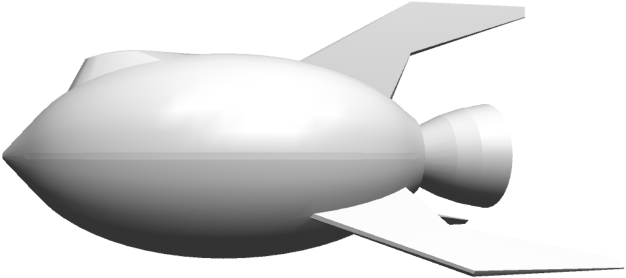
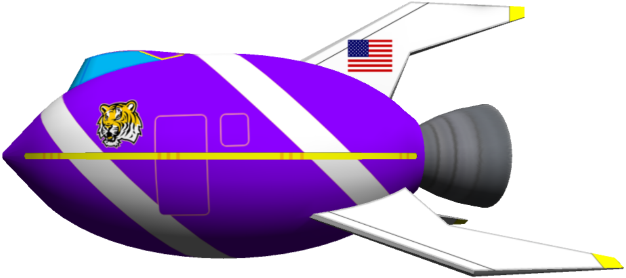
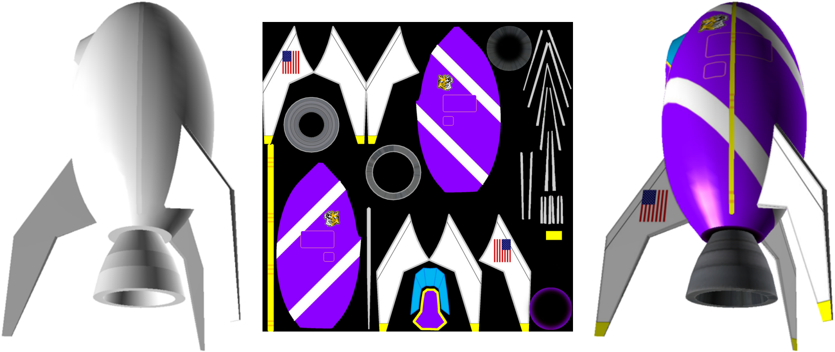
CSC 4356 / ME 4573 Interactive Computer Graphics

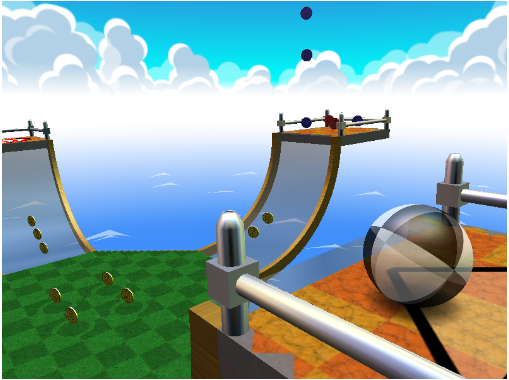
Basic Texturing

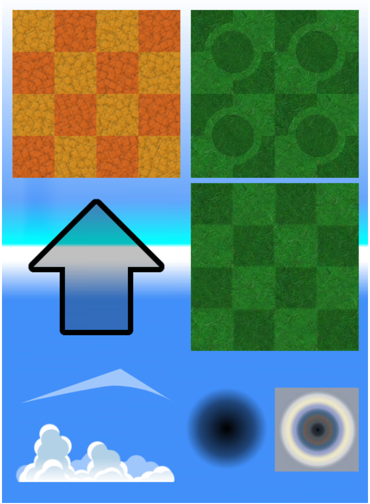




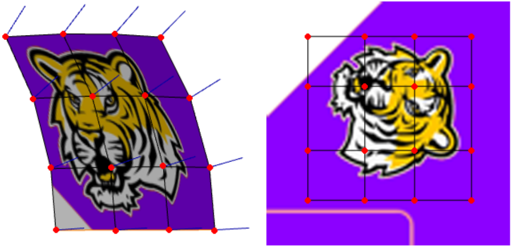


Neverball





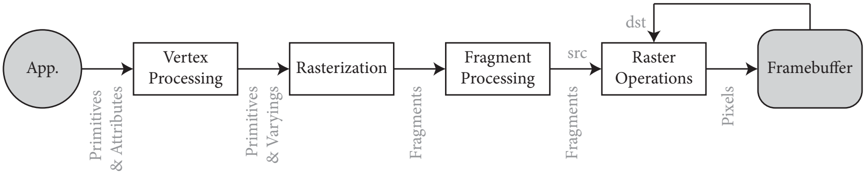
To specify the mapping of an image onto a 3D mesh, we add yet another vertex attribute (which you've seen in cube.c.)



These *texture coordinates* specify a 2D coordinate in $[0.0-1.0, 0.0-1.0]$ giving a position in an image for each vertex.

Texture coordinates

A texture coordinate is yet another *varying* value output from the vertex processing stage.



It is interpolated across a triangle during rasterization, and is an input to the fragment processing stage.

Texture coordinates

Like any other OpenGL vertex attribute, texture coordinates are specified using vertex buffer objects.

```
glEnableClientState(GL_TEXTURE_COORD_ARRAY);
glTexCoordPointer(2, GL_FLOAT, sizeof (vert), offset);
```

Texture coordinates

So that's the texture coordinate vertex attribute.Easy.

The interesting part is the texture *image*.

This is our first opportunity to talk about what goes on in Fragment Processing.

Texture Objects

Like vertex buffer objects, OpenGL texture images are handled via *texture objects*. They are generated and bound in the same fashion.

```
GLuint texture;
glGenTextures(1, &texture);
glBindTexture(GL_TEXTURE_2D, texture);
```

However, we upload data to them differently…

`glTexImage2D(GL_TEXTURE_2D,` *level*, *internal-form*, *width*, *height*, *border*, *external-form*, *type*, *texels* `)`

level	*mipmap* level (usually 0).
internal-form	VRAM data format
width	image width (an integer)
height	image height (an integer)
border	image border (usually 0)
external-form	source data format
type	source data type
texels	source data pointer

Image Parameters

For best compatibility, image width w and height h should be powers of two. That is, for some integers n and m,

$$w = 2^n \quad \text{and} \quad h = 2^m.$$

Most modern hardware allows values as large as 4096, some up to 8192.

Data Parameters

Common data types:

8-bit	GL_UNSIGNED_BYTE
16-bit	GL_UNSIGNED_SHORT

Common data formats:

4-chan	GL_RGBA
3	GL_RGB
2	GL_LUMINANCE_ALPHA
1	GL_LUMINANCE

There are many more exotic data formats and types for use in special circumstances. See the documentation for an exhaustive list.

Image Module

The Image Module is a C source file and header that makes it easy to load, store, and use images in OpenGL applications. It provides:

- Pixel data
- Image width & height
- Channel count & depth
- Data type & format

Image Module

```
int w, h, c, b;

void *p = image_read(name, &w, &h, &c, &b);

int i = image_internal_form(c, b);
int e = image_external_form(c);
int t = image_external_type(b);

image_flip(w, h, c, b, p);

glTexImage2D(GL_TEXTURE_2D, 0, i, w, h, 0, e, t, p);

free(p);
```

Rendering with Textures

As with every OpenGL feature, be sure to *enable* texturing before use, and *disable* it if there is any danger of the state leaking into unsuspecting adjacent code.

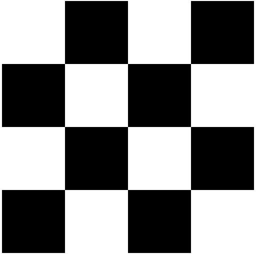
```
glEnable(GL_TEXTURE_2D);
```

To be completely clear on that, texturing must be enabled during *rendering*. You may generate, bind, load, and configure textures without texturing "enabled." Texture object state is orthogonal to its use.

Texture Parameters

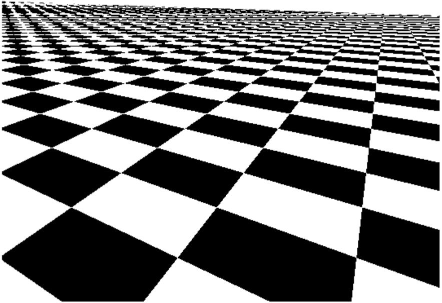
Each texture object has many configurable parameters. We begin with the most useful ones, and show some examples:

- Texture Wrapping
- Texture Filtering



Texture Wrapping

GL_CLAMP_TO_EDGE



GL_REPEAT

Texture Wrapping

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, w);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, w);
```

- `GL_REPEAT` .... repeats straightforwardly.
- `GL_CLAMP_TO_EDGE` ..... clamps to the last texel.
- `GL_CLAMP` .... clamps to the *border* texel.

Remember, these parameters are separately set *per texture object*.

The *border* texel?

The border was dismissed as "usually 0" above. It allows for one *extra* row and column of texels surrounding an image. This enables seamless filtering between disconnected texture images.

$$w = 2^n + 2\,b \qquad h = 2^m + 2\,b$$

You can safely forget about this.

Texture Filtering

Texture *filtering* occurs when the texels of a texture do not map one-to-one onto the pixels of the display (that is, almost always.)

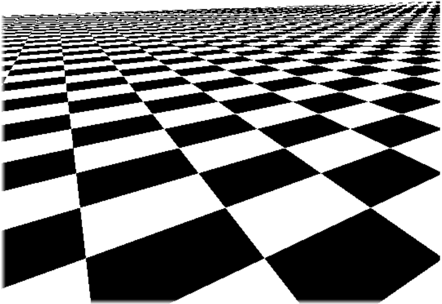
The texture is either *magnified* or *minified*. We can configure each process separately.

Texture Filtering

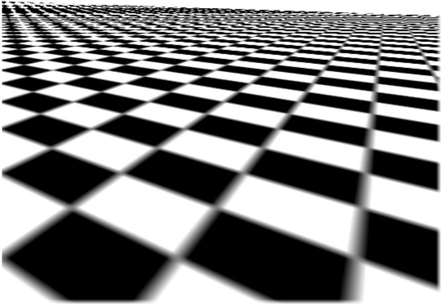
min:
GL_NEAREST
mag:
GL_NEAREST

min:
GL_LINEAR
mag:
GL_LINEAR

Texture Filtering



min: GL_NEAREST
mag: GL_NEAREST



min: GL_LINEAR
mag: GL_LINEAR

Texture Filtering

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, f);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, f);
```

GL_NEAREST ..... selects the closest neighboring texel.
GL_LINEAR ...... interpolates four neighboring texels.

Again, these parameters are separately set *per texture object*.

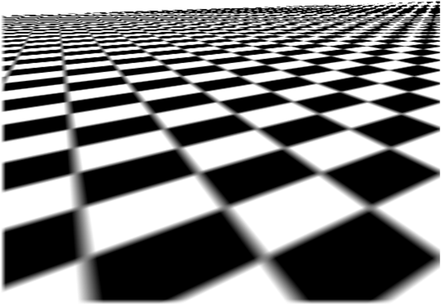
Smarter Texture Filtering

min:
GL_LINEAR
mag:
GL_LINEAR

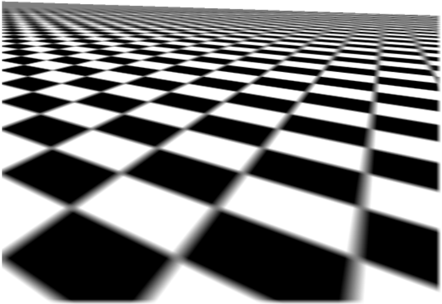
min:
GL_LINEAR
_MIPMAP
_LINEAR
mag:
GL_LINEAR

It’s not very apparent in this slide. See the next one.

Smarter Texture Filtering

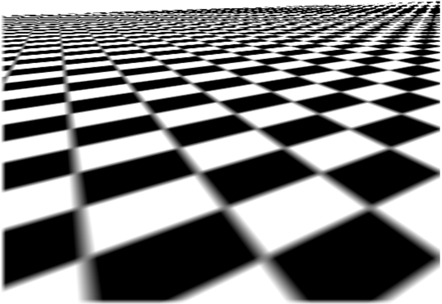


min: GL_LINEAR
mag: GL_LINEAR

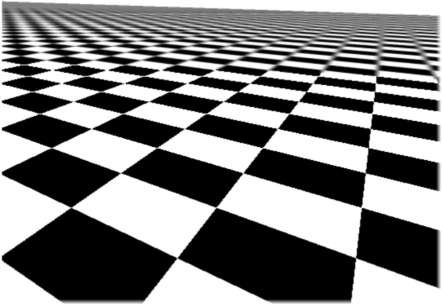


min: GL_LINEAR_MIPMAP_LINEAR
mag: GL_LINEAR

More Better Smarter Texture Filtering



min: GL_LINEAR
mag: GL_LINEAR



min: GL_LINEAR_MIPMAP_LINEAR
mag: GL_NEAREST

Mipmapping

A mipmap is a "pyramid" of texture images, each half the size of the last, with n total levels for a 2^n base image.

There are three ways to create a mipmap. Do…

```
glTexImage2D(GL_TEXTURE_2D}, level, ...);
```

…for each of the *n* image levels. Or…

```
gluBuild2DMipmaps(...);
```

…and link the separate GLU library. Or…

```
glTexParameteri(GL_TEXTURE_2D, GL_GENERATE_MIPMAP, GL_TRUE);
```

…prior to `glTexImage2D`. Stick with this one for now.

Mipmapping

Basic mipmap filtering chooses the texture image level that most closely matches the screen resolution.

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, f);
```

GL_NEAREST_MIPMAP_NEAREST	Nearest-filter the closest image
GL_LINEAR_MIPMAP_NEAREST	Linearly-filter the closest image

Mipmapping

There is seldom a resolution match. Better mipmap filtering samples *two* texture image levels: those just above and just below screen resolution, giving a weighted interpolation of the two.

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, f);
```

`GL_NEAREST_MIPMAP_LINEAR`	Nearest-filter both and interpolate
`GL_LINEAR_MIPMAP_LINEAR`	Linearly-filter both and interpolate

Linear/linear is commonly knows as "tri-linear filtering." Use it.

Coming up. . .

The programmable pipeline.