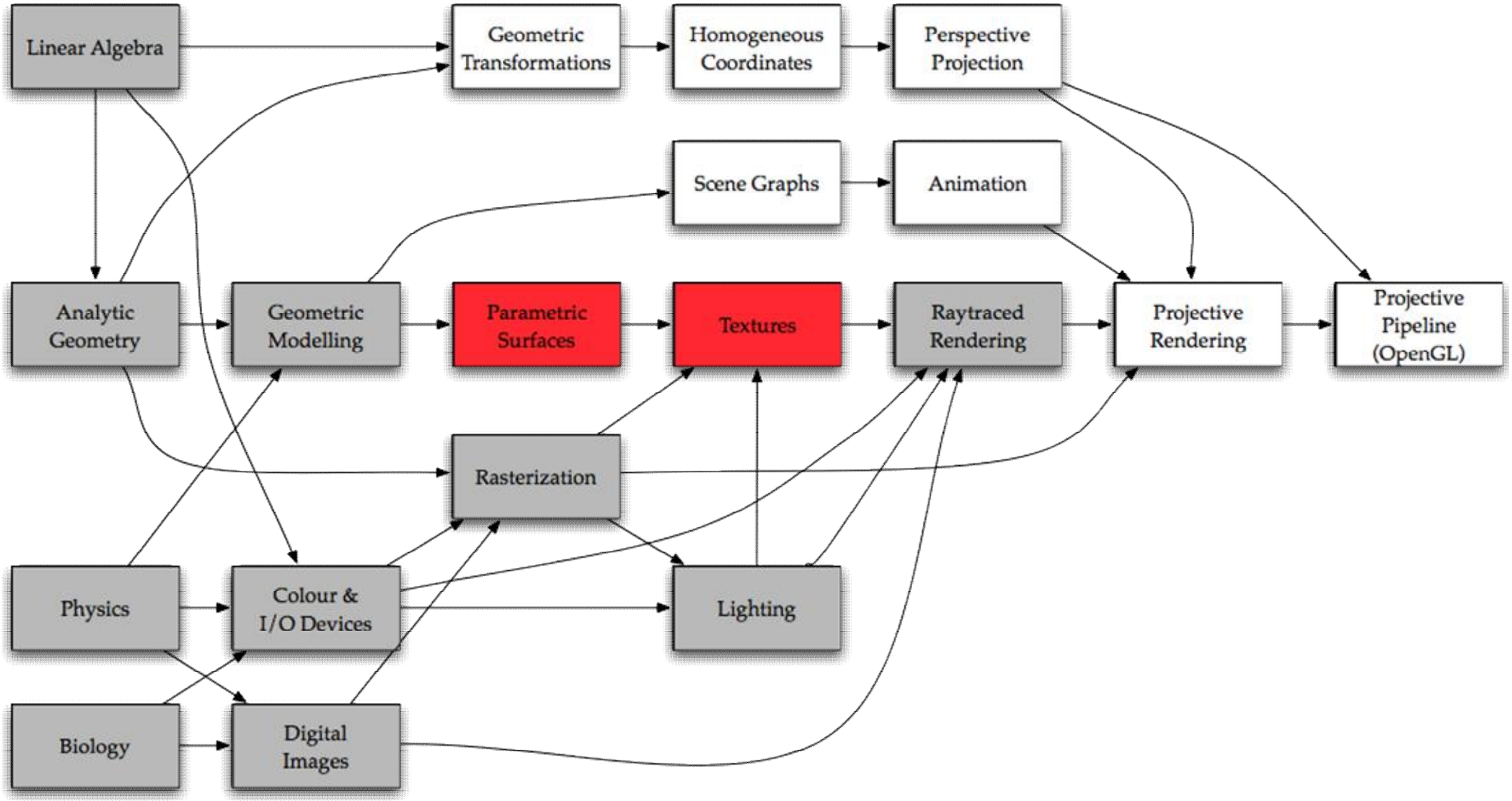
Textures

Where we Are


Linear Algebra
Geometric Transformations
Homogeneous Coordinates
Perspective Projection
Scene Graphs
Animation
Analytic Geometry
Geometric Modelling
Parametric Surfaces
Textures
Raytraced Rendering
Projective Rendering
Projective Pipeline (OpenGL)
Rasterization
Physics
Colour & I/O Devices
Lighting
Biology
Digital Images

Textured Sphere

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 texel for each point
  - a mapping from the surface to the image

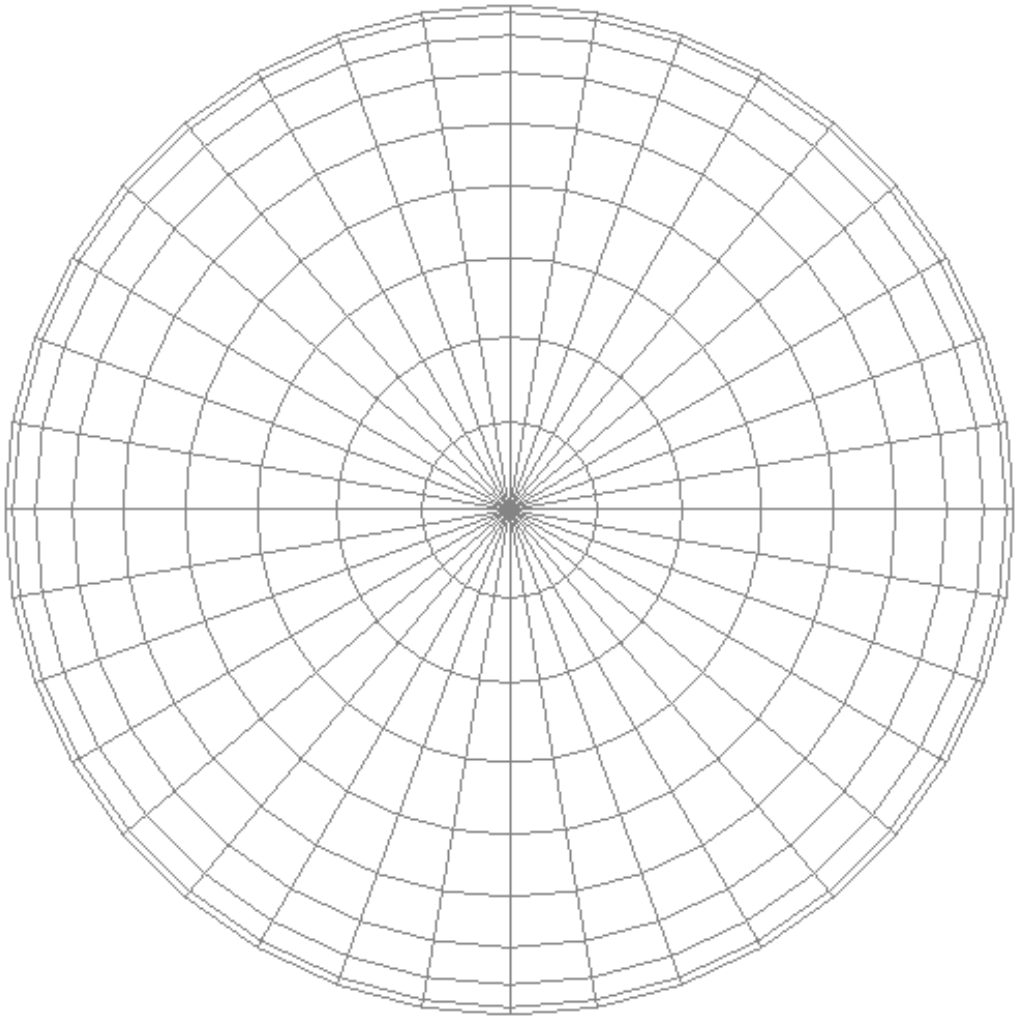
Surface Parametrization

- Images are parametrized
  - each texel has an (i,j) location in texture
- Surfaces can also be parametrized
  - each point on surface has (s,t) location
- So the mapping converts from (s,t) to (i,j)

Texture Steps

- Start with a point p
- Convert to surface parameters (s,t)
- Convert to texel indices (i,j)
- Retrieve texel colour
- Use texel colour for shading

Parametric Sphere



Computing Parameters

- Given point (x,y,z) on sphere
  - Find parameters (Φ,θ) on surface
  - Then find texel coordinates

Computing Parameters

$$x = r\cos\phi\cos\theta$$

$$y = r\cos\phi\sin\theta$$

$$\frac{y}{x} = \frac{r\cos\phi\sin\theta}{r\cos\phi\cos\theta} = \frac{\sin\theta}{\cos\theta} = \tan\theta$$

so:

$$\theta = \arctan\frac{y}{x} \qquad \left(\text{Use C function atan2(y,x)}\right)$$

And:

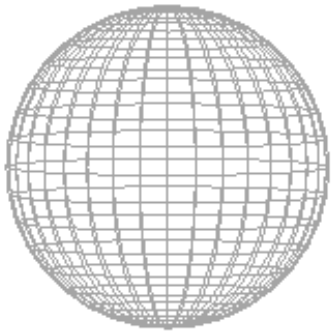
$$z = r\sin\phi$$

so

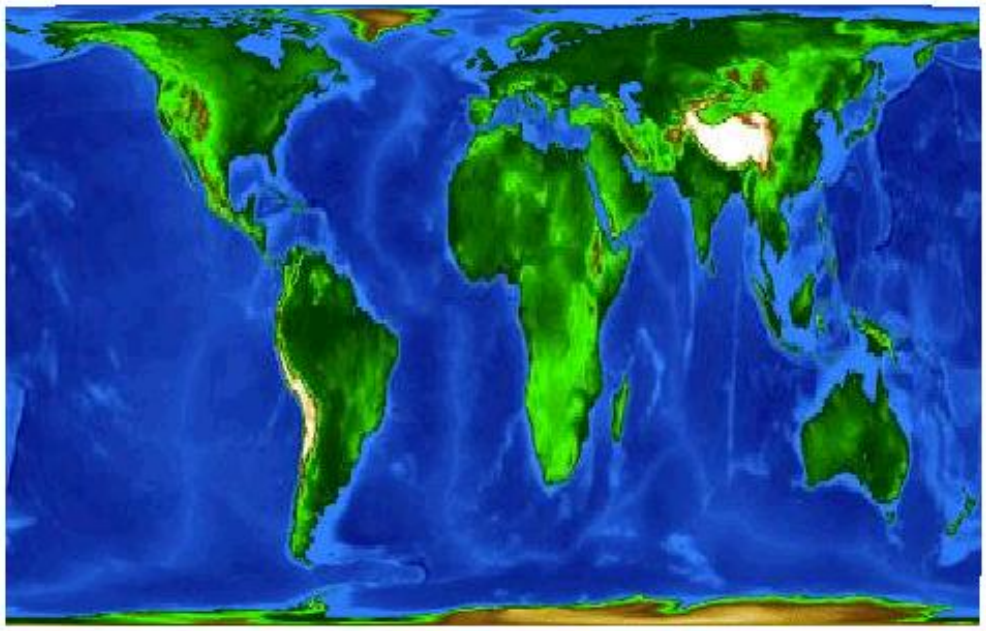
$$\phi = \arcsin\frac{z}{r}$$

Cylindrical Projection

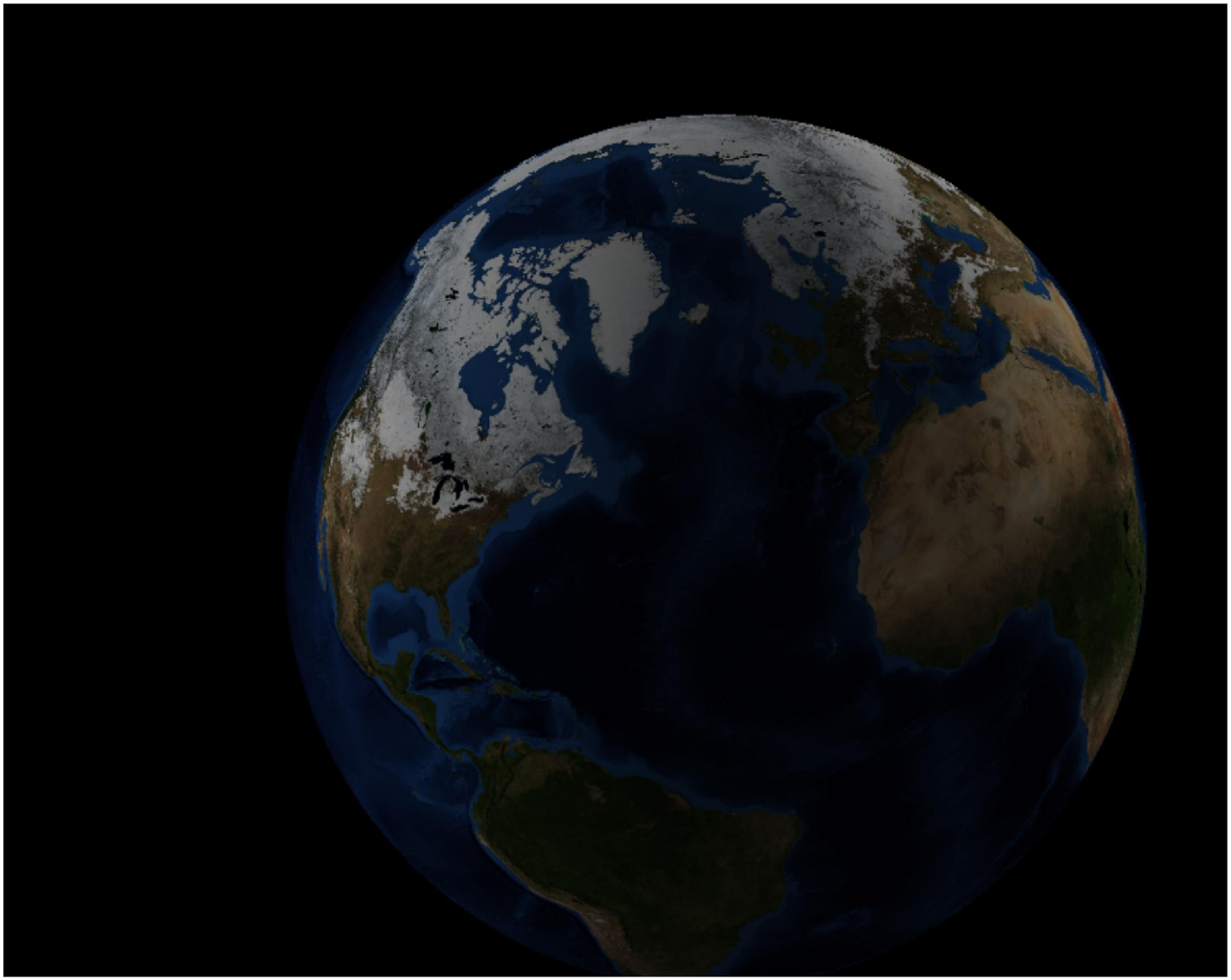
- Simplest map projection: lat. vs. long.



From Map to Globe



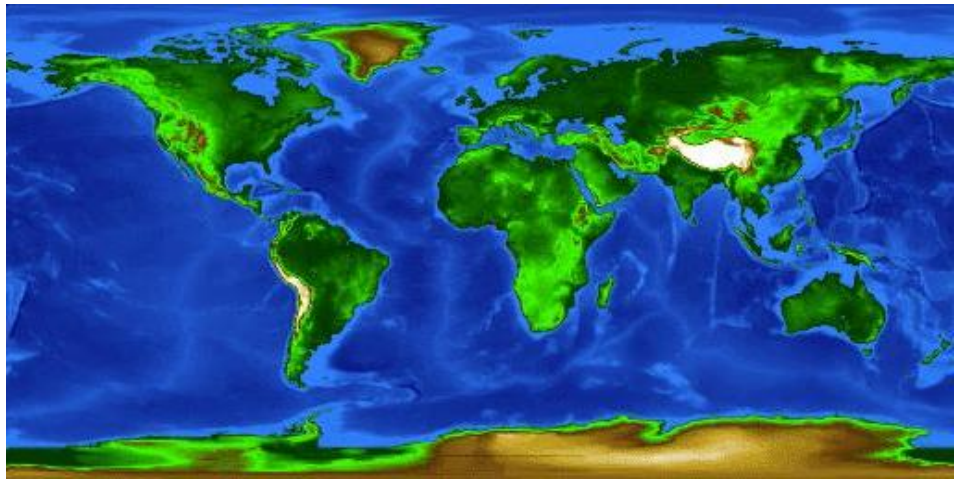
Code in your Assignment 3



Texture Coordinates

Assume that texture has height h, width w

Point	(ϕ,θ)	(s,t)	(i,j)
Top Left	$\left(\frac{\pi}{2},-\pi\right)$	$(1,0)$	$(h,0)$
Top Right	$\left(\frac{\pi}{2},\pi\right)$	$(1,1)$	(h,w)
Bottom Left	$\left(-\frac{\pi}{2},-\pi\right)$	$(0,0)$	$(0,0)$
Bottom Right	$\left(-\frac{\pi}{2},\pi\right)$	$(0,1)$	$(0,w)$



We can compute (i,j) as follows:

$$s=\frac{(\theta+\pi)}{2\pi}=\frac{\theta}{2\pi}+\frac{1}{2} \qquad i=h\cdot s=h\left(\frac{\theta}{2\pi}+\frac{1}{2}\right)$$

$$t=\frac{\left(\phi+\frac{\pi}{2}\right)}{\pi}=\frac{\phi}{\pi}+\frac{1}{2} \qquad j=w\cdot t=w\left(\frac{\phi}{\pi}+\frac{1}{2}\right)$$

Problems

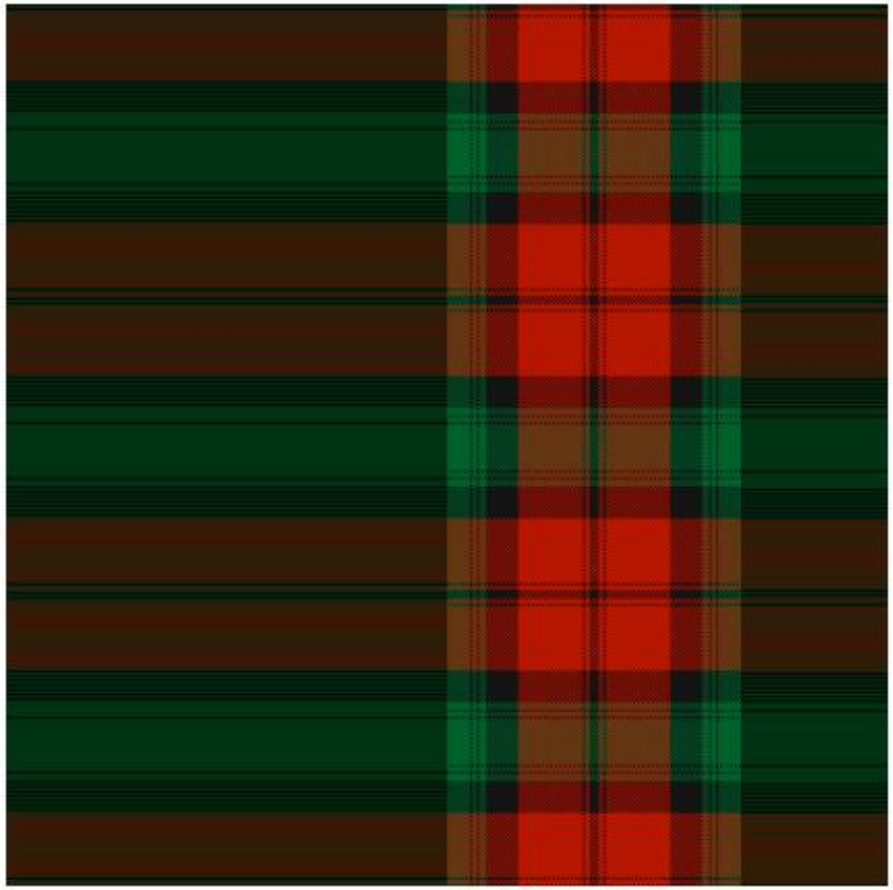
- This works very well if:
  - i & j are both integers
  - s & t are in range [0..1]
- So we need to deal with
  - interpolation - non-integer i, j
  - clamping / repeating - s, t outside range

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



Interpolation

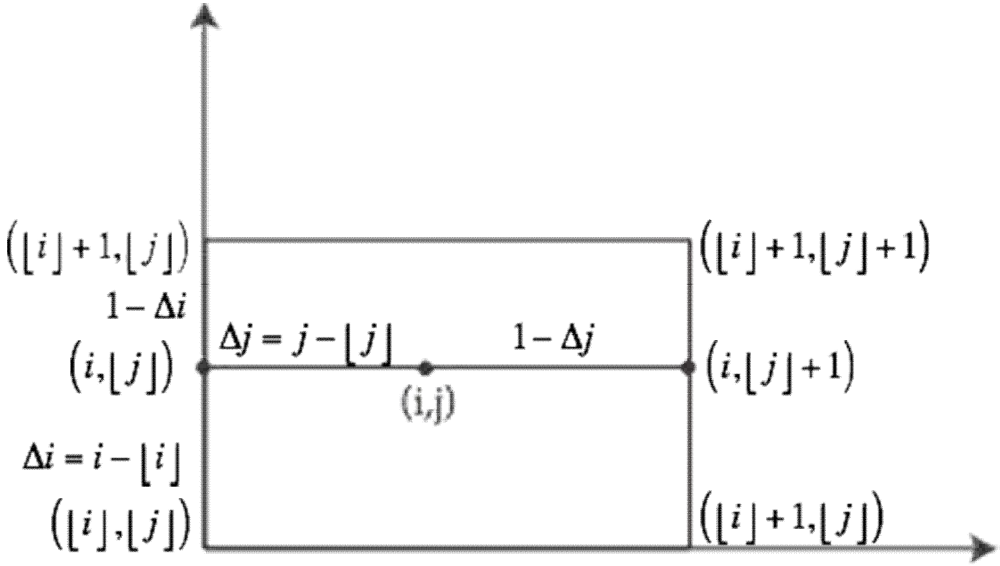
- Texture coordinates are rarely exact
  - land between the texels (texture pixels)
- So interpolate texel values:
  - nearest neighbour
  - bilinear interpolation
  - trilinear interpolation

Nearest Neighbour

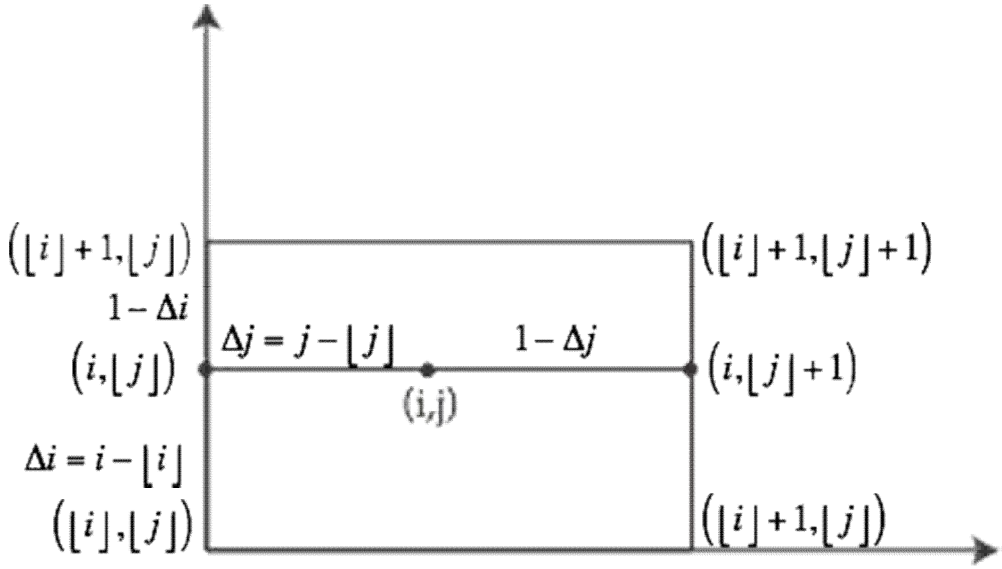
- Simplest form of interpolation:
  - take the nearest available texel
  - e.g. (2.5, 1.38765) maps to (3, 1)
  - preserves sharp edges
  - good for geometric patterns

Bilinear Interpolation

- Texels are arranged on a square grid
- We want to interpolate at (i,j)
- Based on the 4 nearest grid points
- Linear interpolation in *s* then *t*



Development



$$f(i,\lfloor j\rfloor) = (1-\Delta i) f(\lfloor i\rfloor,\lfloor j\rfloor) + \Delta i\ f(\lfloor i\rfloor+1,\lfloor j\rfloor)$$

$$f(i,\lfloor j\rfloor+1) = (1-\Delta i) f(\lfloor i\rfloor,\lfloor j\rfloor+1) + \Delta i\ f(\lfloor i\rfloor+1,\lfloor j\rfloor+1)$$

$$f(i,j) = (1-\Delta j) f(i,\lfloor j\rfloor) + \Delta j\ f(i,\lfloor j\rfloor+1)$$

Pseudo Code

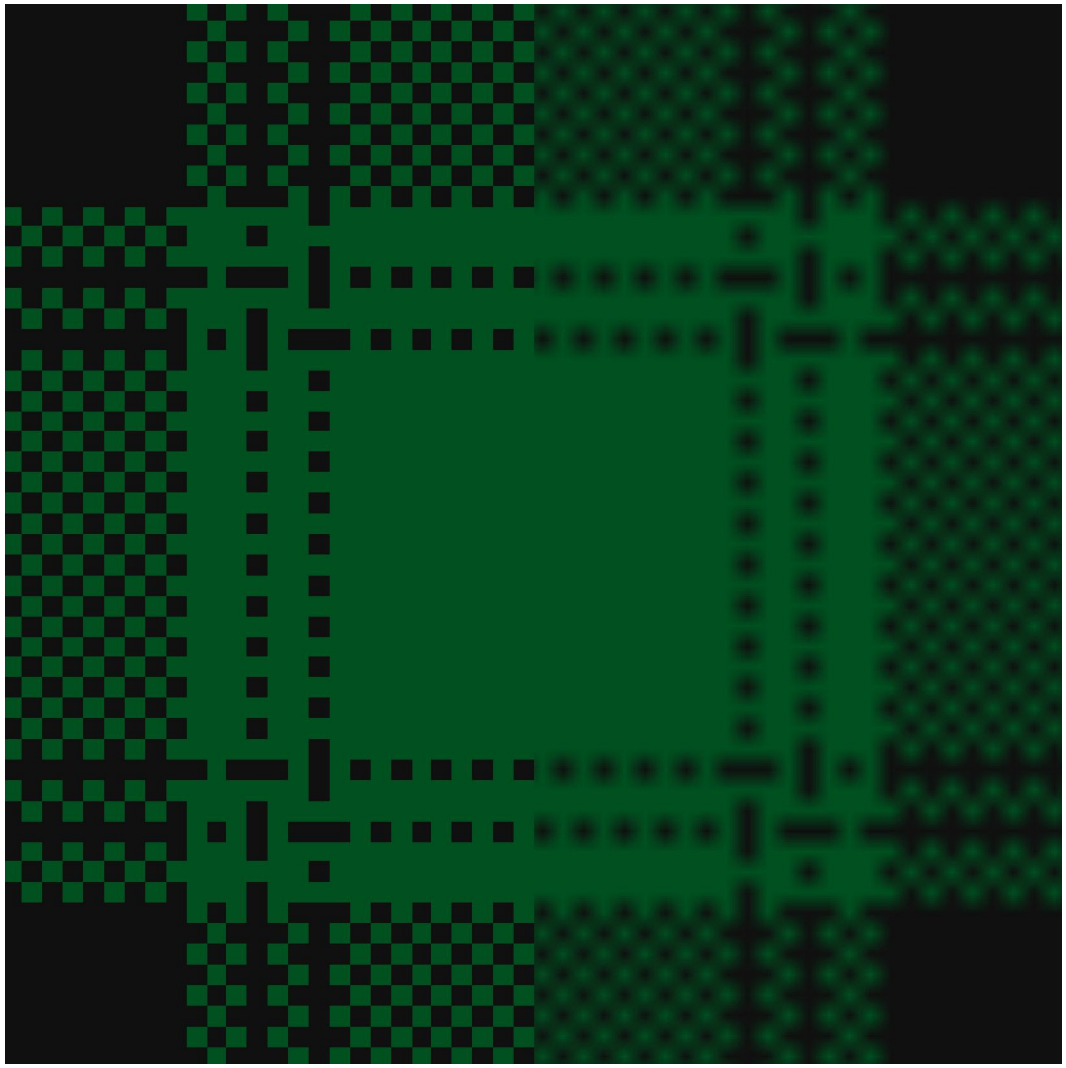
```
RGBValue BilinearLookup(Image tex, float s, float t)
  { // BilinearLookup()
  int i = s;                    // truncates s to get i
  int j = t;                    // truncates t to get j
  float sParm = s - i;          // compute s parameter for interpolation
  float tParm = t - j;          // compute t parameter for interpolation

  // grab four nearest texel colours
  RGBValue colour00 = tex[i][j];
  RGBValue colour01 = tex[i][j+1];
  RGBValue colour10 = tex[i+1][j];
  RGBValue colour11 = tex[i+1][j+1];

  // compute colours on edges
  RGBValue colour0 = colour00 + tParm * (colour01 - colour00);
  RGBValue colour1 = colour10 + tParm * (colour11 - colour10);

  // compute colour for interpolated texel
  return colour1 + sParm * (colour1 - colour0);
  } // BilinearLookup()
```

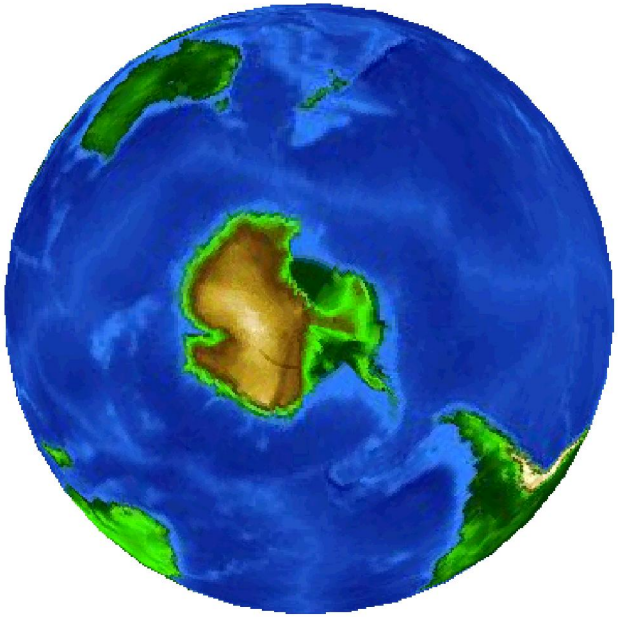
Geometric Texture



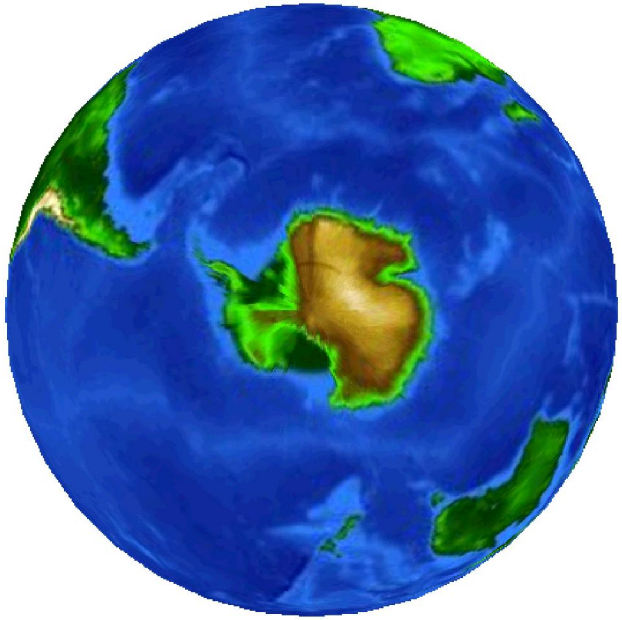
Nearest Neighbour

Bilinear

Non-Geometric Texture



Nearest Neighbour



Bilinear

Texture Modulation

- Colour can be used with or without lighting
  - Texture can replace lighting calculation

$$Colour_{out} = Colour_{texture}$$

  - Or it can modulate lighting calculation

$$Colour_{out} = Colour_{texture} \cdot Colour_{shading}$$

    - Surface colour usually white for this

Texture Mode

Replace

Modulate

Texture Operations

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

Image Coordinates

- OpenGL indexes texels 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

- Using LWJGL & OpenGL state applies as usual

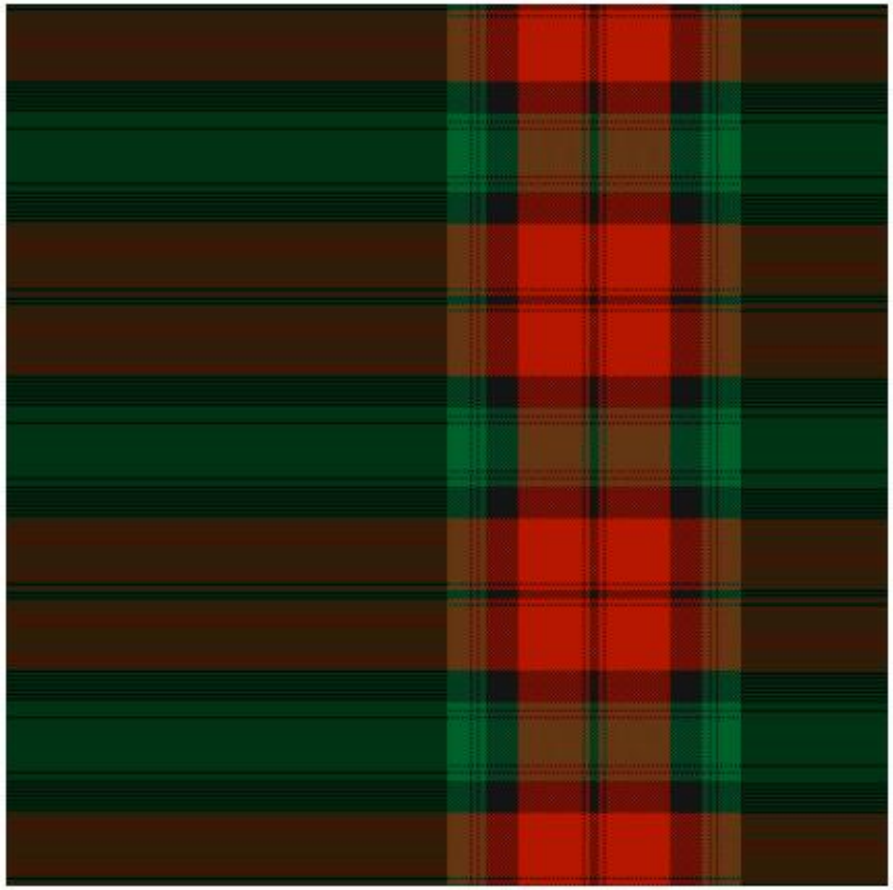
myTexture.bind();

```
GL11.glBegin(GL11.GL_QUADS);
    GL11.glTexCoord2f(0,0); // odd bug , sometimes not needed
    GL11.glVertex2f(100,100);
    GL11.glTexCoord2f(1,0);
    GL11.glVertex2f(100+texture.getTextureWidth(),100);
    GL11.glTexCoord2f(1,1);
    GL11.glVertex2f(100+texture.getTextureWidth(),100+texture.getTextureHeight());
    GL11.glTexCoord2f(0,1);
    GL11.glVertex2f(100,100+texture.getTextureHeight();
GL11.glEnd();
```

Texture Clamping

- Horizontal clamp
- Vertical repeat

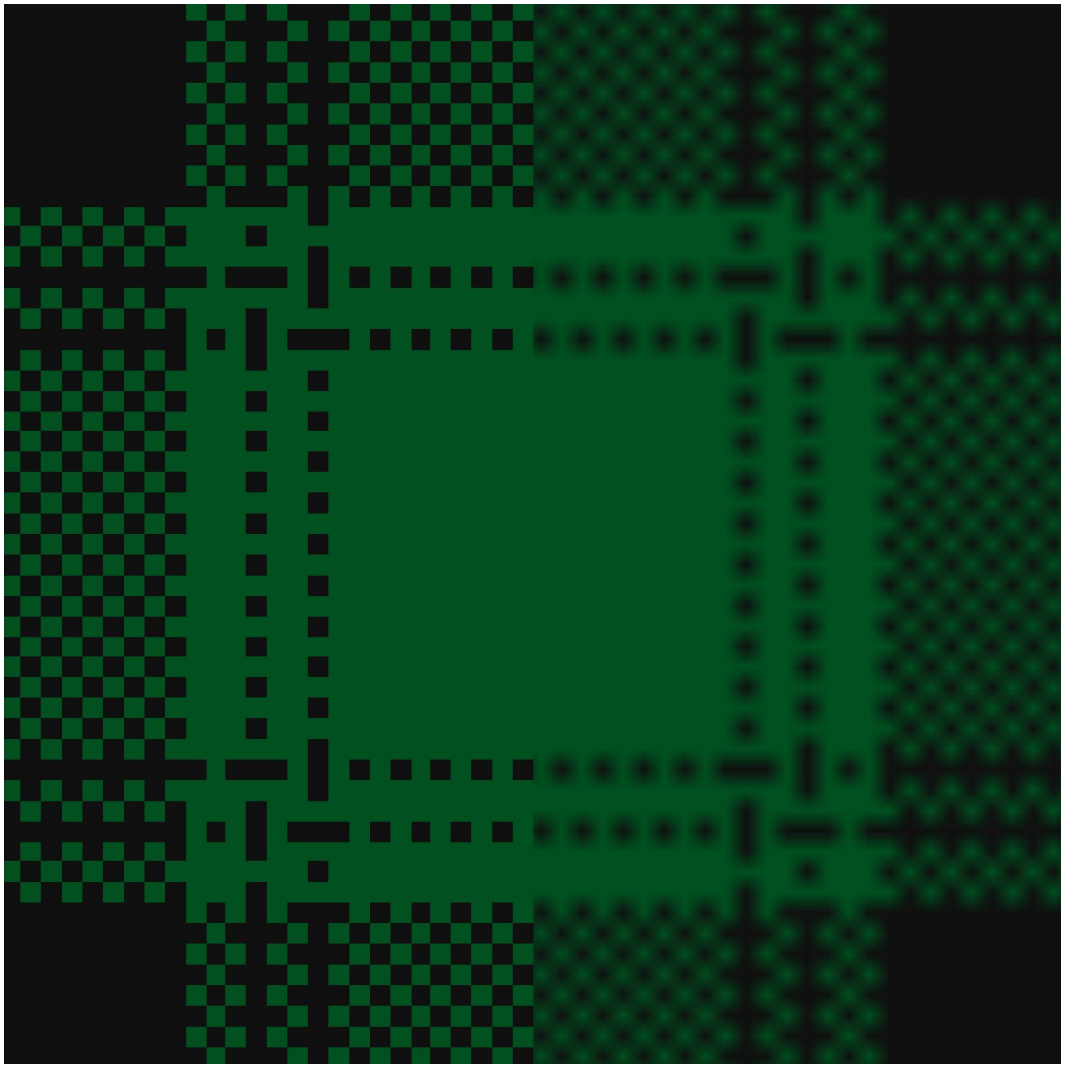
```
GL11.glTexParameteri(
        GL11.GL_TEXTURE_2D,
        GL11.GL_TEXTURE_WRAP_S,
        GL11.GL_CLAMP);
GL11.glTexParameteri(
        GL11.GL_TEXTURE_2D,
        GL11.GL_TEXTURE_WRAP_T,
        GL11.GL_REPEAT);
```



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”

GL_MAG_FILTER



GL_NEAREST GL_LINEAR