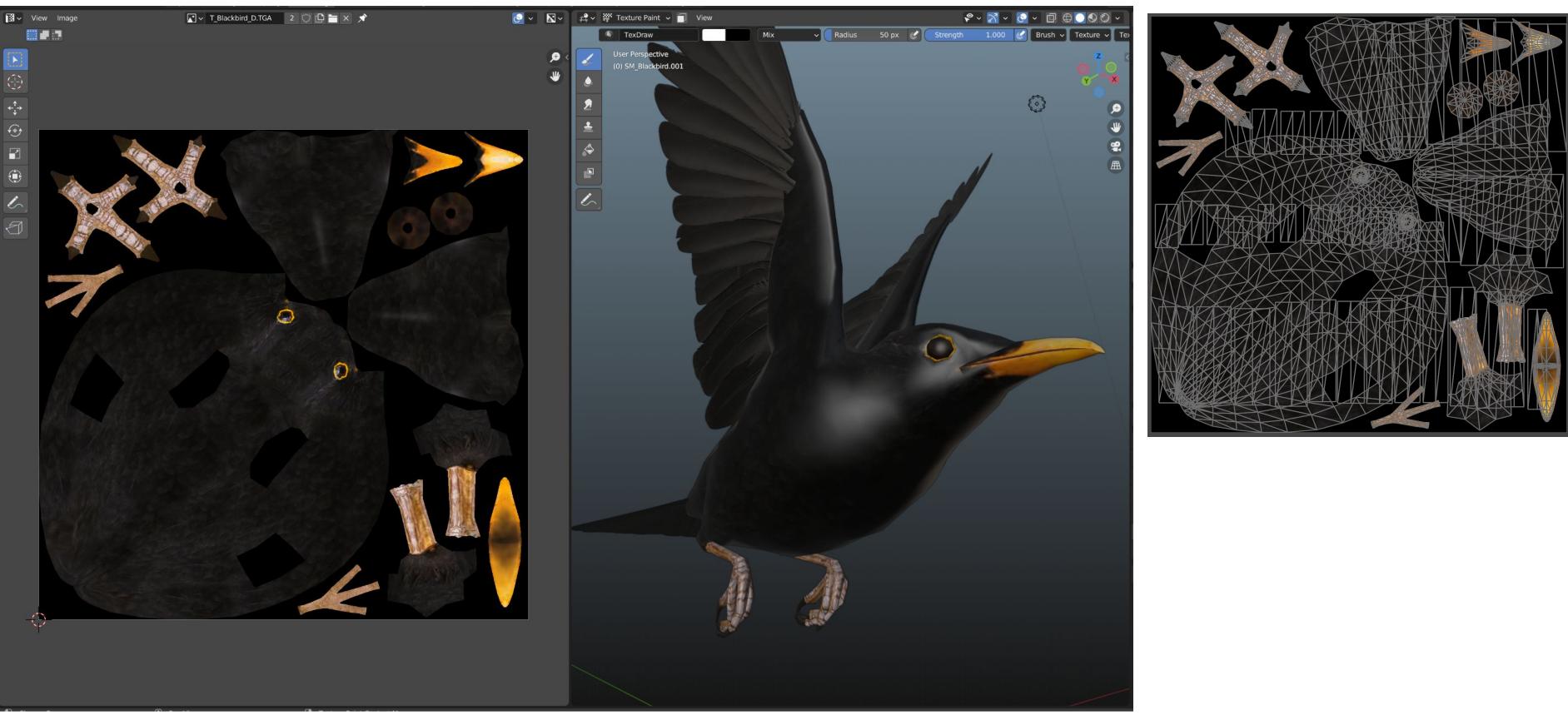
zoom



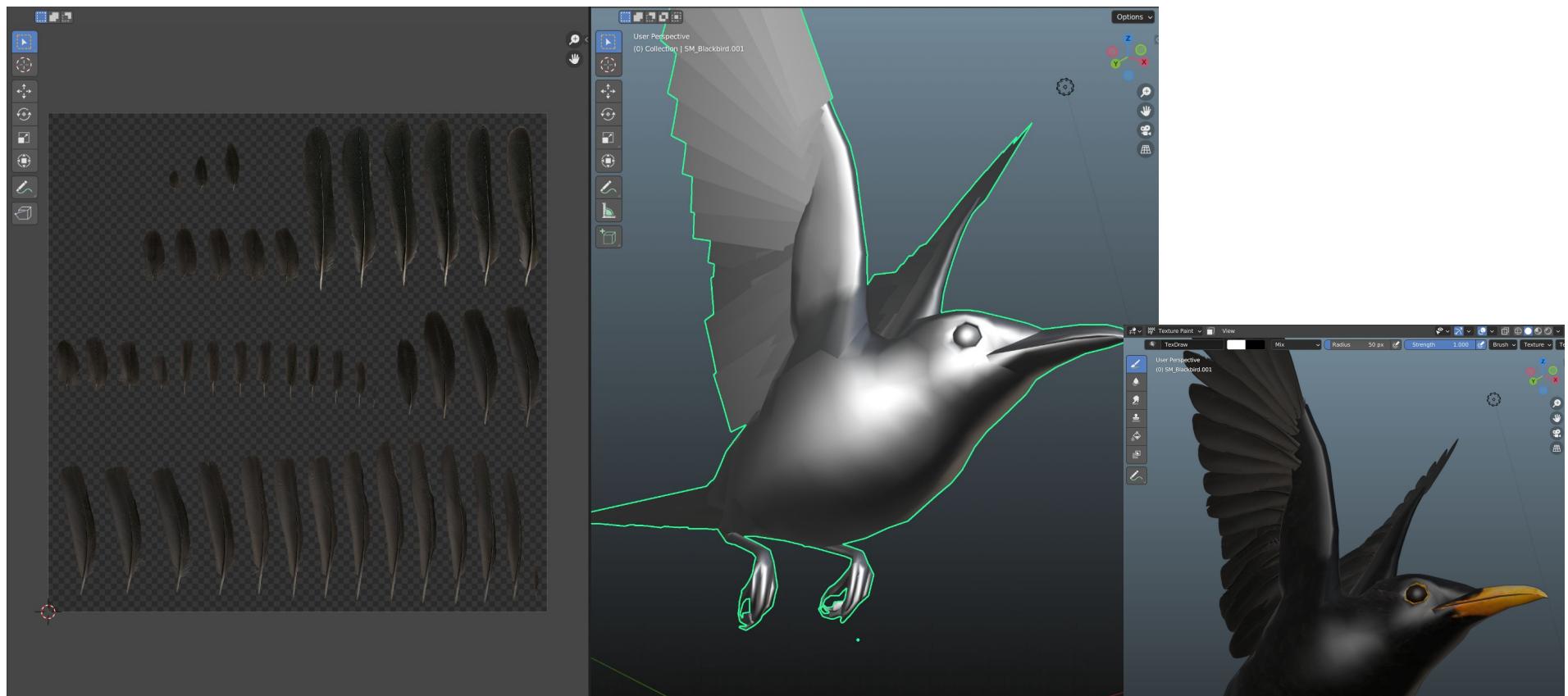
Cada vértice (y por tanto, cada triángulo) debe estar presente en el espacio de la textura.



Esta es una textura para el pajarito y este es el modelo del pajarito.



Así se ve el pajarito con la textura aplicada. ¿Hay algo que les llame la atención?



¡Había una segunda textura! En la parametrización se veían los polígonos de las plumas, pero estas no eran visibles en la textura principal.

Texture mapping

Como una textura es una imagen, entonces:

- Se necesita una librería externa para cargar la imagen en memoria. En Python podemos usar **Pillow** si es manualmente, o bien, **trimesh** se encarga del proceso si leen un modelo 3D.
- Una imagen también se puede generar de manera programática.
- En cualquier caso, la imagen es enviada a la GPU.
- La escena graficada es una imagen, por tanto, **puede ser utilizada como textura**.

Los pixels 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

```
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

Las coordenadas de texturas se escriben como atributos de cada vértice.

En nuestra estructura, podemos reemplazar color por coordenadas de texturas, o bien, agregarlas (ojo con el tamaño de los arrays).

Vertex Shader: simplemente transfiere las coordenadas de textura hacia la siguiente etapa (excepto casos avanzados).

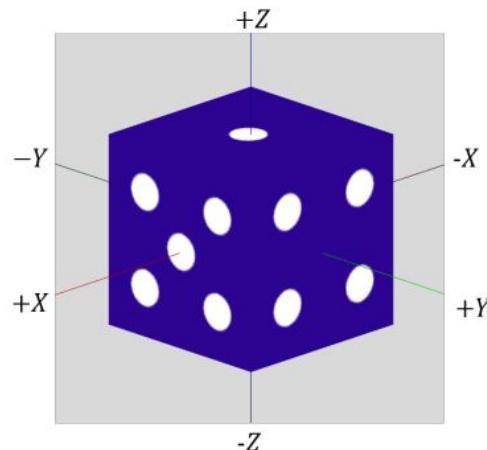
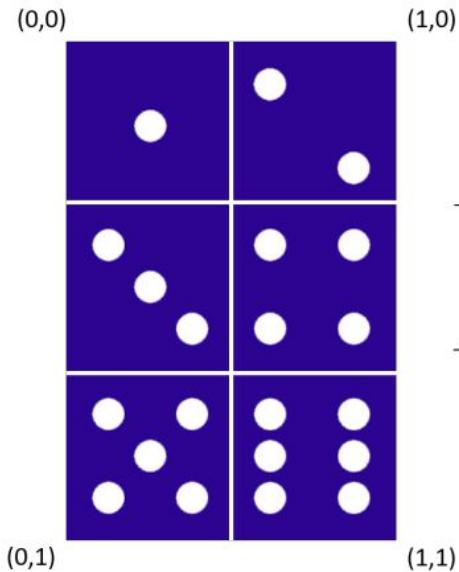
Fragment Shader: Recibe las coordenadas de textura **interpoladas** (a través de **coordenadas baricéntricas**).

En el FS, en vez de (o junto a) 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]
```

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,
```

```
...
```

```
]
```

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;
}
```

Fragment Shader

```
in vec2 outTexCoords;  
  
out vec4 outColor;  
  
uniform sampler2D samplerTex; // interpolador!  
  
void main()  
{  
    outColor = texture(samplerTex, outTexCoords);  
}
```

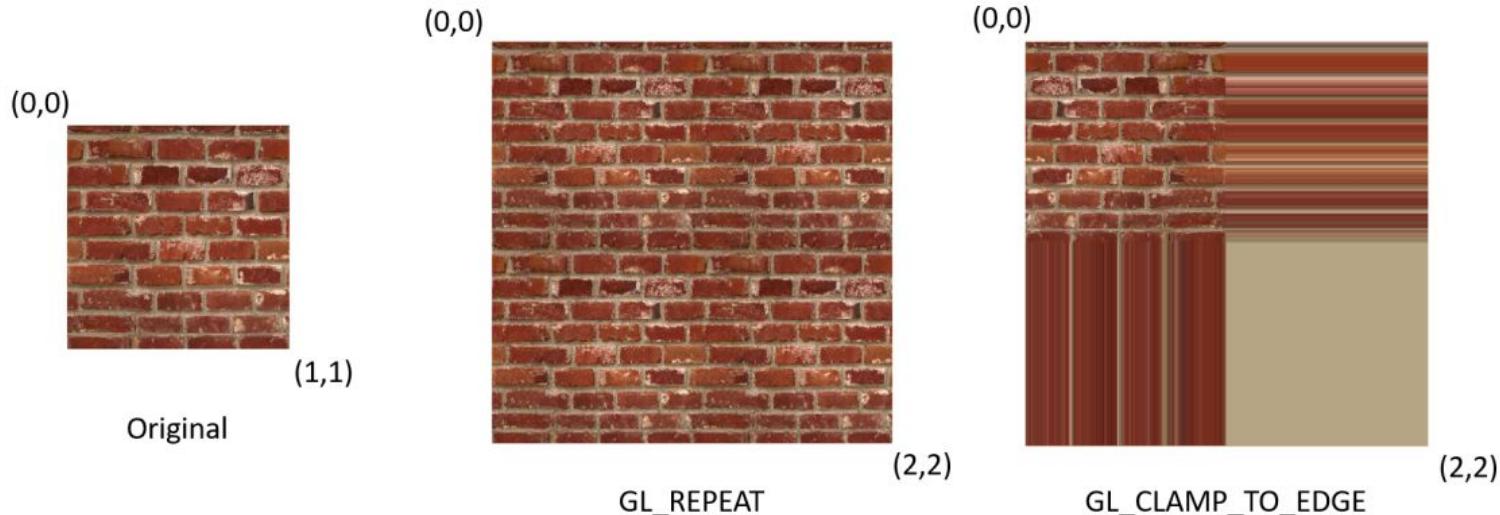
Configuración del VAO

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

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

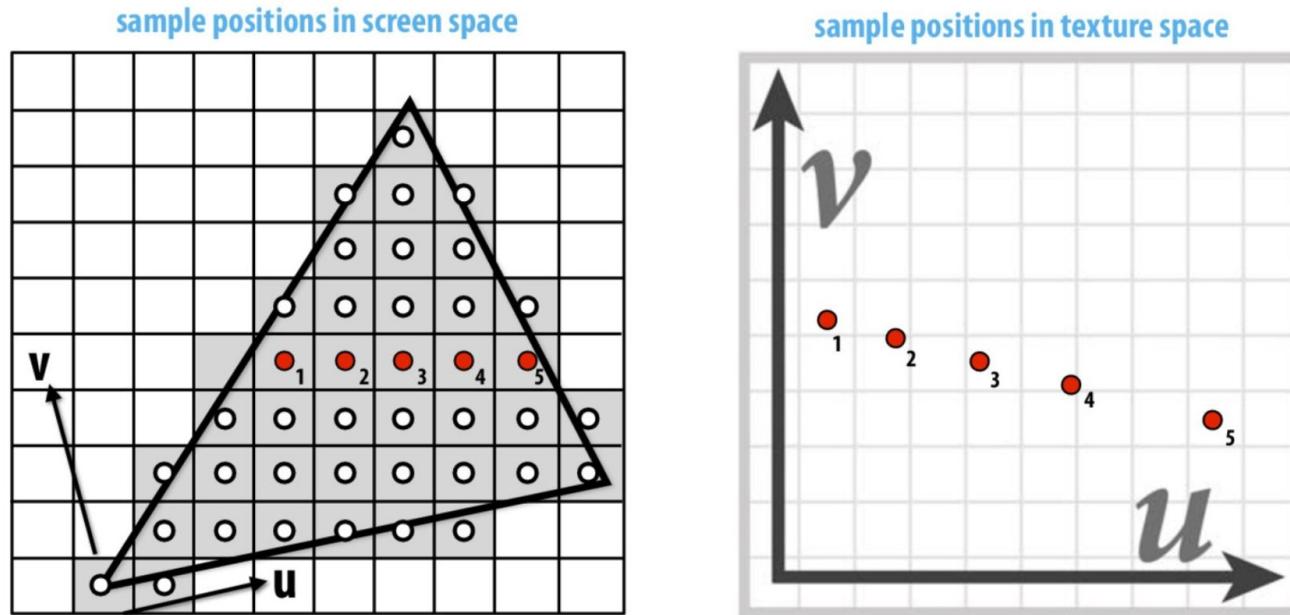


```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT|GL_CLAMP_TO_EDGE)  
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT|GL_CLAMP_TO_EDGE)
```

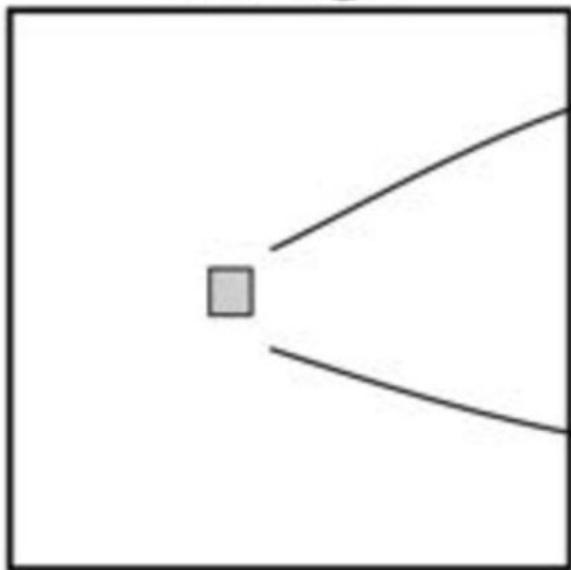


¡La repetición de imágenes tiene un uso!

¿Qué sucede si la cantidad de pixeles de nuestros triángulos no coincide con la cantidad de pixeles de la textura?

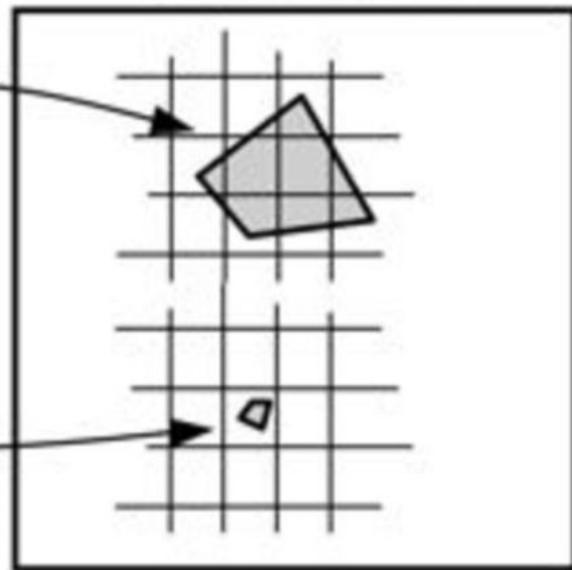


Image



Minification

Texture



Magnification

Modo de interpolación – Magnificación

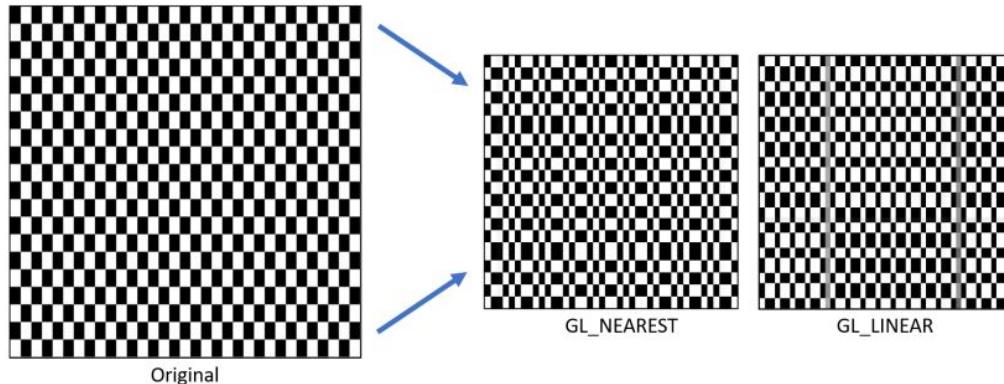
En el caso en que la textura es más pequeña que los pixels que cubre, se habla de filtro de **magnificación**. Un ejemplo sucede cuando la cámara está muy cerca de una superficie.



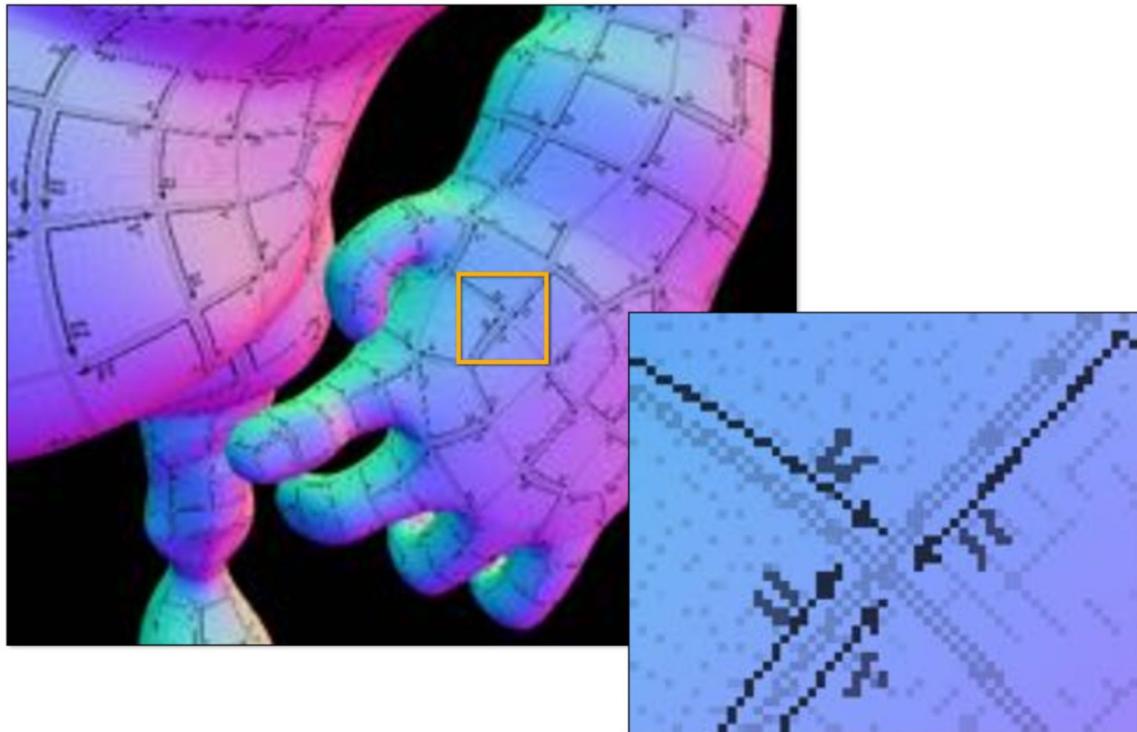
```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR|GL_NEAREST)
```

Modo de interpolación – Minificación

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 minificación. Puede pasar cuando un objeto es muy pequeño o está muy lejos de la cámara.



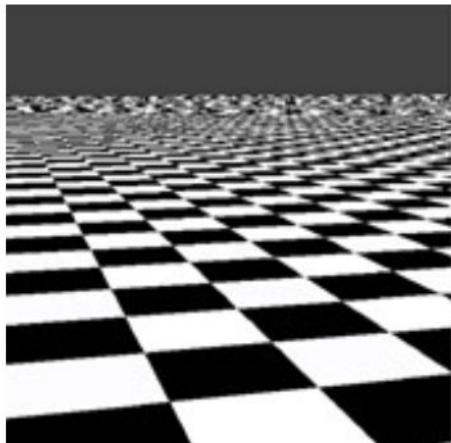
```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST)
```



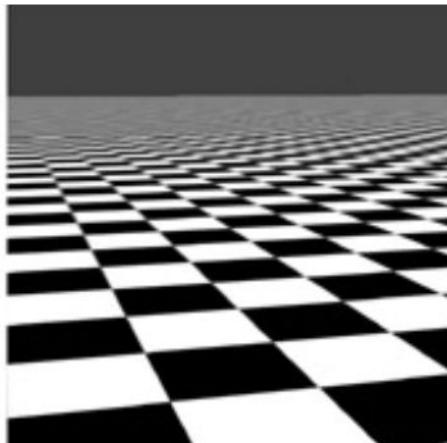
Aliasing debido a minificación.

Aliasing debido a la perspectiva

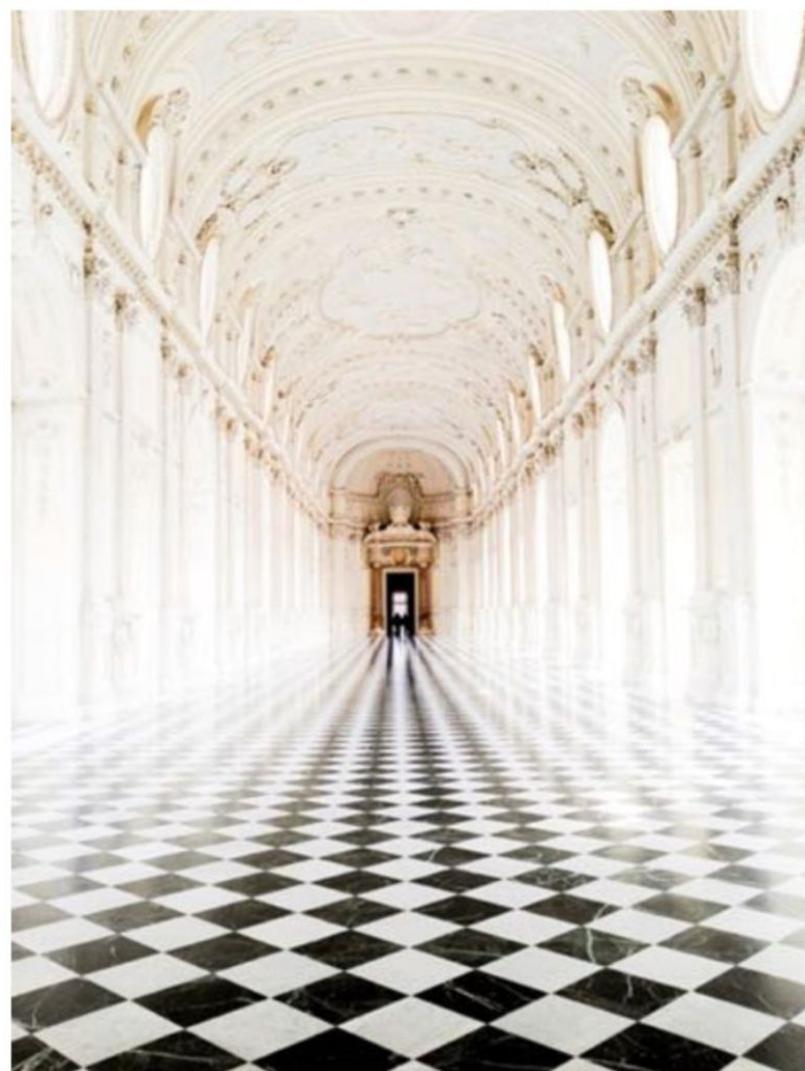
Debido a la perspectiva se genera **aliasing**, de manera similar a lo que vimos en la clase de rasterización.



(a)



(b)

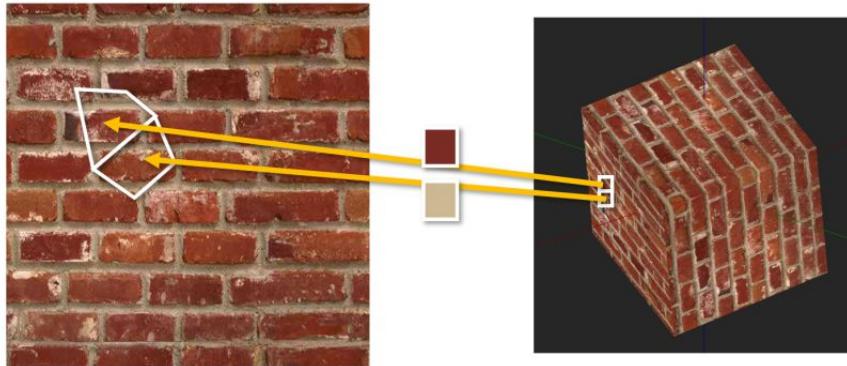


Aliasing en texturas

Si los objetos lejanos poseen texturas grandes, cada pixel tiene asociado muchos texels.

Considerando un único texel por pixel, píxeles vecinos tendrán colores distintos.

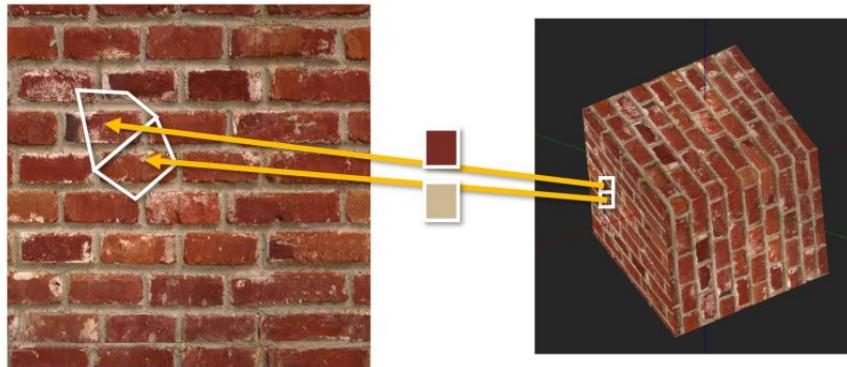
Esta transición brusca es la raíz del problema del aliasing. Porque tenemos una señal con mucho detalle que muestreamos a baja frecuencia.



Aliasing en texturas

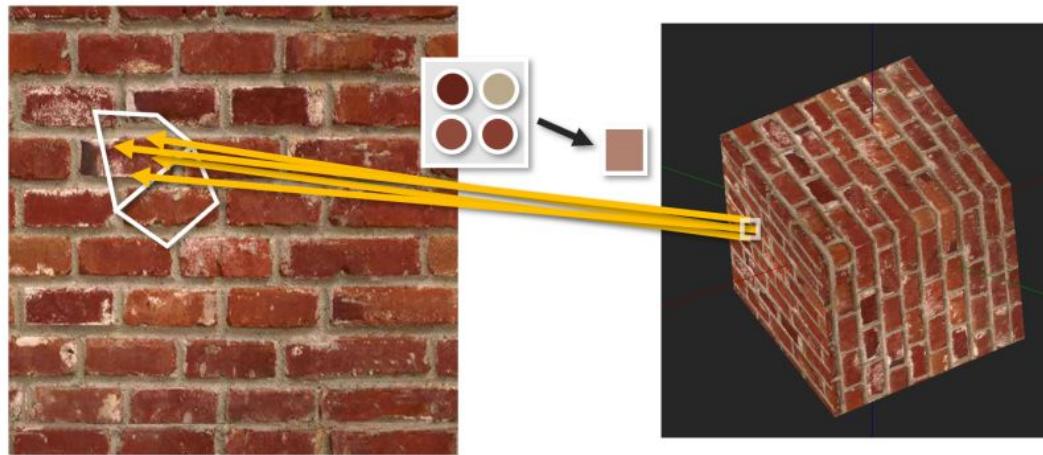
Un remedio paliativo es considerar el color promedio en cada pixel.

Esta idea es **costosa e inviable para rendering en tiempo real**.



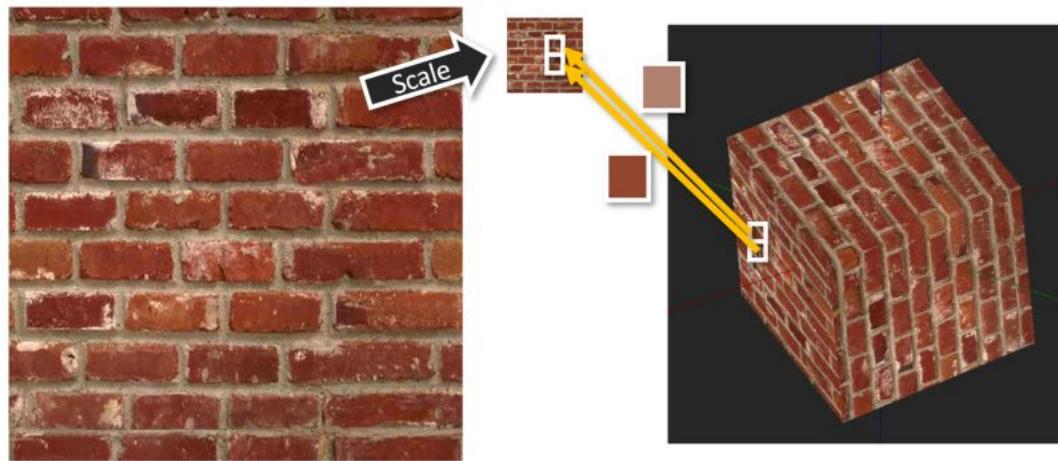
Aliasing en texturas

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.



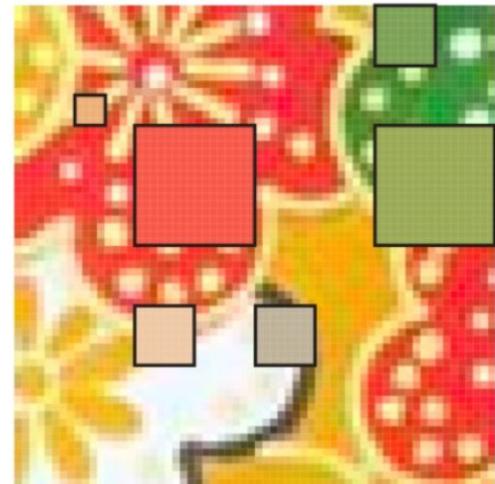
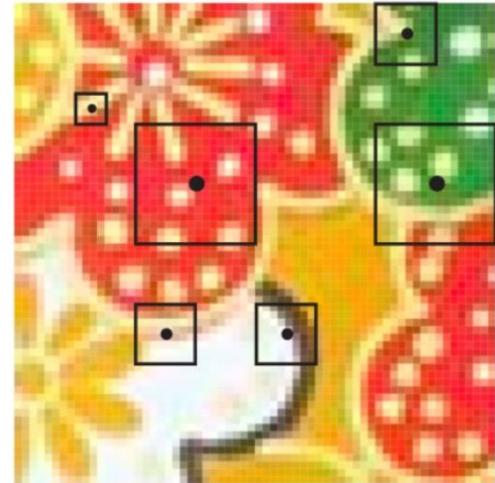
Aliasing en texturas

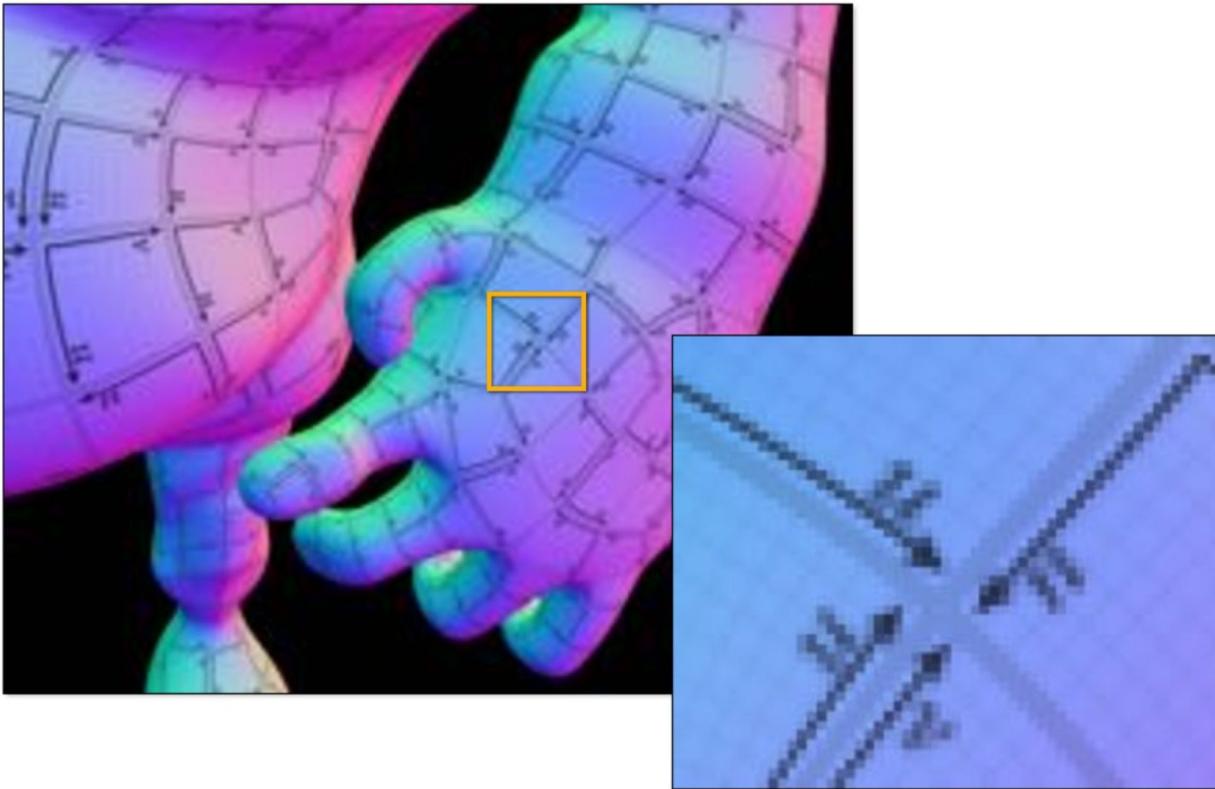
Otra idea es pre-calcular imágenes promediadas más pequeñas, de manera que cada pixel tenga pocos texels de cuales escoger, disminuyendo el problema del aliasing.



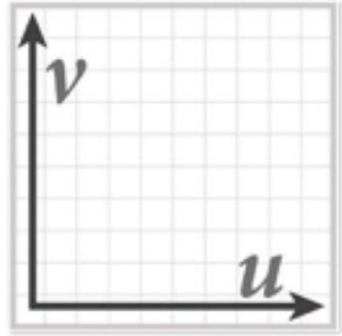
Esta técnica se conoce como “pre-filtering”. Al suplir nosotros la imagen más pequeña (por tanto, promediada), tenemos control sobre la calidad y apariencia de la imagen final.

la manera más rápida de computar es pre-computar

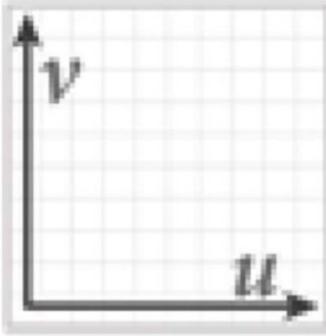




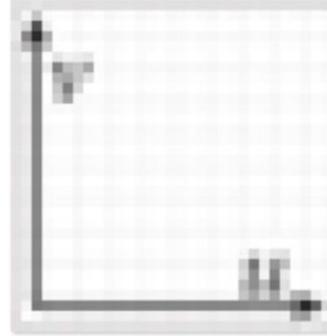
¡Mucho mejor!



Level 0 = 128x128



Level 1 = 64x64



Level 2 = 32x32



Level 3 = 16x16



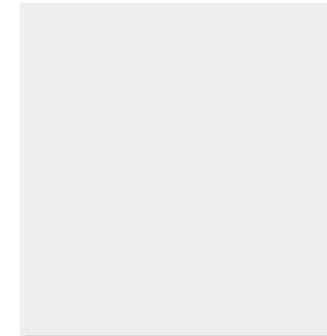
Level 4 = 8x8



Level 5 = 4x4



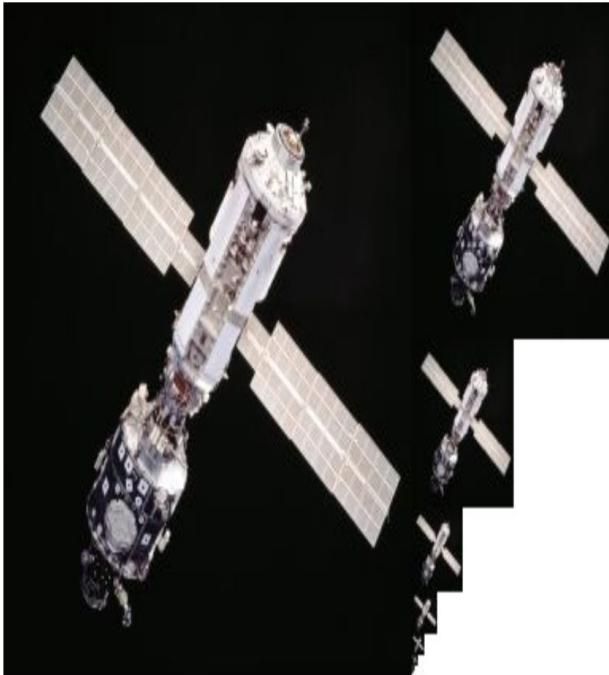
Level 6 = 2x2



Level 7 = 1x1

Ahora bien, para una textura específica, ¿cuántas texturas pre-filtradas existen? No es solamente una. Usualmente se trabaja en potencias de 2.

Esa técnica es llamada
Mip-Mapping



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)
```

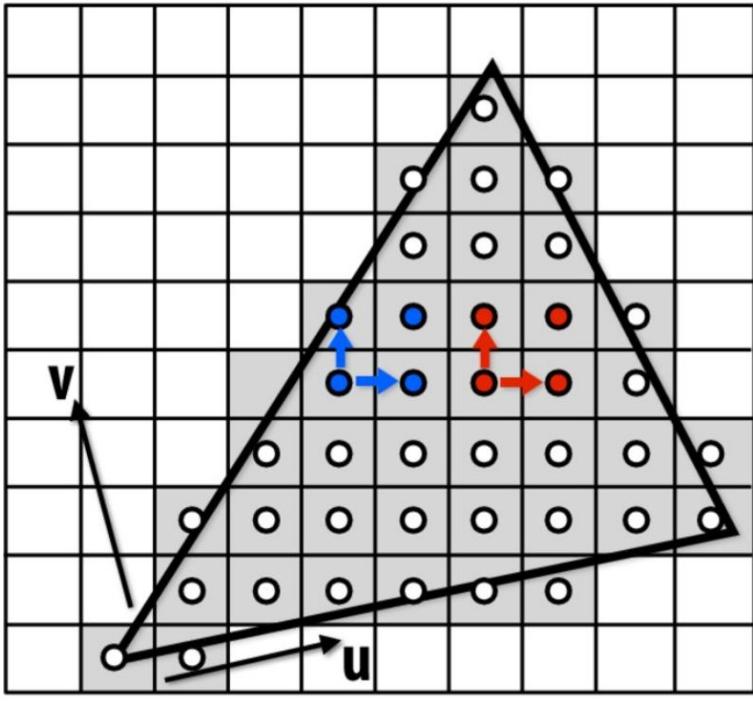
Mip-Maps en OpenGL

GL_NEAREST_MIPMAP_NEAREST: Utiliza el mipmap más cercano al tamaño del pixel y utiliza sobre él una interpolación del vecino más cercano.

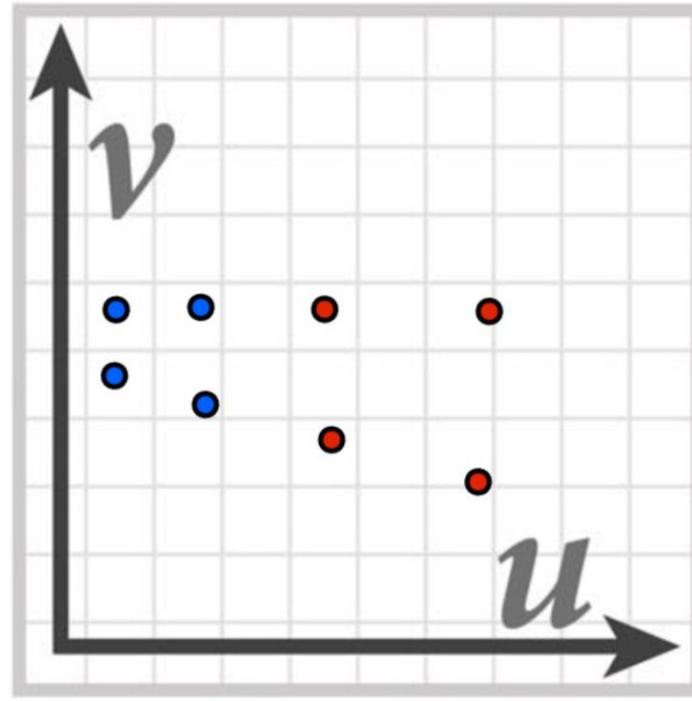
GL_LINEAR_MIPMAP_NEAREST: Utiliza el mipmap más cercano al tamaño del pixel y utiliza una interpolación lineal para escoger color.

GL_NEAREST_MIPMAP_LINEAR: Interpola linealmente los dos mips más cercanos al tamaño del pixel, e interpola este nivel usando el vecino más cercano.

GL_LINEAR_MIPMAP_LINEAR: Interpola linealmente los dos mips más cercanos al tamaño del pixel e interpola linealmente este nuevo nivel para determinar el color del pixel.

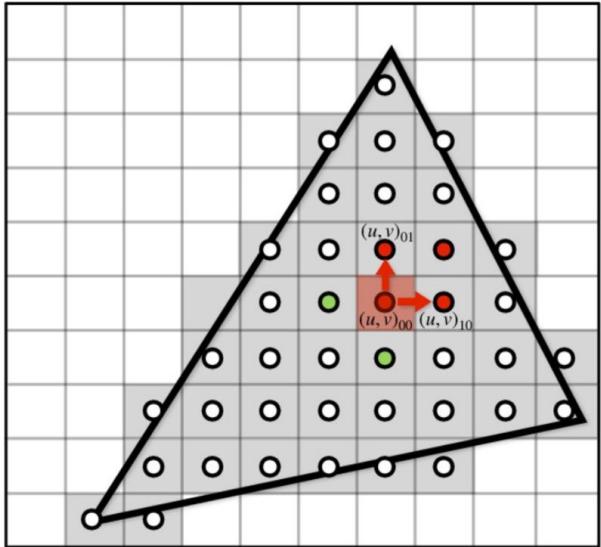


Screen space



Texture space

Los pixeles azules y rojos debiesen utilizar distintos tipos de mipmaps.
¿Cuáles?



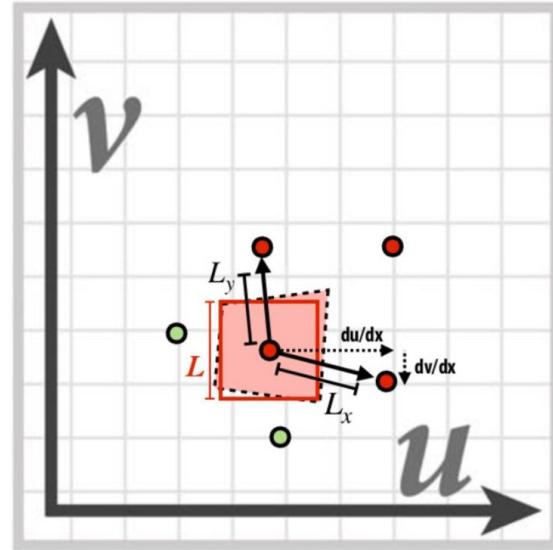
$$\frac{du}{dx} = u_{10} - u_{00} \quad \frac{dv}{dx} = v_{10} - v_{00}$$

$$\frac{du}{dy} = u_{01} - u_{00} \quad \frac{dv}{dy} = v_{01} - v_{00}$$

$$L_x^2 = \left(\frac{du}{dx} \right)^2 + \left(\frac{dv}{dx} \right)^2 \quad L_y^2 = \left(\frac{du}{dy} \right)^2 + \left(\frac{dv}{dy} \right)^2$$

$$L = \sqrt{\max(L_x^2, L_y^2)}$$

mip-map level: $d = \log_2 L$



¡El mipmap a utilizar se determina localmente en cada pixel!

Propuesto: estudiar el **filtro anisotrópico**. Se aplica cuando la extensión de los pixeles abarca múltiples mipmaps.

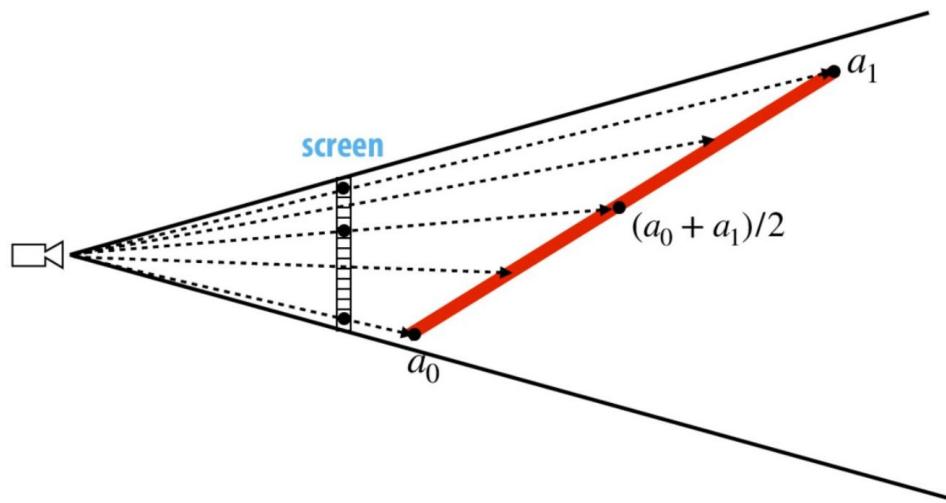
Perspectiva

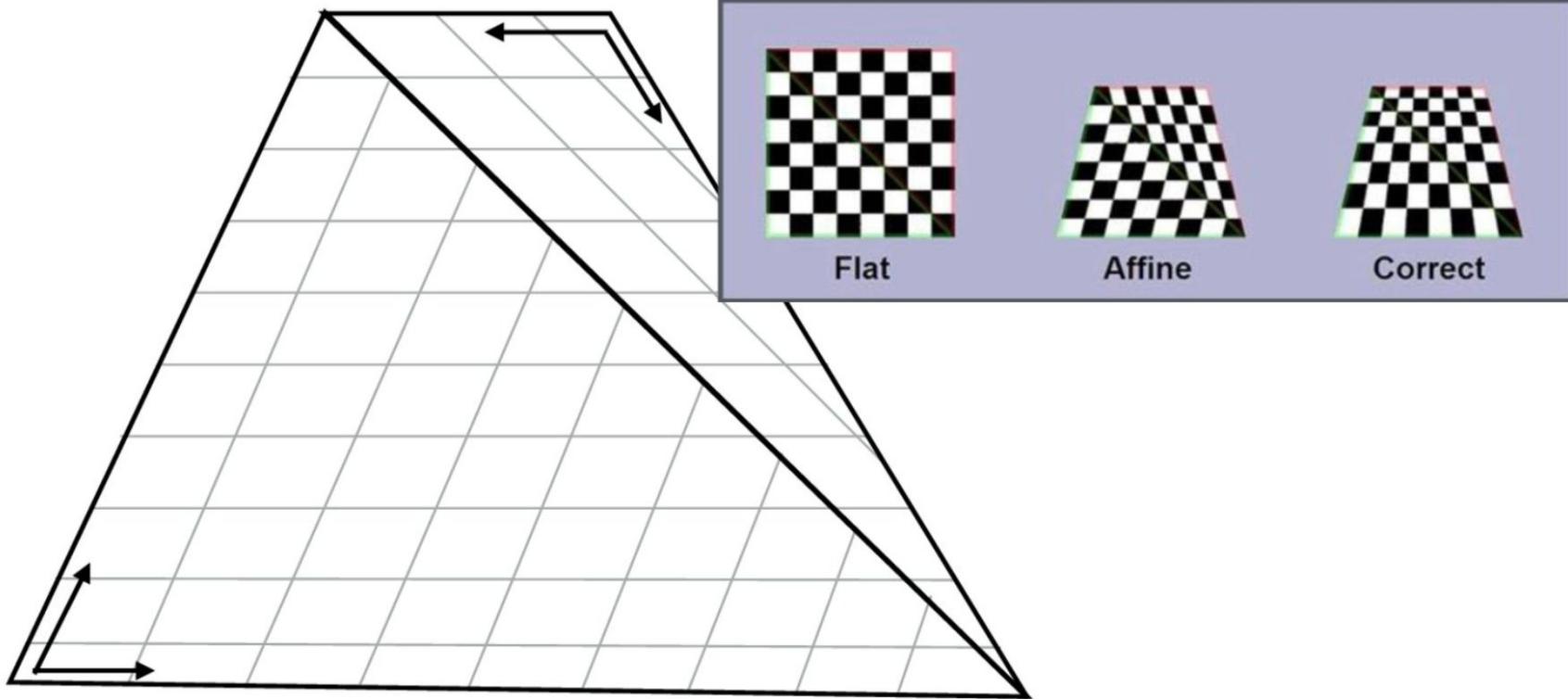
Hemos hablado de interpolación.

Pero lo más probable es que nuestros vértices tengan distintas profundidades (distancia a la cámara) entre sí.

La interpolación se debe hacer, por tanto, en 3D, no en 2D. ¡En 2D nuestra superficie se ha deformado!

Recordar: **geometría proyectiva** y coordenadas homogéneas.





Este cuadrilátero compuesto por dos triángulos es un ejemplo de lo que pasa cuando no se corrige la perspectiva.

Hardware mode, perspective correction set to memory+CPU



Software mode, no perspective correction

Esto era un problema común en los juegos de PSX (Play 1), que no tenía corrección de perspectiva por hardware.

Y corregir era muy **caro**: ¡una división por textura en cada pixel!



Propuesto: encontrar cómo se corrige la perspectiva (esto se preguntó en un control el año pasado).

Hint: recuerden lo que hacen las coordenadas homogéneas.

¿Preguntas?