

---

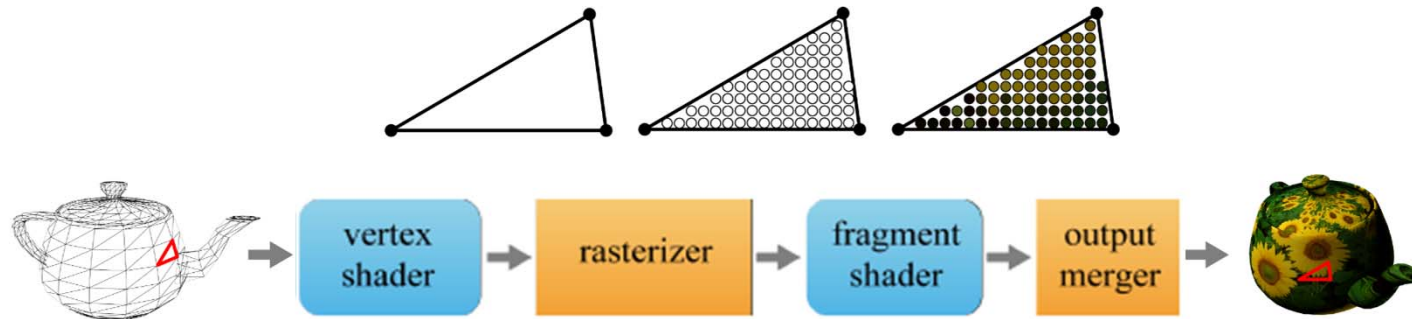
# **Chapters VIII**



## **Image Texturing**

# Where are We?

---

- While the vertex shader outputs the clip-space vertices, the rasterizer outputs a set of fragments at the screen space.



- The per-fragment attributes may include a normal vector, a set of texture coordinates, etc.
  - Using these data, the fragment shader determines the final color of each fragment.
  - Two most important things to do in the fragment shader
    - Lighting 
    - Texturing 
  - Before moving on to the fragment shader, let's see the basics of texturing.
-

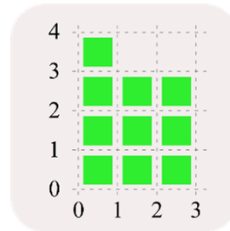
# Texture Coordinates

---

- The simplest among the various texturing methods is *image texturing*.
- An image texture is a 2D array of *texels* (texture elements). A texel's location can be described by its center's coordinates.



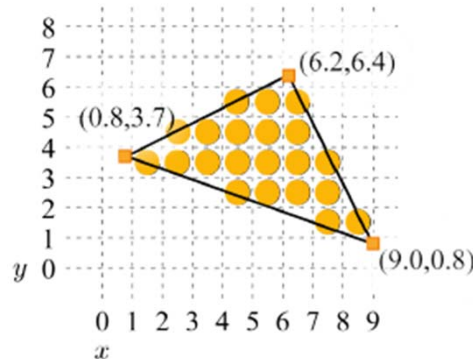
(a)



(b)

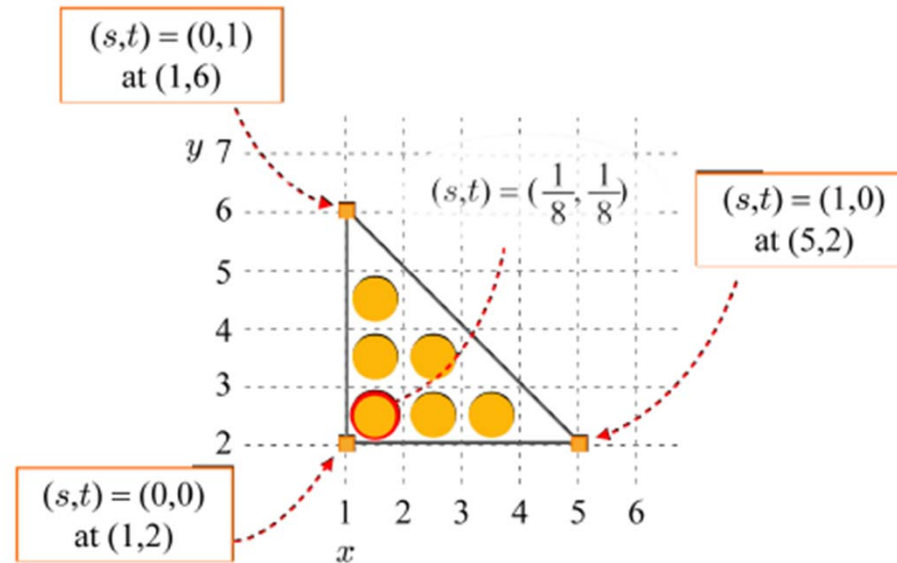


- For each fragment  $p$ , we use its texture coordinates  $(s,t)$  to find a location  $q$  in the texture such that the color at  $q$  can be retrieved and then applied to  $p$ .



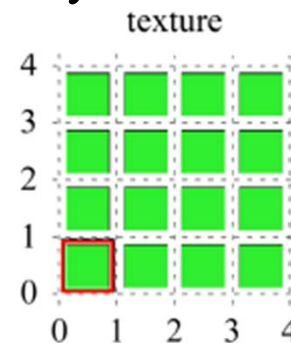
## Texture Coordinates (cont'd)

- Scan conversion is done with the texture coordinates.



- GL multiplies the texture coordinates by the texture image resolution,  $r_x$  and  $r_y$ .

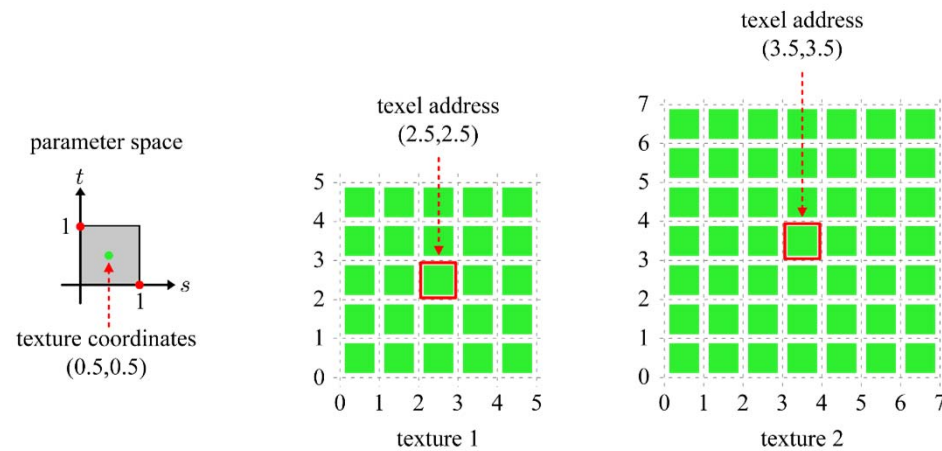
$$s' = s \times r_x$$
$$t' = t \times r_y$$



## Texture Coordinates (cont'd)

---

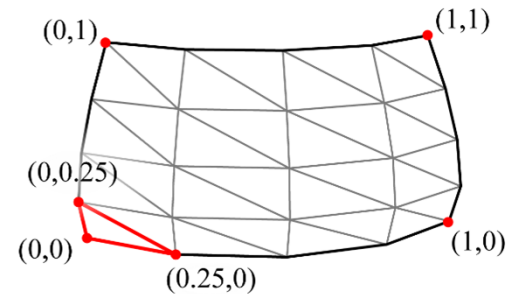
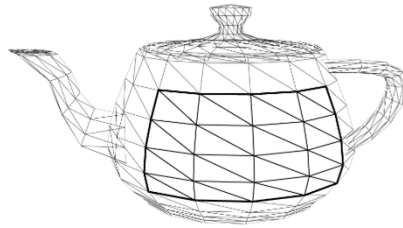
- It is customary to normalize the texture coordinates such that  $s$  and  $t$  range from 0 to 1. The texture coordinates can be freely plugged into different textures.



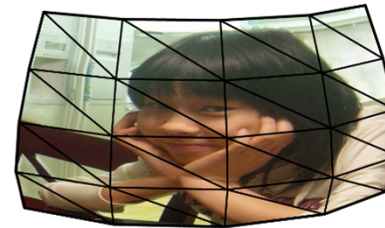
# Texture Coordinates (cont'd)

---

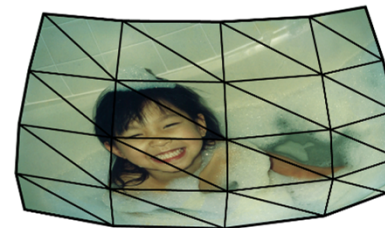
- Multiple images of different resolutions can be glued to a surface without changing the texture coordinates.



(a)



(b)

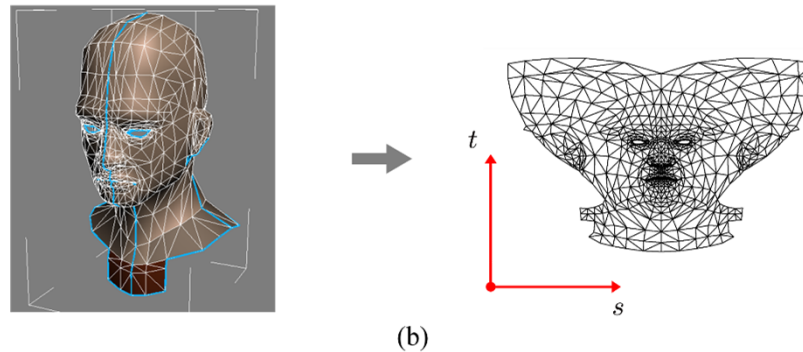
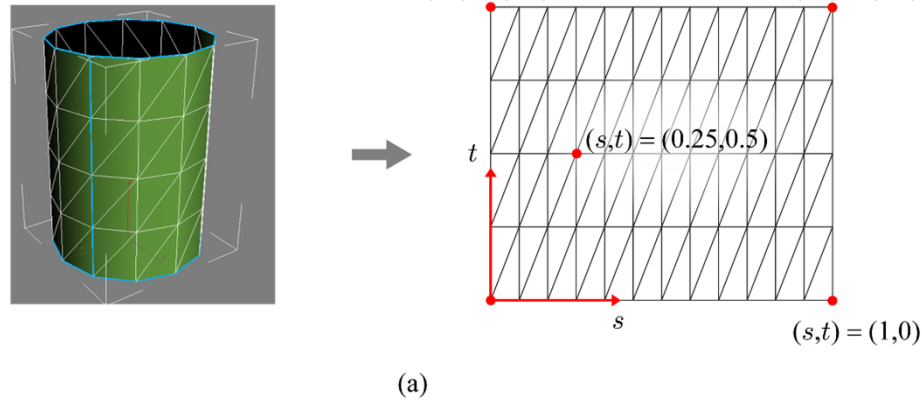


(c)

# Surface Parameterization

---

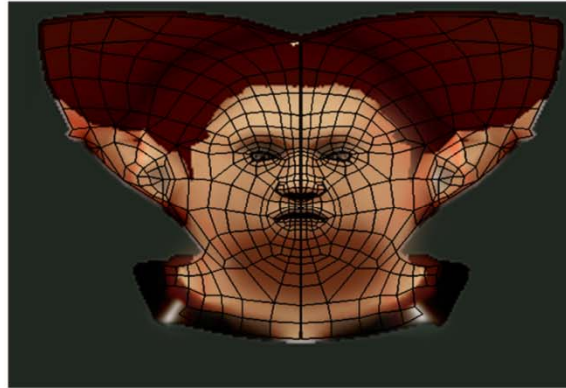
- The texture coordinates are assigned to the vertices of the polygon mesh. This process is called *surface parameterization* or simply *parameterization*.
- In general, parameterization requires unfolding a 3D surface onto a 2D planar domain.



# Chart and Atlas

---

- The surface to be textured is often subdivided into a (hopefully small) number of patches.
- 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*.



(a)



(b)



# *Texturing in GL*

---

- Given an image to be used as a texture, it is first of all loaded into the GL program. Assume that we use `TexData` to describe the structure of the loaded texture and `newTex` is a variable of `TexData`.

```
struct TexData
{
    GLubyte* data;
    GLuint width;
    GLuint height;
    GLint format;
};

TexData newTex;

GLuint oTextures;
glGenTextures(1, &oTextures);
glBindTexture(GL_TEXTURE_2D, oTextures);
glTexImage2D(GL_TEXTURE_2D, 0, newTex.format, newTex.width, newTex.height,
0, newTex.format, GL_UNSIGNED_BYTE, newTex.pixels);
```

- Observe that the above is similar to the process of creating buffer objects for vertex array.
-

## *Texturing in GL (cont'd)*

---

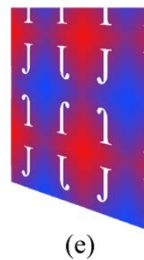
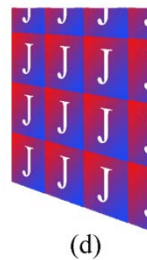
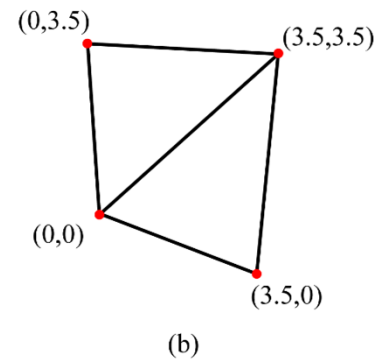
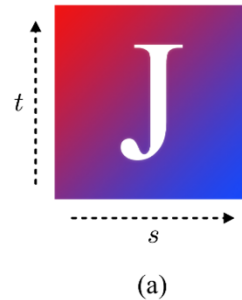
```
void          glTexImage2D(GLenum target,
                           GLint level,
                           GLint internalFormat,
                           GLsizei width,
                           GLsizei height,
                           GLint border,
                           GLenum format,
                           GLenum type,
                           const GLvoid * data);
```

target	GL_TEXTURE_2D for 2D image texture
level	the mipmap level (later on this)
internalFormat	the number of color components such as GL_RGBA
width	the width of the texture image
height	the height of the texture image
border	OpenGL's legacy argument that must be 0.
format	the pixel format such as GL_RGBA
type	the pixel data type such as GL_UNSIGNED_BYTE
data	the pointer to the image data in memory

# Texture Wrapping Mode

---

- The texture coordinates  $(s,t)$  are not necessarily in the range of  $[0,1]$ . The *texture wrapping mode* handles  $(s,t)$  that are outside 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-Repeat*: The texture is mirrored or reflected at every integer junction.We have smooth transition at the boundaries.



## *Texture Wrapping Mode (cont'd)*

---

```
void    glTexParameteri(GLenum target,  
                        GLenum pname,  
                        GLint param);
```

target target is GL\_TEXTURE\_2D for 2D image texture

pname either GL\_TEXTURE\_WRAP\_S or GL\_TEXTURE\_WRAP\_T for texture wrapping

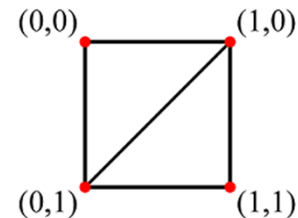
param either GL\_CLAMP\_TO\_EDGE, GL\_REPEAT, or GL\_MIRRORED\_REPEAT

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

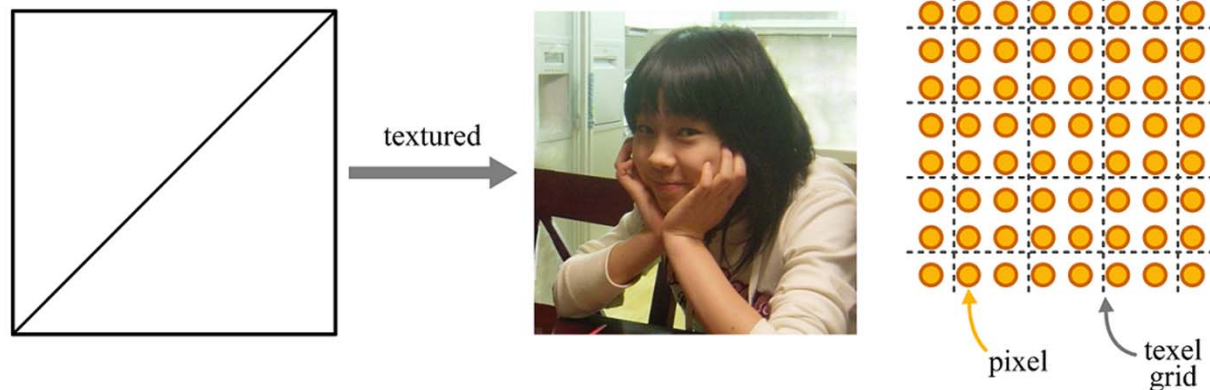
# Texture Filtering – Magnification & Minification

---

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

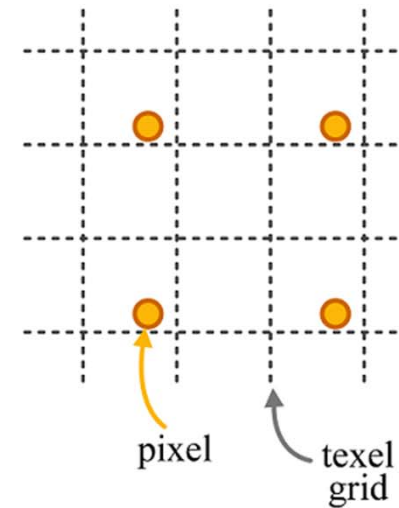


- Note that  $(s',t')$  are floating-point values in almost all cases.
- Consequently, texels around  $(s',t')$  are sampled. This sampling process is called *texture filtering*.
- The screen-space quad may appear larger than the image texture, and therefore the texture is *magnified* so as to fit to the quad. There are more pixels than texels.



## Texture Filtering – Magnification & Minification (cont'd)

- 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.



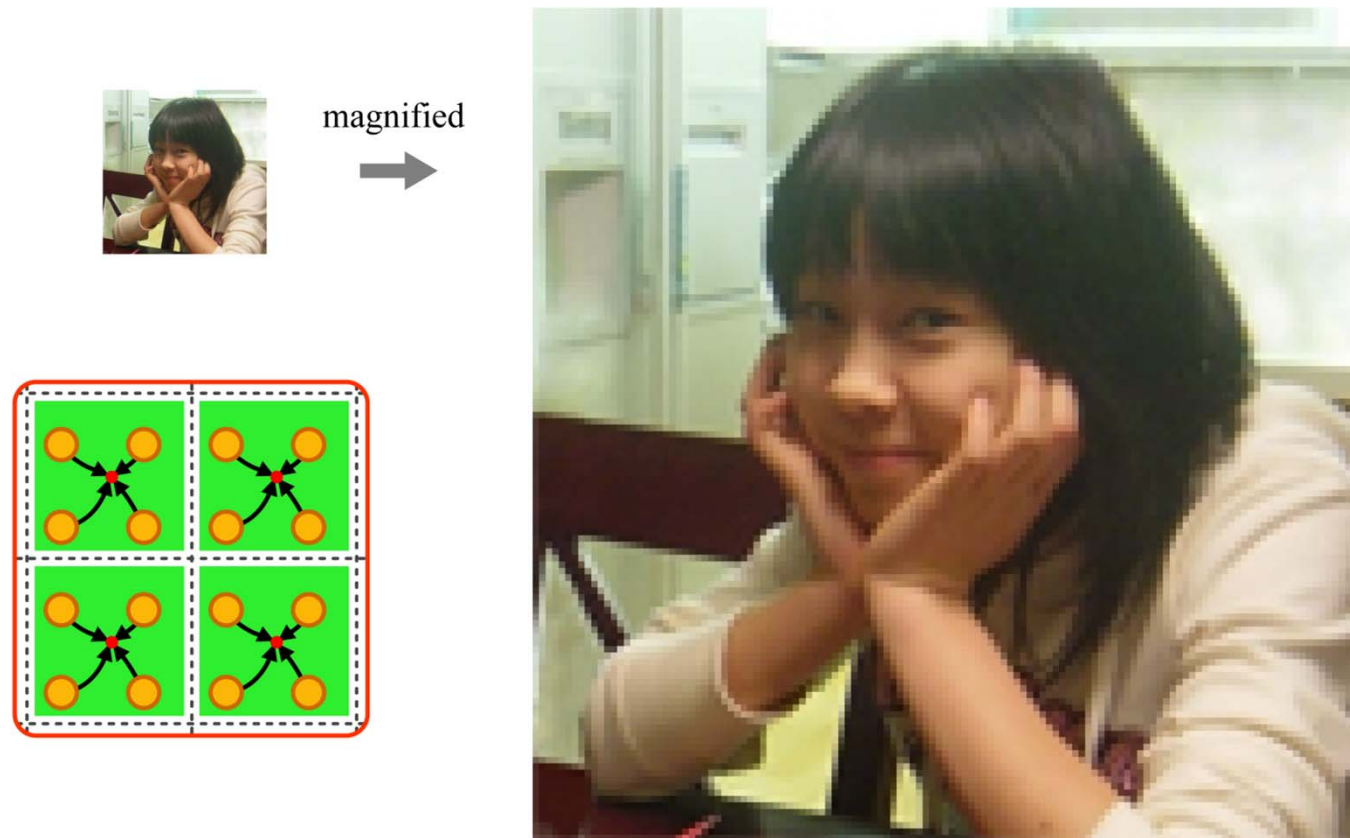
- Summary
  - Magnification: more pixels than texels
  - Minification : less pixels than texels



# *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 produced.



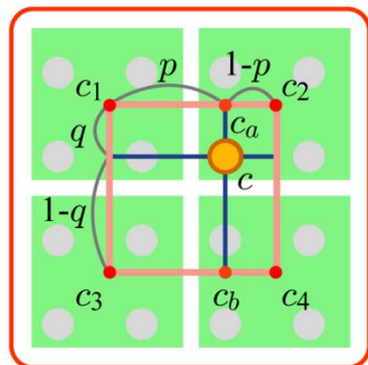
## *Filtering for Magnification (cont'd)*

---

- 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



$$\begin{aligned}c_a &= (1-p)c_1 + pc_2 \\c_b &= (1-p)c_3 + pc_4 \\c &= (1-q)c_a + qc_b\end{aligned}$$



## *Filtering for Minification*

---

- Consider a checker-board 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.

