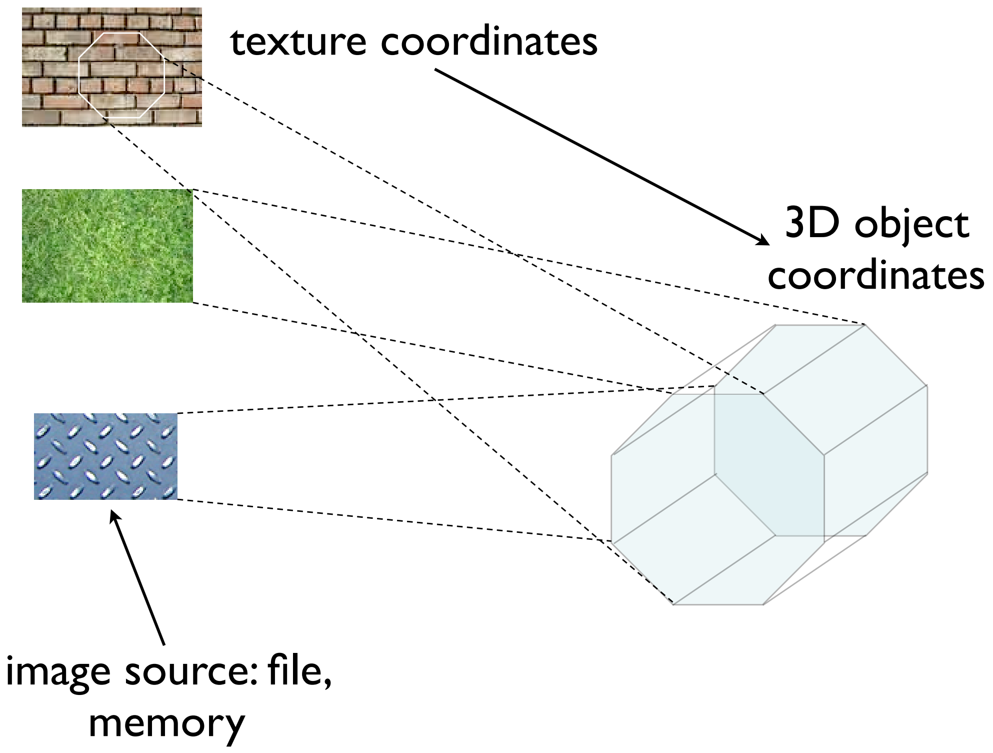
Texture Mapping

- Progression of rendering sophistication:
  - Absolute color, without shading due to lights
  - Lighting results in a new approach to setting color (i.e. "materials" with many attributes)
- Textures add one more layer to the mix
  - In its most generic form, a *texture* is a separate data source of color or material information
  - *Texture mapping* is the act of determining what color from this data source should be applied to a particular point in the corresponding 3D object



General Steps

- Load/initialize the texture data
    - ◇ Typically a 2D image, but can be 1D or 3D as well
    - ◇ Typically read from a file, but of course this requires image file decoding libraries (like Java's ImageIO API)
        - Note that the file-reading phase is distinct from texture mapping in general!
        - For testing purposes, you can always try building a texture in memory first

- Next, set how the texture colors are "converted" into the 3D space
    - ◇ Is texture color == final 3D color?
    - ◇ Should texture color be affected by lighting?
    - ◇ Should texture color be blended with any other color?
- Set how the texture space "maps" into the 3D space
    - ◇ Analogous to applying a "cookie cutter" or "stencil" to the texture data that corresponds to the shape that is being drawn by *glVertex*()*

OpenGL Texture Specifics

- State machine, as usual — any texture-related values that you set will remain active until you explicitly set them to something else
- Textures are "objects" — you need to ask OpenGL to make room for them first with *glGenTextures()*
- OpenGL needs to know how to "read" the image data that you will use for a texture
  - *glPixelStore*()* sets how the bytes are read into pixels
  - Part of *glTexImage2D()* specifies the format of those pixels

- Texture settings are provided in various ways:
  - *glTexParameter*()* — like *glMaterial*()* and *glLight*()*
  - *glMatrixMode(GL_TEXTURE)* — texture transforms
- Match the texture to its core image data: *glTexImage2D()*
- Specify the "current" texture: *glBindTexture()*

- **Don't panic!** Texture mapping is one of those techniques that *really* requires a deeper understanding of computer graphics — particularly 2D image representation — before it makes complete sense

Other Texture Features

Texture-related abilities that may be of interest…

- OpenGL provides a huge variety of ways to interpret byte sequences into texture data
- There is also a correspondingly huge variety of ways to convert the "raw" color in a texture to the "final" color in the object
- *Subimages* — instead of loading up multiple textures, load a large composite image as a single texture, and grab only subsets of that image

- The textures you see here and in the sample code are two-dimensional; 3D textures are possible, and these map into 3D spaces!
- *Lighting options* — particularly specular lighting: before or after texture mapping?
- *Mipmaps* allow some degree of resolution independence — store multiple sizes of the same texture, depending on the size of the 3D object

We won't go into these in detail, but they are listed here in case you have an individual or personal interest in exploring them further (because one or more of these specifically apply to your project, for example).

In the “Because We Can” Department: Fog

- Conceptually very simple; checking out the Nate Robins tutorial (or reading the red book) should be sufficient to learn how to use it

- Complications arise mainly in the “fog algorithm,” not in how to use the feature