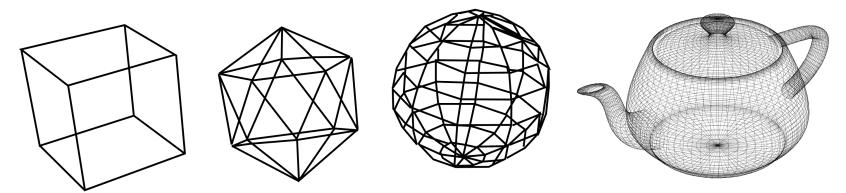




#### **Lecture 6: Texturing**

Lecturer: Dr. NGUYEN Hoang Ha



Reference: JungHyun Han. 2011. 3D Graphics for Game Programming (1st ed.), chapter 8



#### Agenda

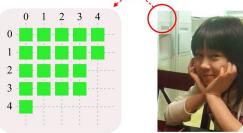
- Revisiting texture concept
- Texture Addressing Mode
- Texture Filtering
- Mipmapping
- Anisotropic Filtering

#### **Texture Coordinates**

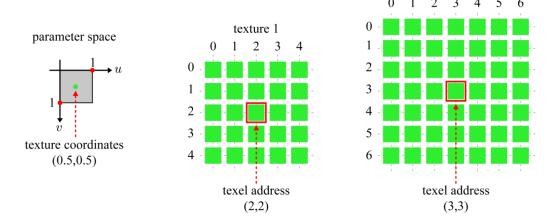


An image texture is a 2D array of *texels* (texture elements). Each texel has a unique address, i.e., 2D array index.





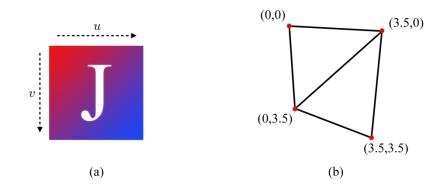
■ Use normalized texture coordinates (u,v) in [0,1] → multiple images of different resolutions can be glued to a surface without changing the texture coordinates<sup>2</sup>

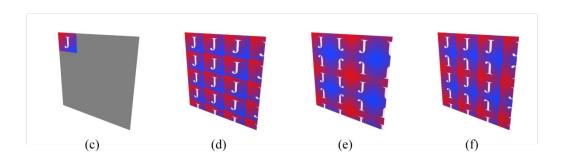


#### Texture Addressing Mode



- The texture coordinates (u,v) are not necessarily in the range of [0,1]. The *texture addressing mode* handles (u,v)s outside the range.
  - Border: The out-of-range coordinates are rendered with a separately defined border color.
  - Wrap, repeat, or tile: The texture is tiled at every integer junction.
  - Mirror: The texture is mirrored or reflected at every integer junction. We have smooth transition at the boundaries.





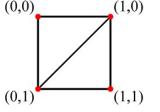
# Texture Filtering – Magnification & Minification



texel

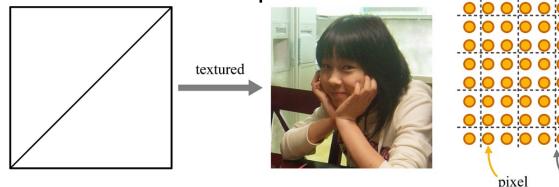
grid

Consider a quad. For each fragment located at (x,y) in the screen, its texture coordinates (u,v) are mapped to the texel address  $(t_x, t_y)$ . We say that the fragment at (x,y) is projected onto  $(t_x, t_y)$ .





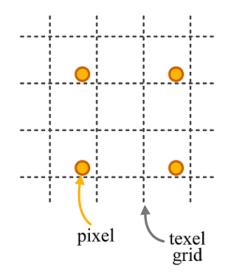
- $(t_x, t_y)$  are floating-point values  $\rightarrow$  texels around  $(t_x, t_y)$  are sampled. This sampling process is called *texture filtering*.
- The screen-space quad may appear larger than the image texture, and therefore the texture is *magnified* so as to fit to the quad. There are more pixels than texels.





In contrast, the screen-space quad may appear smaller than the image texture, and the texture is minified. The pixels are *sparsely projected* onto the texture space.







- Magnification: dense pixels
- Minification : sparse pixels

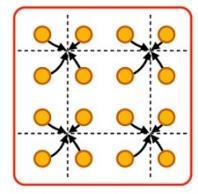
## Filtering for Magnification



- Option 1: Nearest point sampling
  - A block of pixels is 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 produced.



magnified



 $(\lfloor t_x + 0.5 \rfloor, \lfloor t_y + 0.5 \rfloor)$ 



## Filtering for Magnification (cont')

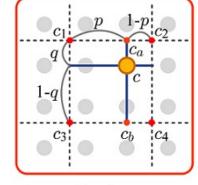


- Option 2: Bilinear interpolation
  - It is preferred to nearest point sampling not only because the final result suffers much less from the blocky image problem but also because the graphics hardware is usually optimized for bilinear interpolation.



magnified

-



$$p = (t_x - \lfloor t_x \rfloor)$$
$$q = (t_y - \lfloor t_y \rfloor)$$

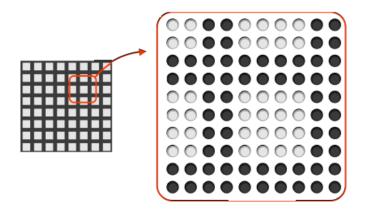
 $c_a = (1-p)c_1 + pc_2$   $c_b = (1-p)c_3 + pc_4$  $c = (1-q)c_a + qc_b$ 



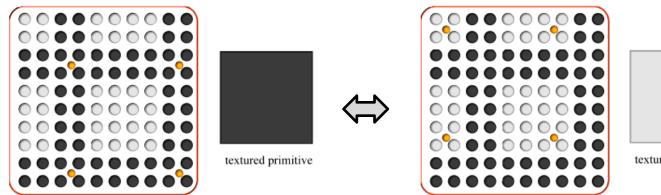
# Filtering for Minification



Consider the following checker-board image texture.



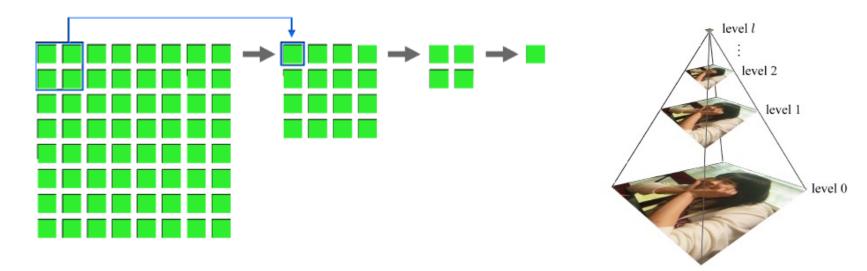
If all pixels are surrounded by dark-gray texels, the textured primitive appears dark gray. If all pixels are surrounded by light-gray texels, the textured primitive appears light gray.







- Reason of minification problem: the texture is larger than the screen-space primitive.
- A solution Decrease the texture size such that there is roughly one-to-one correspondence between pixels and texels.
- For decreasing the texture size, down-sampling is adopted.



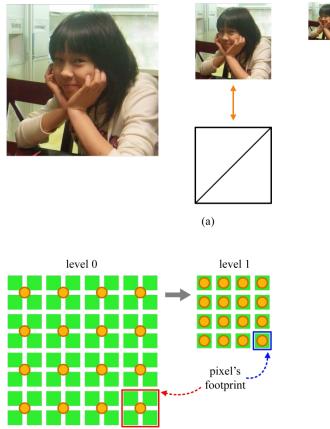
- Given an original texture of 2<sup>1</sup>x2<sup>1</sup> resolution, a texture pyramid of (1 + 1) levels is constructed, where the original texture is located at level 0.
- The pyramid is called a mipmap.
- The level to filter is named *level of detail*, and is denoted by  $\lambda$ .



In the example, the quad and level-1 texture have the same size. A pixel's footprint covers 2x2 texels in level 0, but covers a single texel in level 1.

-2

• When a pixel footprint covers mxm texels of level-0 texture,  $\lambda$  is set to  $\log_2 m$ .





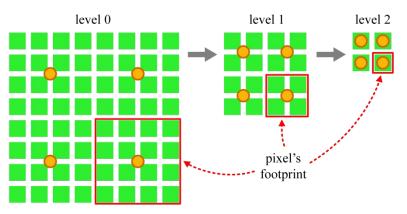
In the example, the screen-space quad and level-2 texture have the same size. A pixel's footprint covers exactly a single texel in level-2 texture, which is then filtered.





level 3

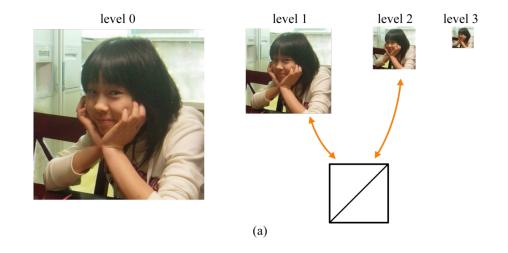
(a)

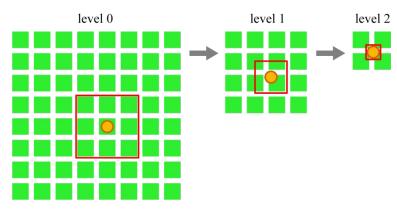


(b)



E.g:  $\lambda = \log_2 3 = 1.585..$ , and so we see levels 1 and 2, more formally  $\lfloor \lambda \rfloor$  and  $\lceil \lambda \rceil$ .





(b)

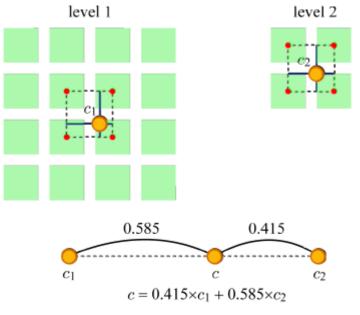
 $\log_2 3 \simeq 1.585$ 



- Between  $\lfloor \lambda \rfloor$  and  $\lceil \lambda \rceil$ , which one to choose?
  - We can take the nearest level.

 $i_{\lambda} = \lfloor \lambda + 0.5 \rfloor$ 

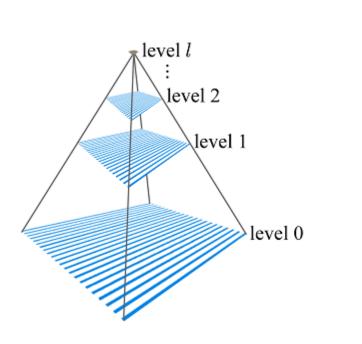
 We can take both of them and linearly interpolate the filtering results. This is called trilinear interpolation.

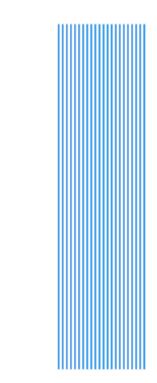


#### Mipmapping - Examples



- Consider a mipmap of stripe texture, and a long thin quad.
- When the quad is orthogonal to the view direction, the texturing result will be as follows.

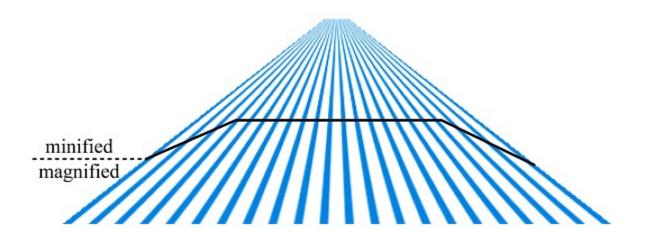




## Mipmapping – Examples (cont')



- What if the quad is angled obliquely away from the viewer?
- In general, for each pixel of the screen-space primitive, it is determined whether the texture is magnified or minified.
- The black lines shown below partition the quad into the magnified and minified parts.
  - Magnification occurs at the lower part of the quad, and level-0 texture is used for texturing.
  - In contrast, minification occurs at the upper part of the quad, and the upper levels of the mipmap may be involved.



# Mipmapping – Examples (cont')

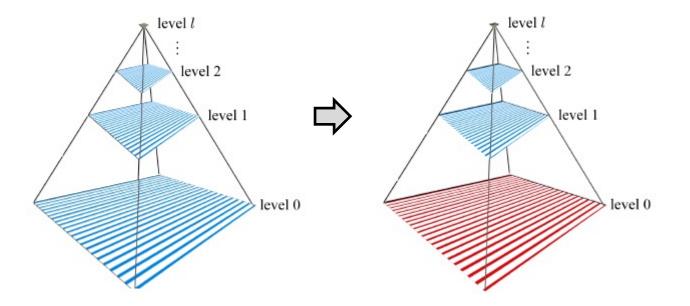


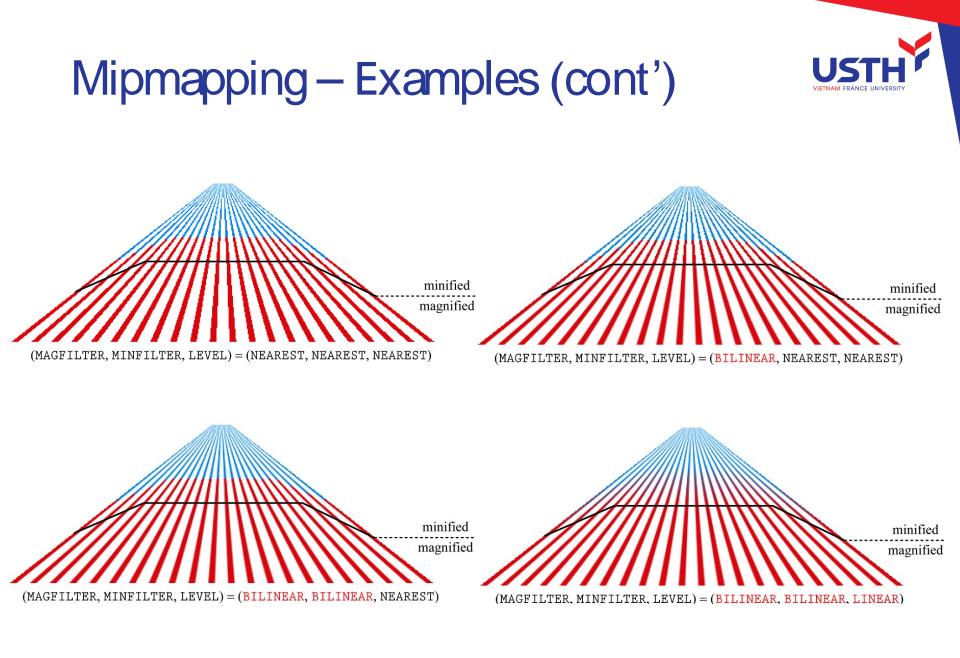
- Mipmapping is implemented in hardware. However, the graphics APIs allow us to control the mipmap filtering mechanism.
- The major control is over magnification filter (MAGFILTER), minification filter (MINFILTER), and mipmap level selection (LEVEL).
  - MAGFILTER
    - Nearest point sampling (NEAREST)
    - Bilinear interpolation (BILINEAR)
  - MINFILTER
    - Nearest point sampling (NEAREST)
    - Bilinear interpolation (BILINEAR)
  - LEVEL
    - Nearest level (NEAREST)
    - Two levels to be linearly interpolated (LINEAR)

#### Mipmapping – Examples (cont')



Let us replace level-0 texture for visualization purpose solely

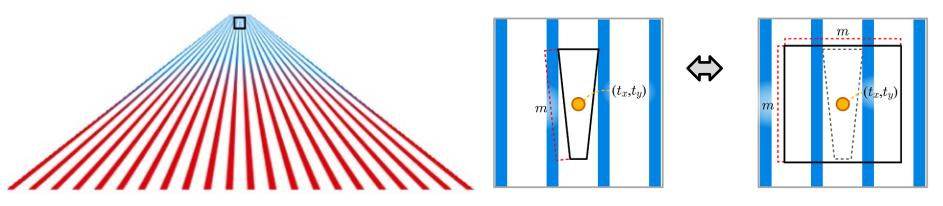




#### Anisotropic Filtering



- Consider a square pixel and its *footprint* in the texture space.
  - A pixel's footprint is an anisotropic quadrilateral in general.
  - To compute  $\lambda$ , traditional mipmapping algorithms often use the longest edge of the anisotropic footprint. Suppose that its length is m. Then,  $\lambda$  is set to  $\log_2 m$ . Filtering the mipmap with such  $\lambda$  is equivalent to taking the isotropic footprint, the size of which is mxm.
  - Observe that the mxm-sized isotropic footprint contains blue texels whereas the anisotropic one does not. Consequently, the textured color of the pixel will be a blend of blue and white even though it is supposed to be just white.
  - This explains why the top part of the textured quad is over-blurred.

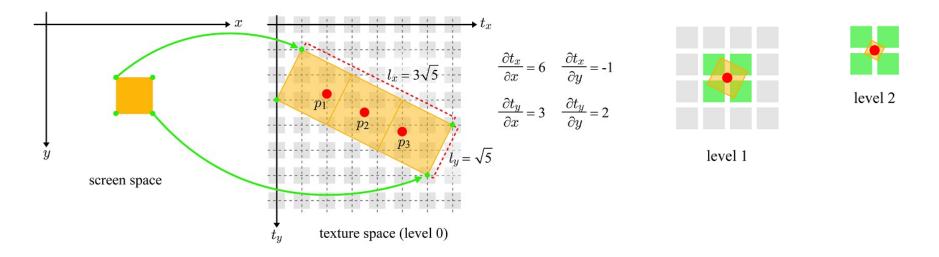


What is desired is *anisotropic filtering*, which filters only the area covered by the anisotropic footprint.

#### Anisotropic Filtering (cont')



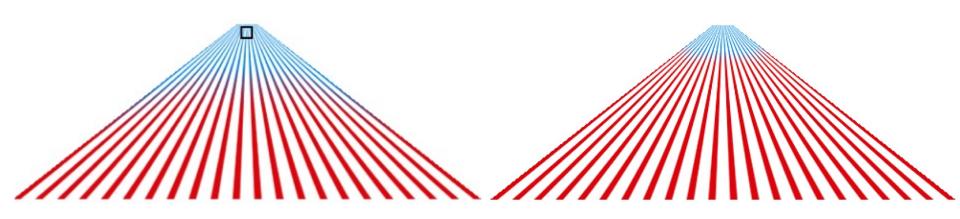
- Anisotropic filtering algorithm an example with a parallelogram footprint
  - The anisotropic footprint is divided into smaller nearly-isotropic footprints, using the lengths of the longer edge and shorter edge.
  - Each of the isotropic footprints is processed separately, and is assigned a sample point.
  - The length of the shorter edge of the original footprint determines the level of detail λ.
  - For the directed level(s), each sample point is used for filtering.
  - All the results are averaged.



#### Anisotropic Filtering (cont')



- Compare the results.
  - The top part of the quad The textured image created only with trilinear interpolation appears blurry toward the horizon. In contrast, the anisotropic filtering preserves the sharpness of the image texture at the area.
  - The area covered by the low-level textures The area of the quad affected by level-0 texture is expanded because λ is determined by the length of the anisotropic footprint's shorter edge, and therefore the lower levels of the mipmap are more frequently sampled.



#### Trilinear Interpolation vs. Anisotropic Filtering





(a)



(b)

#### Texturing with OpenGL



```
data = new unsigned char [imageSize]; // Create a buffer
// then Read the actual data fread(data,1,imageSize,file);
GLuint textureID; // Create one OpenGL texture
glGenTextures(1, &textureID);
// "Bind" the newly created texture : all future texture functions will modify this texture
glBindTexture(GL_TEXTURE_2D, textureID);
```

// Give the image to OpenGL
glTexImage2D(GL\_TEXTURE\_2D, 0,GL\_RGB, width, height, 0, GL\_BGR, GL\_UNSIGNED\_BYTE, data);

```
// ... nice trilinear 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_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
// ... which requires mipmaps. Generate them automatically.
glGenerateMipmap(GL_TEXTURE_2D);
```

```
// Bind our texture in Texture Unit 0
```

```
glBindTexture(GL_TEXTURE_2D, Texture);
```