COSC363 Computer Graphics

4

Texture Mapping

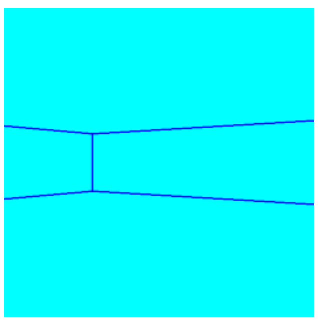
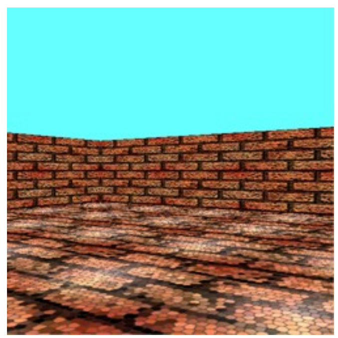
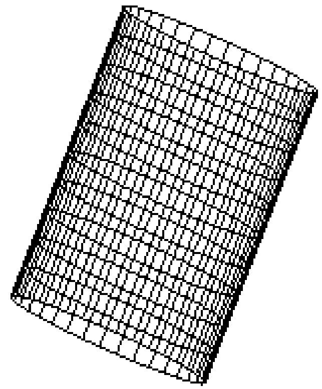
Make an impression!

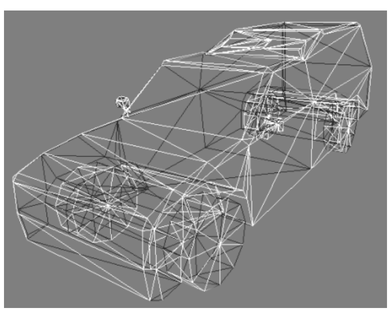
R. Mukundan (mukundan@canterbury.ac.nz)
Department of Computer Science and Software Engineering
University of Canterbury, New Zealand.

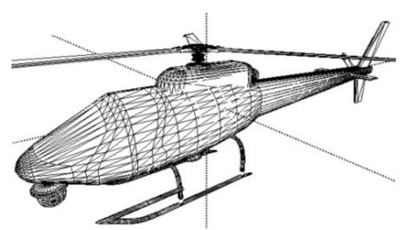
Basic Texture Mapping

Basic texture mapping refers to the process of applying an image or a set of images to an object or a primitive.

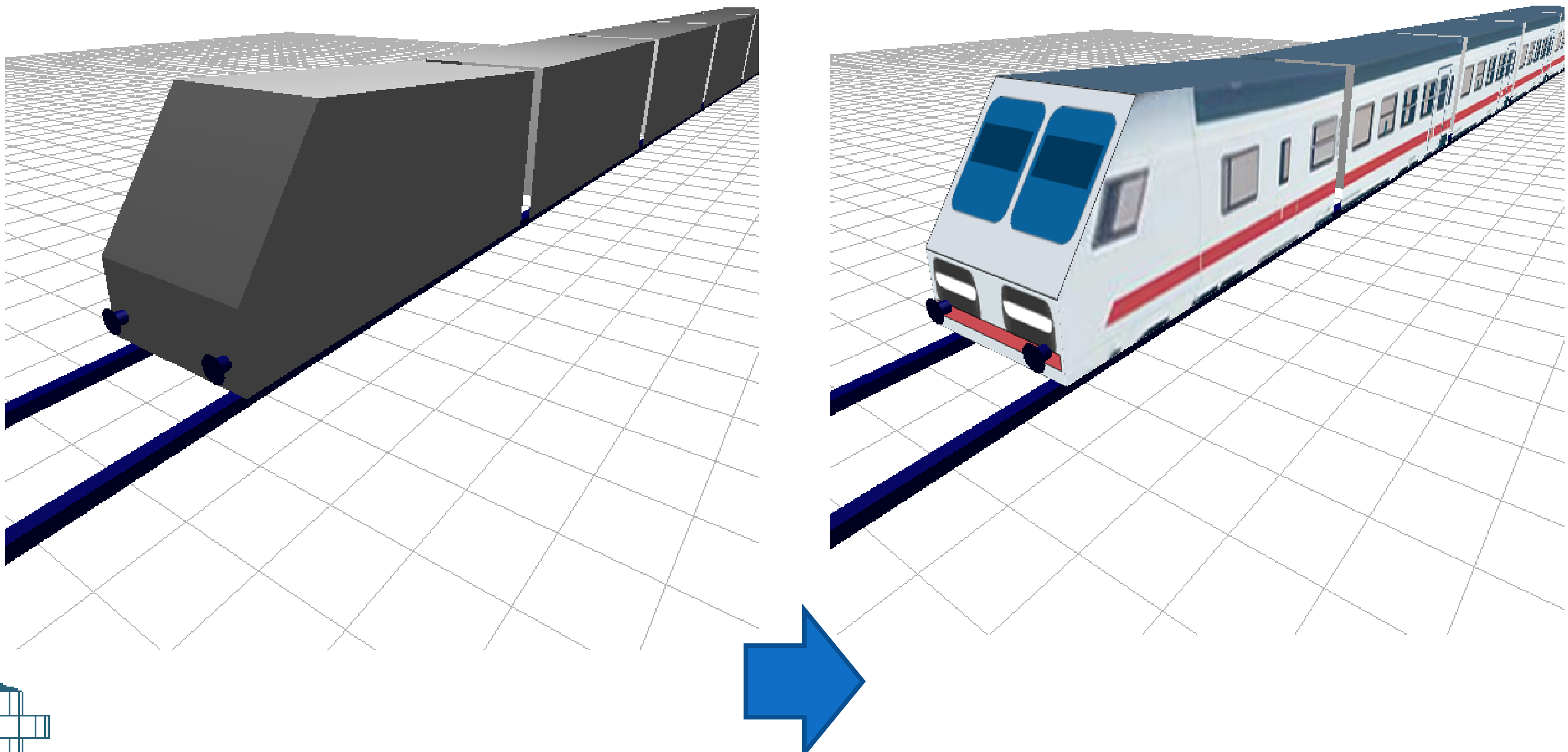
- Adds colour based surface features to polygons
- Makes objects and scenes appear more realistic

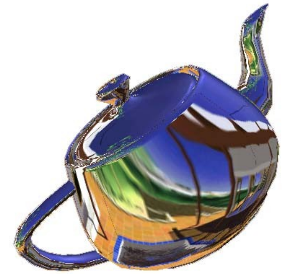



Basic Texture Mapping



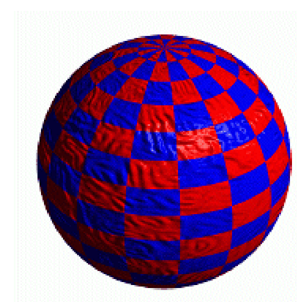
Advanced Applications

- Environment Mapping: Simulates reflections in an object that suggest the "world" surrounding that object.



- Billboarding: View oriented texture mapped polygons commonly used in place of models of trees.

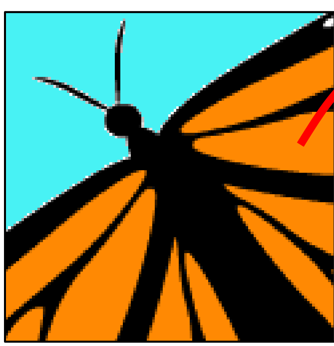
- Bump Mapping: Simulates surface displacements without modifying the geometry, to create the appearance of bumps and wrinkles.



Textures

- For most texture mapping applications, we require images.
- Depending on the image type, we require a *loader* to parse the data contained in the image file, and to load the image data to texture memory.
- An image is a consecutive array of byte values. Each pixel in the image may be represented by 1, 3 or 4 bytes.

3-byte representation:



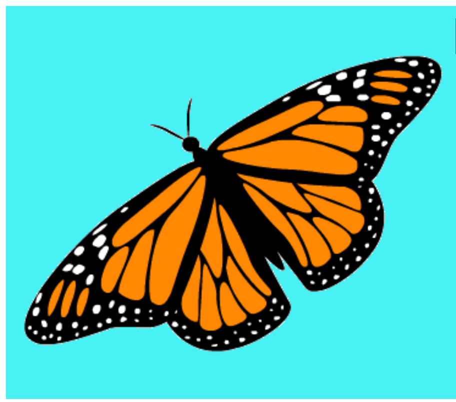
One pixel data

11111111	10000000	00000000
Red	Green	Blue

100x100 image $\Rightarrow$ 100x100x3 bytes

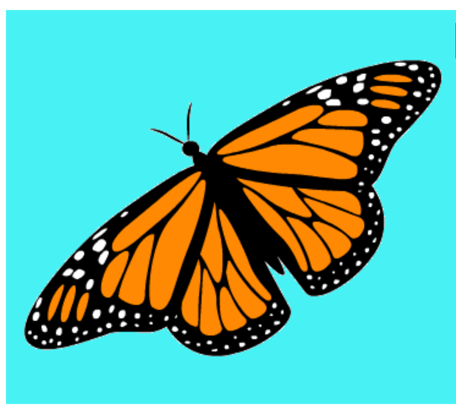
Grey-scale image
1 byte per pixel
Pixel depth (bpp): 8
GL_LUMINANCE

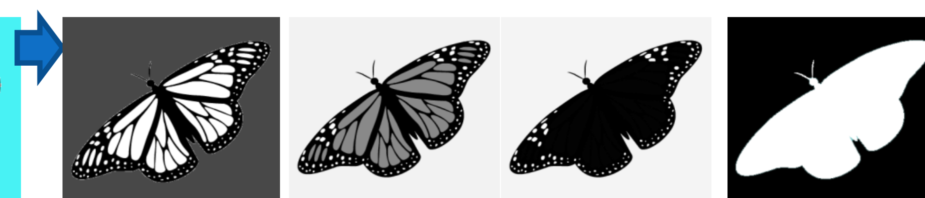
Colour image
3 bytes per pixel
Pixel depth (bpp): 24
GL_RGB



Red Green Blue

Colour image
+ alpha
4 bytes per pixel
Pixel depth (bpp): 32
GL_RGBA





Red Green Blue Alpha

Texture Mapping: Step 1

Generate texture Ids (also referred to as texture names).

- A texture Id is an unsigned integer value (or values) obtained by calling the function `glGenTextures`.
- The texture Ids are then used in the function `glBindTexture` to specify the texture in use.

Example: 1 Texture

```
Gluint texId;
glGenTextures(1, &texId);
glBindTexture(GL_TEXTURE_2D, texId);
...
```

Example: 3 Textures

```
Gluint texId[3];
glGenTextures(3, texId);
glBindTexture(GL_TEXTURE_2D, texId[0]);
...
glBindTexture(GL_TEXTURE_2D, texId[1]);
...
glBindTexture(GL_TEXTURE_2D, texId[2]);
...
```

Texture Mapping: Step 2

Load a texture by calling the function:

```
glTexImage2D (GL_TEXTURE_2D,  0,
  n,   //No. of colour components(1, 3, 4)
  wid, //Image width,  a power of 2
  hgt, //Image height, a power of 2
  0,   //Border
  format, //GL_LUMINANCE,GL_RGB or GL_RGBA
  type,   //GL_UNSIGNED_BYTE
  imgData // Pointer to image data
   );
```

Loading Textures

Scene.tga
256x256
24 bpp
Uncompressed

loadTGA.h

loadTGA(“Scene.tga”); → calls → glTexImage2D(...)

Example:

```
#include "loadTGA.h"
...
Gluint texId;
glGenTextures(1, &texId);
glBindTexture(GL_TEXTURE_2D, texId);
loadTGA("Scene.tga");
...
```

Loading Textures

Earth.bmp
256 x 128
24 bpp
Windows Bitmap

loadBMP.h

loadBMP("Earth.bmp");

calls

glTexImage2D(...)

Example:

```
#include "loadBMP.h"
...
Gluint texId;
glGenTextures(1, &texId);
glBindTexture(GL_TEXTURE_2D, texId);
loadBMP("Earth.bmp");
...
```

Texture Mapping: Step 3

Set texture sampling parameters:

- Minification and magnification filters (discussed later)
- Wrapping mode.

Example:

```
#include "loadTGA.h"
...
Gluint texId;
glGenTextures(1, &texId);
glBindTexture(GL_TEXTURE_2D, texId);
loadTGA("Scene.tga");
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
...
```

Texture Mapping: Step 4

Set texture environment parameters

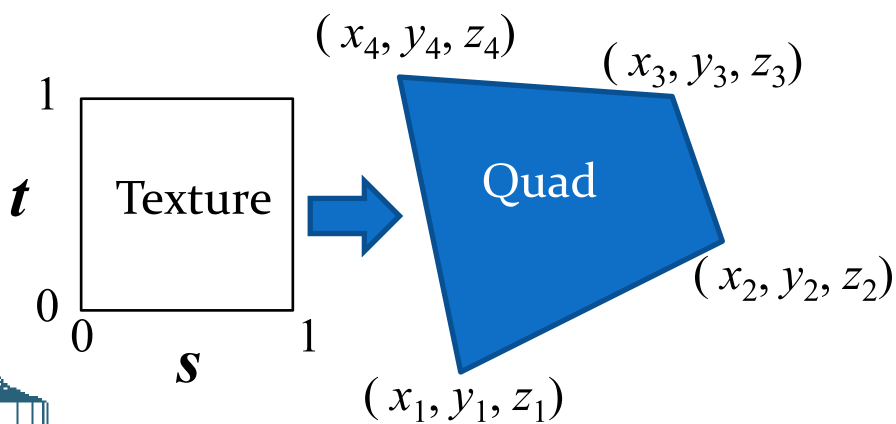
- GL_REPLACE: Texture colour replaces the fragment's colour
- GL_MODULATE: Texture colour is multiplied by fragment's colour


```
#include "loadTGA.h"
...
Gluint texId;
glGenTextures(1, &texId);
glBindTexture(GL_TEXTURE_2D, texId);
loadTGA("Scene.tga");
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
```

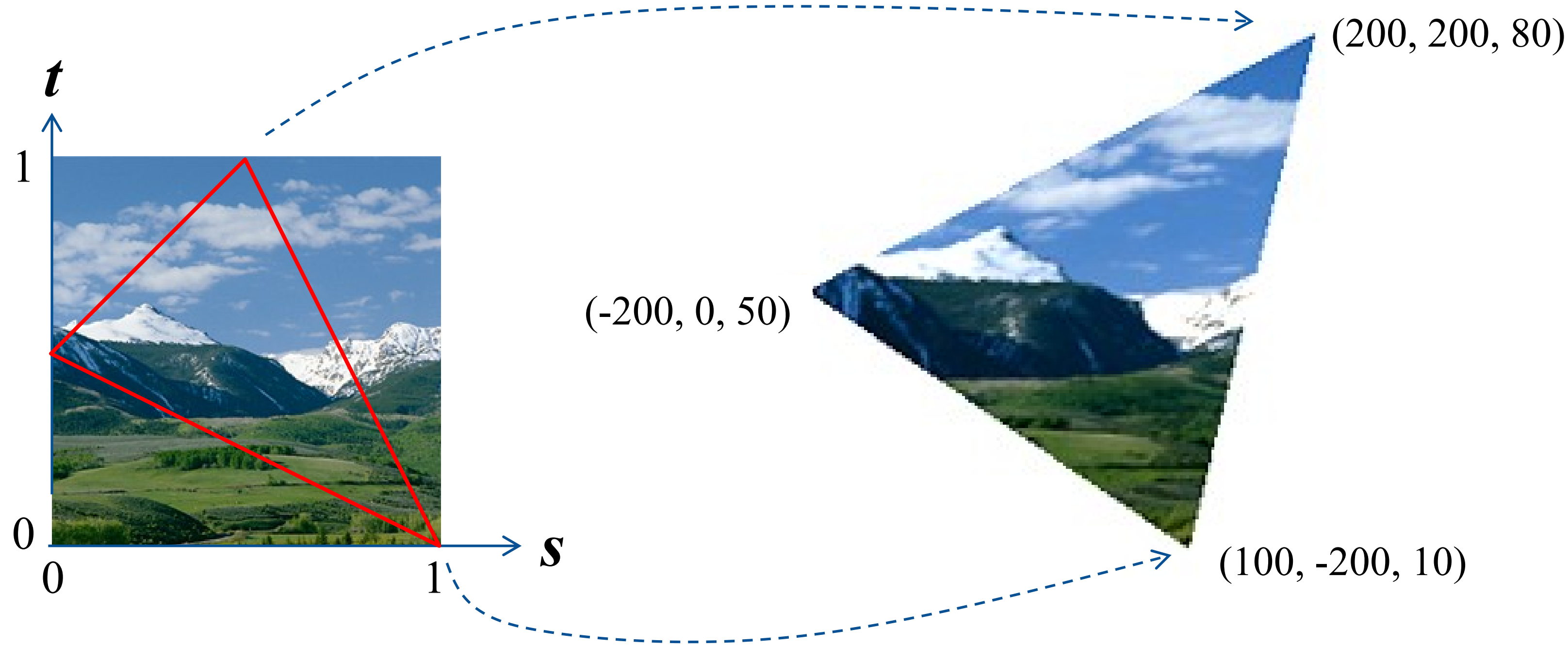
Texture Mapping: Step 5

Enable texturing and assign texture coordinates to vertices.

- Texture coordinates (s, t) are defined in the image space with the origin at the bottom-left corner of the image, and a value 1 at image extremities, <u>independent of image size</u>.
- The user specifies the image region to be mapped to a primitive by associating a pair of texture coordinates with each vertex.



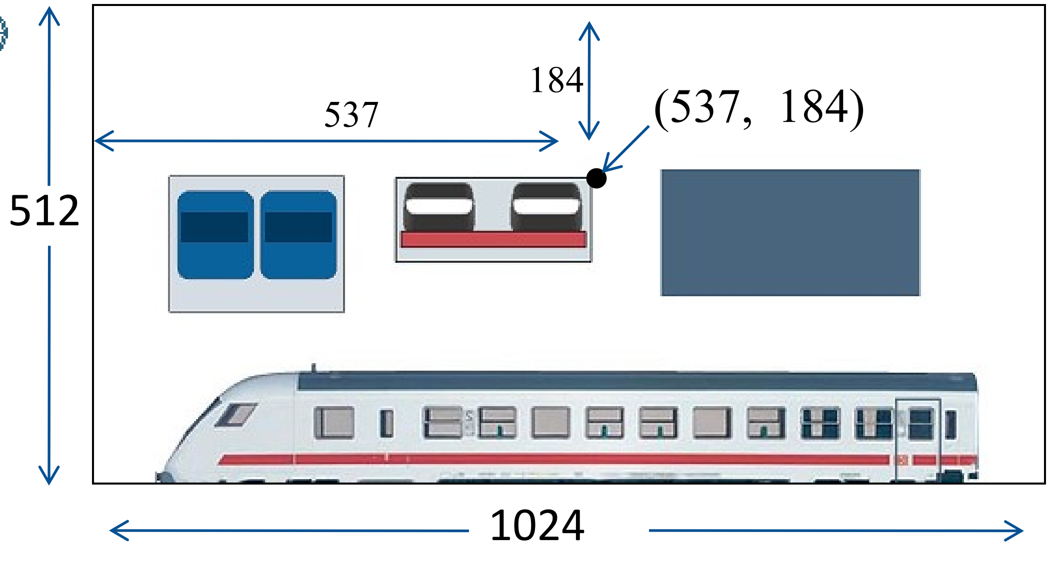

```
glEnable(GL_TEXTURE_2D);
...
glBegin(GL_QUADS);
  glTexCoord2f(0., 0.);
  glVertex3f(x1, y1, z1);
  glTexCoord2f(1., 0.);
  glVertex3f(x2, y2, z2);
  glTexCoord2f(1., 1.);
  glVertex3f(x3, y3, z3);
  glTexCoord2f(0., 1.);
  glVertex3f(x4, y4, z4);
glEnd();
```

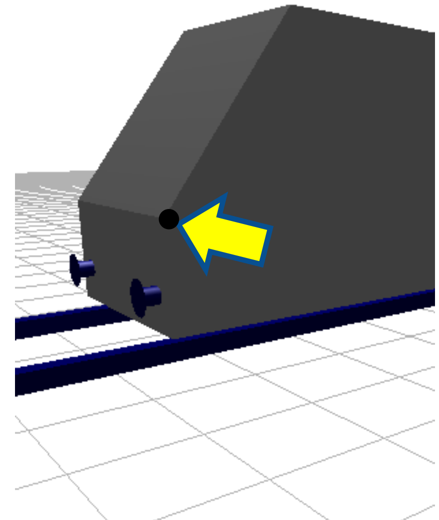



```
glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, texId);
...
glBegin(GL_TRIANGLES);
  glTexCoord2f(0.0, 0.5);  glVertex3i(-200, 0, 50);
  glTexCoord2f(1.0, 0.0);  glVertex3i(100, -200, 10);
  glTexCoord2f(0.5, 1.0);  glVertex3i(200, 200, 80);
glEnd();
```

Another Example

A single texture containing several sections





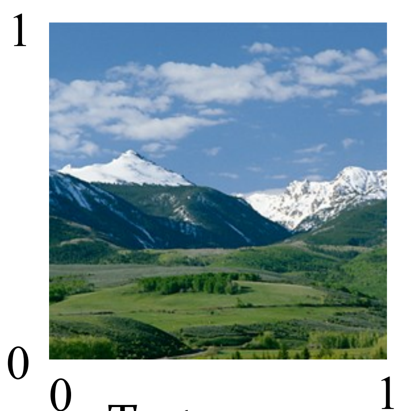
$$(537,\ 184) \Rightarrow (537,\ 328)$$
$$\Rightarrow (537/1024,\ 328/512)$$
$$\Rightarrow \mathbf{(0.5244,\ 0.6406)}$$

```
glBegin(GL_QUADS);
 ...
glNormal3f(0, 0, 1);    //Lights
glTexCoord2f(0.3212, 0.4628);
glVertex3f(-6.5, 0, 22.5);
glTexCoord2f(0.5244, 0.4628);
glVertex3f(6.5, 0, 22.5);
glTexCoord2f(0.5244, 0.6406);
glVertex3f(6.5, 6., 22.5);
glTexCoord2f(0.3212, 0.6406);
glVertex3f(-6.5, 6., 22.5);
```

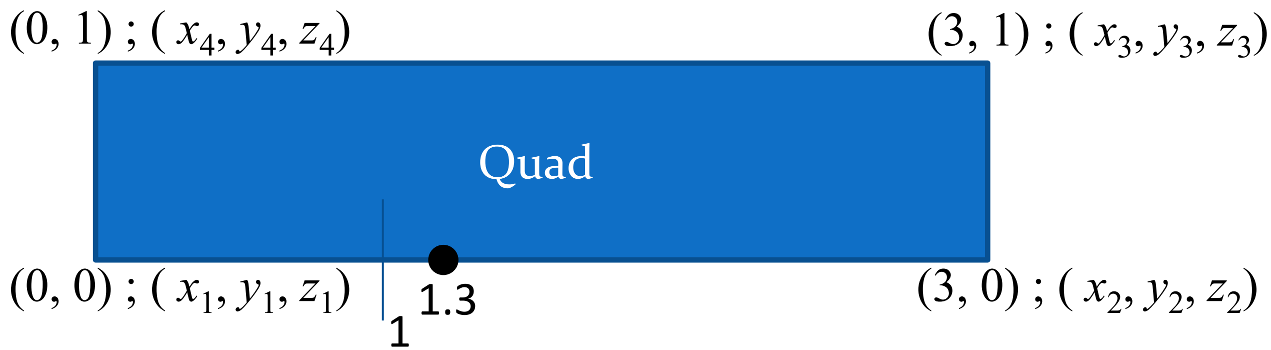
Texture Tiling

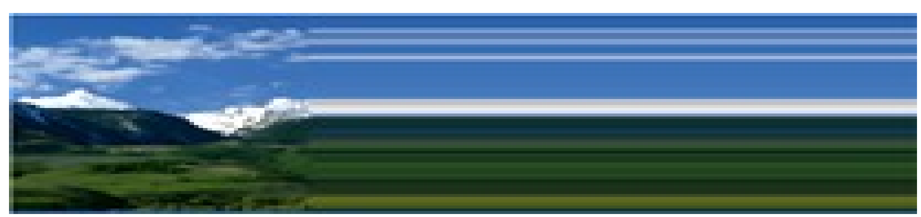
If the wrap parameter for a texture axis is set to GL_CLAMP, then the coordinate value is clamped to the range [0, 1]. (eg., a texture coordinate value 1.3 is treated as 1).

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, **GL_CLAMP**);



Texture

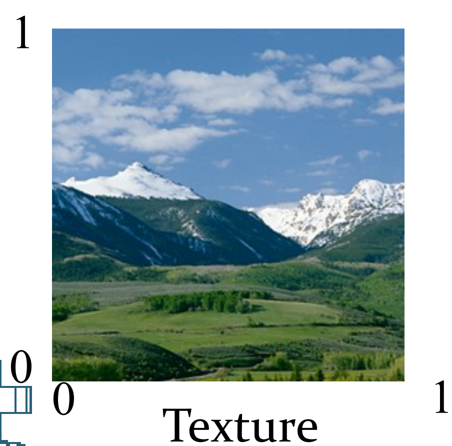


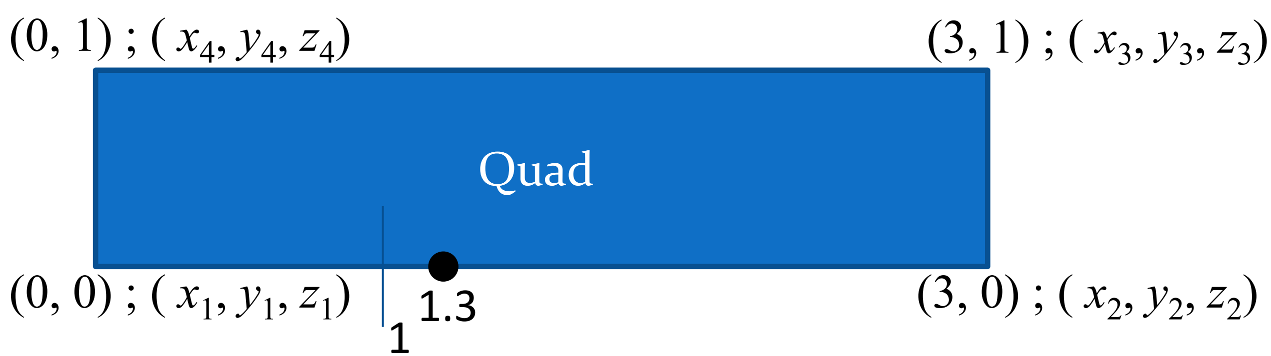


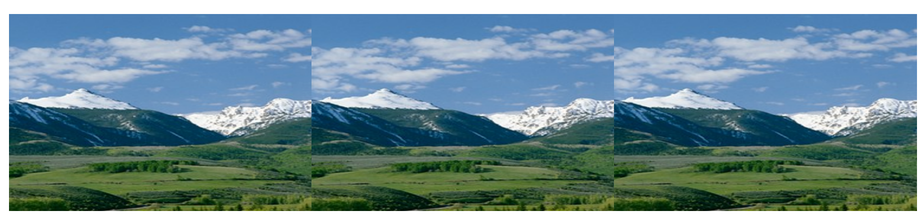
Texture Tiling

- Texture coordinates assigned to a vertex can have values greater than 1. Such values can be used for tiling.
  - If the wrap parameter for a texture axis is set to GL_REPEAT, then the integer part of the texture coordinate along that axis is ignored. (eg. A value 1.3 is treated as 0.3). This results in the tiling of the image along that axis. [Default]

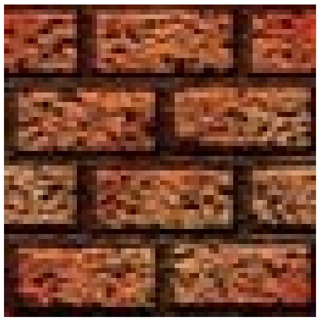
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, **GL_REPEAT**);

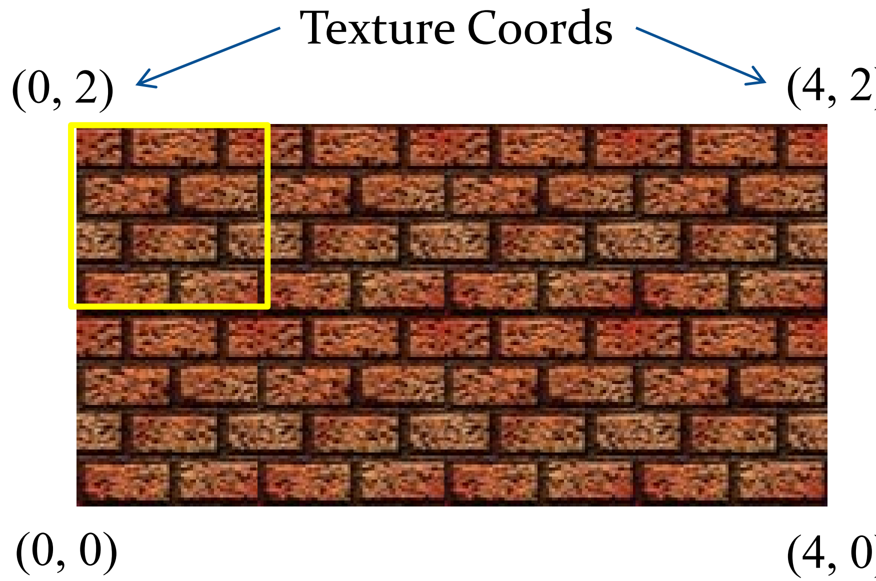






Seamlessly Tileable Textures





(www.textures.com)

Texture Parameters GL_NEAREST, GL_LINEAR

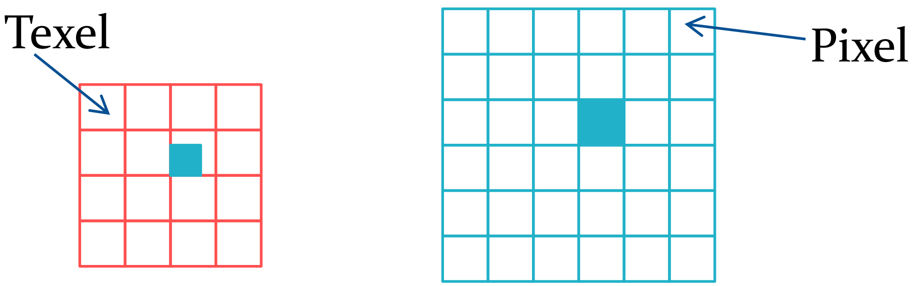
```
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER, GL_NEAREST)
                                                     GL_LINEAR

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

- GL_NEAREST: Returns the texel value nearest to the centre of the pixel.
- GL_LINEAR: Returns the weighted average of four texel values closest to the centre of the pixel.

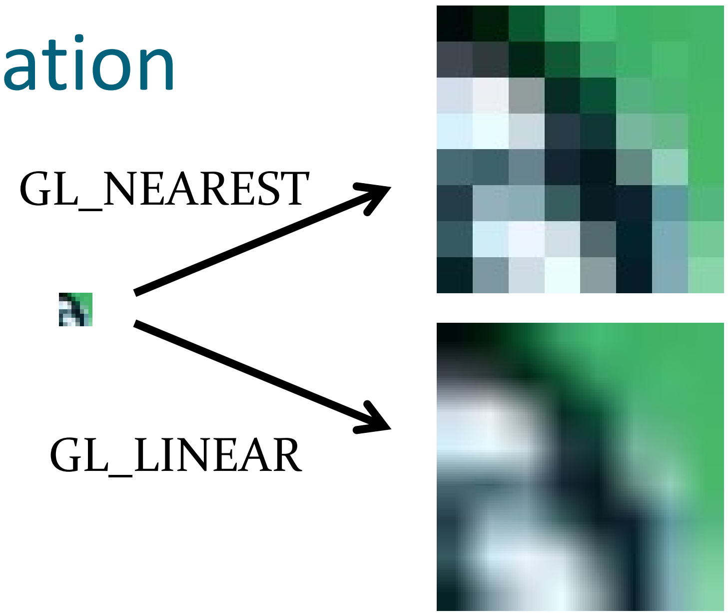
Note:

The pixel value of a texture is often called a "texel".

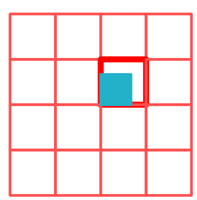
Texture Magnification



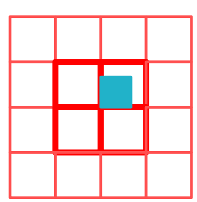
Texture Polygon



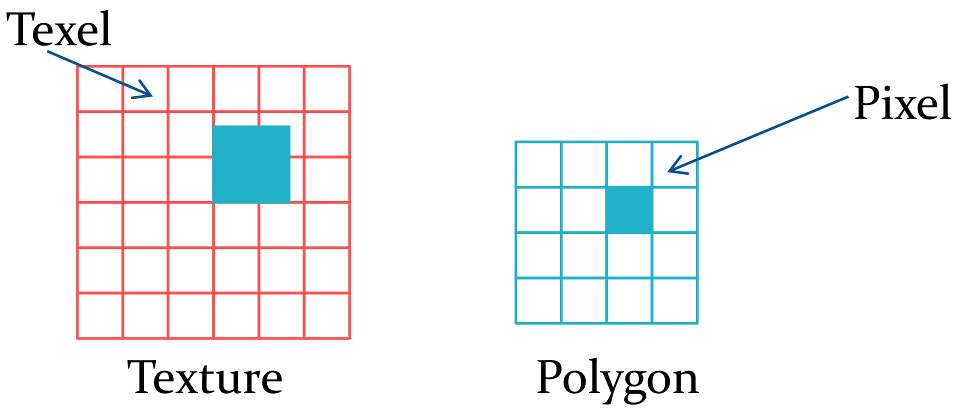
GL_NEAREST: The pixel gets the colour of the texel value nearest to the centre of the pixel.



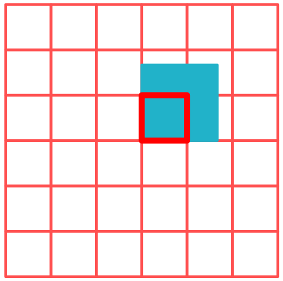
GL_LINEAR: The pixel gets the weighted average of four texel values closest to the centre of the pixel.



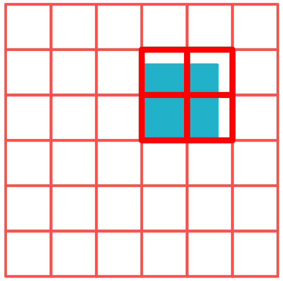
Texture Minification



GL_NEAREST: The pixel gets the colour of the texel value nearest to the centre of the pixel.

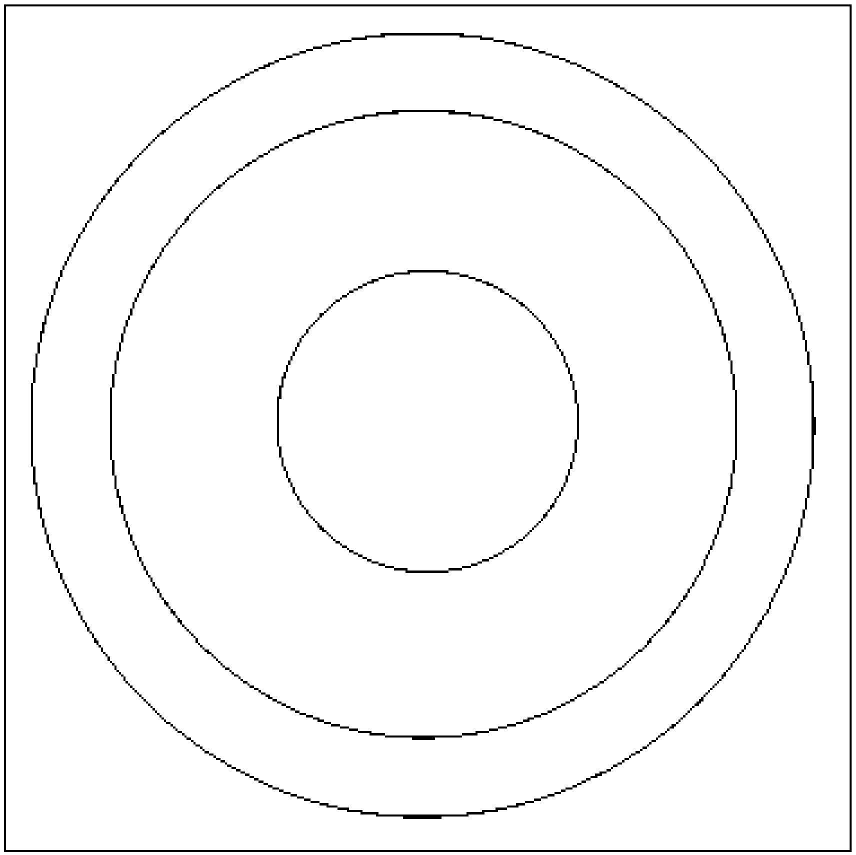


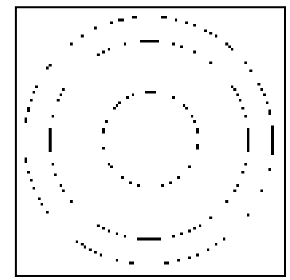
GL_LINEAR: The pixel gets the weighted average of four texel values closest to the centre of the pixel.



Texture Minification

Thin lines often disappear when a texture is mapped to a region containing fewer pixels.

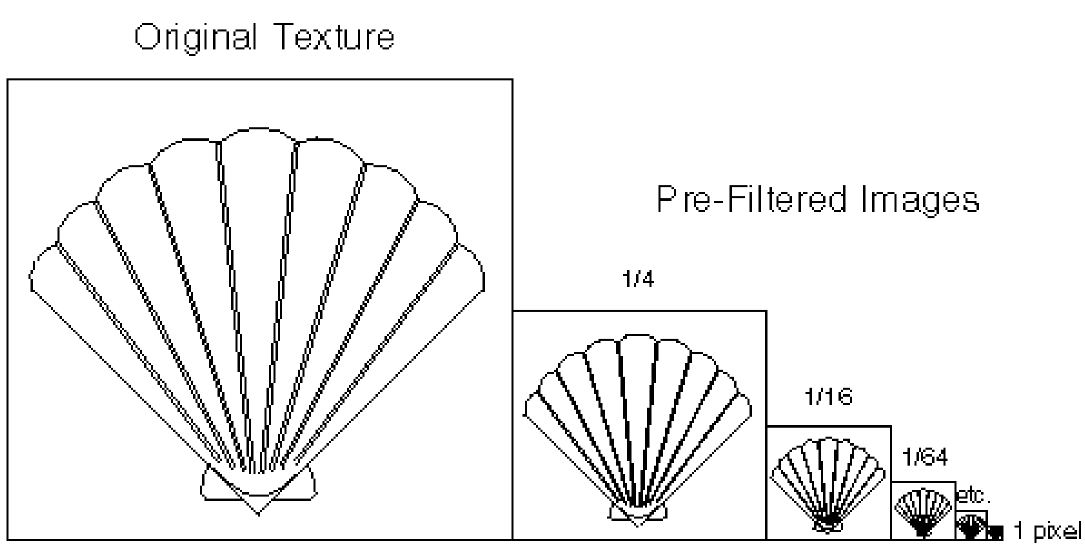




Both GL_NEAREST and GL_LINEAR settings produce similar images

Texture Mipmaps

- MIP = Multum In Parvo = “Much in a small place”
- A mipmap is a set of prefiltered versions of the same image at different scales (resolutions)
- The problem of disappearing lines when a texture is mapped to a small region can be solved by using a mipmap.
- Mipmapping requires additional processing, and 33% extra texture storage space.



Texture Mipmaps

```
glTexParameteri(GL_TEXTURE_2D,
                GL_TEXTURE_MIN_FILTER,
                GL_LINEAR_MIPMAP_LINEAR)

glTexImage2D(GL_TEXTURE_2D, 0, 3, 64,64, 0, GL_RGB,
GL_UNSIGNED_BYTE, img1)
glTexImage2D(GL_TEXTURE_2D, 1, 3, 32,32, 0, GL_RGB,
GL_UNSIGNED_BYTE, img2)
glTexImage2D(GL_TEXTURE_2D, 2, 3, 16,16, 0, GL_RGB,
GL_UNSIGNED_BYTE, img3)
...
glTexImage2D(GL_TEXTURE_2D, 6, 3, 1,1, 0, GL_RGB,
GL_UNSIGNED_BYTE, img7)
```

Texturing a Quadric Surface

Using GLU library, the texture coordinates can be automatically generated for a quadric surface:

```
GLUquadric *q = gluNewQuadric();
gluQuadricDrawStyle ( q, GLU_FILL );
gluQuadricNormals( q, GLU_SMOOTH );
gluQuadricTexture( q, GL_TRUE );
gluSphere ( q, 3.0, 18, 12 );
```

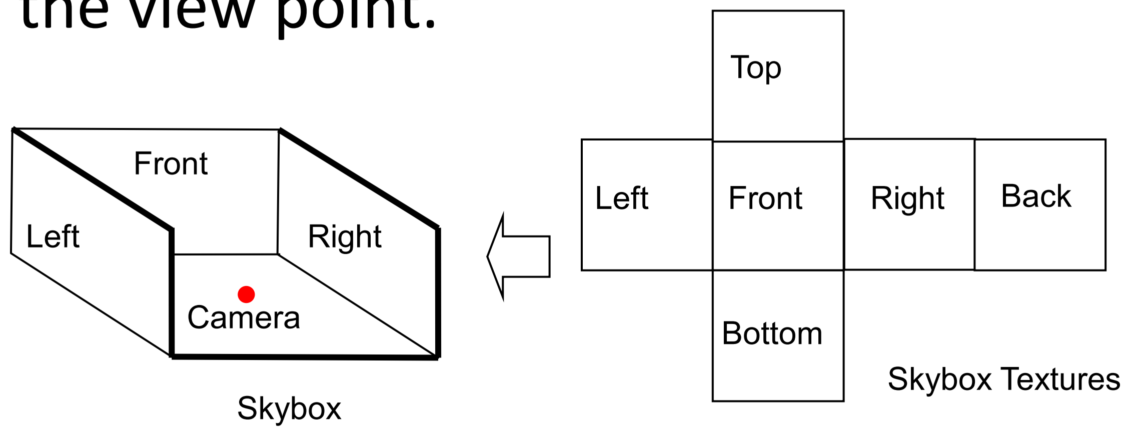
Texturing and Lighting

- Lighting computation is a per-vertex operation, whereas texturing is done later at the fragment processing stage.
- If GL_REPLACE is used as the texturing environment (See slide 11), the colour values got from lighting computation would be replaced with texture colours.
- In order to see the variation of diffuse reflections from the surface, the texture values must be modulated with the already computed fragment colour (GL_MODULATE)
- Modulation will reduce the effect of specular highlights. To get a strong specular highlight on a textured surface, select the following light model:

```
glLightModeli(GL_LIGHT_MODEL_COLOR_CONTROL,
              GL_SEPARATE_SPECULAR_COLOR);
```

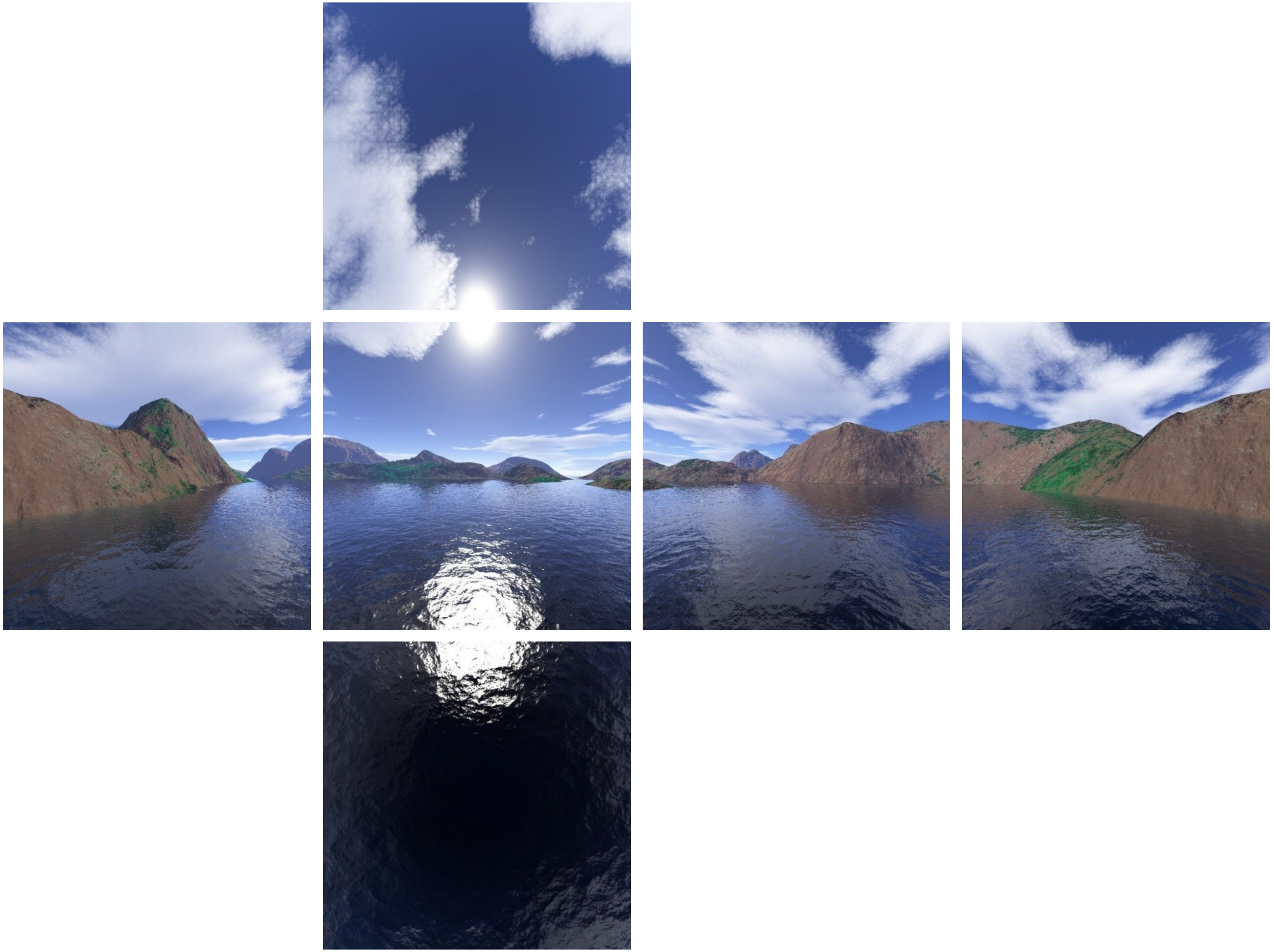
Sky Boxes

- The surrounding environment is displayed as textures on the faces of a large cube, and the cube is rendered centered around the view point.



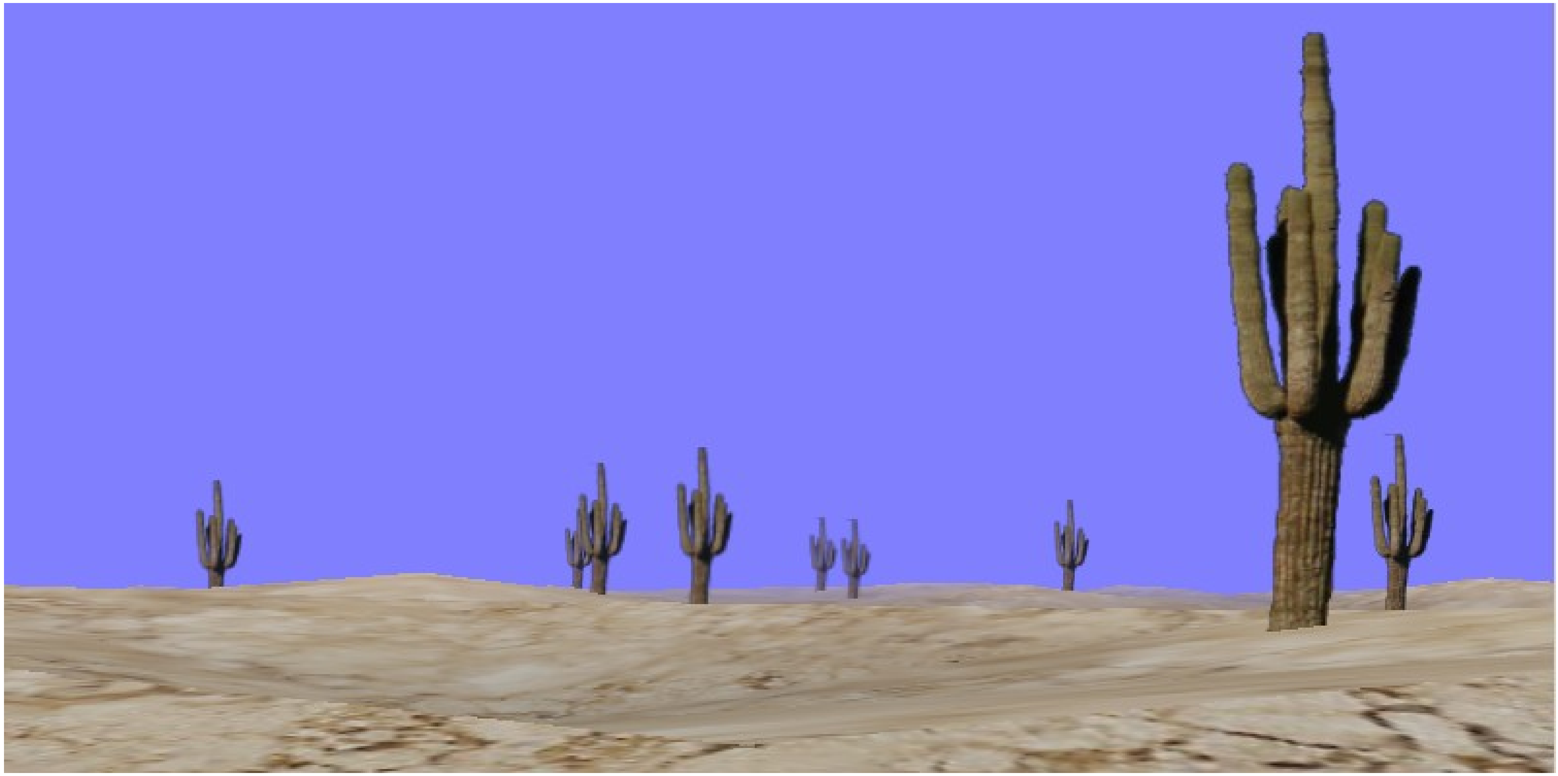
- Try to minimise perspective distortions by
  - Adjusting the focal length ("near" value in gluPerspective) and the field of view ("fov" value in gluPerspective)
  - Adjusting the size of the cube used for texture mapping
  - Not moving the camera very close to the four sides of the cube

Sky Box Textures



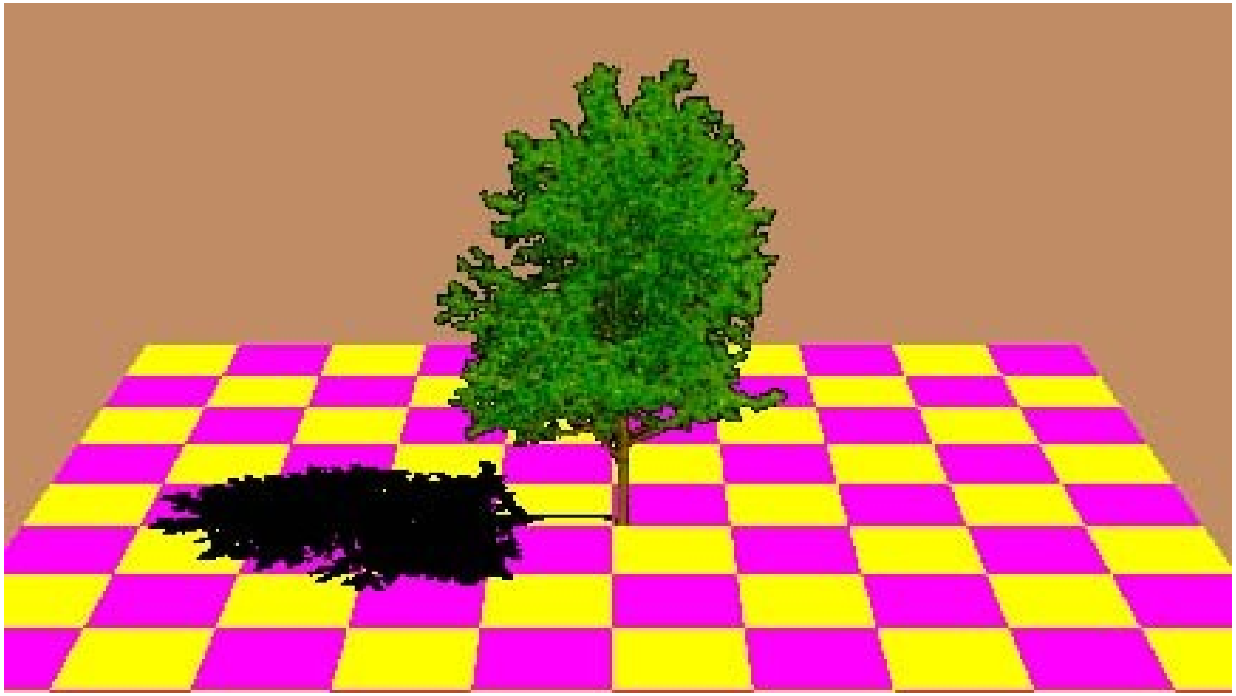
Billboarding

Billboarding is a technique that changes the orientation of a pre-rendered image (usually on quads) in a 3D environment depending on the eye point location and orientation.



Alpha Texturing

- A textured image should appear as being part of the surrounding scene, and not part of a rectangular ‘board’.
- Use the alpha channel of an image to transfer only those pixels on the object.



Alpha Texturing

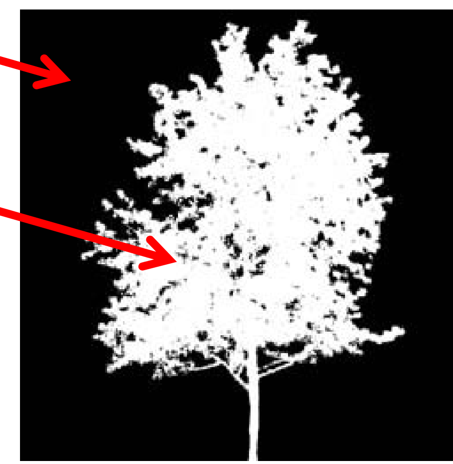
```
glEnable(GL_TEXTURE_2D);
glEnable(GL_ALPHA_TEST);
glAlphaFunc(GL_GREATER,0);
glBindTexture(GL_TEXTURE_2D, texId);
drawBillboard();
glDisable(GL_TEXTURE_2D);
glDisable(GL_ALPHA_TEST);
```

RGB

Background pixels in the image have alpha value 0.

Foreground pixels in the image have alpha value greater than 0.

When the image is mapped to a quad, only those portions of the quad where the mapped texel has an alpha value greater than zero are displayed (see previous slide).



Alpha