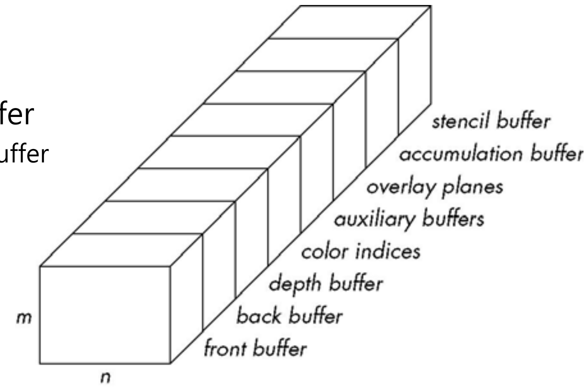
Buffer, Image, and Texture Mapping

Fall 2021
11/16/2021
Kyoung Shin Park
Computer Engineering
Dankook University

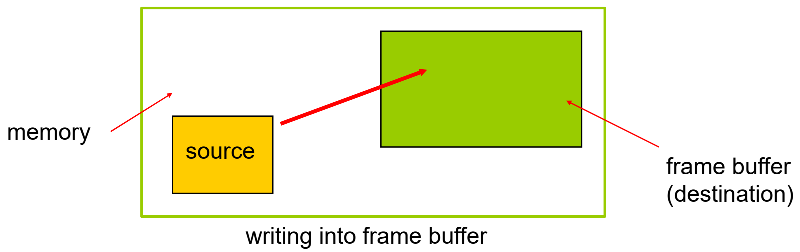
OpenGL Frame Buffer

- Color buffers
  - Front buffer
  - Back buffer
  - Auxiliary buffer
  - Overlay plane
- Depth buffer
- Accumulation buffer
  - High resolution buffer
- Stencil buffer
  - Holds masks



Writing in Buffers

- Conceptually, we consider all of memory as a large two-dimensional array of pixels
- We read and write rectangular block of pixels
  - Bit block transfer (bitblt) operations
- The frame buffer is part of this memory



Buffer Selection

- OpenGL can draw into or read from any of the color buffers (front, back, auxiliary)
- Default to the back buffer
- Change with glDrawBuffer and glReadBuffer
- Note that format of the pixels in the frame buffer is different from that of processor memory and these two types of memory reside in different places
  - Need packing and unpacking
  - Drawing and reading can be slow

Pixel Maps

- OpenGL works with rectangular arrays of pixels called pixel maps or images
- Pixels are in one byte (8 bit) chunks
  - Luminance (gray scale) images 1 byte/pixel
  - RGB 3 bytes/pixel
- Three functions
  - Draw pixels: processor memory to frame buffer
  - Read pixels: frame buffer to processor memory
  - Copy pixels: frame buffer to frame buffer

OpenGL Pixel Functions

```
glReadPixels(x,y,width,height,format,type,myimage)
```

start pixel in frame buffer — size — type of image — type of pixels — pointer to processor memory

```
GLubyte myimage[512][512][3];
glReadPixels(0,0, 512, 512, GL_RGB,
      GL_UNSIGNED_BYTE, myimage);

glDrawPixels(width,height,format,type,myimage)
```

starts at raster position

OpenGL Buffer Management Functions

- Buffer clear
  - glClear(GLbitfield mask); // clear a specified buffer
  - GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT | GL_ACCUM_BUFFER_BIT | GL_STENCIL_BUFFER_BIT
  - glClearBuffer();
- Buffer clear value set
  - glClearColor(); glClearDepth(); glClearDepthf(); glClearStencil();
- Buffer mask (i.e., enabled or disabled)
  - glColorMask[i](GLboolean red, GLboolean green, GLbooleanblue, GLboolean alpha); // set r,g,b,a color in frame buffer
  - glDepthMask(GLboolean flag); // set depth in depth buffer
  - glStencilMask(GLuint mask); // set bit mask in stencil buffer

Images

- Read image data from file or create the image data
- General image format:
  - JPEG, TIFF, PNG, GIF, RGB, EPS, BMP, etc
- Image format:
  - Color channel: greyscale, RGB, RGBA
  - Bit resolution
  - Compression: lossy coding, lossless coding

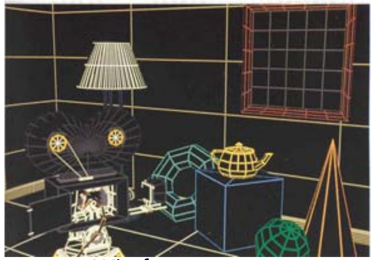
COIN3D simage

- http://www.coin3d.org/lib/simage
- COIN3D simage library support following image format
  - JPEG, TIFF, PNG, PIC, TGA, EPS, GIF, RGB, etc
- To COIN3D simage library, need to add additional library and include directory in your project
  - Project -> Properties(ALT+F7) -> Configuration Properties -> C/C++ -> General ->Additional Include Directories -> add **.₩include**
  - Project -> Properties(ALT+F7) -> Configuration Properties -> C/C++ -> Preprocessor -> Preprocessor Definitions add ;**SIMAGE_DLL**
  - Project -> Properties(ALT+F7) -> Configuration Properties -> Linker -> General -> Additional Library Directories -> add **.₩lib₩debug**
  - Project -> Properties(ALT+F7) -> Configuration Properties - > Linker -> Input -> Additional Dependencies -> add **simage1.lib**

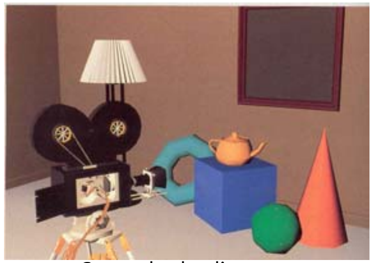
COIN3D simage Example

```
unsigned char *imgPtr;
unsigned char *imageData;
unsigned char *rescaledImageData;
int imageWidth = 0, imageHeight = 0, numComponents = 0;
imageData = simage_read_image (filename, &imageWidth,
                                    &imageHeight, &numComponents); // read
GLsizei xdim2,ydim2;          // if the image size is not the power of 2, resize it
GLenum type;
xdim2 = 1;
while (xdim2 <= imageWidth)
        xdim2 *= 2;
xdim2 /= 2;
ydim2 = 1;
while (ydim2 <= imageHeight)
        ydim2 *= 2;
ydim2 /= 2;
if ((imageWidth != xdim2) || (imageHeight != ydim2))   {
       rescaledImageData = simage_resize(imageData, imageWidth, imageHeight,
                                       numComponents, xdim2, ydim2);
        imgPtr = rescaledImageData;
}    else
        imgPtr = imageData;
```

Texture Mapping


Wireframe


Flat shading


Smooth shading


Texture mapping

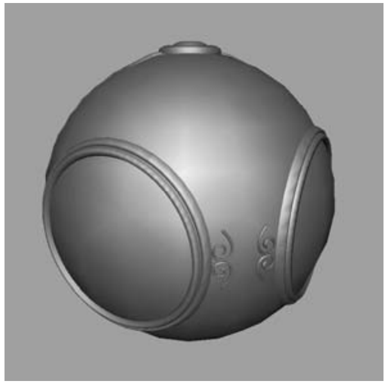
The Limits of Geometric Modeling

- Although graphics cards can render over 10 million polygons per second, that number is insufficient for many phenomena
  - Clouds
  - Grass
  - Terrain
  - Skin
- Texture Mapping
  - Two-dimensional image is applied directly to a surface
  - In real-time graphics rendering where a limited number of polygons must be used, texture mapping is a technique that can significantly increase the realism with a relatively small additional cost.

Three Types of Mapping

- Texture Mapping
  - Uses images to fill inside of polygons.
- Environment/Reflection mapping
  - Uses a picture of environment for texture maps.
  - Allows simulation of highly specular surfaces.
- Bump mapping
  - Emulates altering normal vectors during the rendering process.

Texture Mapping

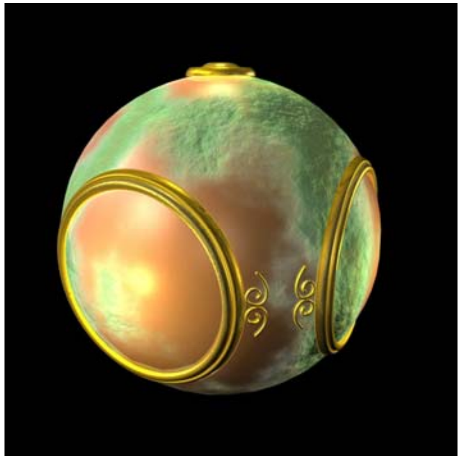


geometric model

texture mapped

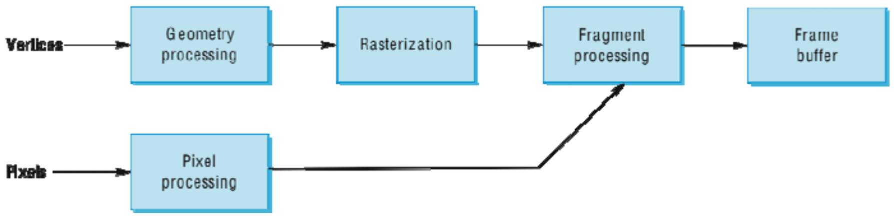
Environment Mapping

Bump Mapping



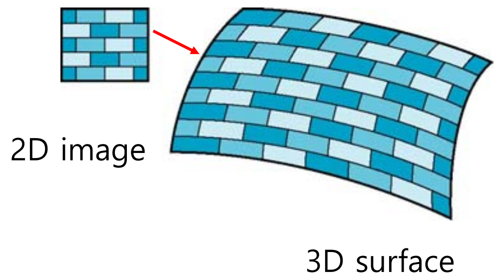
Where does texture mapping take place?

- Mapping techniques are implemented at the end of the rendering pipeline
  - Very efficient because few polygons make it past the clipper



Is it simple?

- Although the idea is simple - map an image to a surface.
  - There are 3 or 4 coordinate systems involved



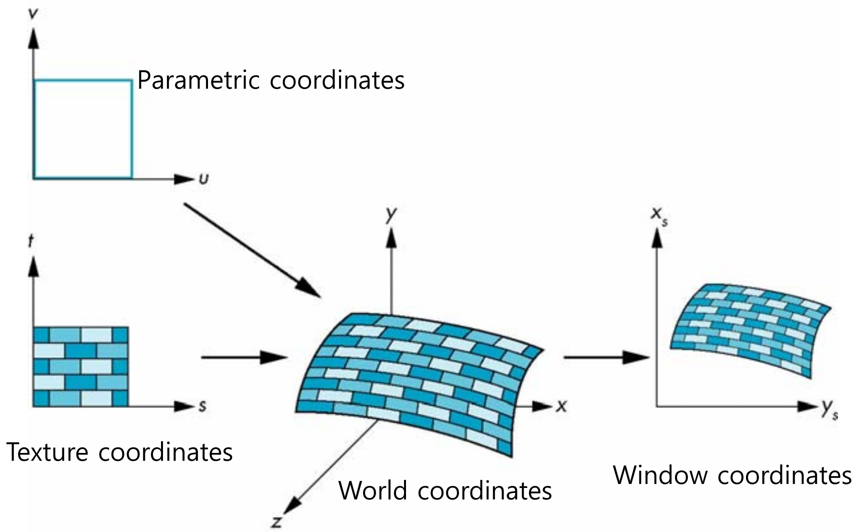
Texture Mapping

- Conceptual 2D texture mapping process
  - Surface parameterization
    - How to apply a texture image to an object?
    - The coordinates of the texture image mapped to each point of the object are required. (x_0, y_0, z_0) => (x_t, y_t)
  - Geometric transformation
    - Geometric transformation determines the mapping relationship between each point of an object and its position on the projection screen. (x_0, y_0, z_0) => (x_s, y_s)
  - Rasterization
    - The process of finding pixels on which each geometric object is projected
  - Texture color calculation
    - The process of painting each pixel with a texture color appropriately
    - How to calculate the texture color visible through each pixel?
    - How to blend the calculated texture color with the original color of the object?

Coordinate Systems

- Parametric coordinates (u, v)
  - May be used to model curves & surfaces
- Texture coordinates (s, t)
  - Used to identify points in the image to be mapped
- Object or World Coordinates (x, y, z)
  - Conceptually, where the mapping takes place
- Window Coordinates (x_s, y_s)
  - Where the final image is really produced

Texture Mapping



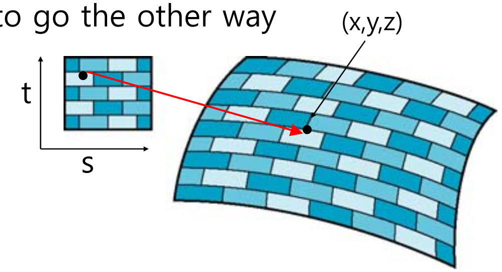
Mapping Functions

- Basic problem is how to find the maps
- Consider mapping from texture coordinates to a point a surface
- Appear to need three functions

  x = x(s,t)

  y = y(s,t)

  z = z(s,t)
- But we really want to go the other way

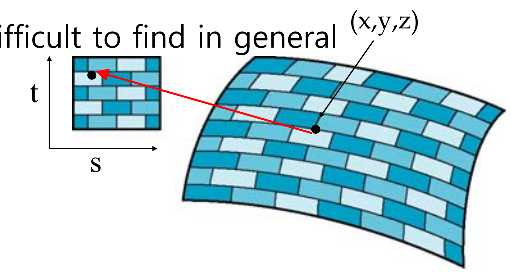


Backward Mapping

- We really want to go backwards
  - Given a pixel, we want to know to which point on an object it corresponds
  - Given a point on an object, we want to know to which point in the texture it corresponds
- Need a map of the form

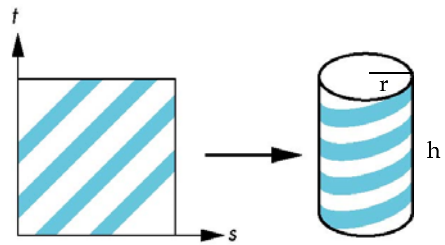
  s = s(x,y,z)

  t = t(x,y,z)
- Such functions are difficult to find in general



Two-part mapping

- Two-part mapping
  - One solution to the mapping problem is to first map the texture to a simple intermediate surface
- Example: first, map to cylinder



Cylindrical Mapping

- Parametric cylinder

  $x = r \cos 2\pi u$ u: (0,1)

  $y = r \sin 2\pi u$ v: (0,1)

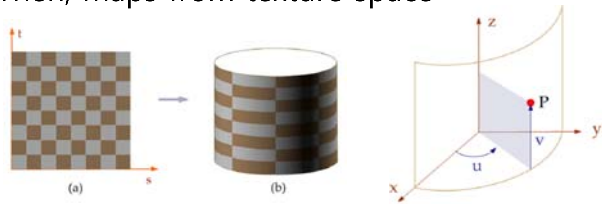
  $z = v/h$

- Maps rectangle in u,v space to cylinder of radius r and height h in world coordinates

  $s = u$

  $t = v$

- Then, maps from texture space



$x = r \cos 2\pi s$
$y = r \sin 2\pi s$
$z = t/h$

Spherical Map

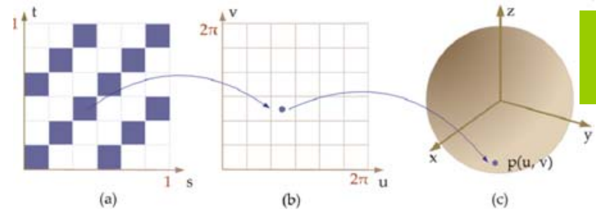
- We can use a parametric sphere

  $x = r \cos 2\pi u$

  $y = r \sin 2\pi u \cos 2\pi v$

  $z = r \sin 2\pi u \sin 2\pi v$

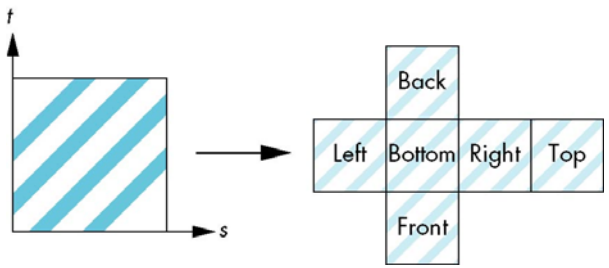
- In a similar manner to the cylinder but have to decide where to put the distortion
  - Mercator projection creates the largest distortion at both poles.
- Spherical mapping is used in environmental maps.



$x = r \cos 2\pi s$
$y = r \sin 2\pi s \cos 2\pi t$
$z = r \sin 2\pi s \sin 2\pi t$

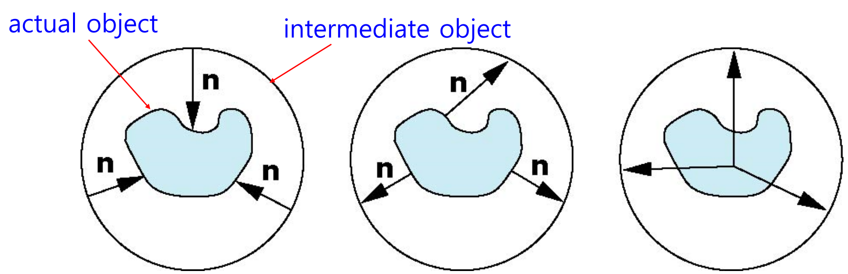
Box Mapping

- Easy to use with simple orthographic projection
- Also used in environment maps



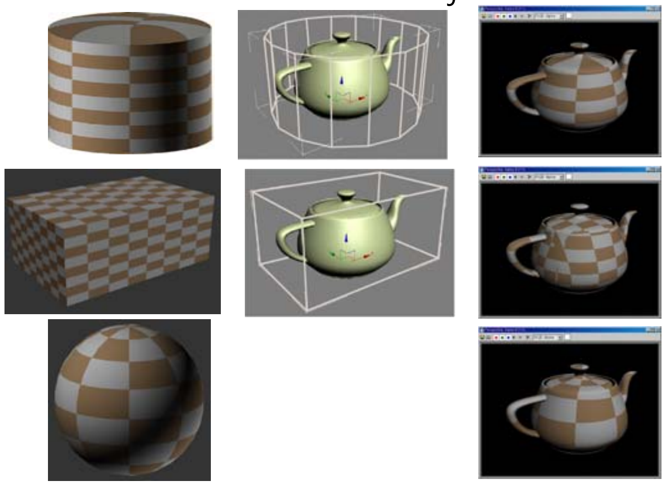
Second Mapping

- Map from intermediate object to actual object
  - Normals from intermediate to actual
  - Normals from actual to intermediate
  - Vectors from center of intermediate



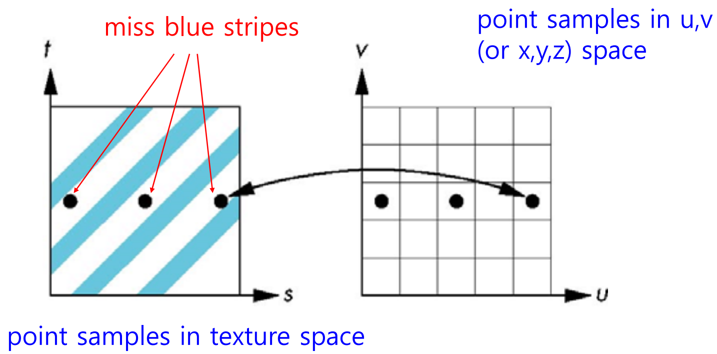
Second Mapping

- Put the object inside the mediation surface and apply texture to the surface of the object.



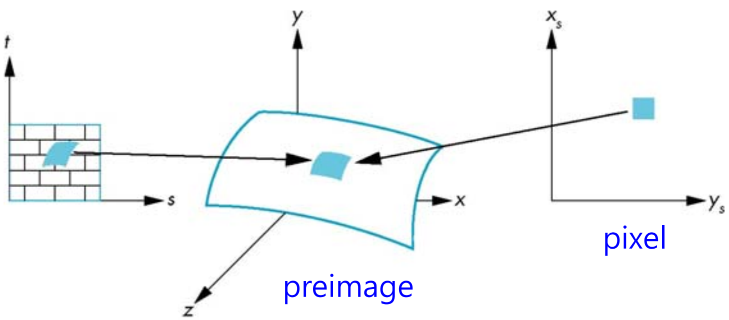
Aliasing

- Point sampling of the texture can lead to aliasing errors
  - Point sampling – point to point mapping



Area Averaging

- A better but slower option is to use area averaging
  - Area Averaging – area to area mapping

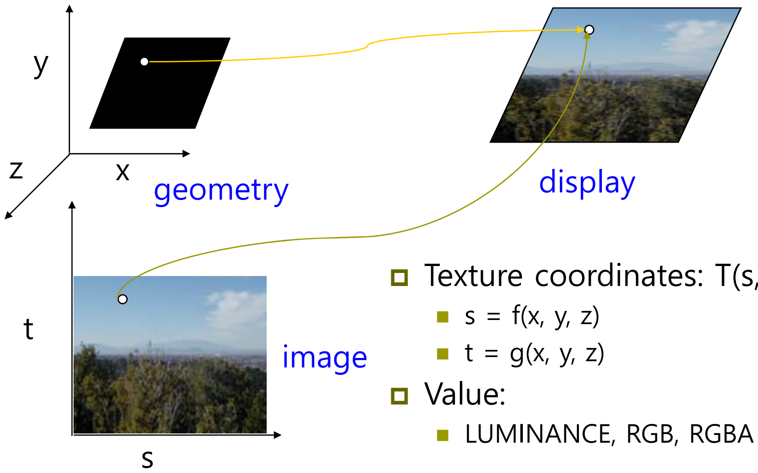


  - Note: the *preimage* of pixel is curved

Basic Strategy

- Three steps to applying a texture
  1. Specify the texture
     - read or generate image
     - assign to texture
     - enable texturing
  2. Assign texture coordinates to vertices
     - Proper mapping function is left to application
  3. Specify texture parameters
     - wrapping
     - filtering

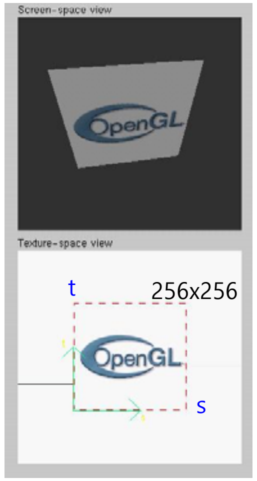
Texture Mapping



- Texture coordinates: T(s, t)
  - s = f(x, y, z)
  - t = g(x, y, z)
- Value:
  - LUMINANCE, RGB, RGBA

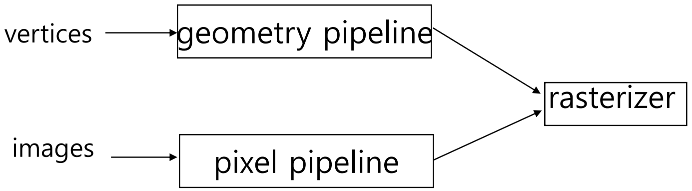
OpenGL Texture Example

- The texture (below) is a 256 x 256 image that has been mapped to a rectangular polygon which is viewed in perspective.



Texture Mapping in the OpenGL Pipeline

- Images and geometry flow through separate pipelines that join at the rasterizer
  - "complex" textures do not affect geometric complexity



Specifying a Texture Image in OpenGL

- Define a texture image from an array of texels (texture elements) in CPU memory
  - GLubyte imageData[512][512];
- Define as any other pixel map
  - Scanned image
  - Generate by application code
- Enable texture mapping
  - glEnable(GL_TEXTURE_2D)
  - OpenGL supports 1-4 dimensional texture maps

Define Image as a Texture

- glTexImage2D(target, level, components, width, height, border, format, type, texels);
  - target: texture type, e.g., GL_TEXTURE_2D
  - level: mipmapping level
  - components: texel components, e.g., RGB
  - width, height: texel width and height (in pixels)
  - border: used for smoothing
  - format, type: texel format and type
  - texels: texels pointer

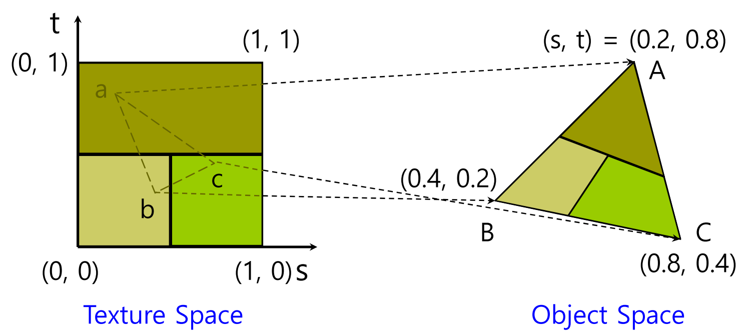
```
glTexImage2D(GL_TEXTURE_2D, 0, RGB, imageWidth, imageHeight,
             0, GL_RGB, GL_UNSIGNED_BYTE, imageData);
```

Converting A Texture Image

- OpenGL requires texture dimensions to be powers of 2
  - 64x64, 64x128, 512x512, …
- If dimensions of image are not powers of 2
  - gluScaleImage(format, w_in, h_in, type_in, *data_in, w_out, h_out, type_out, *data_out);
    - data_in – the original image data
    - data_out – the resized image data
- Image interpolated and filtered during scaling

Mapping a Texture

- Based on parametric texture coordinates
- Texture coordinates must be specified for each vertex



Mapping a Texture

- Define texture coordinates per vertex

```
// Square Vertices Positions
squareVertices.push_back(glm::vec3(-0.75f, -0.75f, 0.0f));
squareVertices.push_back(glm::vec3(0.75f, -0.75f, 0.0f));
squareVertices.push_back(glm::vec3(0.75f, 0.75f, 0.0f));
squareVertices.push_back(glm::vec3(-0.75f, 0.75f, 0.0f));
// Square Vertices Texture Coordinates
squareTextureCoords.push_back(glm::vec2(0.0f, 0.0f));
squareTextureCoords.push_back(glm::vec2(1.0f, 0.0f));
squareTextureCoords.push_back(glm::vec2(1.0f, 1.0f));
squareTextureCoords.push_back(glm::vec2(0.0f, 1.0f));
```

- Note: use vertex array for code efficiency

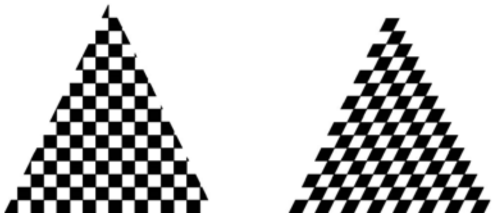
Interpolation

- OpenGL uses interpolation to find proper texels from specified texture coordinates
- Can be distortions

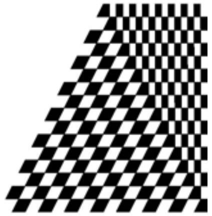
good selection of tex coordinates

poor selection of tex coordinates

texture stretched over trapezoid showing effects of bilinear interpolation



Checkerboard texture mapping on a triangle



Checkerboard texture Mapping on a trapezoid

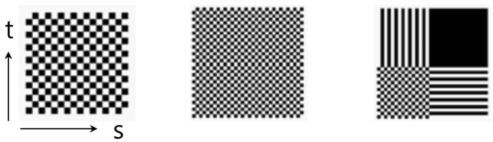
Texture Parameters

- OpenGL has a variety of parameters that determine how texture is applied
  - Wrapping – Wrapping parameters determine what happens if s and t are outside the (0,1) range, e.g. CLAMP, REPEAT
  - Filter modes – Filter modes allow us to use area averaging instead of point samples
  - Mipmapping – Mipmapping allows us to use textures at multiple resolutions
  - Environment parameters – Environment parameters determine how texture mapping interacts with shading

Wrapping Mode

- Clamp: adjustment of the value within the range (0, 1)
  - If s and t are greater than 1, use 1
  - If s and t are less than 0, use 0
- Repeat: repeat texture for values outside the range (0, 1)
  - Use s %1 and t % 1

```
glTexParameteri(GL_TEXTURE_2D,
                GL_TEXTURE_WRAP_S, GL_CLAMP)
glTexParameteri(GL_TEXTURE_2D,
                GL_TEXTURE_WRAP_T, GL_REPEAT)
```



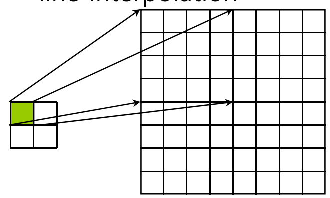
texture

GL_REPEAT wrapping

GL_CLAMP wrapping

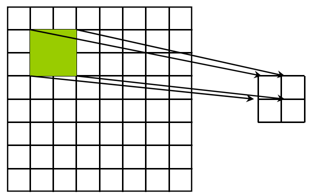
Magnification and Minification

- More than one texel can cover a pixel (**minification**)
- More than one pixel can cover a texel (**magnification**)
- Can use point sampling (nearest) or linear filtering
  - linear filtering – use the weighted average of texel groups including neighbors of texels determined by point sampling
  - nearest – use the nearest texel value to the value calculated by line interpolation



Texture Polygon

Magnification



Texture Polygon

Minification

Filter Modes

- Define min/mag filters
  - glTexParameteri(target, type, mode)
  - glTexParameteri(GL_TEXTURE_2D, GL_TEXURE_MAG_FILTER, GL_NEAREST);
  - glTexParameteri(GL_TEXTURE_2D, GL_TEXURE_MIN_FILTER, GL_LINEAR);

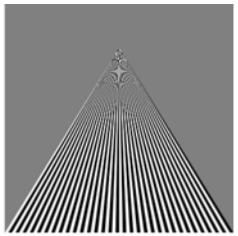
Note that linear filtering requires a border of an extra texel for filtering at edges (border = 1)

Mipmapped Textures

- **Mipmapping** allows for prefiltered texture maps of decreasing resolutions
- Lessens interpolation errors for smaller textured objects
- Declare mipmap level during texture definition
  - glTexImage2D(GL_TEXTURE_*D, level, …)
- Automatically create mipmap textures
  - **glGenerateMipmap(GL_TEXTURE_2D)**
- Use the following options to take advantage of optimal mipmapping and point sampling in OpenGL
  - glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST_MIPMAP_NEAREST)
  - glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR)

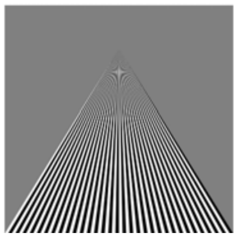
Aliasing Example

point sampling



linear filtering

mipmapped point sampling



mipmapped linear filtering

Texture Environment

- Controls how texture is applied
  glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, mode);
  - GL_TEXTURE_ENV_MODE modes:
    - GL_MODULATE: modules with computed shade
    - GL_DECAL: use only texture color
    - GL_BLEND: blends with an environmental color
    - GL_REPLACE: use only texture color
  - GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE
- Set blend color with GL_TEXTURE_ENV_COLOR

Generating Texture Coordinates

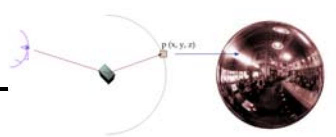
- OpenGL can generate texture coordinates automatically
  glTexGen{ifd}[v](GL_S/T, GL_TEXTURE_GEN_MODE, modes);
  - Specify a plane – generate texture coordinates based upon distance from the plane
  - Generation modes:
    - GL_OBJECT_LINEAR
    - GL_EYE_LINEAR
    - GL_SPHERE_MAP (used for environmental maps)

```
Glfloat planes[] = {0.5, 0.0, 0.0, 0.5} // s=x/2 + ½
Glfloat planet[] = {0.0, 0.5, 0.0, 0.5} // t=y/2 + ½
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);
glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);
glTexGenfv(GL_S, GL_OBJECT_LINEAR, planes);
glTexGenfv(GL_T, GL_OBJECT_LINEAR, planet);
```

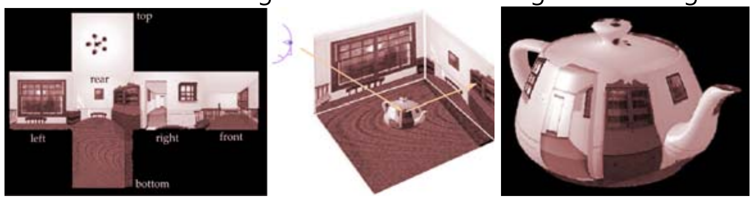
Texture Objects

- Texture is part of the OpenGL state
  - If we have different textures for different objects, OpenGL will be moving large amounts data from processor memory to texture memory
- Recent versions of OpenGL have texture objects
  - One image per texture object
  - Texture memory can hold multiple texture objects

Environment Mapping

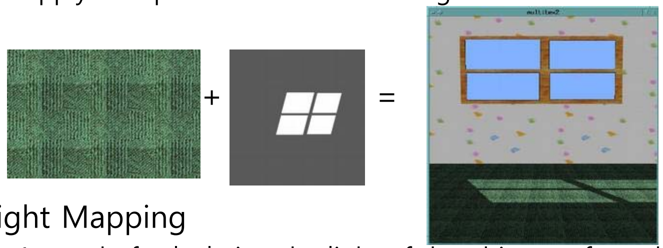


- Environment Maps
  - Start with image of environment through a wide angle lens
    - Can be either a real scanned image or an image created in OpenGL
  - Use this texture to generate a spherical map
  - Use automatic texture coordinate generation
  - Spherical environment mapping – Using automatic texture coordinate generation after creating a spherical map from an environment image taken with a 180 degree wide angle lens

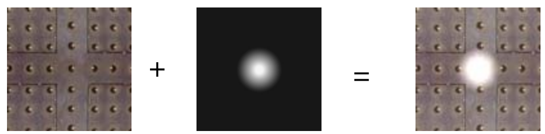


Multitexturing

- Multitexturing
  - Apply a sequence of textures through cascaded texture units



- Light Mapping
  - Instead of calculating the light of the object surface, the texture and bright image are mixed and the resulting image is directly applied to the object surface



Reference

- https://www.glprogramming.com/red/chapter10.html
- https://www.khronos.org/opengl/wiki/Default_Framebuffer