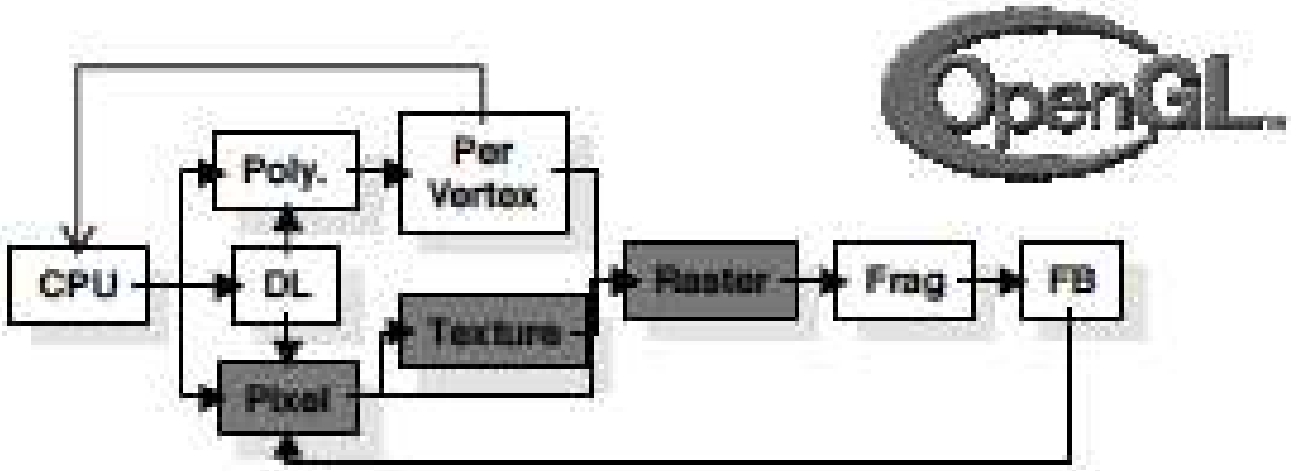
Texture Mapping in OpenGL

Professor Raymond Zavodnik

Fachhochschule Regensburg, Germany, May 13, 2012

What is Texture Mapping?

- Until now only (shaded) color has been applied to object surfaces
- We can also in some sense augment those colors with images which can be
  - Computer-generated, e.g. a checkerboard
  - Camera images, e.g. a digital photograph
- This can greatly simplify scene geometry
- Instead of writing down vertices for a brick wall, we can just apply a texture to a surface of that wall
- We can also apply images to achieve image-warping, e.g. a (rectangular) photograph can be applied to a sphere
- Texture mapping allows simulating material properties of real-world surface


OpenGL
Poly.
Per Vertex
CPU
DL
Texture
Raster
Frag
FB
Pixel

Types of Textures

There are basically four types:

1. One Dimensional, for use when there is only one degree of variation
2. Two Dimensional, which is the usual case for use of photographs
3. Three Dimensional, used when there are three dimensions of variations, e.g. visualizing tomography data
4. Environmental, used when the environment surrounding a scene is reflected on objects in the scene (Cube Map)
5. Can also be used for placing shadows of objects in lit scenes

Development

- OpenGL 1.1: Texture Objects have both texture images and state parameters. Allow caching textures in a working set in texture memory
- OpenGL 1.3 Multiple textures were introduced allowing several textures to be applied to a simgle primitive, allowing shadows or lighting effects
- OpenGL 1.4 Support for depth-map shadow algorithms

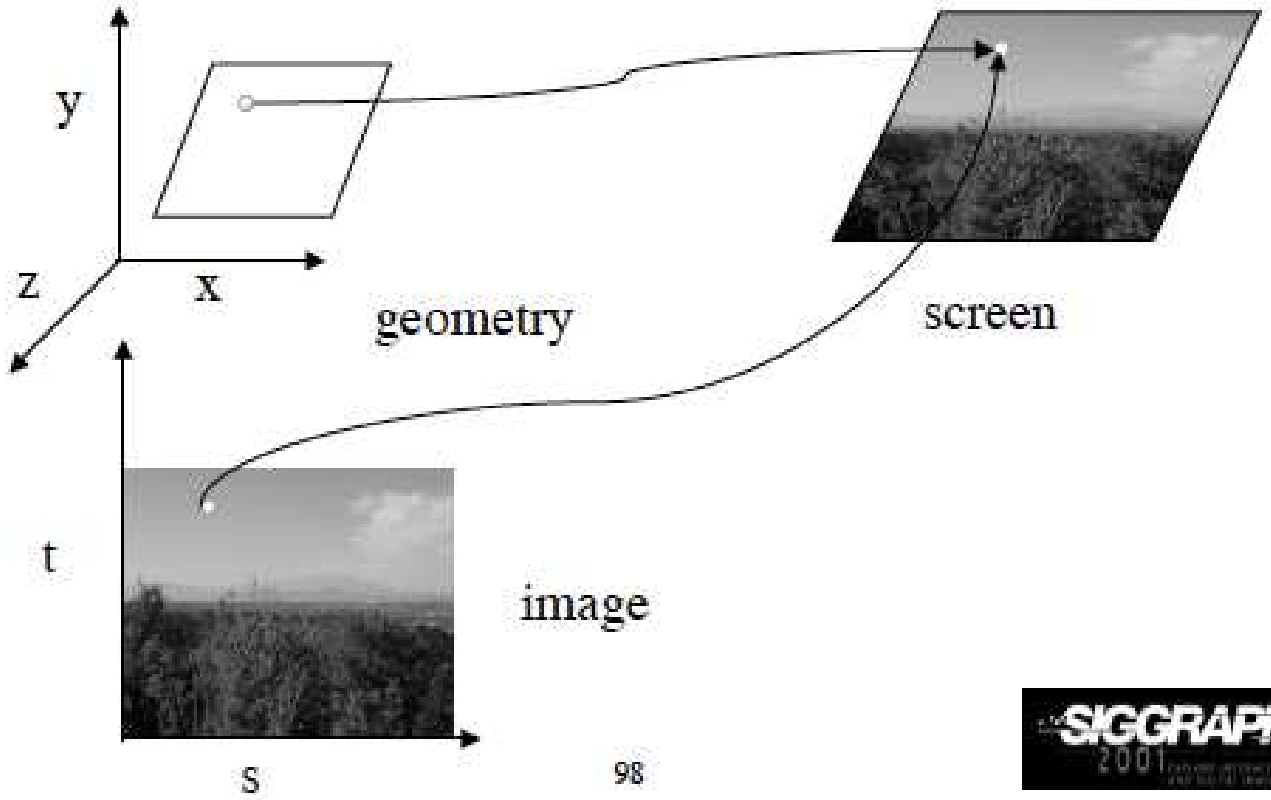

y
z
x
geometry
screen
t
image
s
98

Image Maps and Textures

- The pixels of the (2D) image textures are called **Texels**
- In 3D they are called **Voxels**
- OpenGL does not provide tools for creating textures
- Try `DevIL`, which can be downloaded from an OpenGL subsidiary
- We will use a C-program that has the (converted) image data
- There are some restrictions, e.g. The image size in both directions must be a power of 2 (but not the same power)

How Textures Are Generated

- Image pixels and primitive vertices flow through separate pipelines that come together at the rasterizer stage
- Visual detail is included in the image, not in the geometry
- Texture is added to the geometry during the rasterization of its fragments
- Texture is employed together with lighting and material calculation
- OpenGL keeps separate coordinates for textures (s, t)
- Texture coordinates are applied to vertices, just like color, normals, etc.

Steps in Texture Mapping

These are listed for 2D-Textures:

1. Obtain an unused **Texture Object** identifier (`glGenTextures()`) and create a **Texture Object** (`glBindTexture()`) in an initialization phase
2. Set the texture object's **State Parameters**. This involves how texture is to be wrapped and colors filtered
3. Specify the actual texture image (`glTexImage2D()`), giving size information, texture location, etc.
4. Prepare for rendering by **binding** the texture to the rendering context
5. Also enable the texture mapping (`glEnable(GL_TEXTURE_2D);`)
6. Send the geometry to OpenGL with assignment of texture coordinates to the sequence of vertices

Texture Objects

- Set byte alignment with: `glPixelStorei(GL_UNPACK_ALIGNMENT, 1);`
- A **Texture Object** is a structure that stores a texture and its associated states
- These allow switching between textures in an efficient manner
- Some OpenGL implementations allow the use of **Working Sets** to store textures in dedicated texture memory for fast access and application

Texture Objects

- In OpenGL 1.1 it was necessary to specify the texture image and its associated state before each use
- To use texture objects you only need to refer to their **Identifiers**
- These are allocated with `glGenTextures(GL_size n, GLuint* textures);`

Obtaining Texture Objects

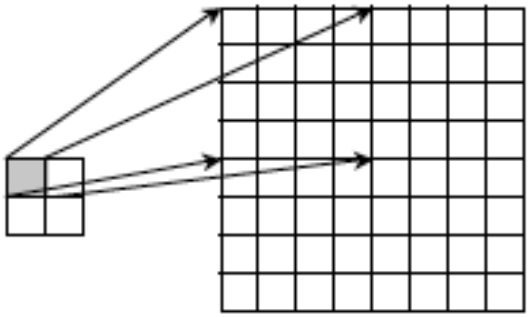
```
GLuint texId[2];
glGenTextures(2, &texId);
```

- During initialization bind it with, for example, `glBindTexture(GL_TEXTURE_2D, texId[1]);`. This is only once use of this function
- Bind and rebind later when setting the rendering context
- Later calls activate the texture and its state
- Do this immediately before the actual redraw

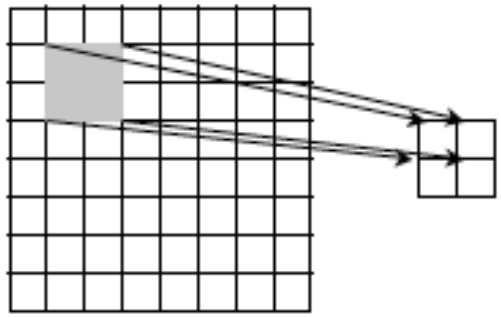
Setting the Texture Object State

- This is where **Wrapping Modes** and **Filtering** are applied
- Admittedly complex, but necessary
- There are two issues that must be addressed:
  1. The texture may not fit onto the geometry, so that there is not a 1-1 correspondence between the primitive fragments and the textures pixels, or better, texels
  2. The texture may need to be **tiled** onto the primitive
- There are also image quality issues
- There are issues with **Aliasing Artefacts**

The Need For Filtering



Magnification



Minification

110

The Need For Filtering

- Use `glTexParameteri()` with appropriate parameters
- During rasterization, if a fragment covers *less* area than a texel of the texture image then use a value set by `GL_TEXTURE_MAG_FILTER` to help calculate the texture color. For example, the value `GL_LINEAR` to interpolate between texel values. This process is **Magnification**
- During rasterization, if a fragment covers *more* area than a texel of the texture image then use a value set by `GL_TEXTURE_MIN_FILTER` to help calculate the texture color. Here the default is `GL_NEAREST_MIPMAP_LINEAR`, which interpolates between **Mipmap levels**. This process is called **Minification**

What the *** is Mipmapping?

- Create several approximations of the original texture at lower resolution,called *levels*
- Each succeeding level should have resolution one-half the preceeding one
- You can have OpenGL calculate these for you
- Avoids the flashing effect when viewing a texture from a distance
- From OpenGL 1.2 on even the level used is automatic

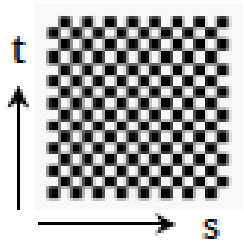
Wrapping Textures

- What to do if the texture does not fill the primitive to which it is applied?
  1. You can **Clamp** it
  2. You can **Wrap** it by tiling
- While tiling the integer parts of the texture coordinates are ignored
- Using clamping the texture coordinates larger than 1.0 are set to 1.0 and values less than 0.0 are set to 0.0
- This is useful when you want a single texture applied
- Borders can be used for more control over fitting textures
- Use `GL_REPEAT` for things like a checkerboard

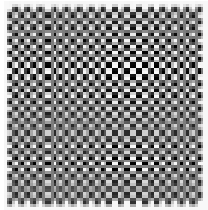
Summary of Wrapping Modes

Parameter	Values
GL_TEXTURE_WRAP_S	GL_CLAMP, GL_CLAMP_TO_EDGE GL_CLAMP_TO_BORDER,GL_REPEA GL_MIRRORED_REPEAT
GL_TEXTURE_WRAP_T	GL_CLAMP, GL_CLAMP_TO_EDGE GL_CLAMP_TO_BORDER,GL_REPEA GL_MIRRORED_REPEAT
GL_TEXTURE_WRAP_R	GL_CLAMP, GL_CLAMP_TO_EDGE GL_CLAMP_TO_BORDER,GL_REPEA GL_MIRRORED_REPEAT
GL_TEXTURE_MAG_FILTER	GL_NEAREST,GL_LINEAR

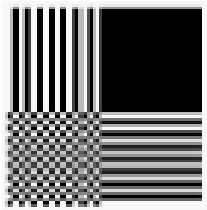
Wrapping Mode



texture



GL_REPEAT
wrapping



GL_CLAMP
wrap

112

Controlling How Texturing is Applied

- Remember: There is (usually) also an underlying color
- How and whether this should be used in calculating the texture
- Use the function `glTexEnv*()`
  - `GL_MODULATE`:Multiply texel and fragment colors
  - GL_BLEND:Linearly blend between texel color and the enironment color, set by `GL_TEXTURE_ENV_COLOR`
  - `GL_REPLACE`:Use the texel color only
- For example:
  ```
  glTexEnvf(GL_TEXTURE_ENV,
  GL_TEXTURE_ENV_MODE, GL_REPLACE);
  ```
  sets the drawing mode to `GL_REPLACE`, which means just use the texture colors

Specifying the Texture

- Use `glTexImage2D*()`
- Parameters are:
  1. `GL_TEXTURE_2D` for now
  2. Mipmap level (just use 0)
  3. Components–use `GL_RGB` or `GL_RGBA`
  4. The image's width and height
  5. An optional border–use 0
  6. Format–use same as Components
  7. Type, usually `GL_UNSIGNED_BYTE`
  8. The address of the image

Enabling Texturing

- Because of the dynamic naturing of assigning active textures it is necessary to call `glBindTexturei()` again, this time to activate the texture and its associated state
- **Enable** the texture with `glEnable(GL_TEXTURE_2D);`, which turns on texturing
- Turn texturing off with `glDisable(GL_TEXTURE_2D);`

Assigning Texture Coordinates

- This is last stage of texturing
- Assign texture coordinates ($0.0 \leq s, t \leq 1.0$)
- Use Vertex Arrays or Buffer Obects to specify the texture coordinates
- Enable texture coordinate generation for s, t with `glEnable(GL_TEXTURE_GEN_{S,T})`