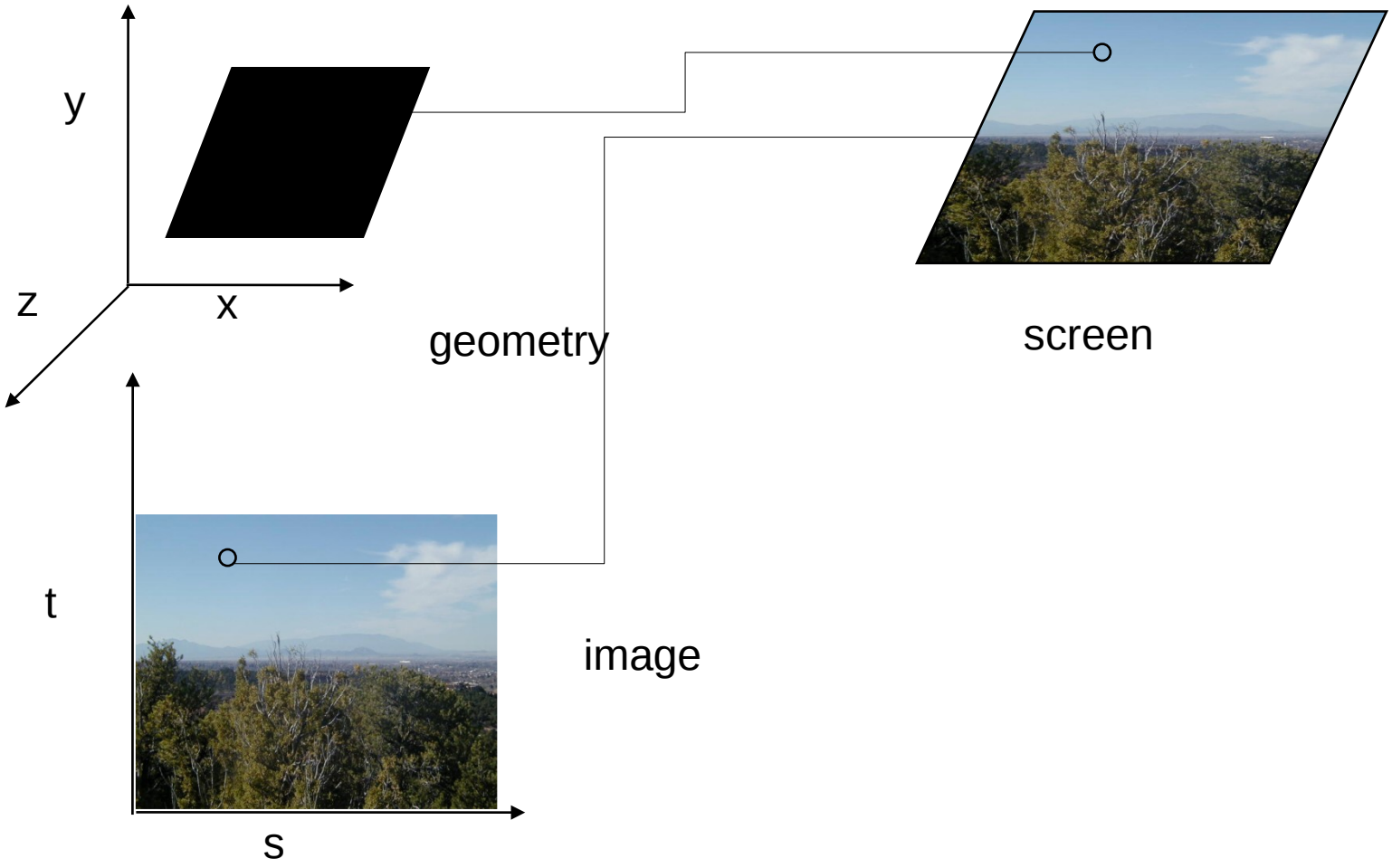
OpenGL Texture Mapping

Benefits of using textures

- A texture is an “image” that is mapped to a polygon.
- Textures are rectangular arrays of data (does not have to be 2D)

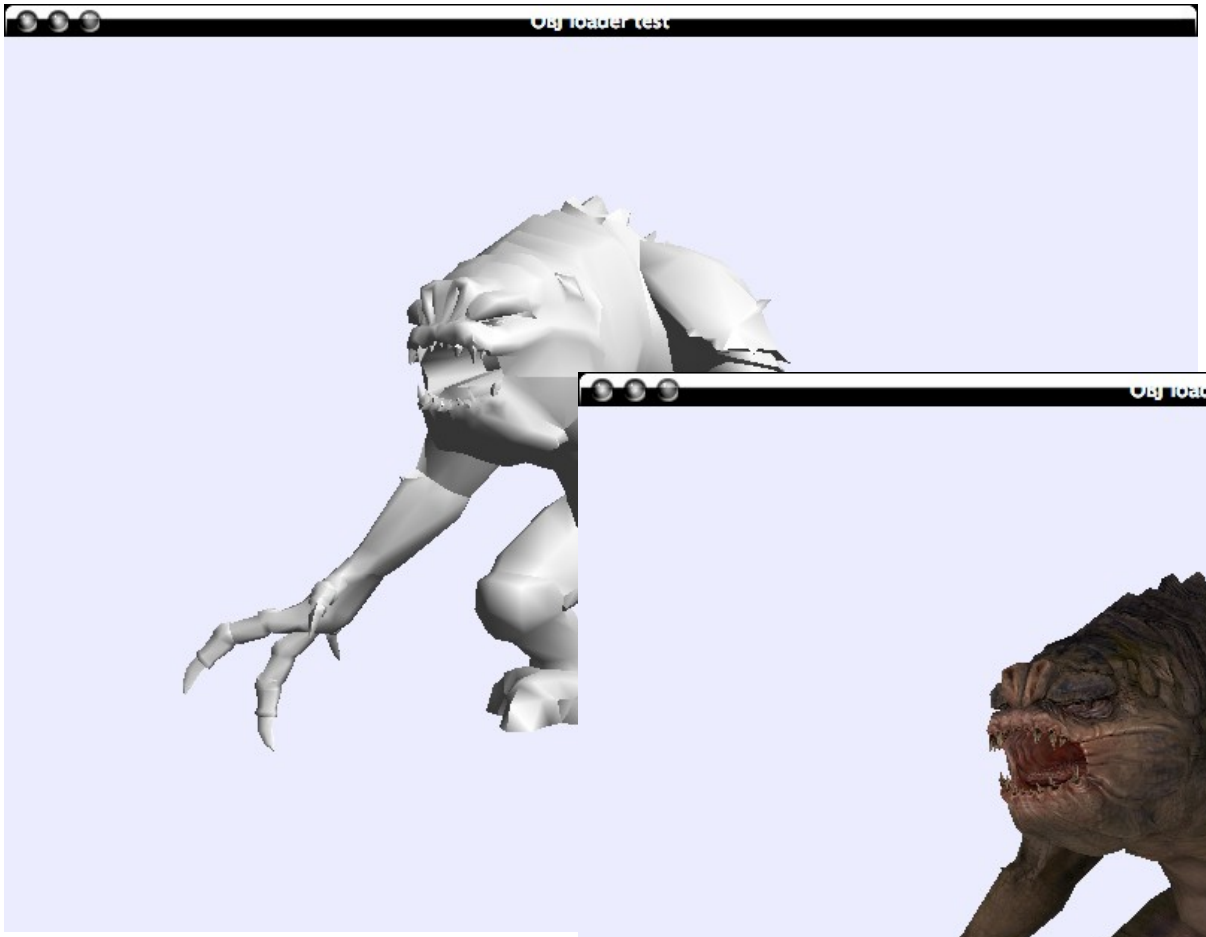
Texture Mapping



Why textures?

- greater realism
  - real objects are not smooth and regular
- save resources
  - imaging rendering a brick wall
  - draw every single brick? that’s a lot of polygons!
  - instead glue an image of a brick wall to one large


Obj loader test


Obj loader test


Obj loader test

- How to apply a texture to a polygon?
  - Texels must somehow be matched to pixels.
  - Polygons can be transformed... what do to with the texture in that case?
- Mapping a rectangular texture to a quad?
- Mapping a rectangular texture to a non-rectangular region?
- Texturing is actually a quite complicated process...

Basic Steps

- Three steps to applying a texture
  - Specify the texture
  - Enable texturing & bind texture(define the one you'll use immediately)
  - Assign texture coordinates to vertices

Basic Steps

- Three steps to applying a texture
  - Specify the texture
  - Enable texturing & bind texture(define the one you'll use immediately)
  - Assign texture coordinates to vertices

Basic Steps

: Specify the texture

- Read or generate image
- Generate a handle to texture memory, and bind it:
  - glGenTextures(1, &texName);
  - glPixelStorei(GL_UNPACK_ALIGNMENT, 1)
  - glBindTexture(GL_TEXTURE_2D, texName);
- Specify texture parameters: wrapping, filtering.
  - glTexParameteri(...)
- Copy the image to texture memory:
  - glTexImage2D(...)

Specify Texture 1 : Read or Generate Image

- You need some way to get a texture image.
  - either loaded from file or procedurally generated
  - SDL_Image
  - image class is your friend
- Textures are usually thought of as 2-dimensional.
  - **`Glubyte my_texels[512][512];`**
  - But they can also be 1-dimensional or 3-dimensional.
    - **`Glubyte my_texels[512][512][3]`**
  - OpenGL requires texture dimensions to be powers of 2
- The data describing a texture can consist of one, two, three, or four elements per texel.
  - 3 or 4 elements usually represent RGB or RGBA
  - 1 element often represents e.g. a modulation constant

Specify Texture 1 : Read or Generate Image

- Define a texture image from an array of *texels* (texture elements) in CPU memory
  - **`Glubyte my_texels[512][512];`**
  - OpenGL requires texture dimensions to be powers of 2

Specify Texture 2: Pixel Storage
glPixelStorei(GL_UNPACK_ALIGNMENT, 1);

- Image data is usually stored in rectangular two-dimensional arrays.
  - some machines have optimized architecture for data that is aligned on two-byte, four-byte, or eight-byte boundaries
  - some machines have different byte order
- **GL_UNPACK_ALIGNMENT** describes how the bitmap data is stored in computer memory
  - 1 means just use next available byte
  - this is what you probably want unless you're working on a more exotic architecture
- **GL_UNPACK_SWAP_BYTES** specifies that endianness must be swapped

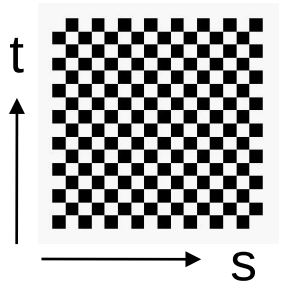
- 2: Enable texturing & bind texture(define the one you'll use immediately)
  - glEnable(GL_TEXTURE_2D)
  - glBindTexture(GL_TEXTURE_2D, texName);
- 3: Assign texture coordinates to vertices
  - u,v [0.0-1.0]

•Specify Texture 3 :Texture Parameters : glTexParameteri(…)

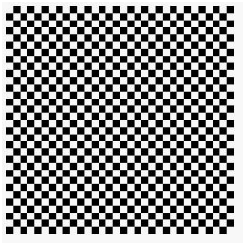
- OpenGL a variety of parameter that determine how texture is applied
  - **Wrapping parameters** determine what happens if s and t are outside the (0,1) range
  - **Filter modes** allow us to use area averaging instead of point samples
  - **Mipmapping** allows us to use textures at multiple resolutions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

Wrapping Mode

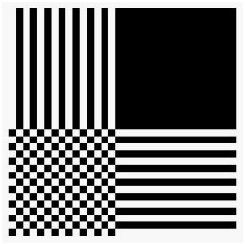
- You can assign texture coordinates outside the range [0; 1]
- What happens?
  - either repeat the texture (**GL_REPEAT)**
    - integer part of the texture coordinates is ignored
  - or clamp (**GL_CLAMP)**
    - values > 1.0 are set to 1.0
    - values < 0.0 are set to 0.0



texture



GL_REPEAT wrapping

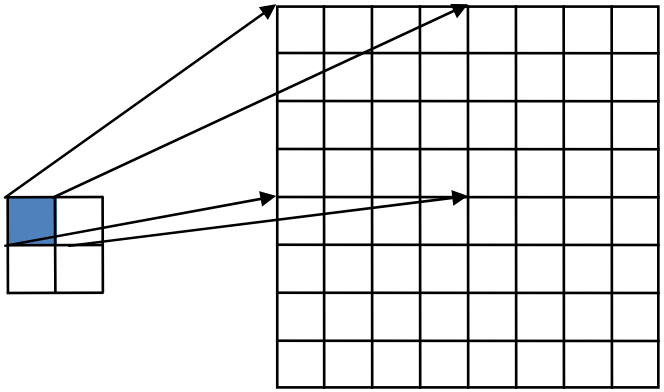


GL_CLAMP wrapping

Filters: Magnification and Minification

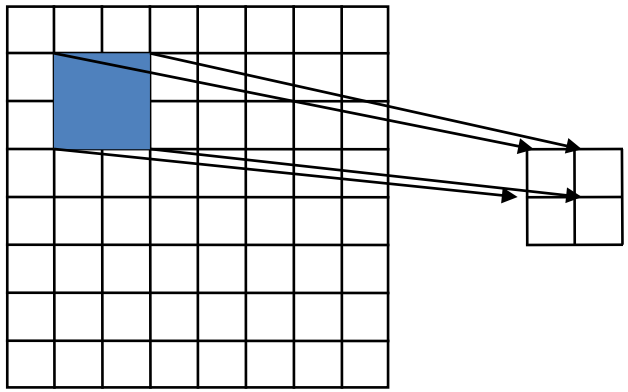
More than one texel can cover a pixel (*minification*) or more than one pixel can cover a texel (*magnification*)

Can use point sampling (nearest texel) or linear filtering (2 x 2 filter) to obtain texture values



Texture Polygon

Magnification



Texture Polygon

Minification

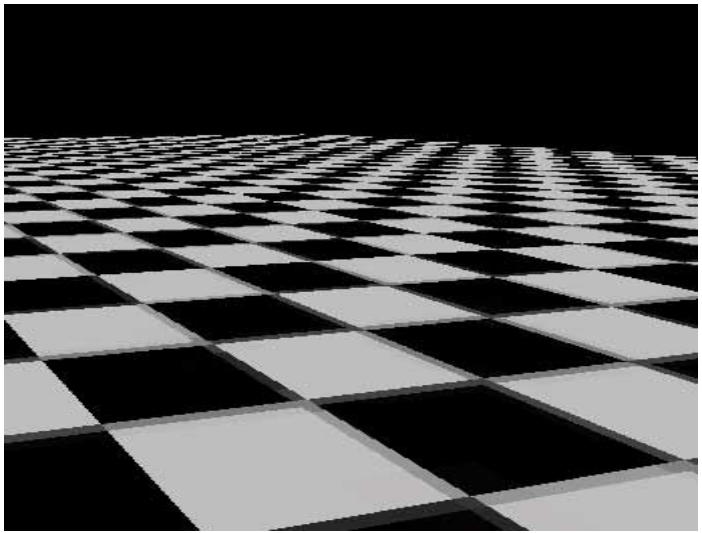
Filter Modes

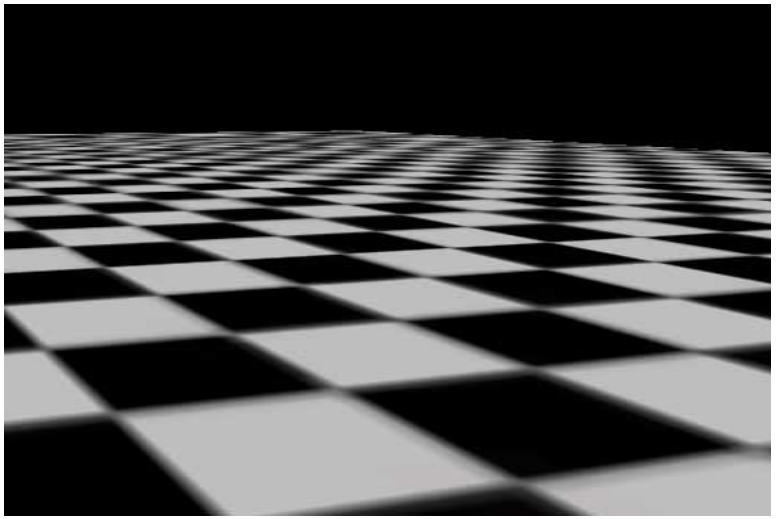
Modes determined by

- **glTexParameteri(target, type, mode)**

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXURE_MAG_FILTER,
                GL_NEAREST);

glTexParameteri(GL_TEXTURE_2D, GL_TEXURE_MIN_FILTER,
                GL_LINEAR);
```





Specify Texture 3 : Copy an Image into a Texture

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

target: type of texture, e.g. **GL_TEXTURE_2D**

level: should be 0 (used for mipmapping)

components: specified which of RGBA are selected for use in modulating/blending

width, height: specify texture image dimensions (texels in pixels)

border: border width (usually 0)

Specify Texture 3 : Copy an Image into a Texture

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

format specifies format

GL_COLOR_INDEX, GL_RGB, GL_RGBA etc.

type: specifies the type

GL_BYTE, GL_UNSIGNED_BYTE, GL_SHORT, GL_UNSIGNED_SHORT, GL_INT etc.

texels: contains the texture-image data

```
glTexImage2D(GL_TEXTURE_2D, 0, 3, 512, 512, 0,
GL_RGB, GL_UNSIGNED_BYTE, my_texels);
```

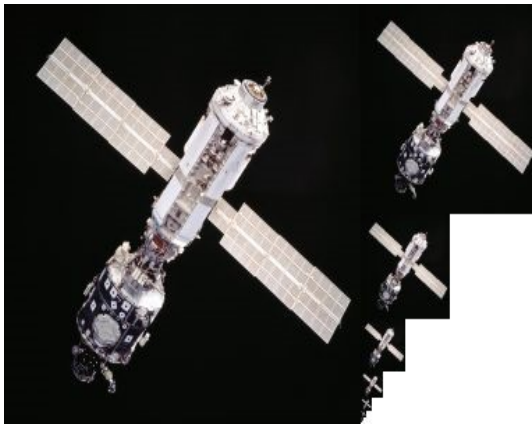
Specify Texture 3 : Texture Parameters glTexParameteri(...)

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

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, … )
```



- GLU mipmap builder routines will build all the textures from a given image

Example

Specify Texture 3 : Texture Parameters glTexParameteri(...)

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

Environment parameters

- How should the final color be computed from the fragment color and the texture-image color?
  - `void glTexEnvf ( GLenum target , GLenum pname , GLfloat param );`
    - **`target`** Specifies a texture environment. Must be **GL_TEXTURE_ENV.**
    - **`pname`** Specifies the symbolic name of a single-valued texture environment parameter. Must be **GL_TEXTURE_ENV_MODE.**
    - **`param`** Specifies a single symbolic constant
      - just use texture color **(GL_DECAL)**
      - use texture to modulate fragment color - useful to combine texturing with lighting **(GL_MODULATE)**
      - blend a constant color with the fragment color based on the texture value **(GL_BLEND)**

Basic Steps

- Three steps to applying a texture
  - Specify the texture
  - Enable texturing & bind texture(define the one you'll use immediately)
    - glEnable(GL_TEXTURE_2D)
    - glBindTexture(GL_TEXTURE_2D, texName);
  - Assign texture coordinates to

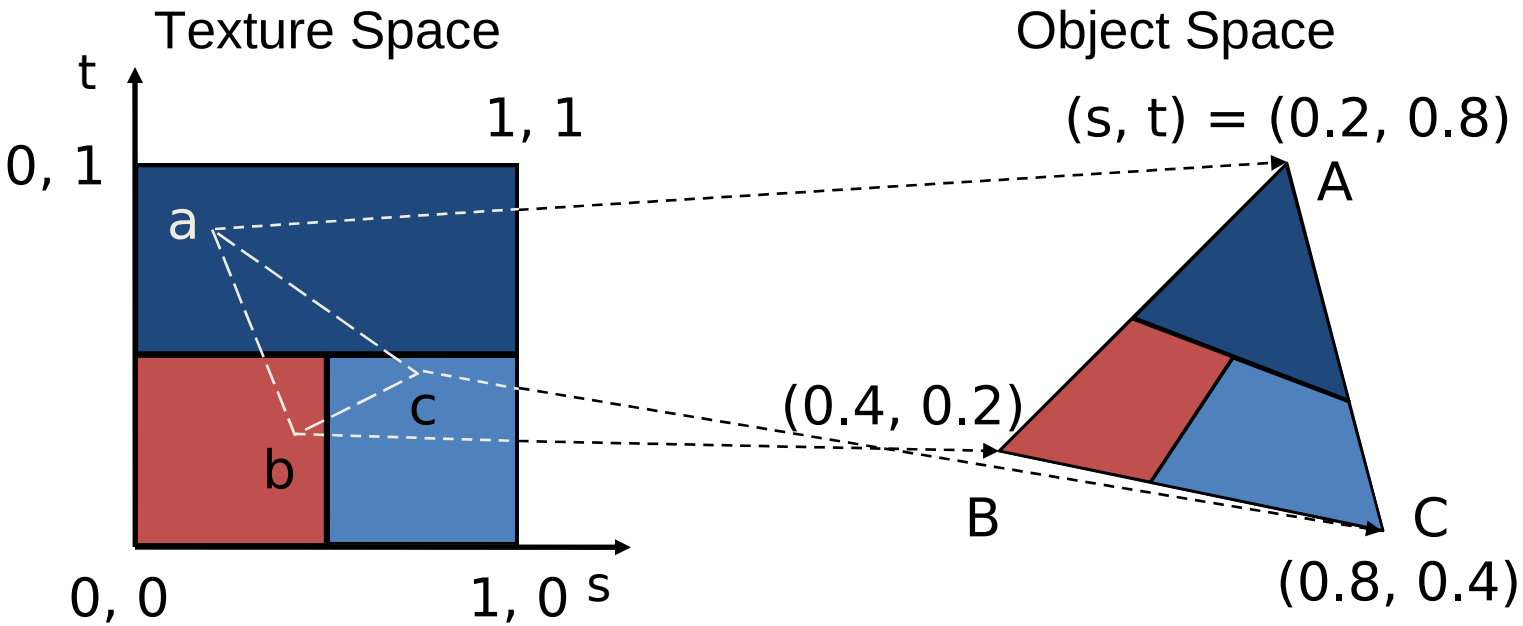
Basic Steps

- Three steps to applying a texture
  - Specify the texture
  - Enable texturing & bind texture(define the one you'll use immediately)
  - Assign texture coordinates to vertices

- Specify both geometric coordinates and texture coordinates for the objects that should be textured.
  - For a 2D texture, texture coordinates are in the range [0; 1].
  - The object's geometric coordinates can be anything.
  - So we need to indicate how the texture should be aligned.
- For example to map a rectangular image onto a quad:
- Use texture coordinates (0, 0), (1, 0), (1, 1), and (0, 1) for the four corners (vertices) of the polygon.

Mapping a Texture

- **`glTexCoord*()`** specified at each vertex



Typical Code

```
glBegin(GL_POLYGON);
   glColor3f(r0, g0, b0);
   glNormal3f(u0, v0, w0);
   glTexCoord2f(s0, t0);
   glVertex3f(x0, y0, z0);
   glColor3f(r1, g1, b1);
   glNormal3f(u1, v1, w1);
   glTexCoord2f(s1, t1);
   glVertex3f(x1, y1, z1);
       .
       .
glEnd();
```

Note that we can use vertex arrays to increase efficiency

Example of Codes

```
#include "image.h“
// texture id
GLuint texid = 0;

// file storing our texture data - must be a power of 2 by a power of 2
// in size, such as 64x64 or 128x256.  100x100 is invalid.
const char *texture_filename = "./texture.bmp";
```

```
Void display(void)
{
        glClear(GL_COLOR_BUFFER_BIT |
GL_DEPTH_BUFFER_BIT);

    // select our texture handle and enable
texturing
        glBindTexture( GL_TEXTURE_2D, texid );
        glEnable( GL_TEXTURE_2D );
        glPushMatrix();
        glRotatef(270,0.0,1.0,0.0);
        drawBox();
        glPushMatrix();
        glBindTexture( GL_TEXTURE_2D, 0 );
    glDisable( GL_TEXTURE_2D );
        glPopMatrix();
        glutSwapBuffers();
}
```

```
void initTexture ()
{
  // read in texture image
  Image *image = new Image();
  image->read( texture_filename );
  if( ! image->good() )
     {
         fprintf( stderr, "ERROR: couldn't load %s\n",
texture_filename );

         fgetc( stdin );
      exit( 1 );
  }
```

...

```
...
//generate texture handle
  glGenTextures(1, &texid );
  // set current texture to our handle
  glBindTexture(GL_TEXTURE_2D, texid );
  // set the magnification / minification filter to linear
sampling
  // (instead of GL_NEAREST for point sampling)
  glTexParameterf(GL_TEXTURE_2D,
GL_TEXTURE_MAG_FILTER, GL_LINEAR);
  glTexParameterf(GL_TEXTURE_2D,
GL_TEXTURE_MIN_FILTER, GL_LINEAR);
  // set the texture coordinate wrap mode to repeat (instead
of GL_CLAMP)
  glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
GL_REPEAT);
  glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
GL_REPEAT);
```

```
..
// pixels will be packed on the byte
  glPixelStorei( GL_UNPACK_ALIGNMENT, 1 );

  // load image data into texture object
  glTexImage2D( GL_TEXTURE_2D, 0, GL_RGB,
                image->getWidth(), image->getHeight(), 0,
                GL_RGB, GL_UNSIGNED_BYTE, image-
>getPixels() );

  // set current texture to none
  glBindTexture( GL_TEXTURE_2D, 0 );

  // free the image memory - no need to store it anymore
  delete image;
}
```

```
Void display(void)
{
        glClear(GL_COLOR_BUFFER_BIT |
GL_DEPTH_BUFFER_BIT);

    // select our texture handle and enable texturing
        glBindTexture( GL_TEXTURE_2D, texid);
        glEnable( GL_TEXTURE_2D );

        glPushMatrix();
        glRotatef(270,0.0,1.0,0.0);
        drawBox();
        glPushMatrix();

        glBindTexture( GL_TEXTURE_2D, 0 );
    glDisable( GL_TEXTURE_2D );
        glPopMatrix();
    // disable texturing to render normally

        glutSwapBuffers();
}
```

```
int
main(int argc, char **argv)
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB |
GLUT_DEPTH);
        glutCreateWindow("red 3D lighted cube");
        glutDisplayFunc(display);
        glutIdleFunc(display);
        initTexture();
        init();
        glutMainLoop();
        return 0;             /* ANSI C requires main to return
int. */
}
```

Q&A

Advanced Texture Mapping Issues

Light Mapping

- Given a complicated lighting condition, how to accelerate and still get real-time frame rates?
- Assuming your environment is diffuse, let’s compute your lighting condition once and then use as texture map

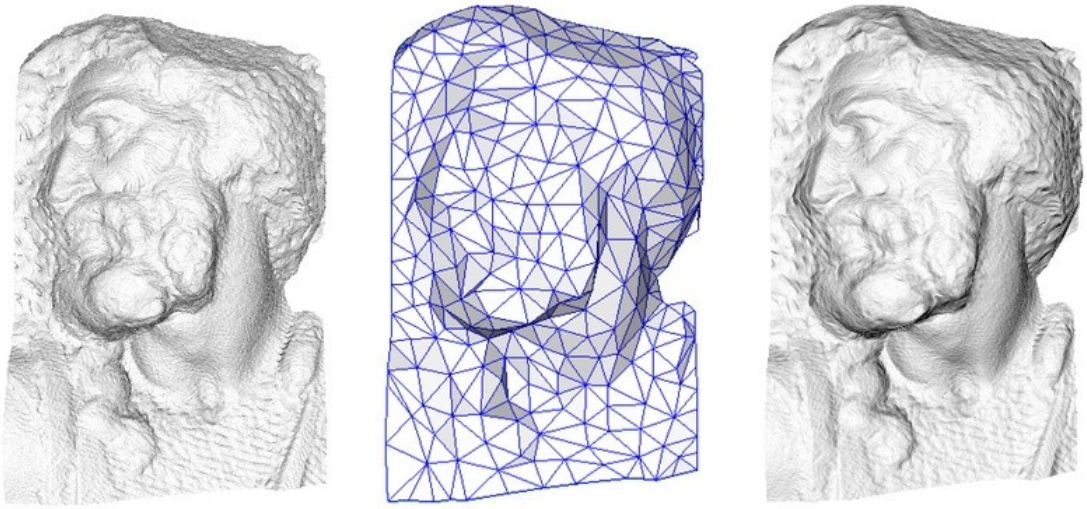
Example

Applying Light Maps

- Use multi-texturing hardware
  - First stage: Apply color texture map
  - Second stage: Modulate with light map
- Pre-lighting textures:
  - Apply the light map to the texture maps as a pre-process
  - When light maps will cause problem?
- Use multi-pass rendering
  - Update the light map dynamically

Normal Mapping

- Store the normals directly in texels.
- Can be generated by sampling the normals of original complex geo
- Ber
  con



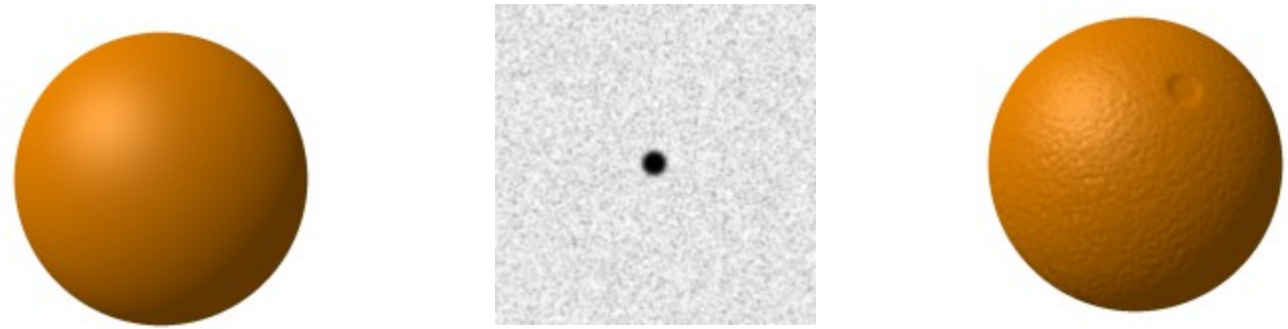
original mesh
4M triangles

simplified mesh
500 triangles

simplified mesh
and normal mapping
500 triangles

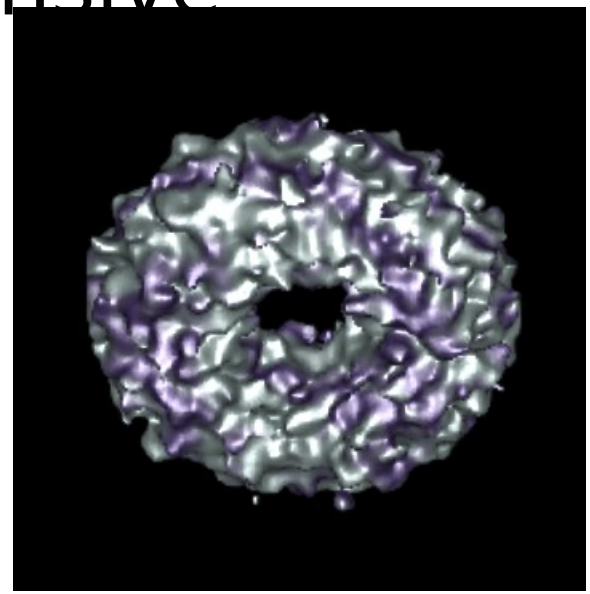
Bump Mapping

- Similar to normal mapping
- Store heightmap in the texture
- Calculate the normal of each pixel based on the heightmap



Displacement Mapping

- Modifies the surface position in the direction of the surface normal.
- Need to generate new geometry, quite expensive



Parallax Mapping

- Aka. Offset Mapping or Virtual Displacement Mapping
- Enhancement of bump/normal mapping.
- Create “deeper” effect
- Uses a height map
- Does not change object geometry

Parallax Mapping

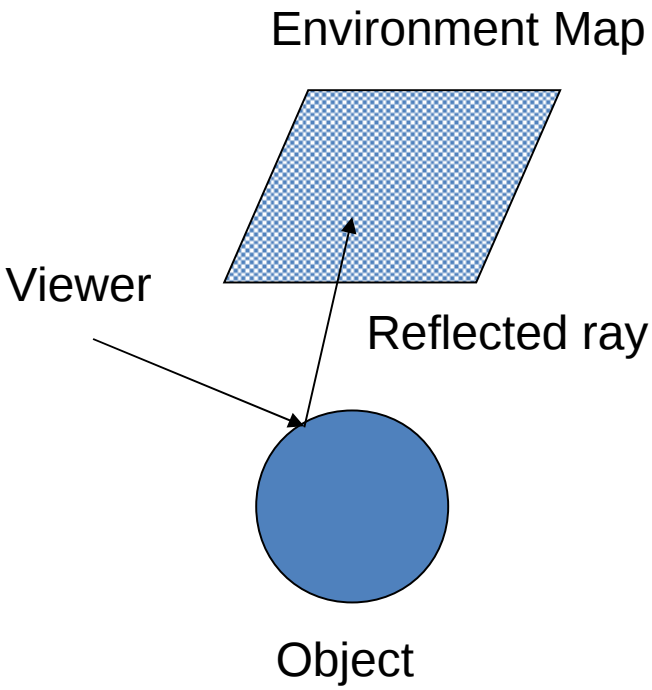
Displacing the texture coordinates at a point on the rendered polygon by a function of the view angle in tangent space (the angle relative to the surface normal) and the value of the height map at that point.

Without Parallex Mapping

With Parallex Mapping

Environment Mapping

- Environment mapping produces reflections on shiny objects
- Texture is transferred in the direction of the reflected ray from the environment map onto the object
- Reflected ray: $\boldsymbol{R}=2(\boldsymbol{N}\cdot\boldsymbol{V})\boldsymbol{N}-\boldsymbol{V}$
- Problems:
  - Flat
  - Near
  - Multi-objects



Environment Mapping