Computer Graphics

Karin Kosina (vka kyrah)

Part 3 (reloaded)

Computer Graphics

Karin Kosina (vka kyrah)

needful things for your toolbox

- fullscreen mode
- repeating key events
- time-based animation
- Mac OS X specific:
  - synchronizing SDL_GL_SwapBuffers() with the vertical refresh

see source code examples...

fullscreen mode

- simply add SDL_FULLSCREEN in SDL_SetVideoMode()

```
if (!SDL_SetVideoMode(width, height, 32, SDL_OPENGL)) ...
```

fullscreen mode

- simply add SDL_FULLSCREEN in SDL_SetVideoMode()

```
if (!SDL_SetVideoMode(width, height, 32, SDL_OPENGL|SDL_FULLSCREEN)) ...
```

fullscreen mode

- simply add SDL_FULLSCREEN in SDL_SetVideoMode()

```
if (!SDL_SetVideoMode(width, height, 32, SDL_OPENGL|SDL_FULLSCREEN)) ...
```

tricolor_fullscreen.cpp

repeating key events

repeating key events

- interaction through key events so far:
  - increase translation/rotation value on key-down

repeating key events

- interaction through key events so far:
  - increase translation/rotation value on key-down
- new and improved interaction through key events:
  - set movement flag on key-down
  - clear movement flag on key-up
  - update animation if movement flag is set

don't do this:

```
int spin = 0;

void mydisplay()
{
  glPushMatrix();
  glRotatef ((float) spin, 0.0, 1.0, 0.0);
  // draw scene here
  glPopMatrix();
}

// in event processing loop

  if (event.type == SDL_KEYDOWN) {
    switch(event.key.keysym.sym) {
      case SDLK_RIGHT:
        spin = (spin + 5) % 360;
        break;
      }
    }
  }
```

movinglight.cpp

```
int spin = 0;
bool spinning = false;

void mydisplay()
{
  if (spinning) spin = (spin + 1) % 360;
  glPushMatrix();
  glRotatef ((float) spin, 0.0, 1.0, 0.0);
  // draw scene here
  glPopMatrix();
}

// in event processing loop

 if (event.type == SDL_KEYDOWN) {
    switch(event.key.keysym.sym) {
      case SDLK_RIGHT:
        spinning =true;
        break;
      }
    }
  } else if (event.type == SDL_KEYUP) {
    switch(event.key.keysym.sym){
    case SDLK_RIGHT:
      spinning = false;
      break;
    }
  }
```

do this instead!

keyrepeat.cpp

time-based animation

- Animation should not depend on frame-rate.
- So far, we did this:

```
// main application loop
bool done = false;
while (!done) {
  mydisplay();
  SDL_GL_SwapBuffers();

  SDL_Event event;
  while (SDL_PollEvent(&event)) {
    if (event.type == SDL_QUIT) {
      done = true;
    } else if (event.type == SDL_KEYDOWN) {
      switch(event.key.keysym.sym) {
      case SDLK_ESCAPE:
        done = true;
      }
    }
  }
}
```

```
void mydisplay()
{
  glClear(GL_COLOR_BUFFER_BIT);
  glLoadIdentity();

  static float rotation = 0.0;
  rotation += 1.0;
  glRotatef(rotation, 0, 0, 1);

  glBegin(GL_TRIANGLES);
  glVertex3f( 0.0f, 1.0f, 0.0f);
  // ... more GL drawing code
}
```

- Changes in framerate are very visible!

time-based animation

- a better way to do it:

```
bool done = false;
unsigned int now, prev = SDL_GetTicks();

while (!done) {
  mydisplay();
  SDL_GL_SwapBuffers();

  SDL_Event event;
  while (SDL_PollEvent(&event)) {
    if (event.type == SDL_QUIT) {
      done = true;
    } else if (event.type == SDL_KEYDOWN) {
      switch(event.key.keysym.sym) {
      case SDLK_ESCAPE:
        done = true;
      }
    }
  }

  now = SDL_GetTicks();
  myanimate(now-prev);
  prev = now;
}
```

```
void myanimate(unsigned int delta)
{
  rotation += delta/10.0;
}


void mydisplay()
{
  glClear(GL_COLOR_BUFFER_BIT);
  glLoadIdentity();

  glRotatef(rotation, 0, 0, 1);

  glBegin(GL_TRIANGLES);
  glVertex3f( 0.0f, 1.0f, 0.0f);
  // ... more GL drawing code
}
```

time-based animation

- a better way to do it:

```
bool done = false;
unsigned int now, prev = SDL_GetTicks();

while (!done) {
  mydisplay();
  SDL_GL_SwapBuffers();

  SDL_Event event;
  while (SDL_PollEvent(&event)) {
    if (event.type == SDL_QUIT) {
      done = true;
    } else if (event.type == SDL_KEYDOWN) {
      switch(event.key.keysym.sym) {
      case SDLK_ESCAPE:
        done = true;
      }
    }
  }

  now = SDL_GetTicks();
  myanimate(now-prev);
  prev = now;
}
```

```
void myanimate(unsigned int delta)
{
  rotation += delta/10.0;
}


void mydisplay()
{
  glClear(GL_COLOR_BUFFER_BIT);
  glLoadIdentity();

  glRotatef(rotation, 0, 0, 1);

  glBegin(GL_TRIANGLES);
  glVertex3f( 0.0f, 1.0f, 0.0f);
  // ... more GL drawing code
}
```

tba.cpp

synchronised swapping

```
// Sync the SDL_GL_SwapBuffers() call with the vertical blank
// (This is for Mac OS X only!)

GLint swap = 1;
CGLSetParameter(CGLGetCurrentContext(), kCGLCPSwapInterval, &swap);
```

textures

textures

textures

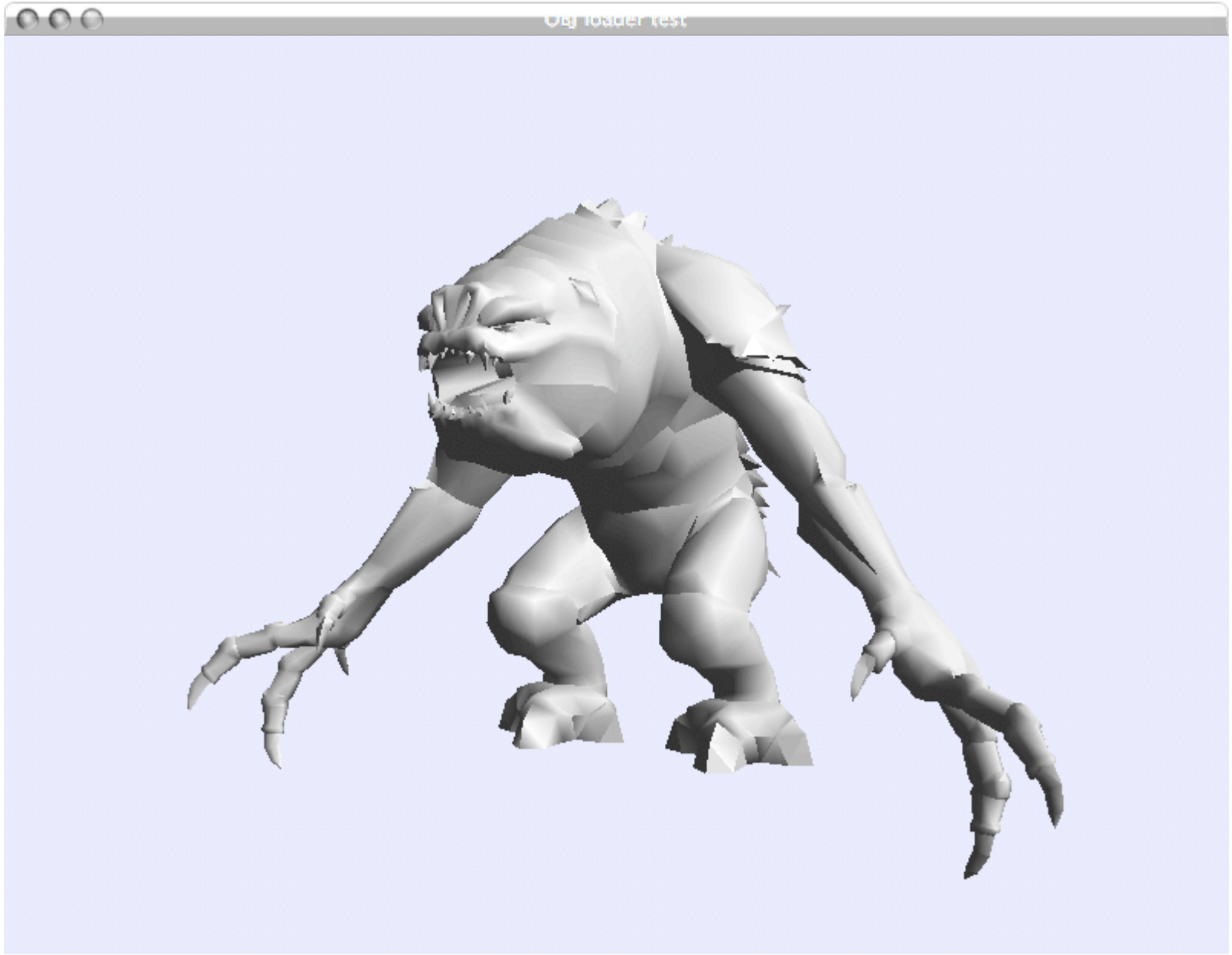
- A texture is an “image” that is mapped to a polygon.

textures

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

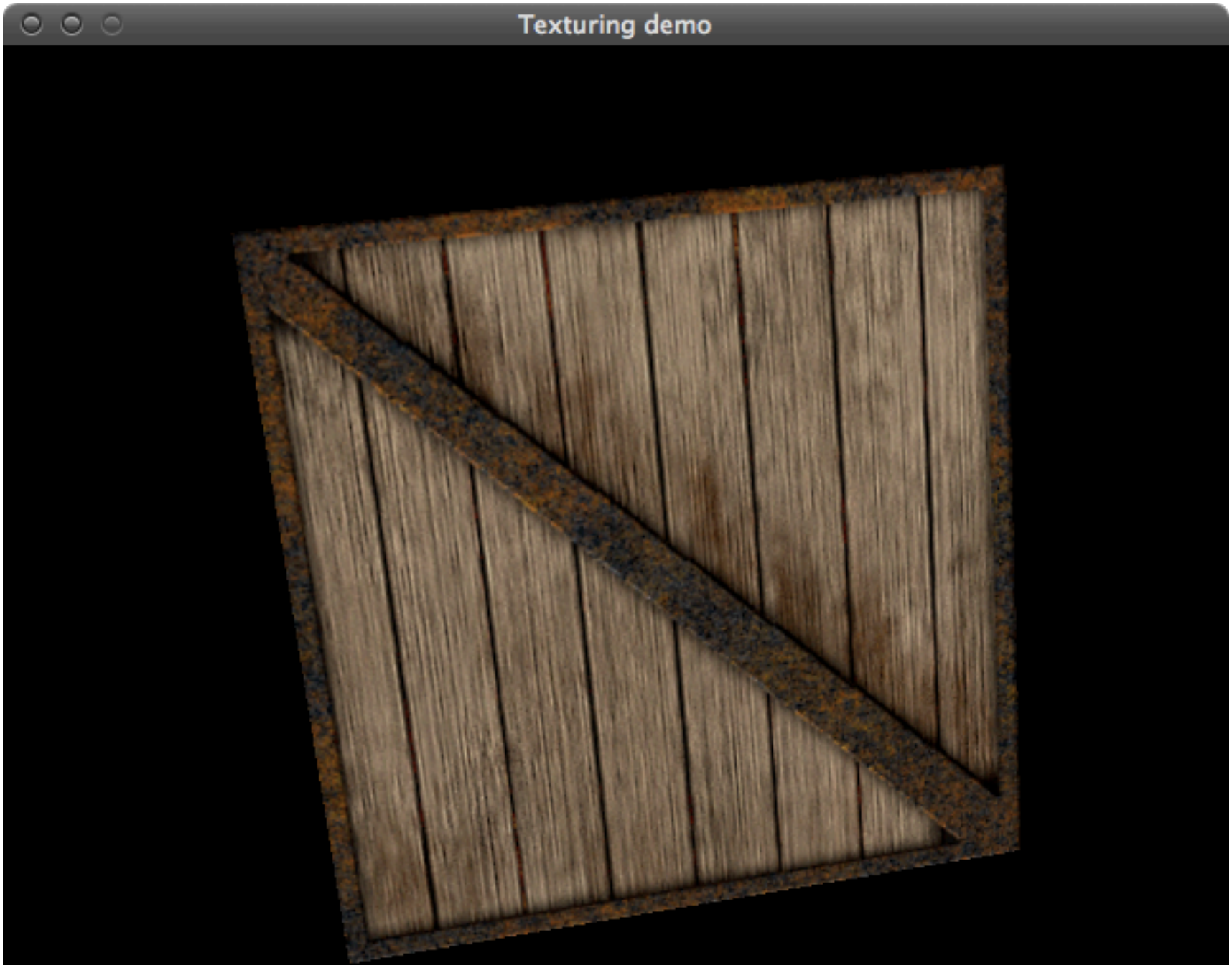
textures

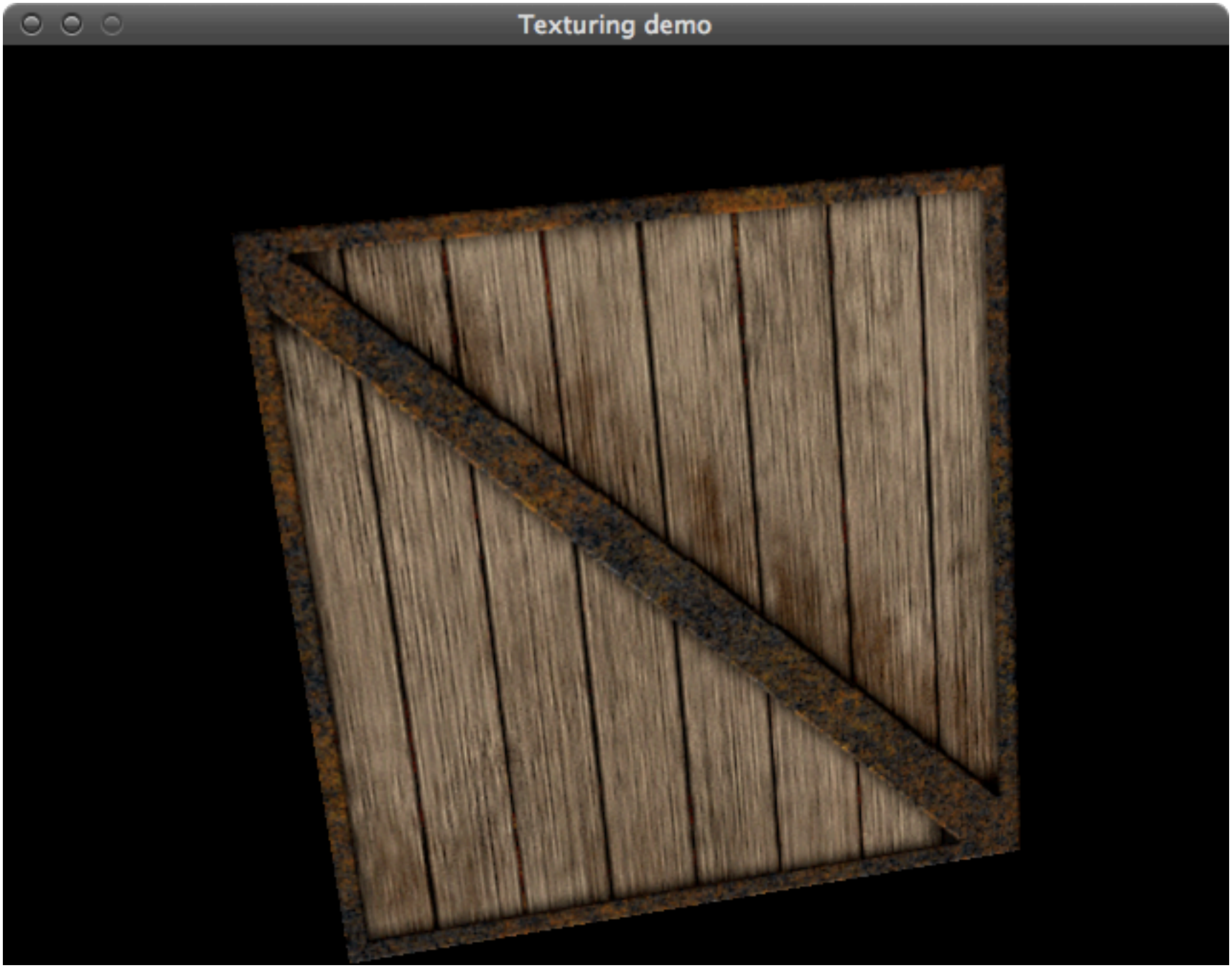
- A texture is an “image” that is mapped to a polygon.
- Textures are rectangular arrays of data (does not have to be 2D)
- Why textures?
  - greater realism
    - real objects are not smooth and regular
  - save resources
    - imaging rendering a brick wall
    - draw every single brick? that’s a lot of polygons!
    - instead glue an image of a brick wall to one large polygon...


Obj loader test


Obj loader test
Obj loader test

texturing example


Texturing demo



texturing.cpp

texturing overview

```
// texture loading - during GL initialisation

Texture * texture = new Texture("crate.png");
texture->load();

// texture usage - during display()

glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, texture->id);

glBegin(GL_QUADS);

  glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, -1.0f,  1.0f);
  glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, -1.0f,  1.0f);
  glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f,  1.0f,  1.0f);
  glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f,  1.0f,  1.0f);

glEnd();

glDisable(GL_TEXTURE_2D);
```

texturing in OpenGL

texturing in OpenGL

behind the scenes

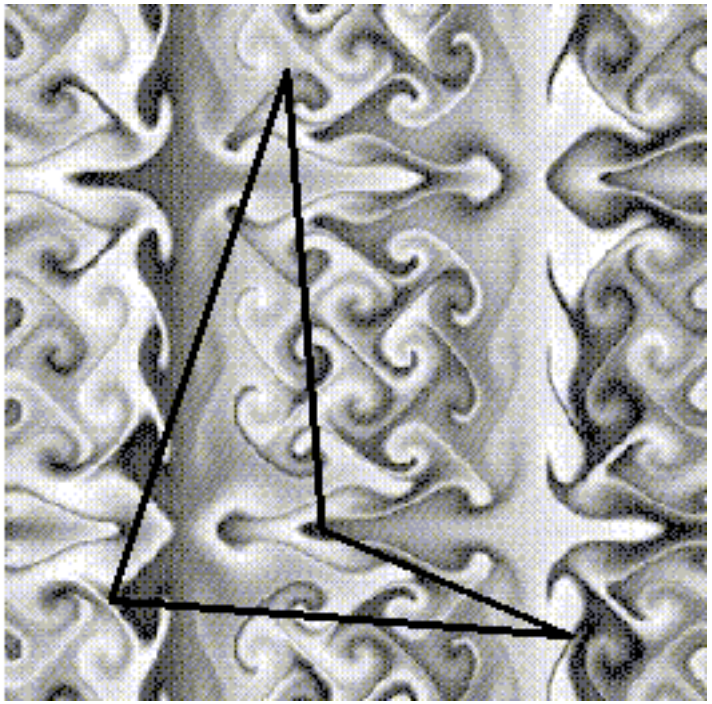
(this is a bit involved)

(so please don't get scared)

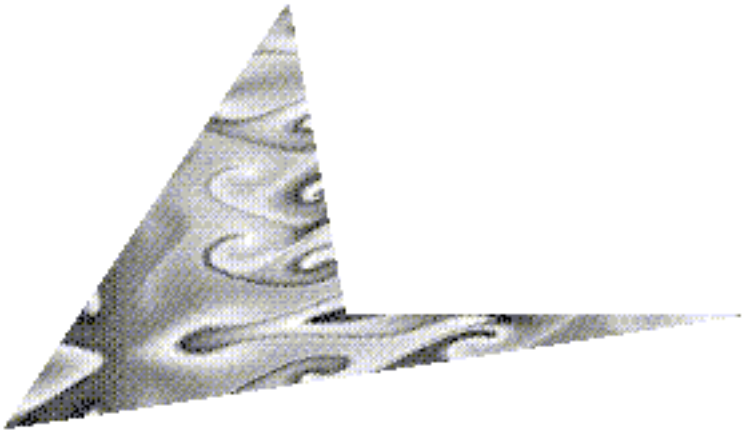
texture mapping

- 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...

texture mapping



Entire texture. Black outline shows quadrilateral and how the texture is mapped to it.



Polygon displayed on the screen. Distorted because of applied transformations. Texture is stretched in the x direction and compressed in the y direction to match this distortion.

texturing in OpenGL: setup

```
Image image("crate.tga");

glGenTextures(1, &texture);
glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
glBindTexture(GL_TEXTURE_2D, texture);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

glTexImage2D(GL_TEXTURE_2D, 0, image.internalFormat(),
             image.w(), image.h(), 0, image.format(),
             GL_UNSIGNED_BYTE, image.pixels());
```

texturing in OpenGL: usage

```
// in mydisplay()

glEnable(GL_TEXTURE_2D);
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
glBindTexture(GL_TEXTURE_2D, texture);

glBegin(GL_QUADS);

// front face
glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, -1.0f,  1.0f);
glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, -1.0f,  1.0f);
glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f,  1.0f,  1.0f);
glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f,  1.0f,  1.0f);

// etc.

glEnd(GL_QUADS);
```

let`s do this step by step

texturing in OpenGL: setup

```
Image image("crate.tga");

glGenTextures(1, &texture);
glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
glBindTexture(GL_TEXTURE_2D, texture);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

glTexImage2D(GL_TEXTURE_2D, 0, image.internalFormat(),
             image.w(), image.h(), 0, image.format(),
             GL_UNSIGNED_BYTE, image.pixels());
```

texturing in OpenGL: setup

```
Image image("crate.tga");

glGenTextures(1, &texture);
glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
glBindTexture(GL_TEXTURE_2D, texture);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

glTexImage2D(GL_TEXTURE_2D, 0, image.internalFormat(),
             image.w(), image.h(), 0, image.format(),
             GL_UNSIGNED_BYTE, image.pixels());
```

texturing in OpenGL: setup

```
Image image("crate.tga");

glGenTextures(1, &texture);
glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
glBindTexture(GL_TEXTURE_2D, texture);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

glTexImage2D(GL_TEXTURE_2D, 0, image.internalFormat(),
             image.w(), image.h(), 0, image.format(),
             GL_UNSIGNED_BYTE, image.pixels());
```

Pixel Storage

- 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

texturing in OpenGL: setup

```
Image image("crate.tga");

glGenTextures(1, &texture);
glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
glBindTexture(GL_TEXTURE_2D, texture);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

glTexImage2D(GL_TEXTURE_2D, 0, image.internalFormat(),
             image.w(), image.h(), 0, image.format(),
             GL_UNSIGNED_BYTE, image.pixels());
```

texturing in OpenGL: setup

```
Image image("crate.tga");

glGenTextures(1, &texture);
glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
glBindTexture(GL_TEXTURE_2D, texture);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

glTexImage2D(GL_TEXTURE_2D, 0, image.internalFormat(),
             image.w(), image.h(), 0, image.format(),
             GL_UNSIGNED_BYTE, image.pixels());
```

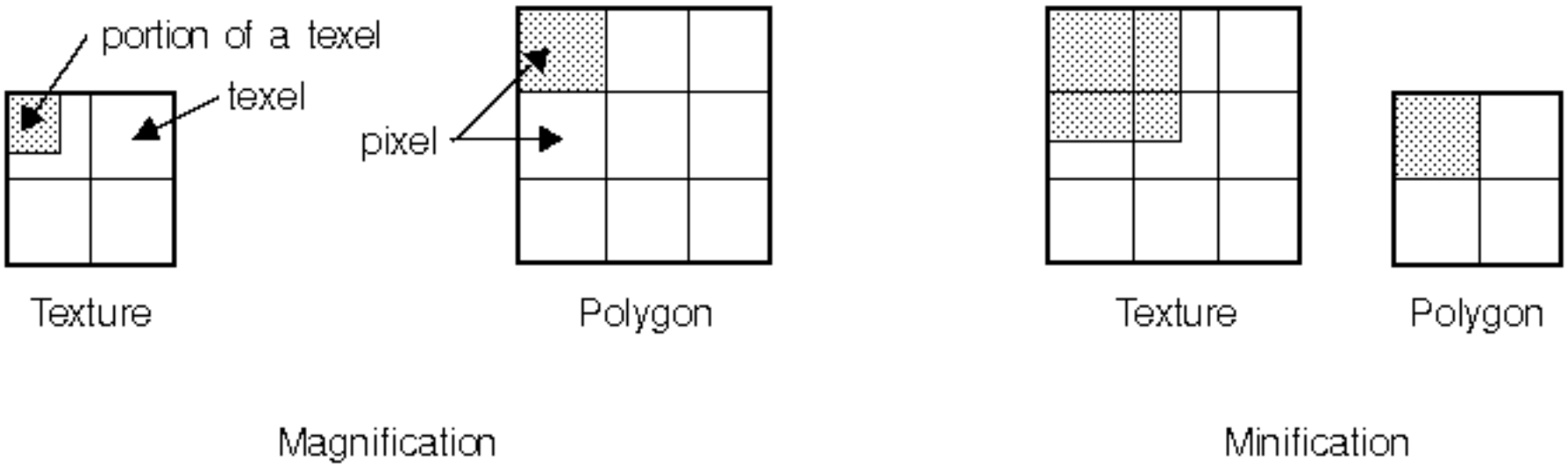
Repeating Textures

- 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

texturing in OpenGL: setup

```
Image image("crate.tga");

glGenTextures(1, &texture);
glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
glBindTexture(GL_TEXTURE_2D, texture);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

glTexImage2D(GL_TEXTURE_2D, 0, image.internalFormat(),
             image.w(), image.h(), 0, image.format(),
             GL_UNSIGNED_BYTE, image.pixels());
```

Filtering



How to map individual texels of a texture to individual pixels on the final screen image?
The texel values must be interpolated/averaged - trade-off between speed and quality

Nearest neighbour interpolation or calculate average by linear interplation?
`GL_NEAREST` vs. **`GL_LINEAR`**

texturing in OpenGL: setup

```
Image image("crate.tga");

glGenTextures(1, &texture);
glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
glBindTexture(GL_TEXTURE_2D, texture);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

glTexImage2D(GL_TEXTURE_2D, 0, image.internalFormat(),
             image.w(), image.h(), 0, image.format(),
             GL_UNSIGNED_BYTE, image.pixels());
```

specifying the texture

- You need some way to get a texture image.
  - either loaded from file or procedurally generated
  - SDL_Image is your friend
- Textures are usually thought of as 2-dimensional.
  - But they can also be 1-dimensional or 3-dimensional.
- 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

specifying the texture

- ```
  void glTexImage2D(GLenum target, GLint level,
                    GLint components, GLsizei width,
                    GLsizei height, GLint border,
                    GLenum format, GLenum type,
                    const GLvoid *pixels);
  ```
  - **`target`** must be **`GL_TEXTURE_2D`**
  - **`level`** should be 0 (needed for MIP-mapping)
  - **`components`** specified which of RGBA are selected for use in modulating/blending
  - **`width/height`** specify texture image dimensions
  - **`border`** specifies the width of the border (usually 0)
  ```

specifying the texture

- ```
  void glTexImage2D(GLenum target, GLint level,
                    GLint components, GLsizei width,
                    GLsizei height, GLint border,
                    GLenum format, GLenum type,
                    const GLvoid *pixels);
  ```
  - **`format`** specifies the 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.
  - **`pixels`** contains the texture-image data
  ```

texturing in OpenGL: usage

```
// in mydisplay()

glEnable(GL_TEXTURE_2D);
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
glBindTexture(GL_TEXTURE_2D, texture);

glBegin(GL_QUADS);

// front face
glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, -1.0f,  1.0f);
glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, -1.0f,  1.0f);
glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f,  1.0f,  1.0f);
glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f,  1.0f,  1.0f);

// etc.
glEnd(GL_QUADS);
```

texturing in OpenGL: usage

```
// in mydisplay()

glEnable(GL_TEXTURE_2D);
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
glBindTexture(GL_TEXTURE_2D, texture);

glBegin(GL_QUADS);

// front face
glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, -1.0f,  1.0f);
glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, -1.0f,  1.0f);
glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f,  1.0f,  1.0f);
glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f,  1.0f,  1.0f);

// etc.
glEnd(GL_QUADS);
```

How to apply the texture?

- How should the final color be computed from the fragment color and the texture-image color?
- 3 modes:
  - 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)

texturing in OpenGL: usage

```
// in mydisplay()

glEnable(GL_TEXTURE_2D);
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
glBindTexture(GL_TEXTURE_2D, texture);

glBegin(GL_QUADS);

// front face
glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, -1.0f,  1.0f);
glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, -1.0f,  1.0f);
glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f,  1.0f,  1.0f);
glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f,  1.0f,  1.0f);

// etc.
glEnd(GL_QUADS);
```

texturing in OpenGL: usage

```
// in mydisplay()

glEnable(GL_TEXTURE_2D);
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
glBindTexture(GL_TEXTURE_2D, texture);

glBegin(GL_QUADS);

// front face
glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, -1.0f,  1.0f);
glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, -1.0f,  1.0f);
glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f,  1.0f,  1.0f);
glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f,  1.0f,  1.0f);

// etc.
glEnd(GL_QUADS);
```

drawing the scene

- 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.

write yourself a texture abstraction class

```
Image image("crate.tga");

glGenTextures(1, &texture);
glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
glBindTexture(GL_TEXTURE_2D, texture);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

glTexImage2D(GL_TEXTURE_2D, 0, image.internalFormat(),
             image.w(), image.h(), 0, image.format(),
             GL_UNSIGNED_BYTE, image.pixels());
```

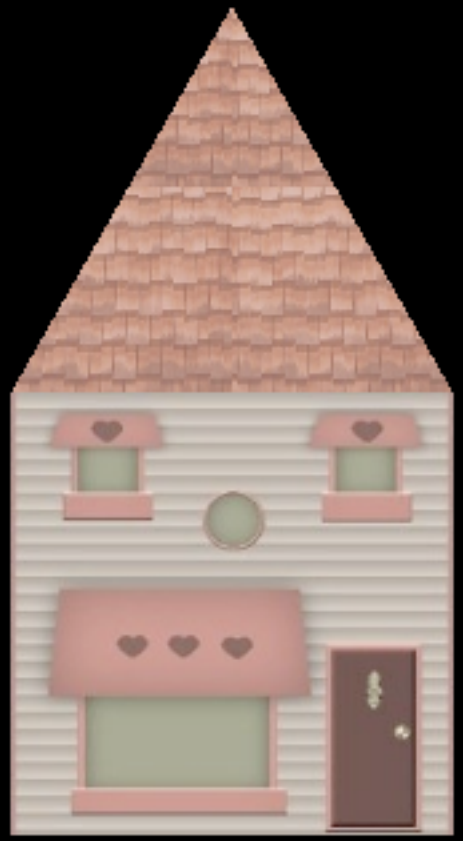
write yourself a texture abstraction class

```
// texture loading
Texture * texture = new Texture("crate.tga");
texture->load();

// texture usage
glBindTexture(GL_TEXTURE_2D, texture->id);
```

write yourself a texture abstraction class

```
// texture loading
Texture * texture = new Texture("crate.tga");
texture->load();

// texture usage
glBindTexture(GL_TEXTURE_2D, texture->id);
```

Feel free to use mine, but be aware that this is not at all production quality software. (If it breaks, you get to keep both halves.)



thanks! :)