Module 5:

- Texture Mapping

Today's Workshop:

- Workshop 5: Texture mapping.

Working with Images in Memory

- An image is a matrix of pixels
- Stored in memory as a matrix or array – each number represents pixel colour
- Pixels are retrieved by accessing the array (eg. pixel at row **R**, column **C**):

2D Matrix

```
– char image[480][640];
– char pixel = image[R][C];
```

Static size Array

```
– char image[640*480];
– char pixel = image[(R-1)*640 + C]; // need to convert row-column format to
                                     // array index
```

Dynamic size Array

```
– char *image = NULL;
– image = new char [640*480];
– char pixel = image[(R-1)*640 + C]; // same as for a  static array
– if (image != NULL) delete [] image;  // after we are done with the image
– image = NULL;                        // free the memory and set to NULL
```

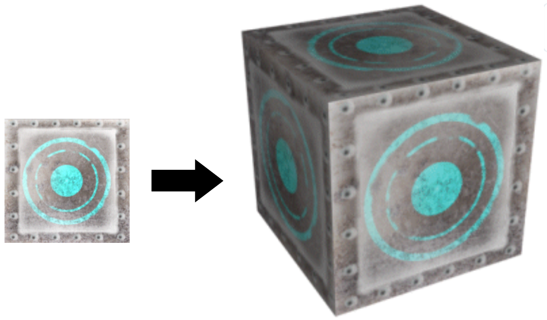
Loading and Saving Images

- Various image storage formats exist:
  - Uncompressed.
  - Lossless and lossy compression.
  - Raw image data v's a header with image information.
  - Open v's Proprietary standards.
- Writing a loader to take an image from a file and put it in a data structure is the easiest for uncompressed images, especially ones with OS support, eg. '.bmp'
- Image handling API's exist to let us work with images without caring about format on disk or how data types are implemented.

Texture Mapping

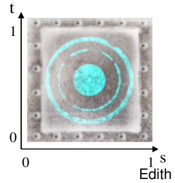
Texture Mapping

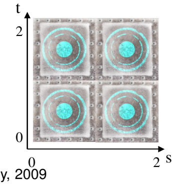
- Texture mapping is the process of applying a picture to a polygon.



Texture Mapping

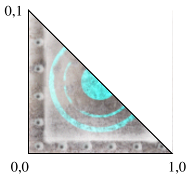
- 2D Texture Map
  - Each pixel in a texture map is called a texel
  - Measured in s, t coordinates
  - s and t range from 0 to 1 for a single picture
- Texture Wrapping (tessellating)
  - When a texture is used as a tessellating texture the u and v co-ordinates going beyond 1 will typically wrap around returning the same image.

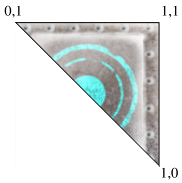


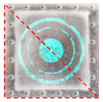


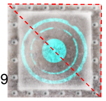
Texture Mapping

- s,t vertex information
  - every vertex is given a s,t coordinate
  - A polygons s,t information defines a polygon on the texture map.
  - Thus the texture can be mapped to a polygon.





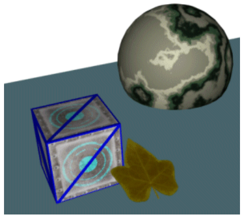


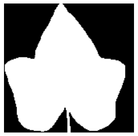


Texture Mapping

- Alpha channel
- Explicit information giving the Opacity of each texel

The leaf in the following scene was modelled using a quad and a texture with an opacity map.





1D & 3D Texture Mapping

- 3D texture maps
  - s,t, r vertex information
  - Texture can be mapped to a polygon by taking 2D slices from the texture.
  - Useful in situations where texture will present different layers
    - Wood, Marble
    - Rock formations
    - Visualising volumetric data – Medical imaging, fog
- 1D texture maps
  - Same as a 2D texture with height of 1 pixel
  - Can be wrapped (repeated) to achieve some consistent effect across a polygon
    - *Used as a ramp to texture an item in a common gradient.*

Texture Mapping in OpenGL

- As with most things related to drawing, OpenGL makes things easy.
  - Once texture mapping is set up, all that needs to be done is to specify the textures [s,t] coordinates at each vertex.
  - OpenGL then takes care of mapping the texture image so that it goes through the same transformations as the polygon and shows up on the screen.
- Textures are images, such as those discussed in the previous section – either loaded in from a file, or generated procedurally.
- To speed up texture mapping we don’t work with the image directly – instead we create an OpenGL texture ‘object’ and put the image in it. OpenGL then manages the object and we can delete the original image data structure.

Working with OpenGL Texture Objects

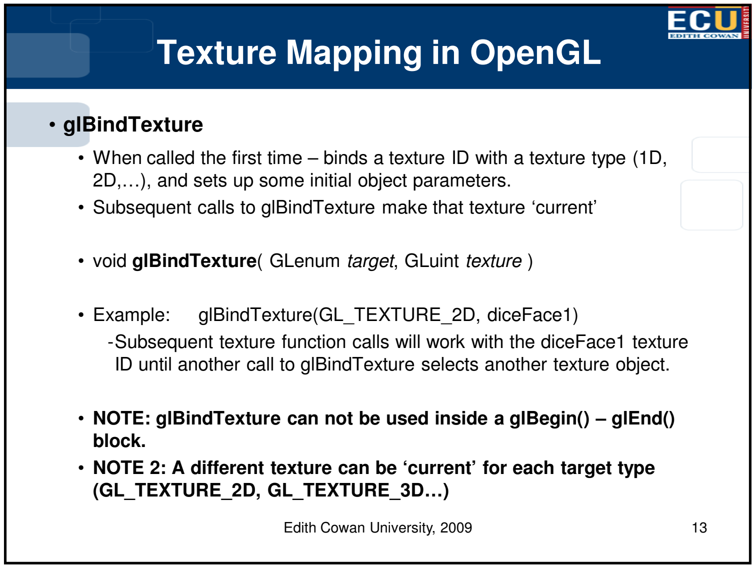
Just like a VBO – its data stored on the graphics card

Assuming the image is already loaded in, this is what we do:

- Enable texture mapping
  - glEnable(GL_TEXTURE_2D); // enabling 2D texture mapping
- Get a unique identifier for the texture object
  - glGenTextures(…
- Make the texture object current and set initial values
  - glBindTexture(…
- Set some texture object parameters
  - glTexParameter(…
- Put the image into the texture object
  - glTexImage2D(… (for 2D textures)
  - After this its all right to delete the original image.
  - The texture can now be used.
- When finished, delete the texture object
  - glDeleteTextures(…

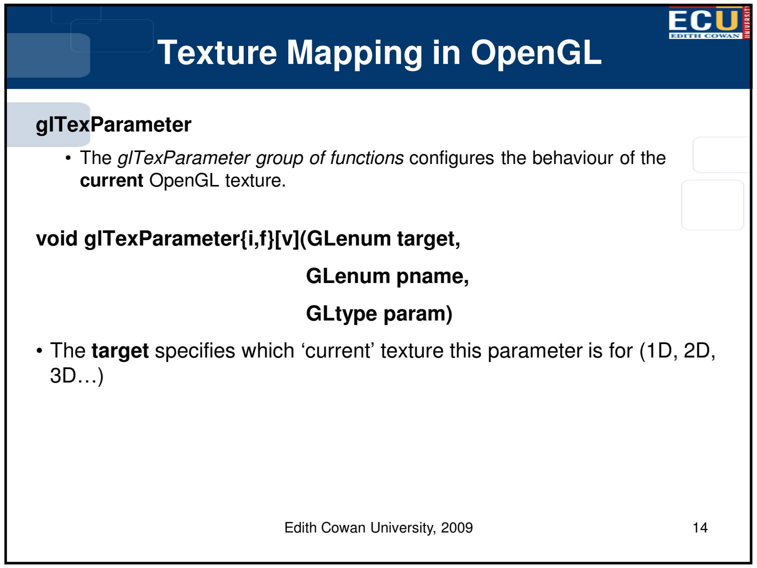
Texture Mapping in OpenGL

- **glGenTextures**
  - Returns a unique unsigned integer(s) corresponding to unused texture object ID(s)
  - void **glGenTextures**(GLsizei n, GLuint **textures*)
  - Example: Get a single free texture ID
    ```
    GLuint diceFace1;
    glGenTextures(1, &diceFace1);
    ```
  - Example: Get a six free texture IDs (stored in an array)
    ```
    GLuint diceFace[6];
    glGenTextures(6, diceFace);
    ```
  - glIsTexture – Determines whether a number is currently assigned to a texture
  - glDeleteTextures – Use before ending the program to prevent resource leaks



Texture Mapping in OpenGL

- **glBindTexture**
  - When called the first time – binds a texture ID with a texture type (1D, 2D,…), and sets up some initial object parameters.
  - Subsequent calls to glBindTexture make that texture ‘current’
  - void **glBindTexture**(GLenum *target*, GLuint *texture*)
  - Example: glBindTexture(GL_TEXTURE_2D, diceFace1)
    - -Subsequent texture function calls will work with the diceFace1 texture ID until another call to glBindTexture selects another texture object.
  - **NOTE: glBindTexture can not be used inside a glBegin() – glEnd() block.**
  - **NOTE 2: A different texture can be ‘current’ for each target type (GL_TEXTURE_2D, GL_TEXTURE_3D…)**



Texture Mapping in OpenGL

glTexParameter

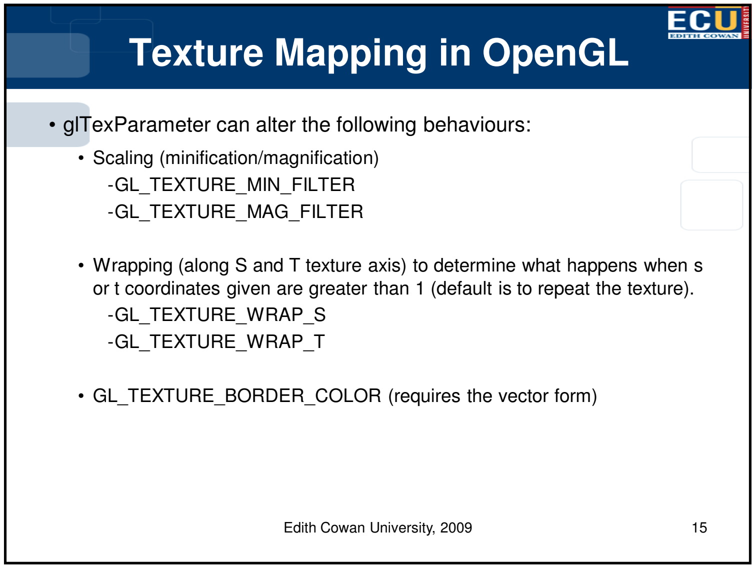
- The *glTexParameter group of functions* configures the behaviour of the **current** OpenGL texture.

void glTexParameter{i,f}[v](GLenum target,

GLenum pname,

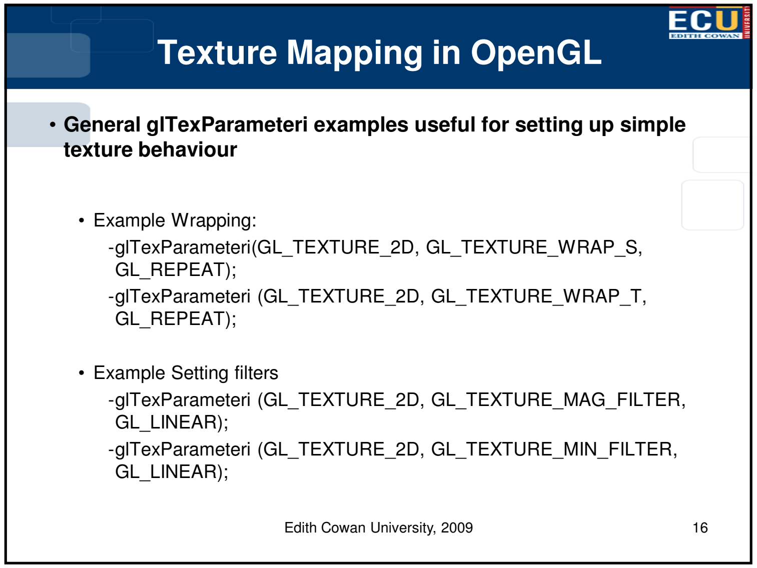
GLtype param)

- The **target** specifies which ‘current’ texture this parameter is for (1D, 2D, 3D…)



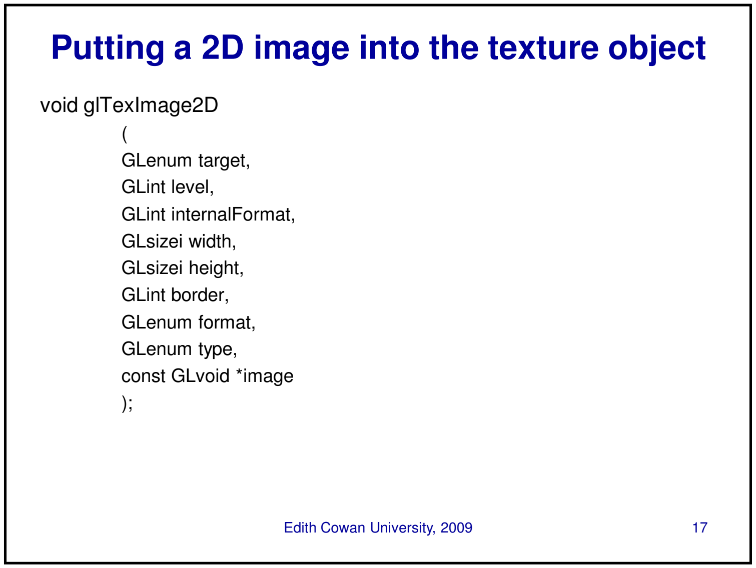
Texture Mapping in OpenGL

- glTexParameter can alter the following behaviours:
  - Scaling (minification/magnification)
    - -GL_TEXTURE_MIN_FILTER
    - -GL_TEXTURE_MAG_FILTER
  - Wrapping (along S and T texture axis) to determine what happens when s or t coordinates given are greater than 1 (default is to repeat the texture).
    - -GL_TEXTURE_WRAP_S
    - -GL_TEXTURE_WRAP_T
  - GL_TEXTURE_BORDER_COLOR (requires the vector form)



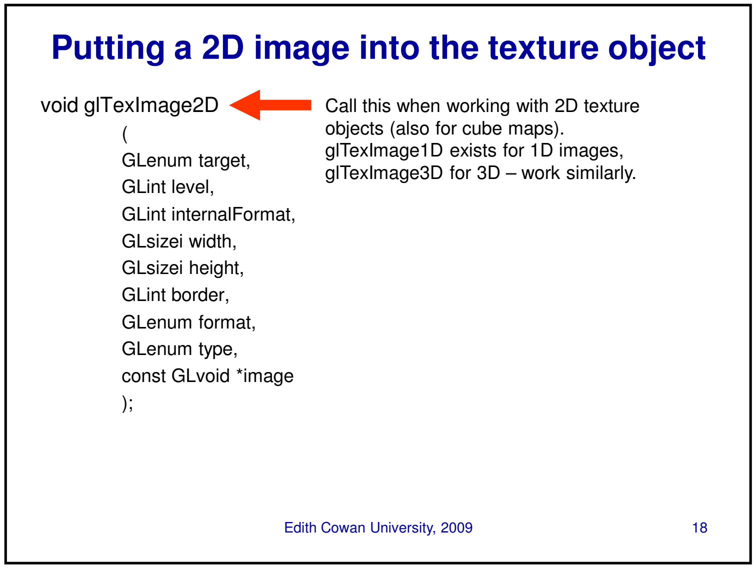
Texture Mapping in OpenGL

- **General glTexParameteri examples useful for setting up simple texture behaviour**
  - Example Wrapping:
    - -glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    - -glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
  - Example Setting filters
    - -glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    - -glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);



Putting a 2D image into the texture object

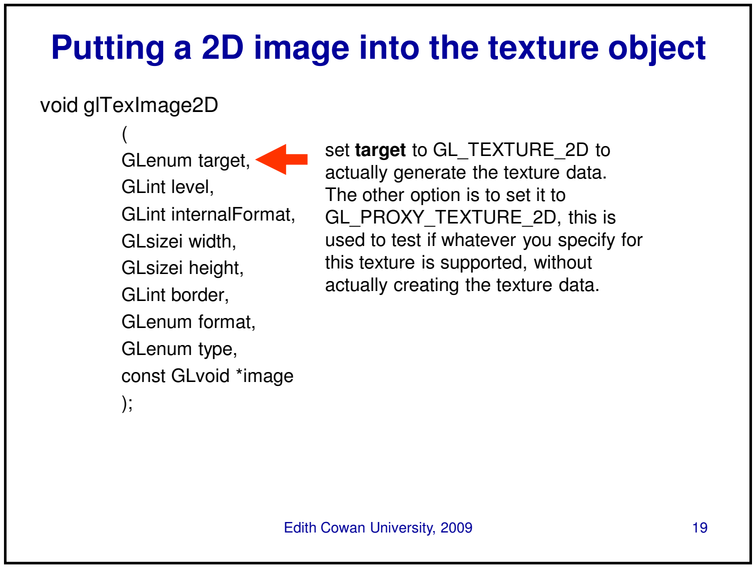
```
void glTexImage2D
        (
        GLenum target,
        GLint level,
        GLint internalFormat,
        GLsizei width,
        GLsizei height,
        GLint border,
        GLenum format,
        GLenum type,
        const GLvoid *image
        );
```



Putting a 2D image into the texture object

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

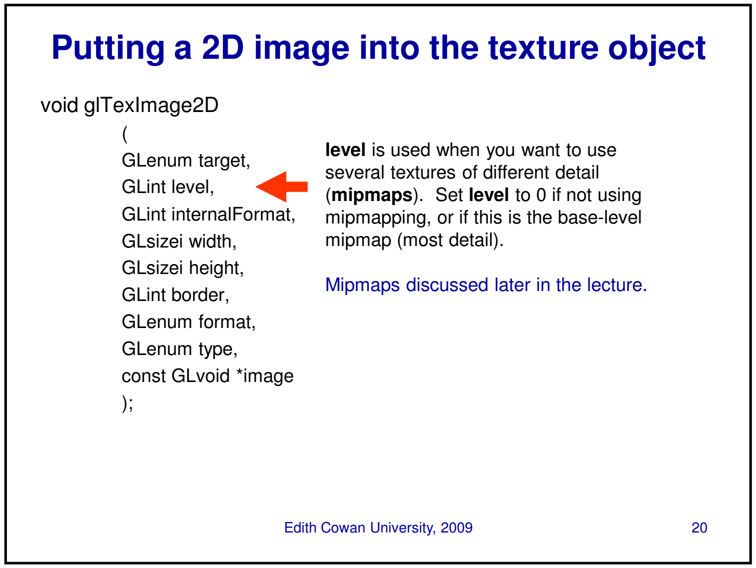
Call this when working with 2D texture objects (also for cube maps). glTexImage1D exists for 1D images, glTexImage3D for 3D – work similarly.



Putting a 2D image into the texture object

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

set **target** to GL_TEXTURE_2D to actually generate the texture data. The other option is to set it to GL_PROXY_TEXTURE_2D, this is used to test if whatever you specify for this texture is supported, without actually creating the texture data.

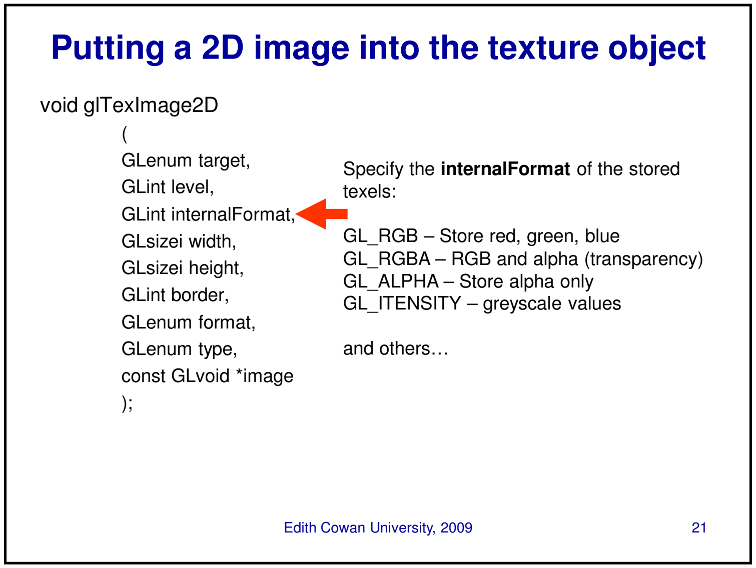


Putting a 2D image into the texture object

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

level is used when you want to use several textures of different detail (**mipmaps**). Set **level** to 0 if not using mipmapping, or if this is the base-level mipmap (most detail).

Mipmaps discussed later in the lecture.



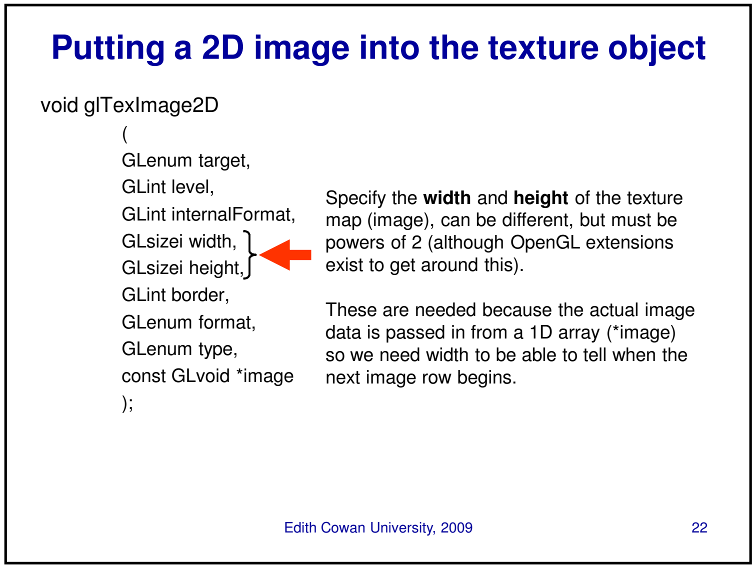
Putting a 2D image into the texture object

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

Specify the **internalFormat** of the stored texels:

GL_RGB – Store red, green, blue
GL_RGBA – RGB and alpha (transparency)
GL_ALPHA – Store alpha only
GL_ITENSITY – greyscale values

and others…

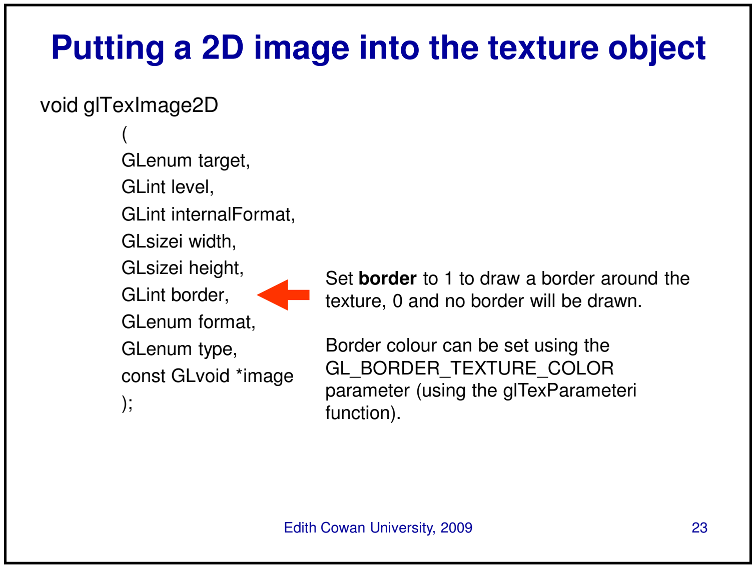


Putting a 2D image into the texture object

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

Specify the **width** and **height** of the texture map (image), can be different, but must be powers of 2 (although OpenGL extensions exist to get around this).

These are needed because the actual image data is passed in from a 1D array (*image) so we need width to be able to tell when the next image row begins.

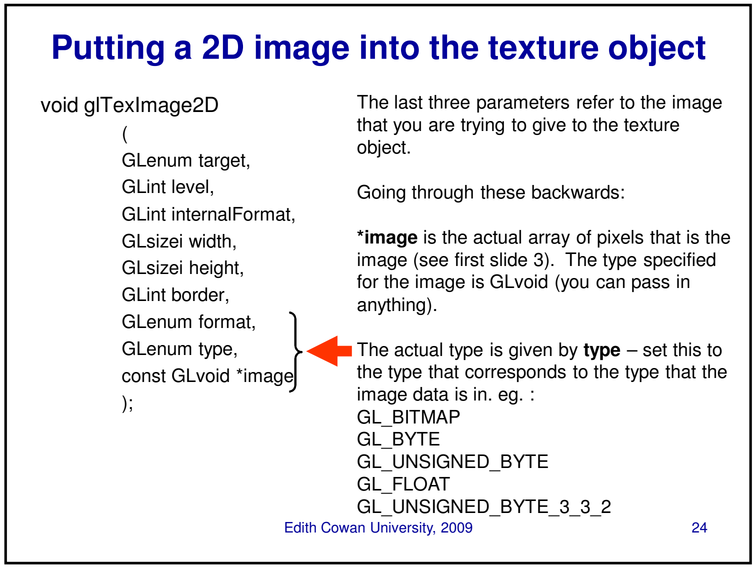


Putting a 2D image into the texture object

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

Set **border** to 1 to draw a border around the texture, 0 and no border will be drawn.

Border colour can be set using the GL_BORDER_TEXTURE_COLOR parameter (using the glTexParameteri function).



Putting a 2D image into the texture object

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

The last three parameters refer to the image that you are trying to give to the texture object.

Going through these backwards:

***image** is the actual array of pixels that is the image (see first slide 3). The type specified for the image is GLvoid (you can pass in anything).

The actual type is given by **type** – set this to the type that corresponds to the type that the image data is in. eg. :
GL_BITMAP
GL_BYTE
GL_UNSIGNED_BYTE
GL_FLOAT
GL_UNSIGNED_BYTE_3_3_2

Putting a 2D image into the texture object

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

The **format** parameter specifies what the data for each pixel means – similar to internalFormat:

GL_RGB
GL_RGBA
GL_COLOR_INDEX

etc…

Immediate Mode Texture Mapping

- The **glTexCoord{1234}{dfis}[v]** group of functions are used to set the texture co-ordinate value that will be applied to the next glVertex Call.
- The **glTexCoord** function can be called between glBegin - glEnd blocks, but note again that glBindTexture can not (program will compile and run, but you wont get the texture you ask for).
- glBindTexture (GL_TEXTURE_2D, diceFace3);

```
glBegin (GL_QUADS);
  glTexCoord2f (0.0, 0.0);
  glVertex3f (0.0, 0.0, 0.0);
  glTexCoord2f (1.0, 0.0);
  glVertex3f (10.0, 0.0, 0.0);
  glTexCoord2f (1.0, 1.0);
  glVertex3f (10.0, 10.0, 0.0);
  glTexCoord2f (0.0, 1.0);
  glVertex3f (0.0, 10.0, 0.0);
glEnd ();
```

VBO Mode Texture Mapping

- See last week's lecture!
- Add each vertices texture coordinates to the buffer when creating it
- When rendering:
  - Enable texture mapping and bind the desired texture

```
glEnable(GL_TEXTURE_2D);
glBindTexture (GL_TEXTURE_2D, m_material);
```

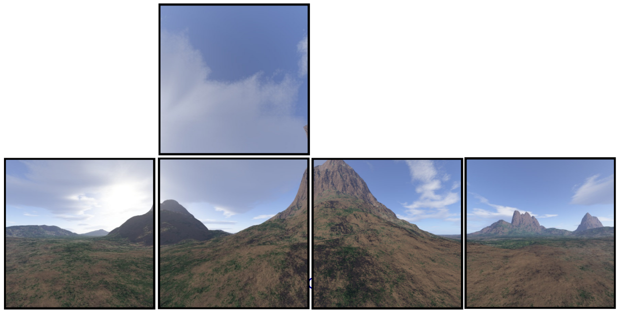
  - Enable texture coordinates and point to the texture coordinates in the Vertex Buffer Object

```
glEnableClientState(GL_TEXTURE_COORD_ARRAY);
glTexCoordPointer(2,GL_FLOAT,0, (float *)NULL + (m_vertexCoordinates.size()* 3));
```

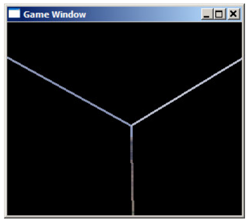
  - Then draw as normal using glDrawElements

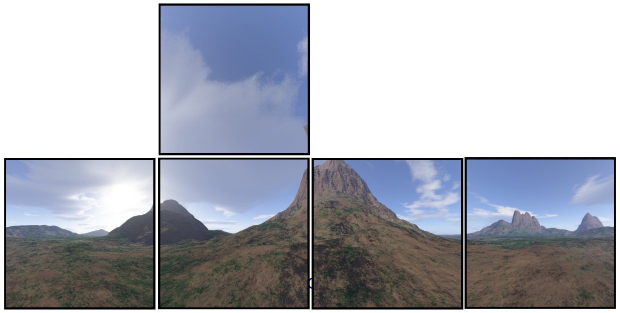
Example

- Five separate images created using Terragen (http://www.planetside.co.uk/terragen/)
- This software lets you generate a 3D landscape and take pictures of it from any angle with different camera settings.

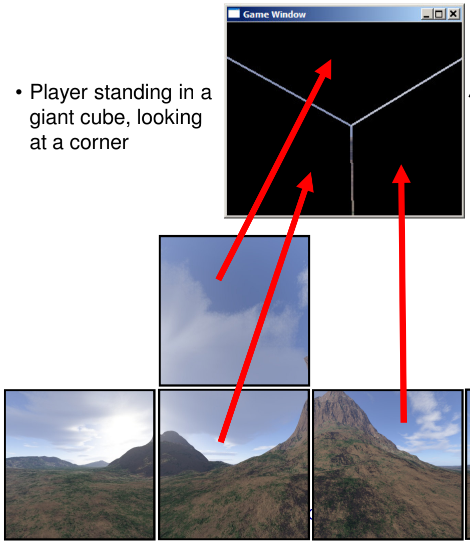


- Player standing in a giant cube, looking at a corner

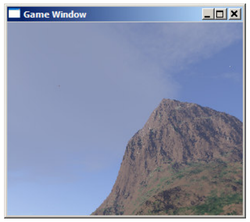




- Player standing in a giant cube, looking at a corner



- What the player sees after texture mapping the images to the cube



Texture Filters

DESCRIPTION

- Texture mapping is a technique that applies an image onto an object's surface. **Texels are mapped onto pixels**. This often involves resizing the image to fit the new surface.
- This involves either
  - Making the image bigger (magnification)
  - Making the image smaller (minification)
- Often the algorithm used to resize images is:
  - a **vital** part of the final image quality.
  - a **major** bottleneck in the rendering pipeline.
- Part of the image resizing process involves filtering the source texture so as to gain a better final scaled image.
- The choice of texture filtering method directly trades quality of the final image for execution speed.
- **glTexParameter** Can be used to set up the use of such filters on a per-texture basis.

Texture Filters

Minification (GL_TEXTURE_MIN_FILTER)

- The texture minifying function is used whenever more than one texel maps to a pixel (eg. trying to map an 8x8 texel texture to a polygon taking up 4x4 pixels on the screen). There are six defined minifying functions to determine what to set the pixel to.
- GL_NEAREST
- GL_LINEAR
- GL_NEAREST_MIPMAP_NEAREST
- GL_LINEAR_MIPMAP_NEAREST
- GL_NEAREST_MIPMAP_LINEAR
- GL_LINEAR_MIPMAP_LINEAR

↑ Faster

↓ Nicer Image

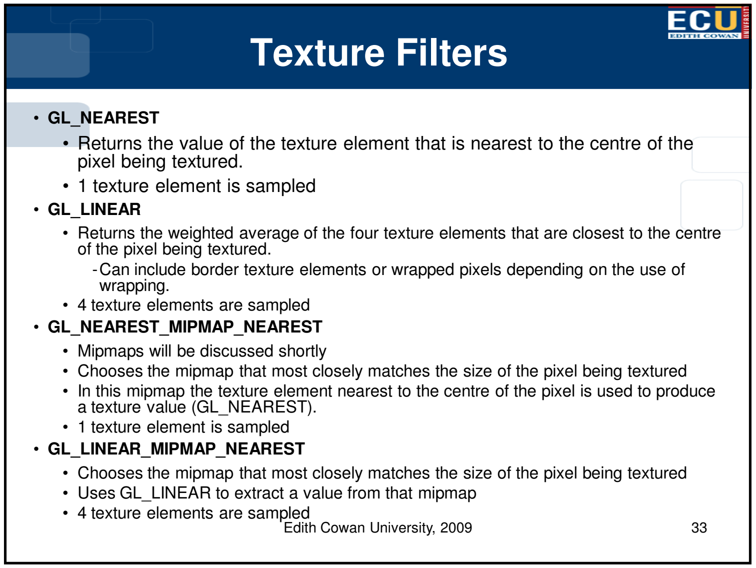
Magnification (GL_TEXTURE_MAG_FILTER)

Magnification of texture images is used when there are more pixels than texels.

- GL_NEAREST
- GL_LINEAR

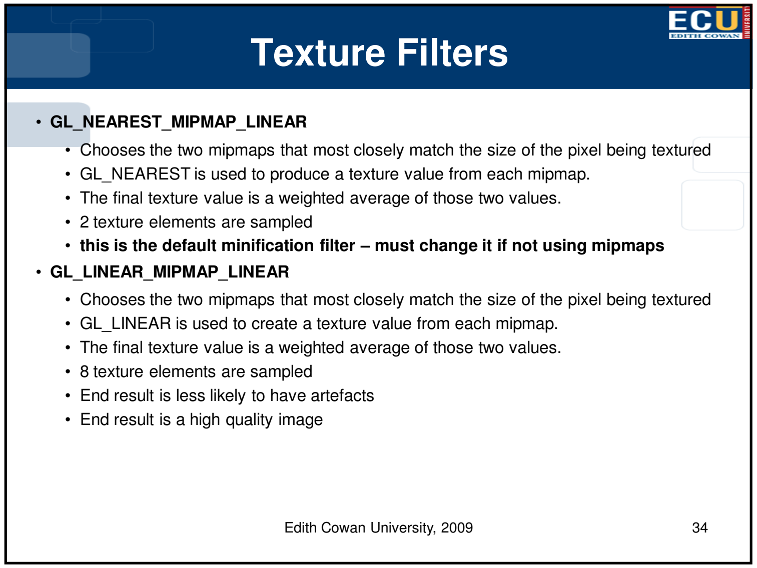
Note:

- Linear refers to pixel values being extrapolated as weighted percentages of source pixels.
- Nearest refers to the selection of a single source pixel being used for output
- Mipmap refers to the use of mipmaps – several lower resolution versions of the texture are specified.



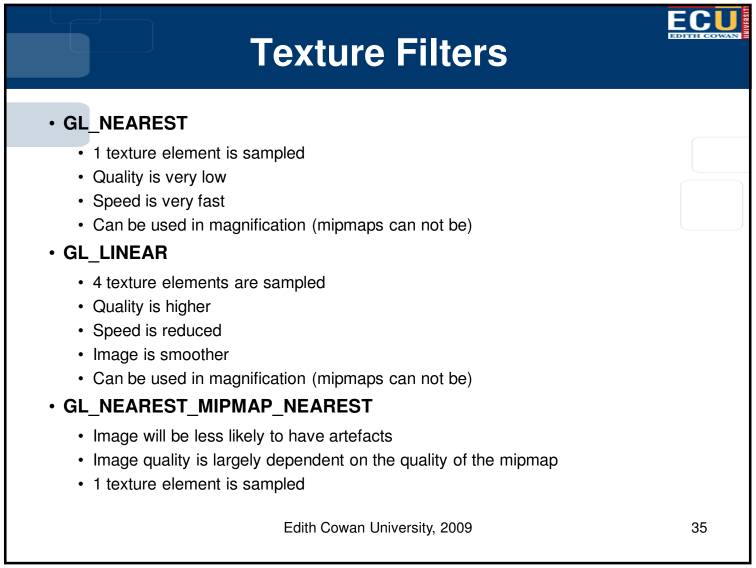
Texture Filters

- **GL_NEAREST**
  - Returns the value of the texture element that is nearest to the centre of the pixel being textured.
  - 1 texture element is sampled
- **GL_LINEAR**
  - Returns the weighted average of the four texture elements that are closest to the centre of the pixel being textured.
    - Can include border texture elements or wrapped pixels depending on the use of wrapping.
  - 4 texture elements are sampled
- **GL_NEAREST_MIPMAP_NEAREST**
  - Mipmaps will be discussed shortly
  - Chooses the mipmap that most closely matches the size of the pixel being textured
  - In this mipmap the texture element nearest to the centre of the pixel is used to produce a texture value (GL_NEAREST).
  - 1 texture element is sampled
- **GL_LINEAR_MIPMAP_NEAREST**
  - Chooses the mipmap that most closely matches the size of the pixel being textured
  - Uses GL_LINEAR to extract a value from that mipmap
  - 4 texture elements are sampled



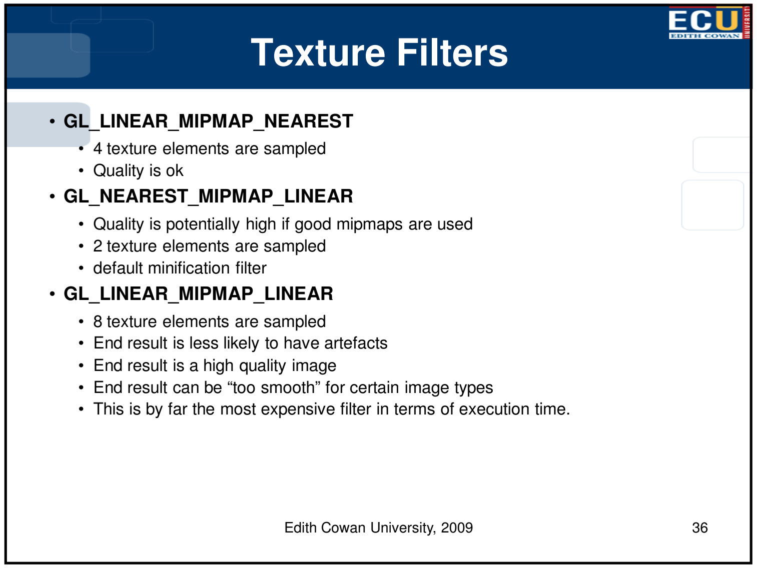
Texture Filters

- **GL_NEAREST_MIPMAP_LINEAR**
  - Chooses the two mipmaps that most closely match the size of the pixel being textured
  - GL_NEAREST is used to produce a texture value from each mipmap.
  - The final texture value is a weighted average of those two values.
  - 2 texture elements are sampled
  - **this is the default minification filter – must change it if not using mipmaps**
- **GL_LINEAR_MIPMAP_LINEAR**
  - Chooses the two mipmaps that most closely match the size of the pixel being textured
  - GL_LINEAR is used to create a texture value from each mipmap.
  - The final texture value is a weighted average of those two values.
  - 8 texture elements are sampled
  - End result is less likely to have artefacts
  - End result is a high quality image



Texture Filters

- **GL_NEAREST**
  - 1 texture element is sampled
  - Quality is very low
  - Speed is very fast
  - Can be used in magnification (mipmaps can not be)
- **GL_LINEAR**
  - 4 texture elements are sampled
  - Quality is higher
  - Speed is reduced
  - Image is smoother
  - Can be used in magnification (mipmaps can not be)
- **GL_NEAREST_MIPMAP_NEAREST**
  - Image will be less likely to have artefacts
  - Image quality is largely dependent on the quality of the mipmap
  - 1 texture element is sampled

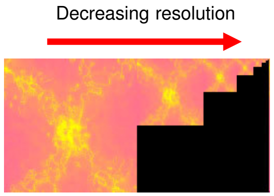


Texture Filters

- **GL_LINEAR_MIPMAP_NEAREST**
  - 4 texture elements are sampled
  - Quality is ok
- **GL_NEAREST_MIPMAP_LINEAR**
  - Quality is potentially high if good mipmaps are used
  - 2 texture elements are sampled
  - default minification filter
- **GL_LINEAR_MIPMAP_LINEAR**
  - 8 texture elements are sampled
  - End result is less likely to have artefacts
  - End result is a high quality image
  - End result can be “too smooth” for certain image types
  - This is by far the most expensive filter in terms of execution time.

Mipmapping

- Mipmapping, lower resolution versions of texture maps.
- Example texture with mipmaps:



Mipmapping

- In most cases the original image (mipmap 0) has dimensions of a power of 2 number
  - 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024
- The following mipmaps are all half the dimensions of the previous mipmap until one of the dimensions is equal to 1.
- Mipmapping does a lot to reduce swimming artefacts – caused by the changing texel interpolation on a moving polygon, but can introduce blurring.

Mipmapping

- How mipmaps are created has a very important effect on the final image.
  - -Loaded in from outside (eg. produced in photoshop)
  - -Created ‘on the fly’ within the program
  - -Automatically generated by OpenGL functions
- It is important when constructing mipmaps that you take use original image scale to make the smaller images.
- If you use the 50% image to create the 25% image and so on then the artefacts introduced by the resize pre-filter will build up on each other creating a very distorted image.

Specifying Mipmaps

- Mipmapping is enabled by specifying one of the _MIPMAP_ minification filters.
- Repeated calls to glTexImage are then used to add different size images to the texture object, using the level parameter to indicate the mipmap level.

```
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 64, 64, 0, GL_RGB, GL_UNSIGNED_BYTE, image0);
glTexImage2D(GL_TEXTURE_2D, 1, GL_RGB, 32, 32, 0, GL_RGB, GL_UNSIGNED_BYTE, image1);
glTexImage2D(GL_TEXTURE_2D, 2, GL_RGB, 16, 16, 0, GL_RGB, GL_UNSIGNED_BYTE, image2);
glTexImage2D(GL_TEXTURE_2D, 3, GL_RGB,  8 ,  8 , 0, GL_RGB, GL_UNSIGNED_BYTE, image3);
glTexImage2D(GL_TEXTURE_2D, 4, GL_RGB,  4 ,  4 , 0, GL_RGB, GL_UNSIGNED_BYTE, image4);
glTexImage2D(GL_TEXTURE_2D, 5, GL_RGB,  2 ,  2 , 0, GL_RGB, GL_UNSIGNED_BYTE, image5);
glTexImage2D(GL_TEXTURE_2D, 6, GL_RGB,  1 ,  1 , 0, GL_RGB, GL_UNSIGNED_BYTE, image6);
```

- By default specify images starting from level 0 (any power of 2 size image) down to the level where the mipmap is of size 1x1. This can be changed by setting the glTexParameter for GL_TEXTURE_BASE_LEVEL or GL_TEXTURE_MAX_LEVEL

Specifying Mipmaps

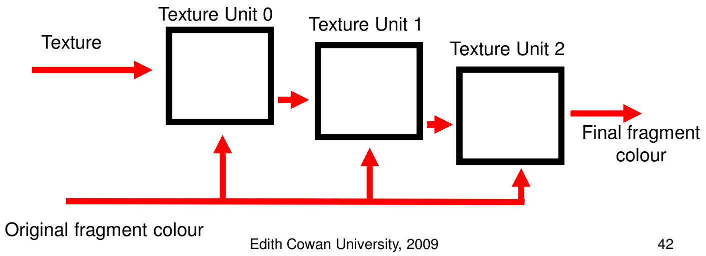
Automatic Mipmaps – OpenGL extension

- A more automated way of producing mipmaps is available as part of the core OpenGL library from version 1.4
- Setting glTexParameter GL_GENERATE_MIPMAP to GL_TRUE enables automatic mipmap generation, then just specify the original size mipmap using one call to glTexImage – OpenGL does the rest.

  -When you change the texture image, OpenGL produces a new set of mipmaps for it.

Texture Environments

- When a texture is applied to a polygon – what happens to the original polygon colour or texture?
  - What happens is not determined by the texture object, but by a texture unit in the texture processing pipeline, each texture unit has its own texture environment.
  - In simple texturing only one texture unit is used, in multi-texturing (mapping several textures to a polygon) additional texture units are used to add in the other textures.



Texture Environments

- A texture environment specifies how texture values are interpreted when a fragment is textured.
- Four basic texture functions are defined (more through extensions)
  - **GL_REPLACE –** texture colours replace previous colours
  - **GL_MODULATE –** texture colours combined with previous colours
  - **GL_DECAL –** texture as a sticker stuck on the polygon, original colour may show through if the sticker has some transparency (alpha)
  - **GL_BLEND** – texture and original fragment colour are combined using colour stored in the texture environment.
- eg. to set replace mode:

```
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
```

- A texture function acts on the fragment to be textured and produces an RGBA colour for that fragment

Texture Filters

Setting the texture functions!

- void **glTexEnvf**(GLenum *target*, GLenum *pname*, GLfloat *param*)
- void **glTexEnvi**(GLenum *target*, GLenum *pname*, GLint *param*)
- void **glTexEnvfv**(GLenum *target*, GLenum *pname*, const GLfloat **params*)
- void **glTexEnviv**(GLenum *target*, GLenum *pname*, const GLint **params*)

parameters

- *target*
  - Specifies a texture environment. Must be GL_TEXTURE_ENV.
- *pname*
  - Specifies the symbolic name of a single-valued texture environment parameter. Accepted values:
    - *GL_TEXTURE_ENV_MODE.*
    - *GL_TEXTURE_ENV_COLOR (vector forms only).*
- *param*
  - Specifies a single symbolic constant, one of:
    - *GL_MODULATE*
    - *GL_DECAL*
    - *GL_BLEND*

Operations on texture objects

- Replace part of a texture objects image with another image
  - glTexSubImage{1,2,3}D(…)
- Copy part of the screen (colour buffer) and use it as the image in the texture object
  - glCopyTexImage{1,2}D(…)
- Replace part of a texture objects image with another image captured from the screen (colour buffer)
  - glCopyTexSubImage{1,2,3}D(…)
- When using sub-images on a mipmapped texture object, enabling automatic mipmap generation saves a lot of trouble.

Matrix Operations on Textures

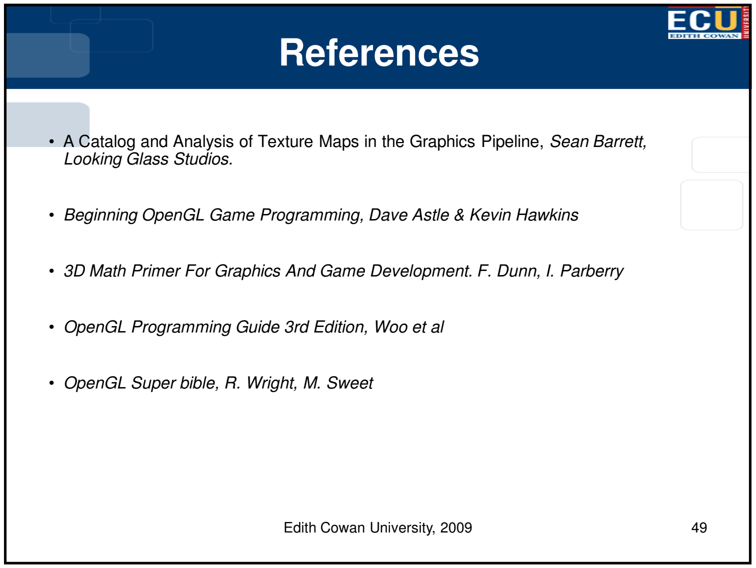
- Matrix operations work on texture coordinates, and there is a texture matrix stack (room for at least 2 matrices) to save transformations on.
- Select the texture matrix stack:
  - glMatrixMode(GL_TEXTURE)
- Transform textures:
  - glTranslate, glRotate, glScale, glMultMatrix, glPushMatrix, glPopMatrix
- Don't forget to return to the model-view matrix when finished
- These operations can be used to animate textures, having the texture rotate or slide across a stationary set of polygons.

Multi-Texturing

- By assigning each texture unit a different texture, and choosing a GL_BLEND filter – multiple textures can be mapped to a single polygon.
- Extra functions to do this: (will need GLEW or similar)
  - glActiveTexture(GL_TEXTURE*) ...
    - * can be 0 to the maximum number of texture units on the graphics card.
    - For each texture unit activated, bind a texture
  - For specifying texture coordinates of vertices, instead of glTexCoord, use: glMultiTexCoord{1234}{sifd}(GL_TEXTURE*, ...

Questions

- **In this lecture we covered :**
  - 2D Image Drawing
  - Texture Mapping
- **Further reading:**
  - The red-book covers pixel drawing (chapter 8) and texture mapping (chapter 9)
- Any Questions?



References

- A Catalog and Analysis of Texture Maps in the Graphics Pipeline, *Sean Barrett, Looking Glass Studios.*
- *Beginning OpenGL Game Programming, Dave Astle & Kevin Hawkins*
- *3D Math Primer For Graphics And Game Development. F. Dunn, I. Parberry*
- *OpenGL Programming Guide 3rd Edition, Woo et al*
- *OpenGL Super bible, R. Wright, M. Sweet*