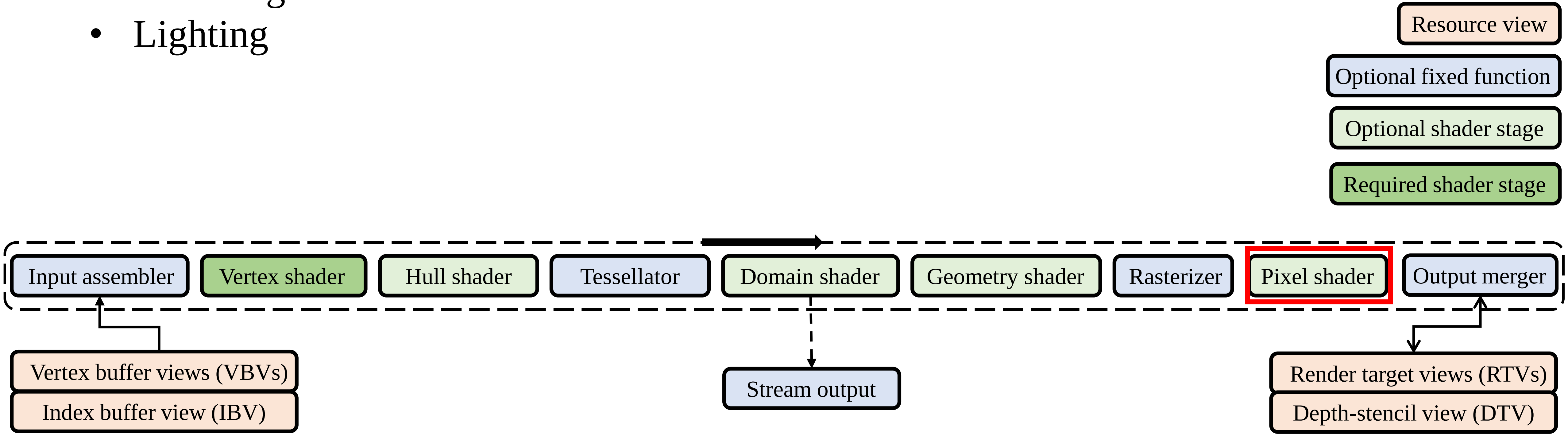
7. Image Texturing

Prof. HyeongYeop Kang
siamiz@khu.ac.kr
YouTube: HKang IIIXR LAB
IIIXR LAB

Pixel Shader

Rendering pipeline (revisited)

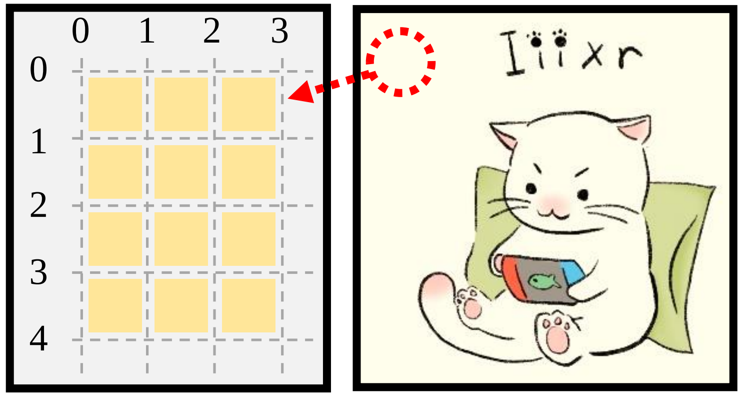
- The per-pixel attributes produced by the rasterizer usually include a normal vector and texture coordinates. Using the attributes, the pixel shader determines the color of each pixel.
  - Texturing
  - Lighting



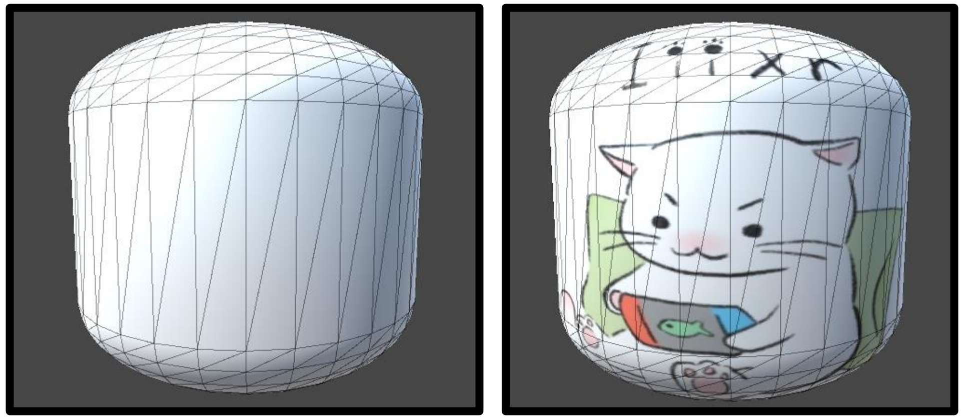
Texture Coordinates

Image texturing

- The simplest among the various texturing methods is image texturing.
- Image texturing is often described as pasting an image on an object's surface.
- A texture is typically represented as a 2D array of texels (texture elements).
- A texel's location can be described by the coordinates of its center.
  - For example, the upper-right corner texel is located at (0.5, 0.5), and its neighbor on the right is at (1.5, 0.5).



Sample image texture

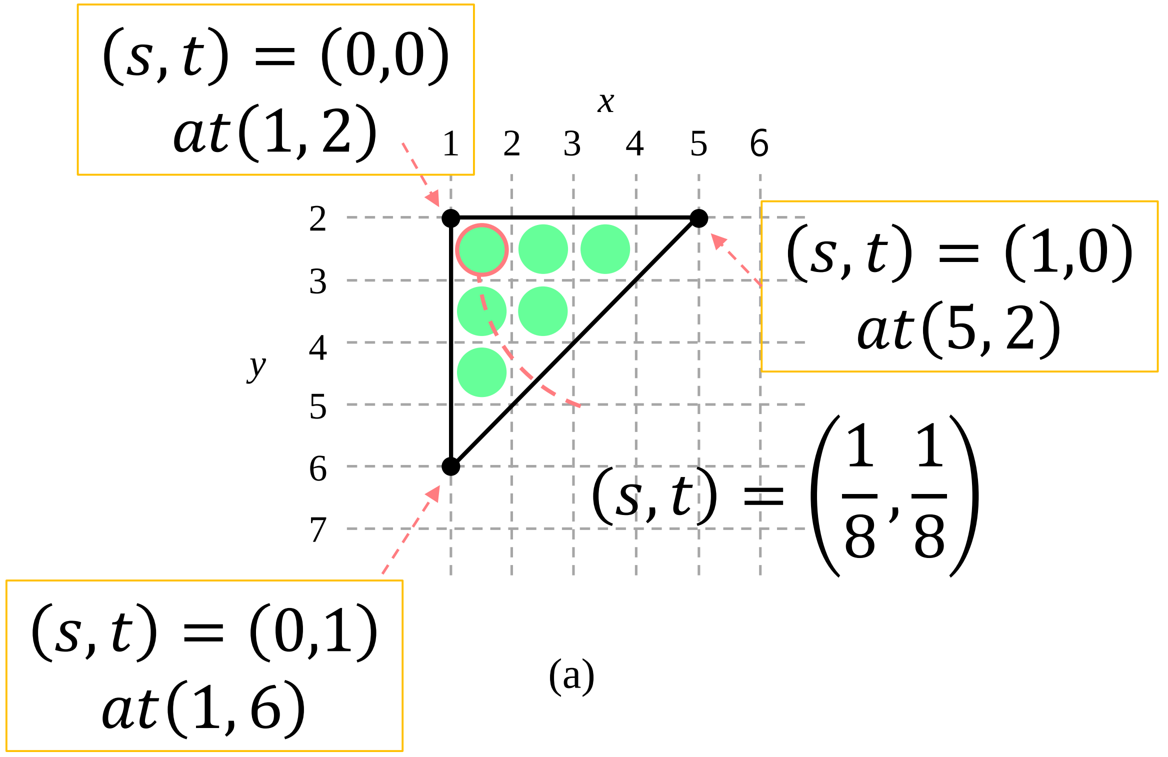


The image is pasted on the surface

Texture Coordinates

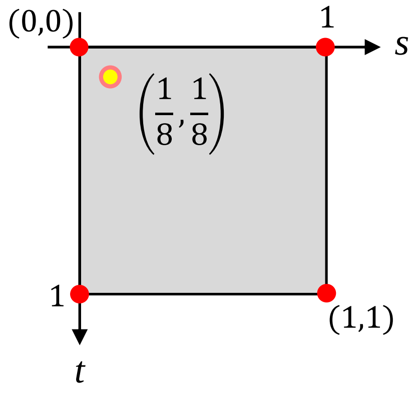
For texturing, we assign the texture coordinates, (s, t), to each vertex of the polygon mesh "at the modeling step."

- The rasterizer interpolates them for the pixel.
- The normalized texture coordinates (s, t) are projected into the texture space.
  - $(s`, t`) = (s, t) \cdot (width_{tex}, height_{tex})$

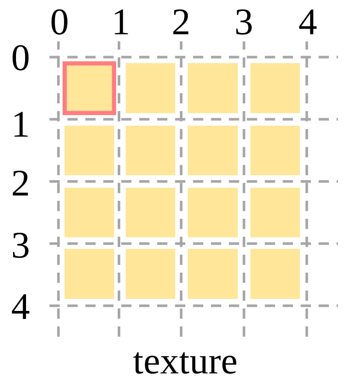


$(s,t) = (0,0)$ $at(1,2)$
$(s,t) = (1,0)$ $at(5,2)$
$(s,t) = (0,1)$ $at(1,6)$
$(s,t) = \left(\frac{1}{8},\frac{1}{8}\right)$
x
1 2 3 4 5 6
y
2 3 4 5 6 7

(a)



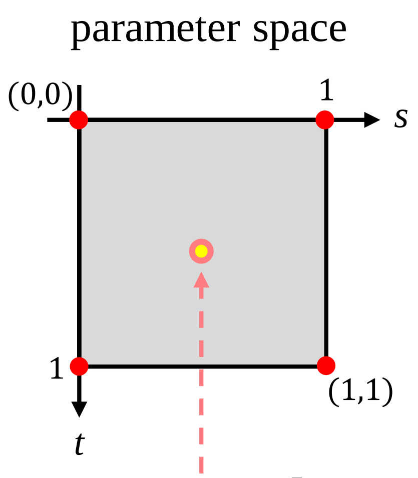
(b)



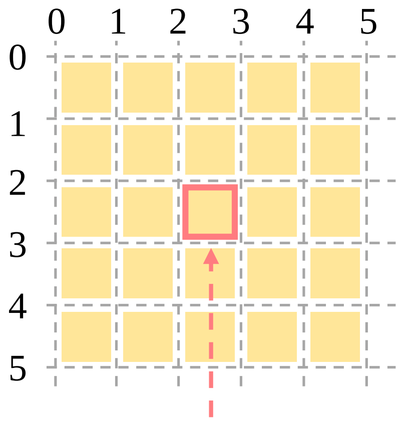
(c)

Texture Coordinates

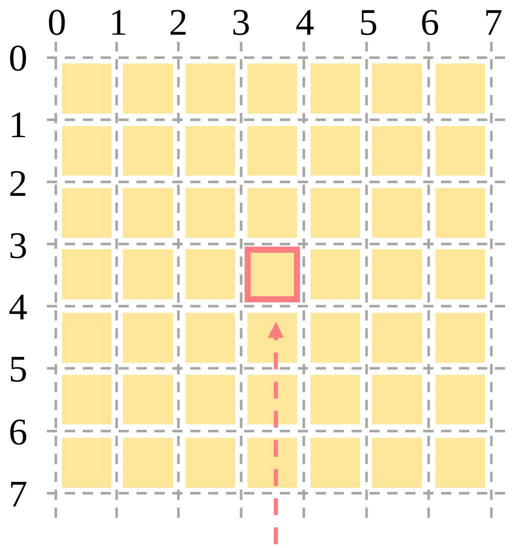
It is customary to normalize the texture coordinates such that *s* and *t* range from 0 to 1. The normalized texture coordinates do not depend on a specific texture resolution and can be freely plugged into various textures.



texture coordinates (0.5,0.5)



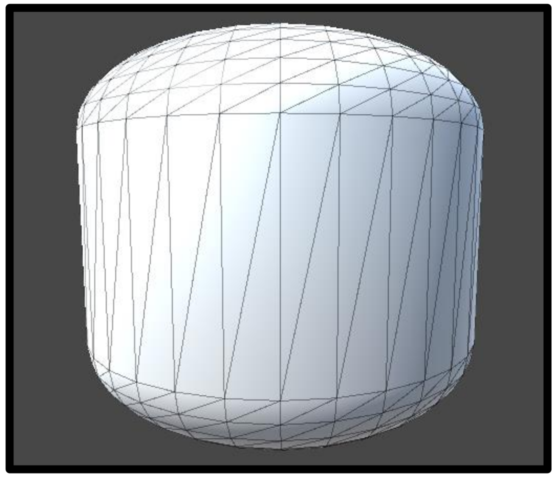
texel at (2.5,2.5)
texture 1



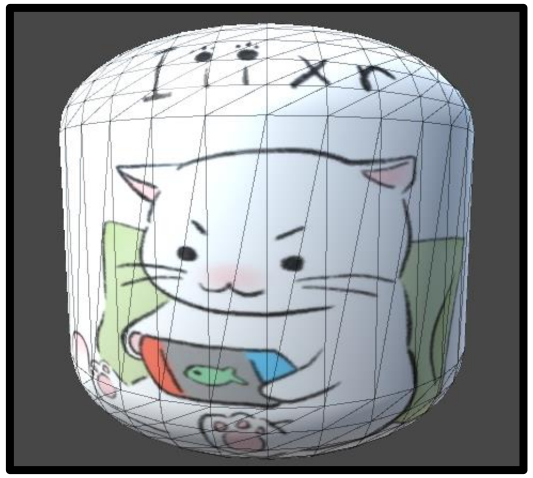
texel at (3.5,3.5)
texture 2

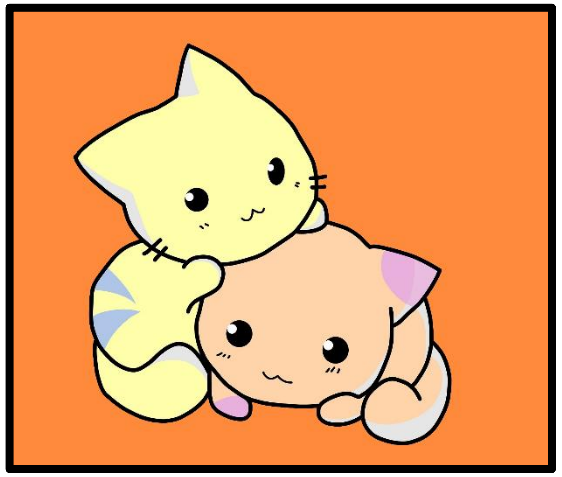
Texture Coordinates

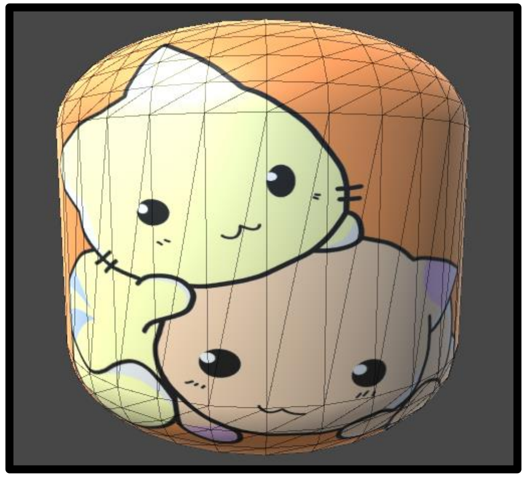
Multiple images of different resolutions can be glued to a surface without changing the texture coordinates.



texture can be pasted without coordinate modification.



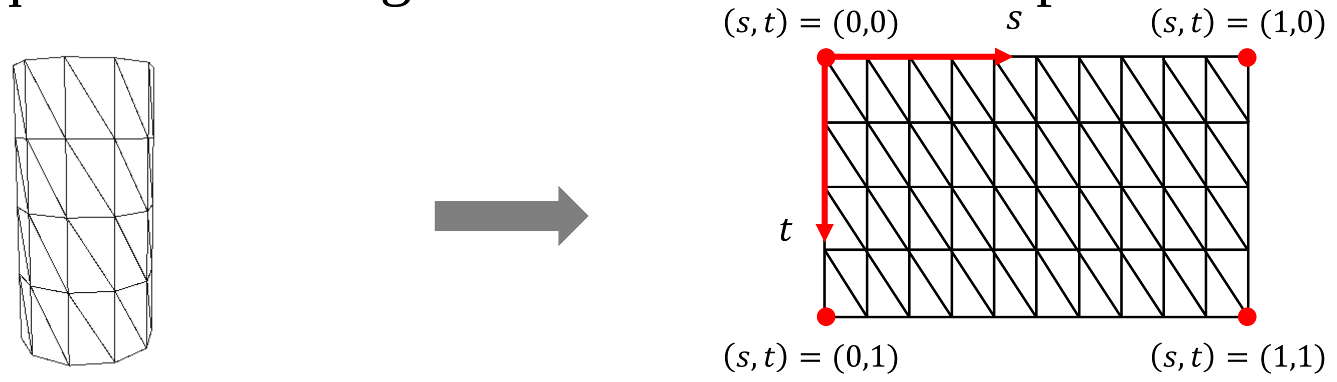




Surface Parameterization

Parameterization

- The texture coordinates are assigned to the vertices of the polygon mesh.
- This process is called surface parameterization or simply parameterization. In general, parameterization requires unfolding a 3D surface onto a 2D planar domain.



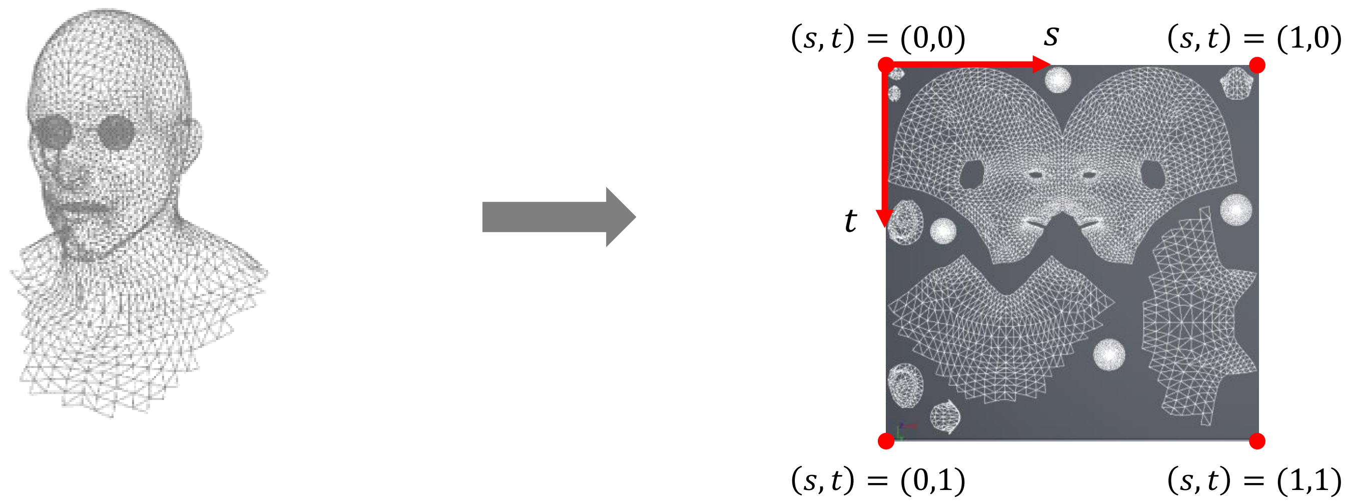
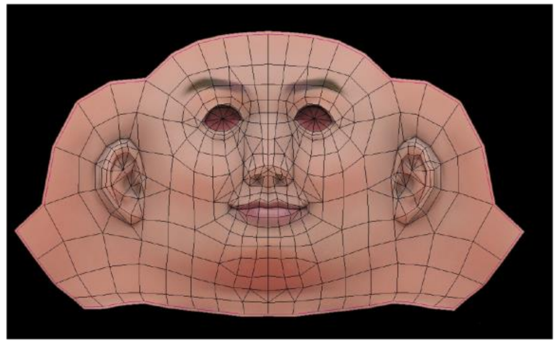


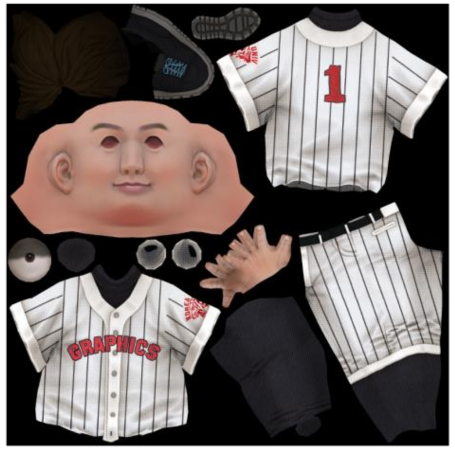
Chart and Atlas

Patch, chart, and atlas

- A complex polygon mesh is usually subdivided into a number of patches such that the patches are unfolded individually (e.g. face, body, arm, …)
- Each patch is unfolded and parameterized. Then, the artist draws an image for each patch. An image for a patch is often called a chart.
- Multiple charts are usually packed and arranged in a texture, which is often called an atlas.



chart

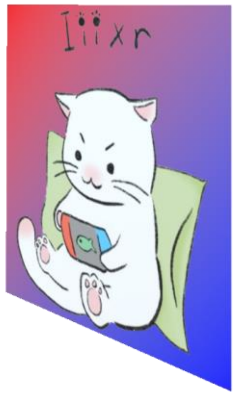


atlas

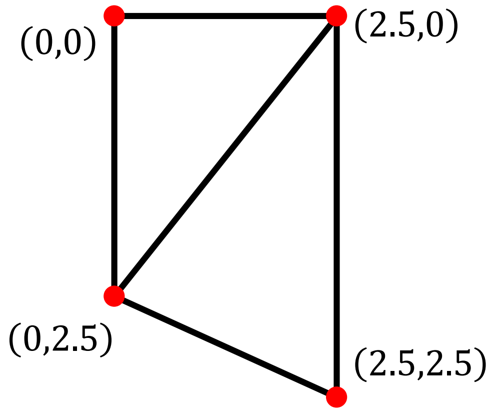
Texture Wrapping

The texture coordinates (s,t) are not necessarily in the range of [0,1]. The *texture wrapping mode* handles (s,t) that are outside of the range.

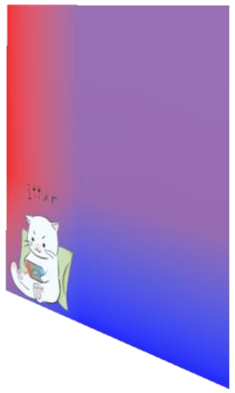
- *Clamp-to-Edge*: The out-of-range coordinates are rendered with the edge colors.
- *Repeat*: The texture is tiled at every integer junction.
- *Mirrored-Repaet*: The texture is mirrored or reflected at every integer junction.



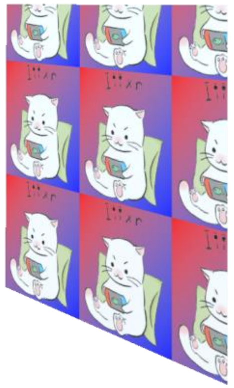
(a)



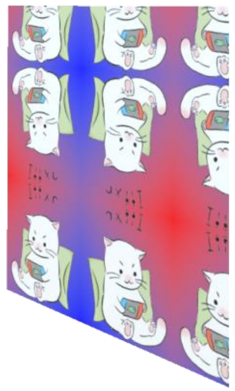
(b)



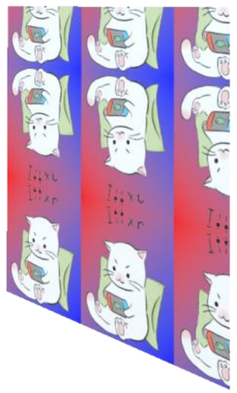
(c)



(d)



(e)

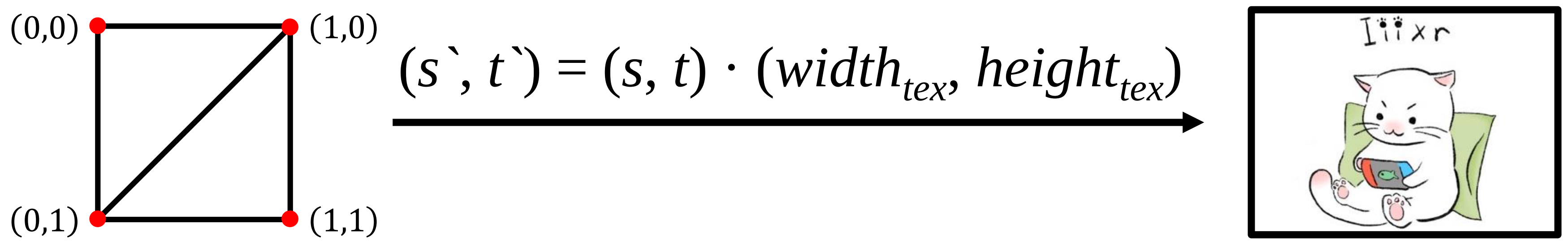


(f)

Texture Filtering

Consider a quad. For each **pixel** located at (x, y) in the screen, its texture coordinates (s, t) are projected onto (s', t') in the texture space.

- Note that (s', t') are floating-point values in almost all cases.
- Consequently, texels around (s', t') are sampled. This sampling process is called texture filtering.

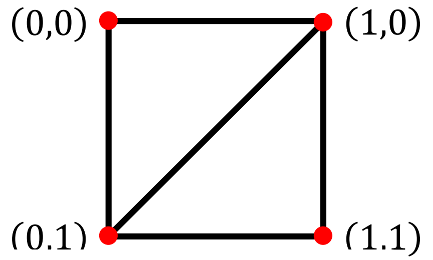


$$(s`, t`) = (s, t) \cdot (width_{tex}, height_{tex})$$

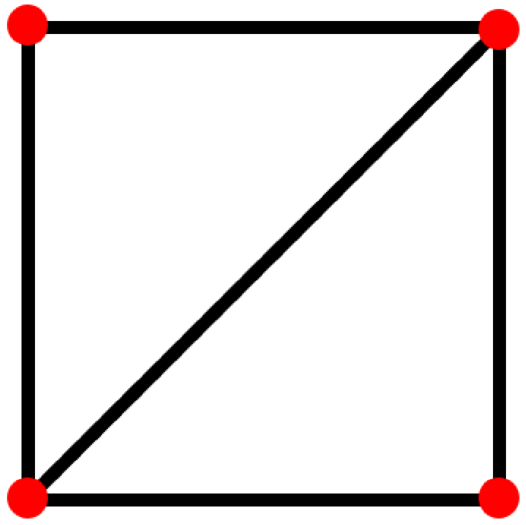
Texture Filtering

Magnification

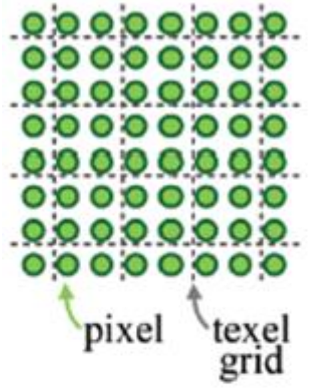
- The screen-space quad may appear larger than the image texture, and therefore the texture is magnified so as to fit to the quad.
- There are more pixels than texels.



$$(s`, t`) = (s, t) \cdot (width_{tex}, height_{tex}) \xrightarrow{\qquad}$$



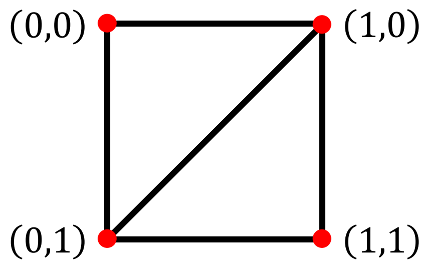
$$\xrightarrow{\qquad}$$



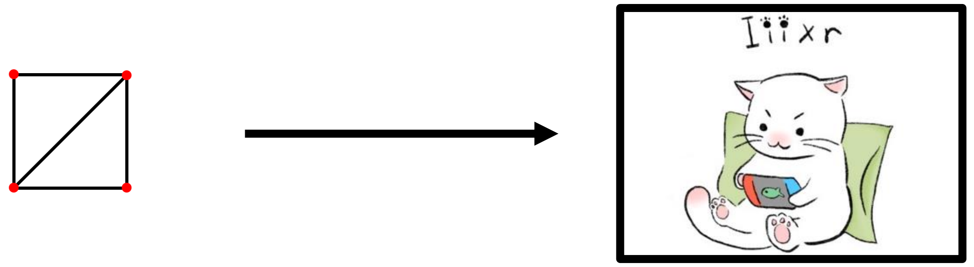
Texture Filtering

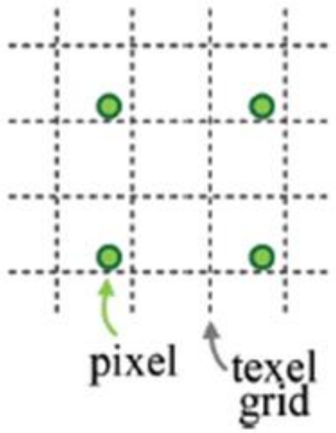
Minification

- In contrast, the screen-space quad may appear smaller than the image texture, and the texture is minified.
- Sparsely projected onto the texture space, the pixels take large jumps in the space.
- There are less pixels than texels.



$$(s`, t`) = (s, t) \cdot (width_{tex}, height_{tex}) \xrightarrow{\qquad}$$

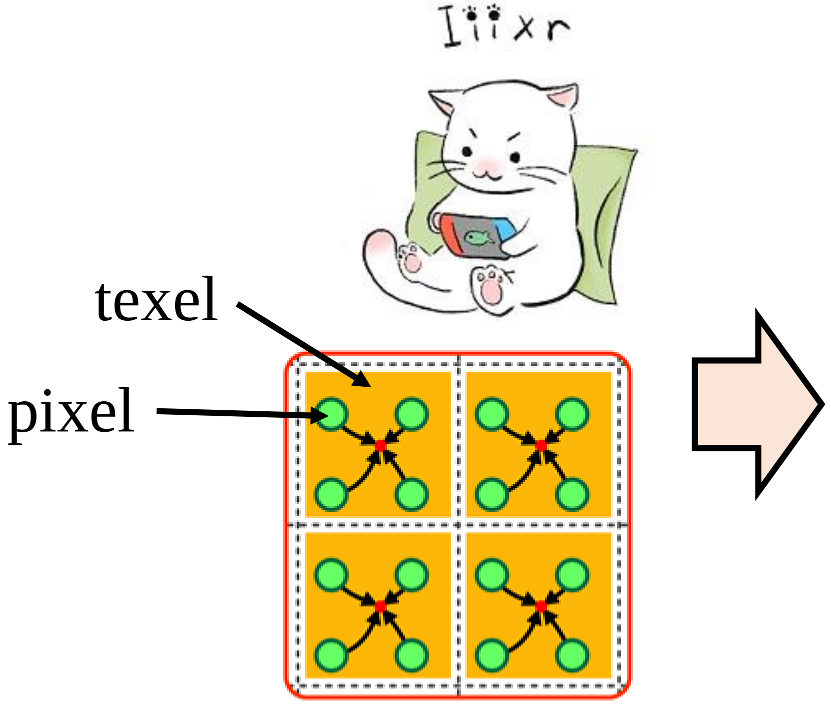




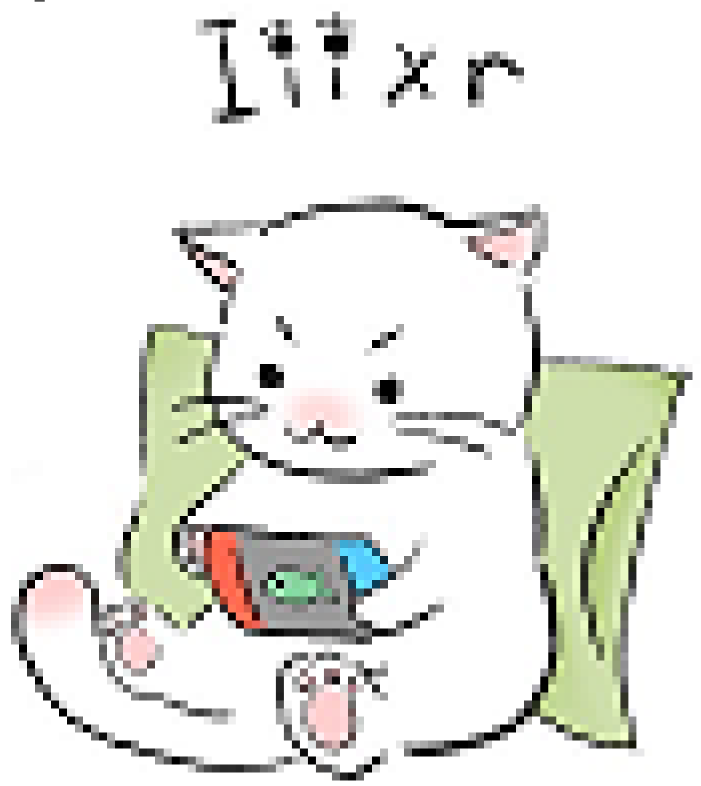
Filtering for Magnification

Option 1: Nearest point sampling

- A block of pixels can be mapped to a single texel.
- Consequently, adjacent pixel blocks can change abruptly from one texel to the next texel, and a blocky image is often produ



four pixels obtain the color from the same texel.

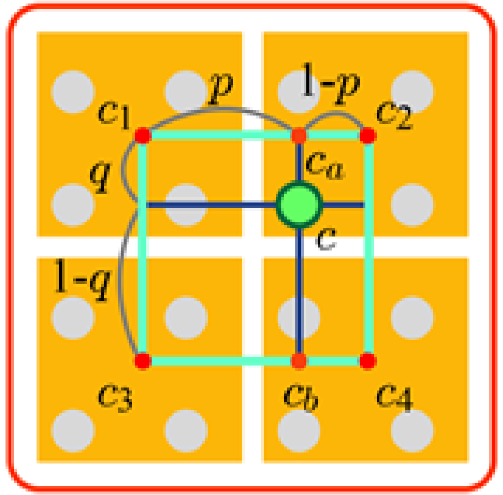


blocky image is generated

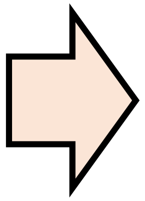
Filtering for Magnification

Option 2: Bilinear interpolation

- It is preferred to nearest point sampling not only because the final result suffers much less from the blocky image problem but also because the graphics hardware is usually optimized for bilinear interpolation.

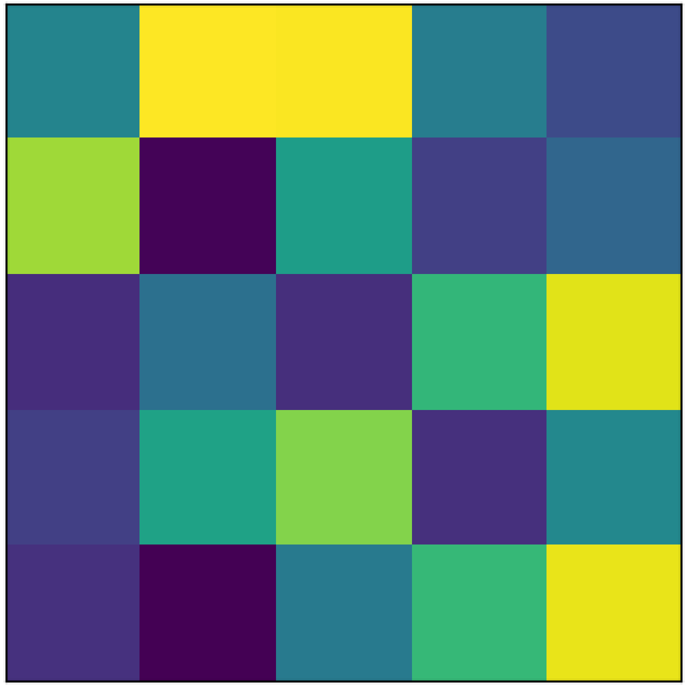


$$c_a = (1-p)c_1 + pc_2$$
$$c_b = (1-p)c_3 + pc_4$$
$$c = (1-q)c_a + qc_b$$

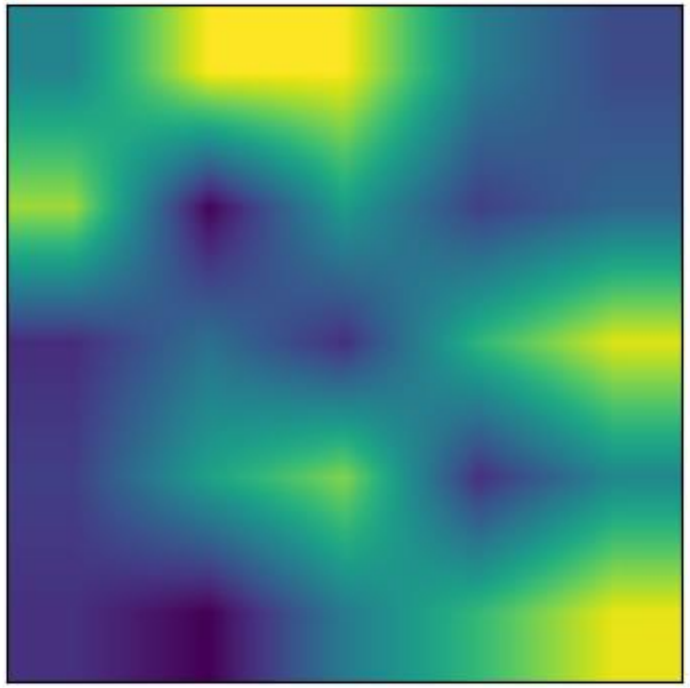


Filtering for Magnification

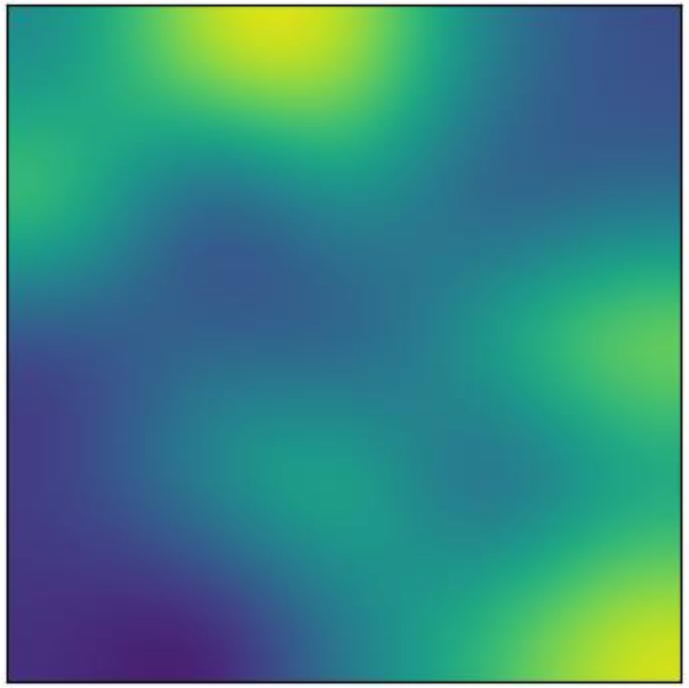
Visualization of filtering algorithms



2D nearest-neighbor



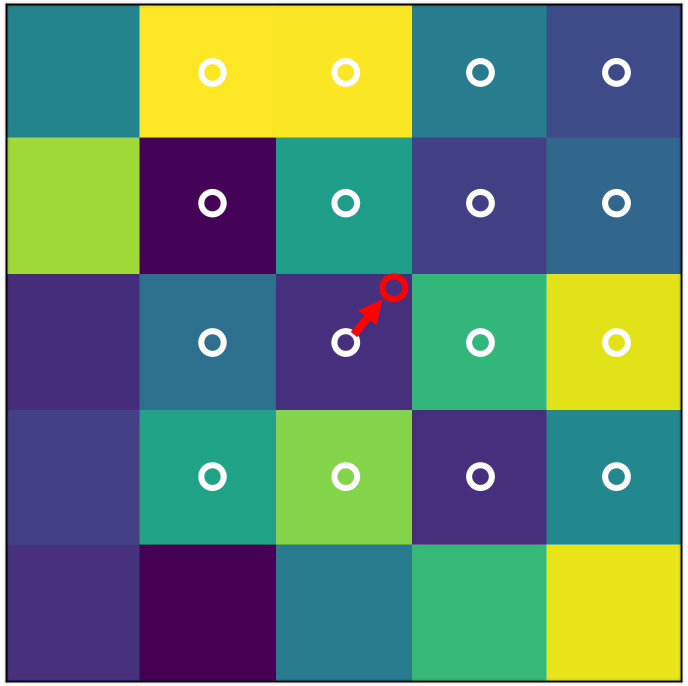
Bilinear



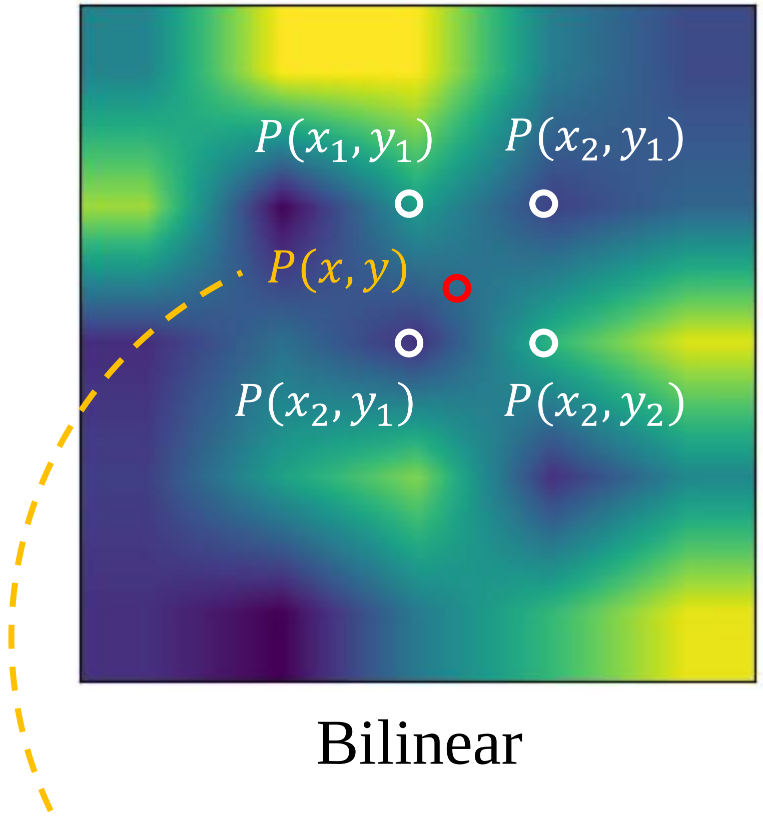
Bicubic

Filtering for Magnification

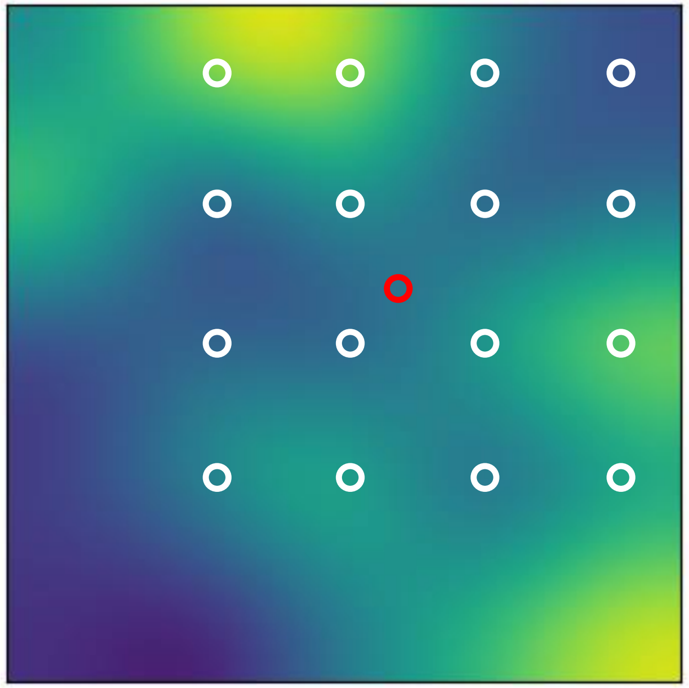
Visualization of filtering algorithms



2D nearest-neighbor



Bilinear



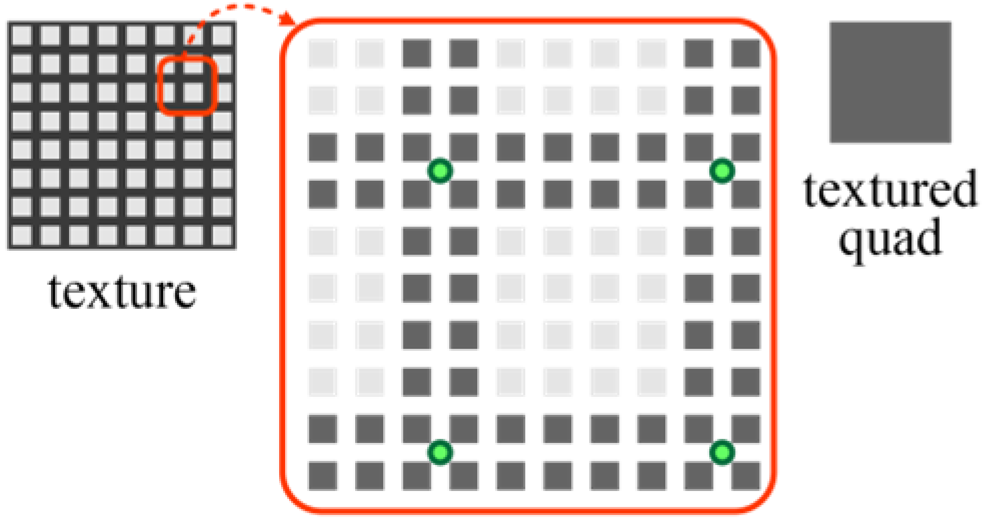
Bicubic

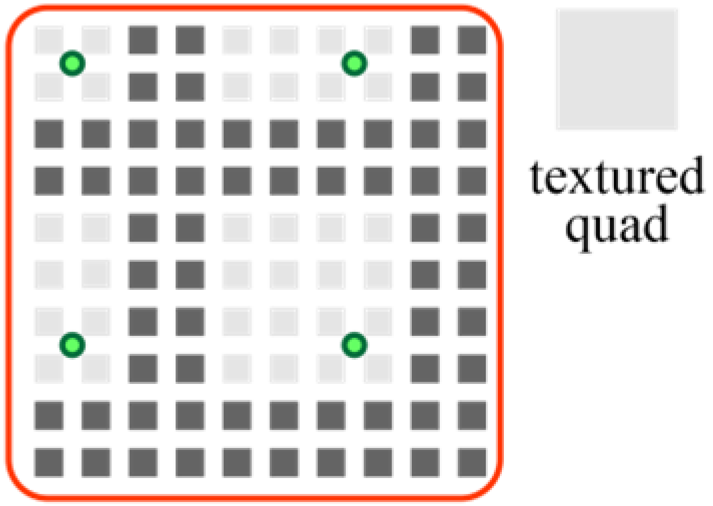
$$P(x,y)=\frac{y_2-y}{y_2-y_1}\left(\frac{x_2-x}{x_2-x_1}P(x_1,y_1)+\frac{x-x_1}{x_2-x_1}P(x_2,y_1)\right)+\frac{y-y_1}{y_2-y_1}\left(\frac{x_2-x}{x_2-x_1}P(x_1,y_2)+\frac{x-x_1}{x_2-x_1}P(x_2,y_2)\right)$$

Filtering for Minification

Aliasing in minification

- In the minification case, the pixels are sparsely projected onto the texture space.
- Consider the checkerboard image texture.
  - If all pixels are surrounded by dark-gray texels, the textured quad appears dark gray.
  - If all pixels are surrounded by light-gray texels, the textured quad appears light gray.
- This problem is an instance of *aliasing*.

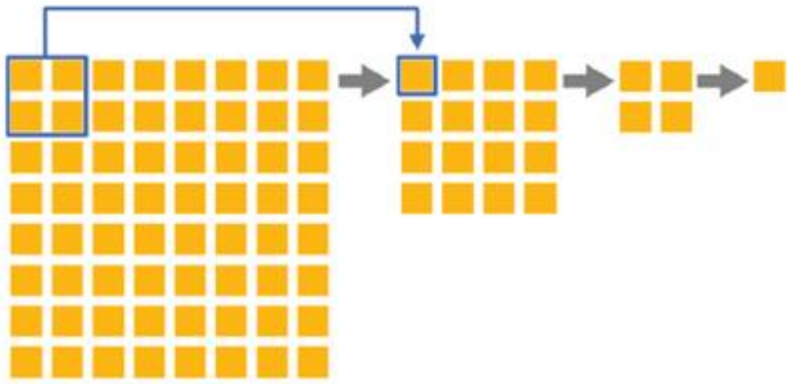


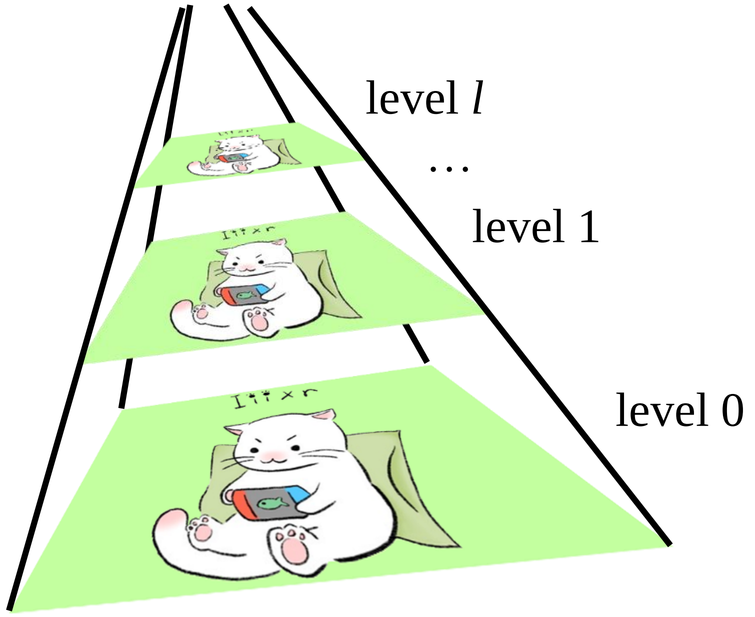


Mipmap

Mipmap

- The aliasing problem observed in minification is caused by the fact that we have more texels than pixels. The pixels take large jumps in the texture space, leaving many texels not involved in texturing.
- Then, a simple solution is to reduce the number of texels such that the texel count becomes as close as possible to the pixel count. For decreasing the texture size, down-sampling is adopted.
- Given an original texture of 2^lx2^l resolution, a pyramid of (l+ 1) levels is constructed, where the original texture is located at level 0.
- The pyramid is called a *mipmap*.
- The level to filter is named *level of detail*, and is denoted by λ. We will look for the level whose texel count is close to the pixel count.

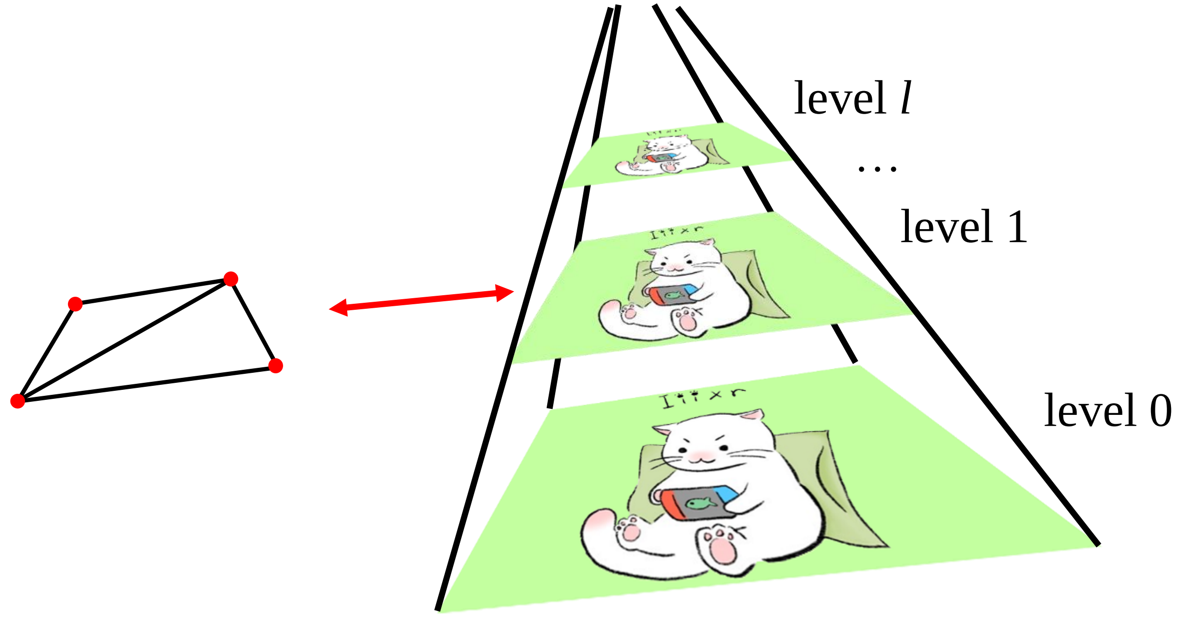


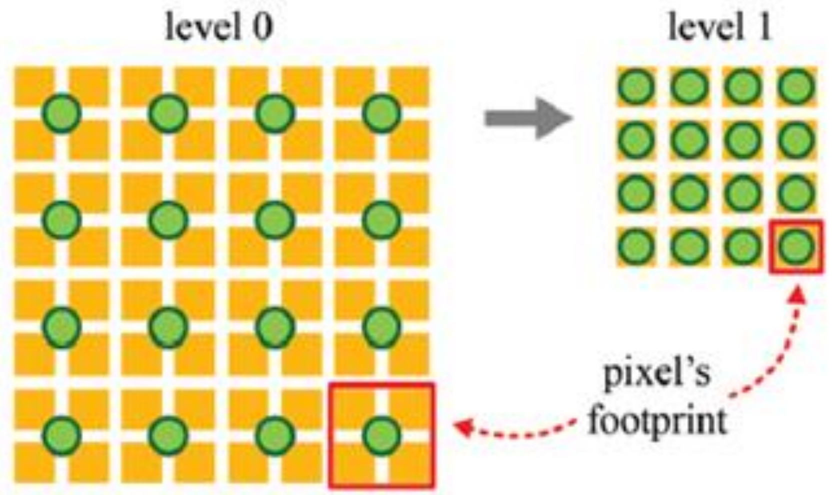


Mipmapping

Level 1 example

- A pixel covers an area on the screen.
- For simplicity, take the area as square. Then, a pixel's projection onto the texture space is not a point but an area centered at (s', t'). The projected area is called the *footprint* of the pixel.



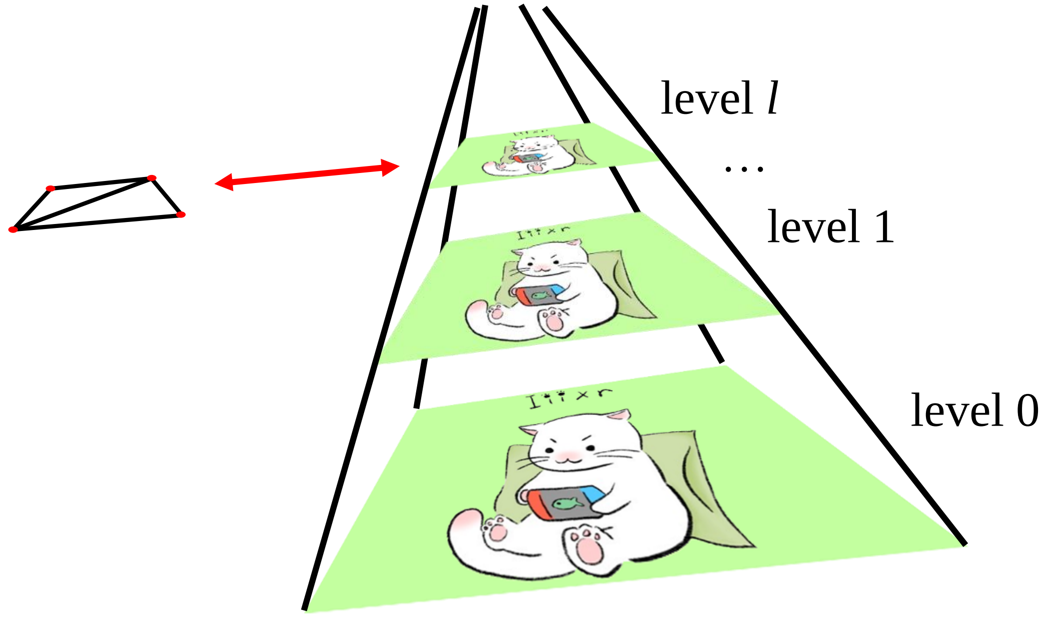


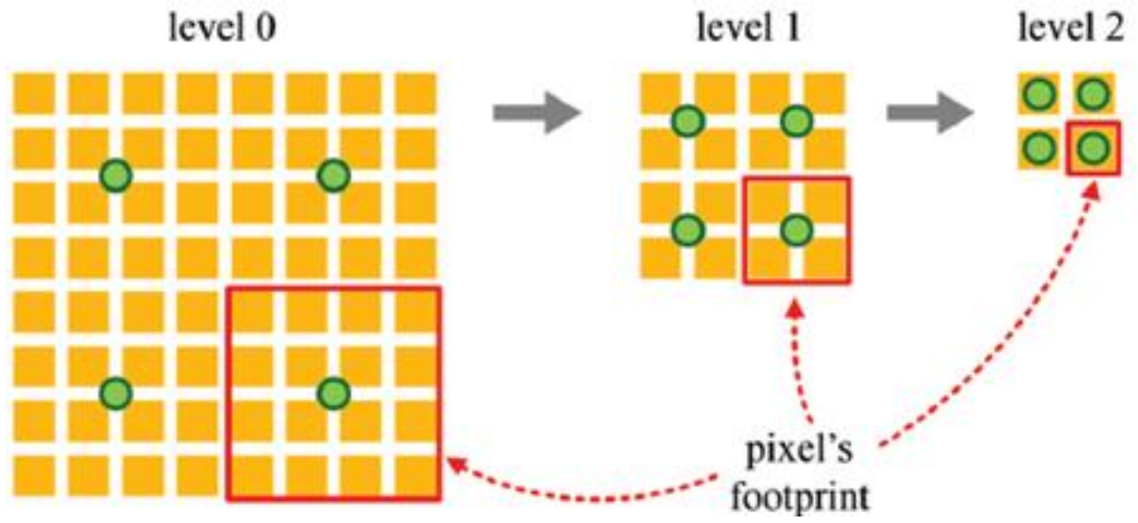
- In this contrived example, the quad and level 1 texture have the same size. A pixel's footprint covers 2 × 2 texels in level 0, but covers a single texel in level 1.
- When a pixel footprint covers m × m texels of level-0 texture, λ is set to $\log_2 m$.

Mipmapping

Level 2 example

- In this example, the screen-space quad and level-2 texture have the same size. A pixel's footprint covers exactly a single texel in level-2 texture, which is then filtered.
- In this and previous examples, observe that all texels are involved in filtering.

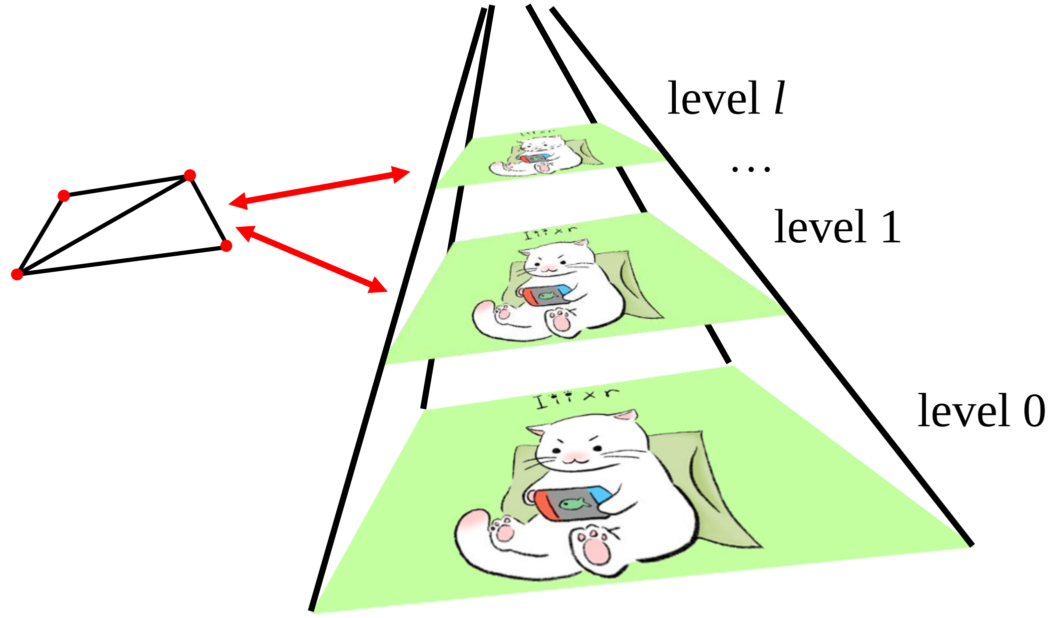


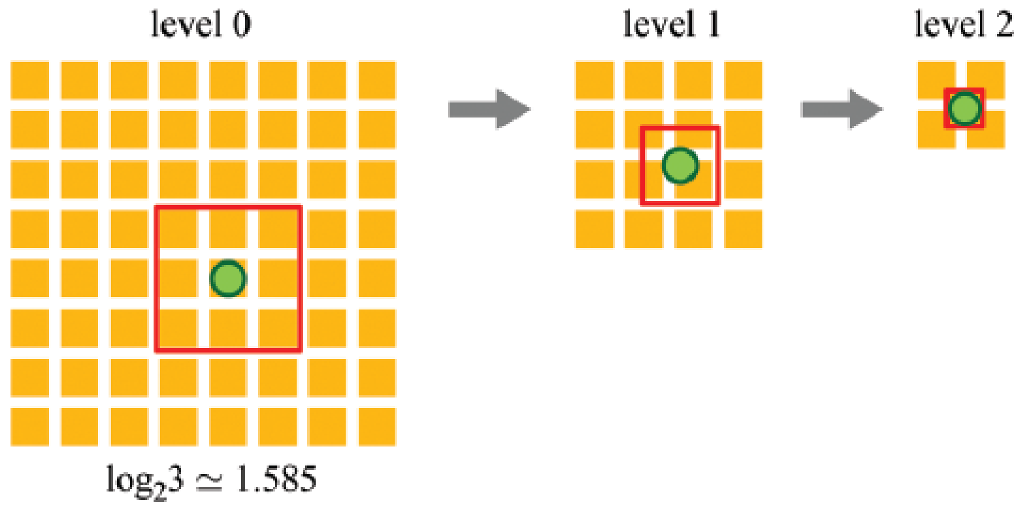


Mipmapping

Level 1.585 example

- In the following example, $\lambda = \log_2 3 ≒ 1.585$, and so we see levels 1 and 2.

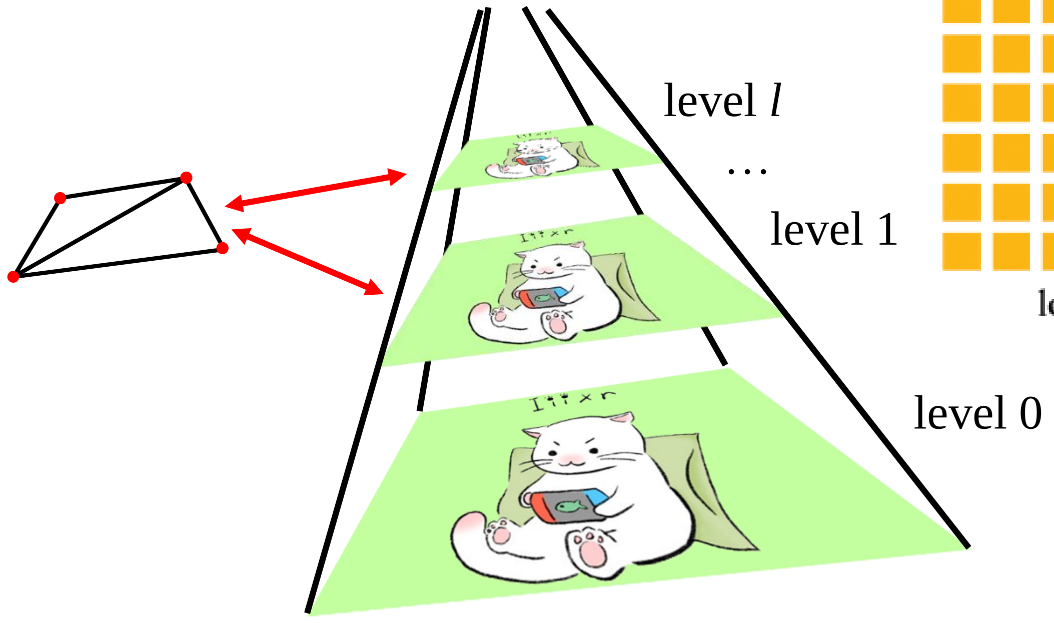


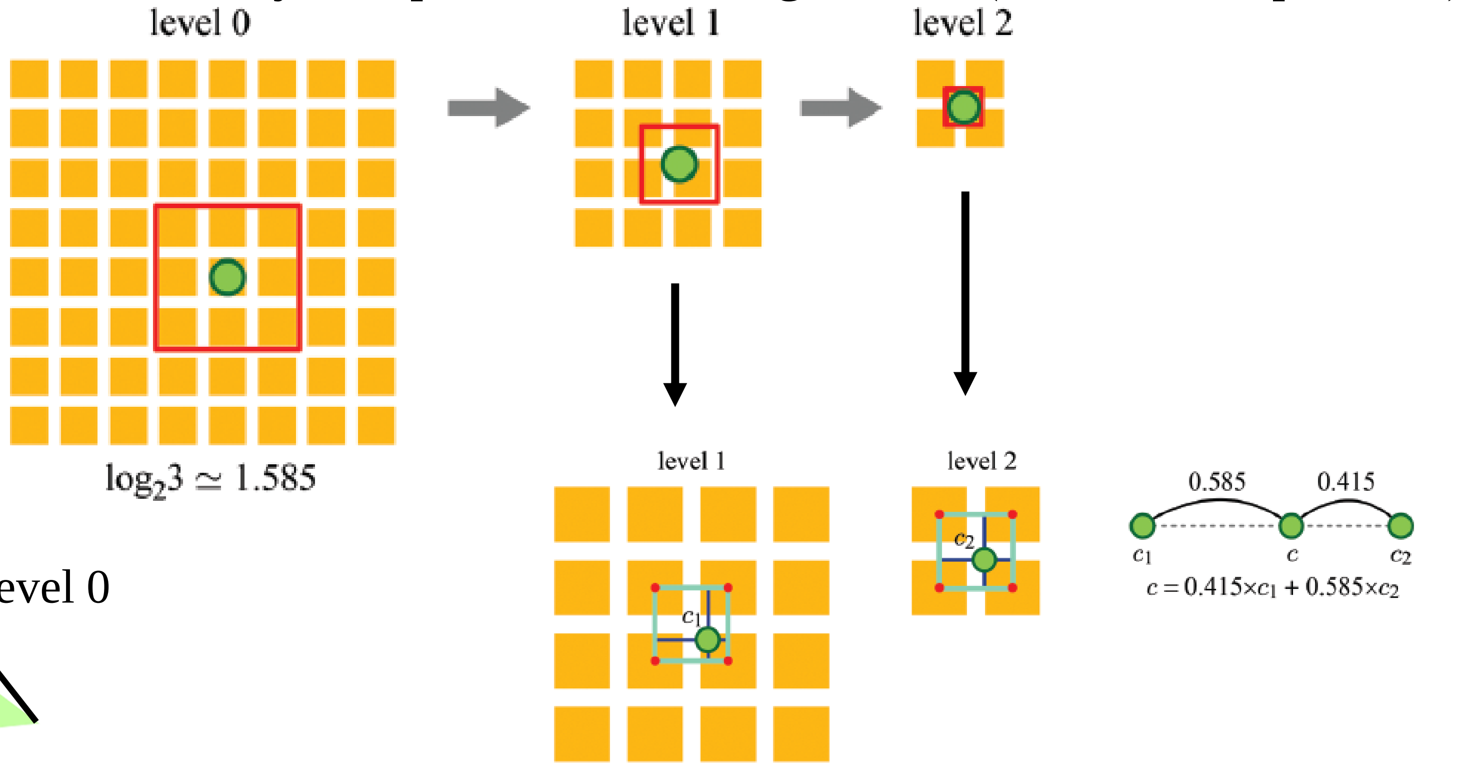


Mipmapping

Level 1.585 example

- Option 1: We can take the nearest level. *l* = round(λ).
- Option 2: we can take both of levels and linearly interpolate the filtering results. (bilinear interpolation)


level *l*
…
level 1
level 0


level 0
level 1
level 2
$\log_2 3 \simeq 1.585$
level 1
c_1
level 2
c_2
0.585
0.415
c_1
c
c_2
$c = 0.415 \times c_1 + 0.585 \times c_2$