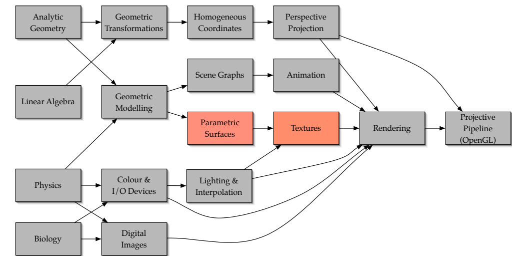
Textured Objects

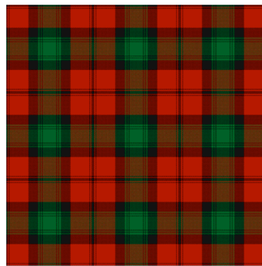
COMP 3003 Autumn 2005

Where We Are



Rendering Tartan

- No geometry
- Only colour
- How many triangles?



Tartan & Triangles

- Model the tartan explicitly:
  - specify colours at vertices
  - takes *two* triangles per stitch
- That's a lot of triangles for a flat surface
- We want to *paint* the tartan *on the surface*

Textures

- A *texture* is an *image* painted on a surface
  - reduces *geometric* complexity of object
  - requires more complex *processing*
- Textures are made of *texels*
  - need to specify where each texel goes
  - i.e. a *coordinate system* on the surface

Texture Parametrisation

- Images are *parametrized*
  - each *texel* has an (i,j) location in image
- Surfaces can also be parametrized
  - each point on surface has (s,t) location
- Define a *mapping* from (s,t) to (i,j)

Texture Mapping

- A *mapping* is just a function
  - converts coordinate systems
- We want mappings of the form:

$$\mathcal{M} : \Re^d \to \Re^e$$
$$1 \le d, e \le 3$$

- Assume these mappings are *linear*

Parametric Plane

- Parametric form of arbitrary plane:
  - p is *any* point in Π
  - u, v are any two *independent* vectors in Π

$$\Pi = p + s\vec{u} + t\vec{v}$$

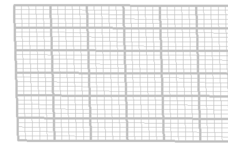
Parametric Surfaces

- A *surface* is a plane that is not flat:
  - Instead of s, t terms, use a function:

$$S = p + \vec{P}(s,t)$$

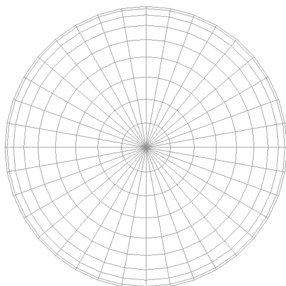
- Every point q on S has corresponding s, t
  - the *parametric coordinates* of q

Parametric Cylinder



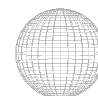
$$Cylinder = \{(\cos t,\ \sin t,\ s) : -1 \le s \le 1,\ 0 \le t \le 2\pi\}$$

Parametric Sphere

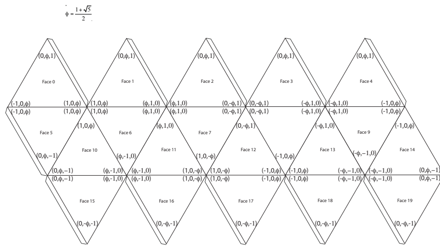


Cylindrical Projection

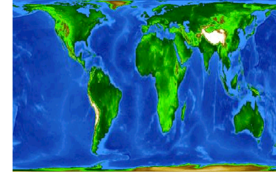
- Simplest map projection: lat. vs. long.



Icosahedron

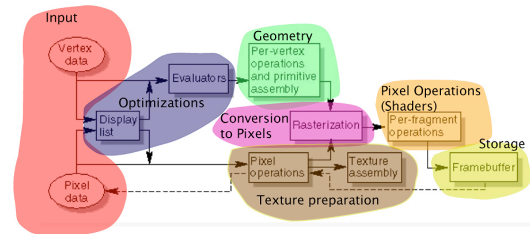


From Map to Globe



Textured Sphere

OpenGL Pipeline



Texture Operations

- *Input* from file to RAM to VRAM
- *Pixel Operations:*
  - *scaling, biasing, & mapping*
  - *clamping*
  - *rasterization*
- *Readback*

Requirements

- We need to:
  - *load* a *texture (image)* into memory
  - *map* the texture onto the surface
    - define *parametric coords* for surface
    - define *parametric coords* for texture
    - give relationship between coords

Image Representation

- A *pixel* is a single element of an image
  - colour is stored for each pixel:
    - 1-bit: black or white
    - 4-/8-bit: ID in *colour look-up table* (CLUT)
    - 16-bit: 5-6 bits each of RGB
    - 24/32-bit: 8 bits (1 byte) each of RGB

File Formats

- Simple file formats (*bitmaps* / *pixmaps*):
  - BMP, PICT, PBM/PGM/PPM, RAW
- Complex file formats (compressed):
  - GIF, JPG, PCX, TGA, TIFF
- Vector file formats (*not* stored as pixels):
  - EPS, PDF, PS

Image Coordinates

- OpenGL indexes pixels with [0..1]
  - assumes image size is 2^m x 2^n
  - actually allows 1, 2, or 3 texture coords
- Surface parameters don't always match
  - so OpenGL has a *texture* matrix
  - for transforming texture coordinates
  - defaults to the identity matrix

Specifying Parameters

- Give surface parameters for each vertex
  - glTexCoord2f();
- Then interpolate between vertices
  - using barycentric coordinates
  - generates parameters for each pixel

Setting the Texture

- OpenGL state applies as usual

```
/*    Specify pixel alignment in memory              */
glPixelStorei(GL_UNPACK_ALIGNMENT, 1);

/*    Read texture from file somehow                 */
textureBytes = ReadTextureFromFile();

/*    Tell OpenGL to use the texture                 */
glTexImage2D( GL_TEXTURE_2D,    /* for now, use this  */
              0,                /* & this             */
              GL_RGBA,          /* format of image    */
              width, height,    /* size of image      */
              0,                /* image border width */
              GL_RGBA,          /* format for v. card */
              GL_UNSIGNED_BYTE, /* type of pixel data */
              textureBytes);    /* pointer to data    */
```

Clamping

- *Texture coordinates* are 0 ... 1
- For other values, we *clamp* or we *repeat:*
  - *clamp* coords to 0 ... 1 (use edge pixels)
  - *repeat* texture (duplicates textures)
- Set separately for horizontal & vertical

Clamping Example

- Horizontal clamp
- Vertical repeat

```
glTexParameteri(
       GL_TEXTURE_2D,
       GL_TEXTURE_WRAP_S,
       GL_CLAMP);
glTexParameteri(
       GL_TEXTURE_2D,
       GL_TEXTURE_WRAP_T,
       GL_REPEAT);
```

Interpolation

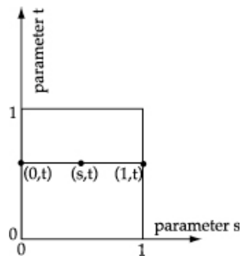
- Texture coordinates are rarely exact
  - land *between* the *texels* (texture pixels)
- So *interpolate* texel values:
  - GL_NEAREST (best if geometric tex.)
  - GL_LINEAR (best if organic tex.)
- GL_MAG_FILTER: for magnification
- GL_MIN_FILTER: for “minification”

Nearest Neighbour

- Simplest form of interpolation:
  - take the nearest available texel
  - e.g. (2.5, 1.38765) maps to (3, 1)
  - preserves sharp edges
  - specified with GL_NEAREST

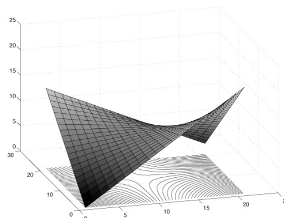
Bilinear Interpolation

- Texels are arranged on a square grid
- Interpolate *f(s,t)* from *f(0,t)* and *f(1,t)*
- Interpolate *f(0,t)* from *f(0,0)* and *f(0,1)*, &c.

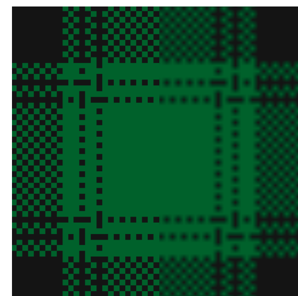


Bilinear Interpolation

- Instead of a flat (linear) surface, we get this:
- But it’s easy to compute
- And it looks good in practice

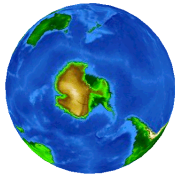


GL_MAG_FILTER

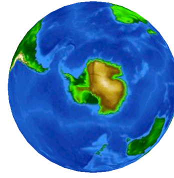


GL_NEAREST GL_LINEAR

GL_MAG_FILTER



GL_NEAREST



GL_LINEAR

Interpolation Issues

- Foreshortening affects interpolation
- More complex (better) forms exist
- Interpolation is how we define surfaces
  - triangles are *linear* interpolation of points on the surface
  - more complex (better) surfaces exist

Foreshortening

- Distant objects are shorter than close ones
  - Interpolation *ought* to account for this
  - But it doesn't (at present)
- Not a big problem for small triangles
  - So break up your large triangles

Perspective-Correct

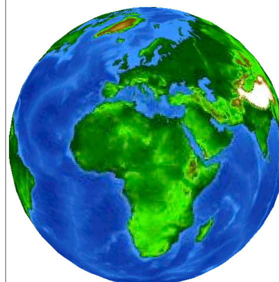
- Given a point P with:
  - barycentric coordinates α, β, γ
  - texture coordinates s_A, s_B, s_C
  - homogeneous coordinates w_A, w_B, w_C
- Interpolate

$$s = \frac{\alpha \frac{s_A}{w_A} + \beta \frac{s_B}{w_B} + \gamma \frac{s_C}{w_C}}{\frac{\alpha}{w_A} + \frac{\beta}{w_B} + \frac{\gamma}{w_C}}$$

Texture Mode

- Colour is used *with* or *without* lighting
- Texture does the same with glTexEnvf()
  - texture *modulates* surface colour
    - use white surface to apply shading
  - texture *replaces* surface colour
    - shading is ignored

Texture Mode



GL_REPLACE

GL_MODULATE

Mapping the Texture

- Use glTexCoord() for texture coords
- As usual, OpenGL state applies
  - specify texture coords for each vertex
- And remember to turn textures on/off:
  - glEnable(), glDisable()

Texturing Code

```
/*    Specify pixel alignment in memory              */
glPixelStorei(GL_UNPACK_ALIGNMENT, 1);

/*    Read texture from file somehow                 */
textureBytes = ReadTextureFromFile();

/*    Tell OpenGL to use the texture                 */
glTexImage2D( GL_TEXTURE_2D, 0, GL_RGBA, width, height,
              0, GL_RGBA, GL_UNSIGNED_BYTE, textureBytes);

/*    Set clamping & interpolation                   */
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

/*    Set texture mode                               */
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
```

Texturing Code II

```
/*    Enable texturing                               */
glEnable(GL_TEXTURE_2D);

/*    Start rendering triangles                      */
glBegin(GL_TRIANGLES);

/*    Render the triangles                           */
glTexCoord2f(0.0, 0.0);  glVertex3f(-5.0, -5.0, 0.0);
glTexCoord2f(0.0, 1.0);  glVertex3f(-5.0,  5.0, 0.0);
glTexCoord2f(1.0, 1.0);  glVertex3f( 5.0,  5.0, 0.0);

glTexCoord2f(0.0, 0.0);  glVertex3f(-5.0, -5.0, 0.0);
glTexCoord2f(1.0, 1.0);  glVertex3f( 5.0,  5.0, 0.0);
glTexCoord2f(1.0, 0.0);  glVertex3f( 5.0, -5.0, 0.0);

/*    Stop rendering triangles                       */
glEnd();

/*    Disable texturing                              */
glDisable(GL_TEXTURE_2D);
```

Texturing Options

- 1D / 3D textures
- Texture matrix stack
- Texture caching (on video card)
- Resident texture sets (optimized caching)
- Fragment shaders
- Mip-mapping (level-of-detail textures)

Texture Applications

- Translucent objects
- Objects with holes
- Bump-mapping
- Shadow-mapping
- Light-mapping
- Billboarding
- Environment mapping (reflections)

Texture Problems

- Texture optimization
- Texture creation
- Texture caching on video card
- Texture distortion
- Surface parametrization
- Texture foreshortening

Texture Distortion