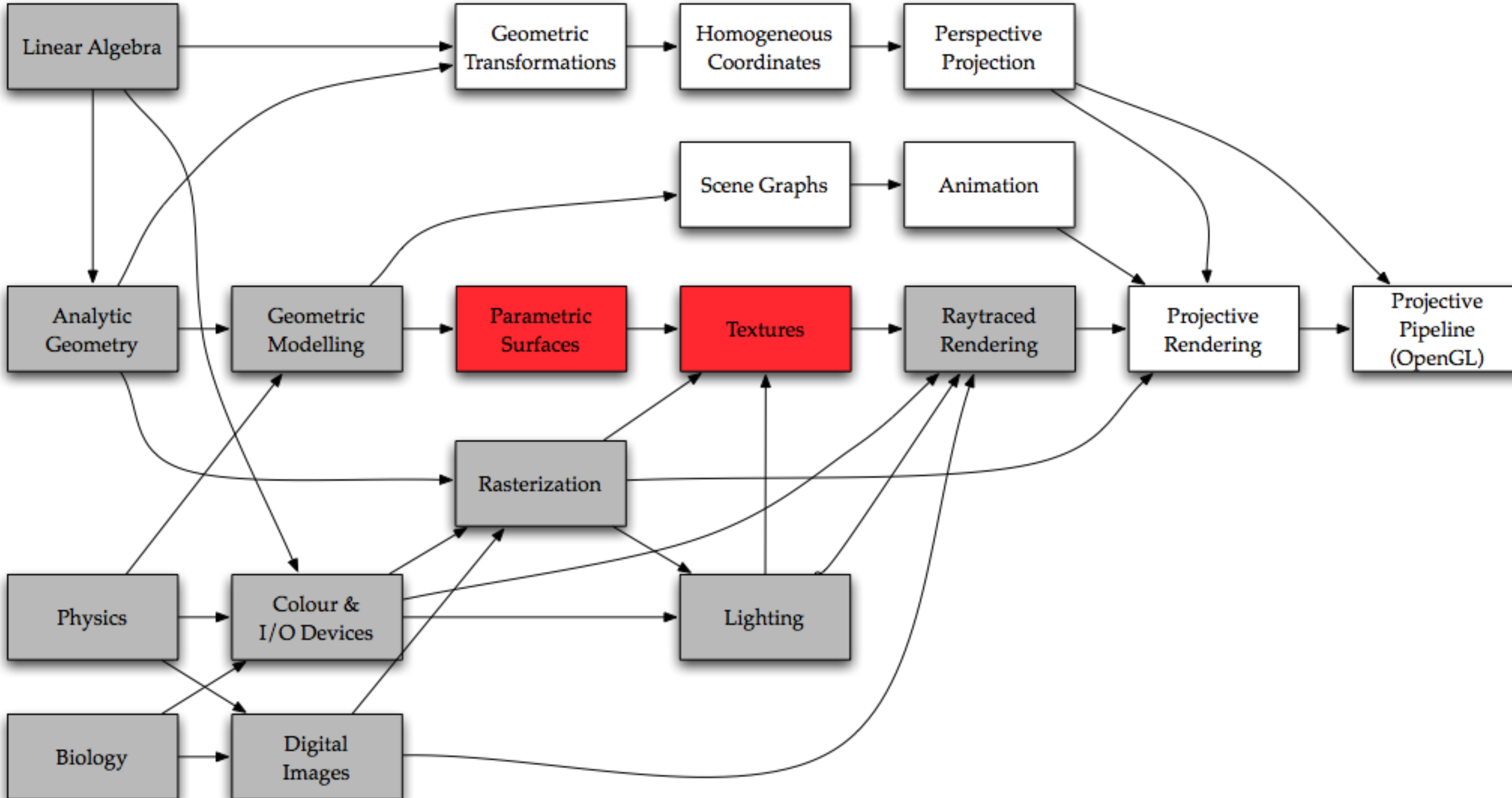
Textures

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

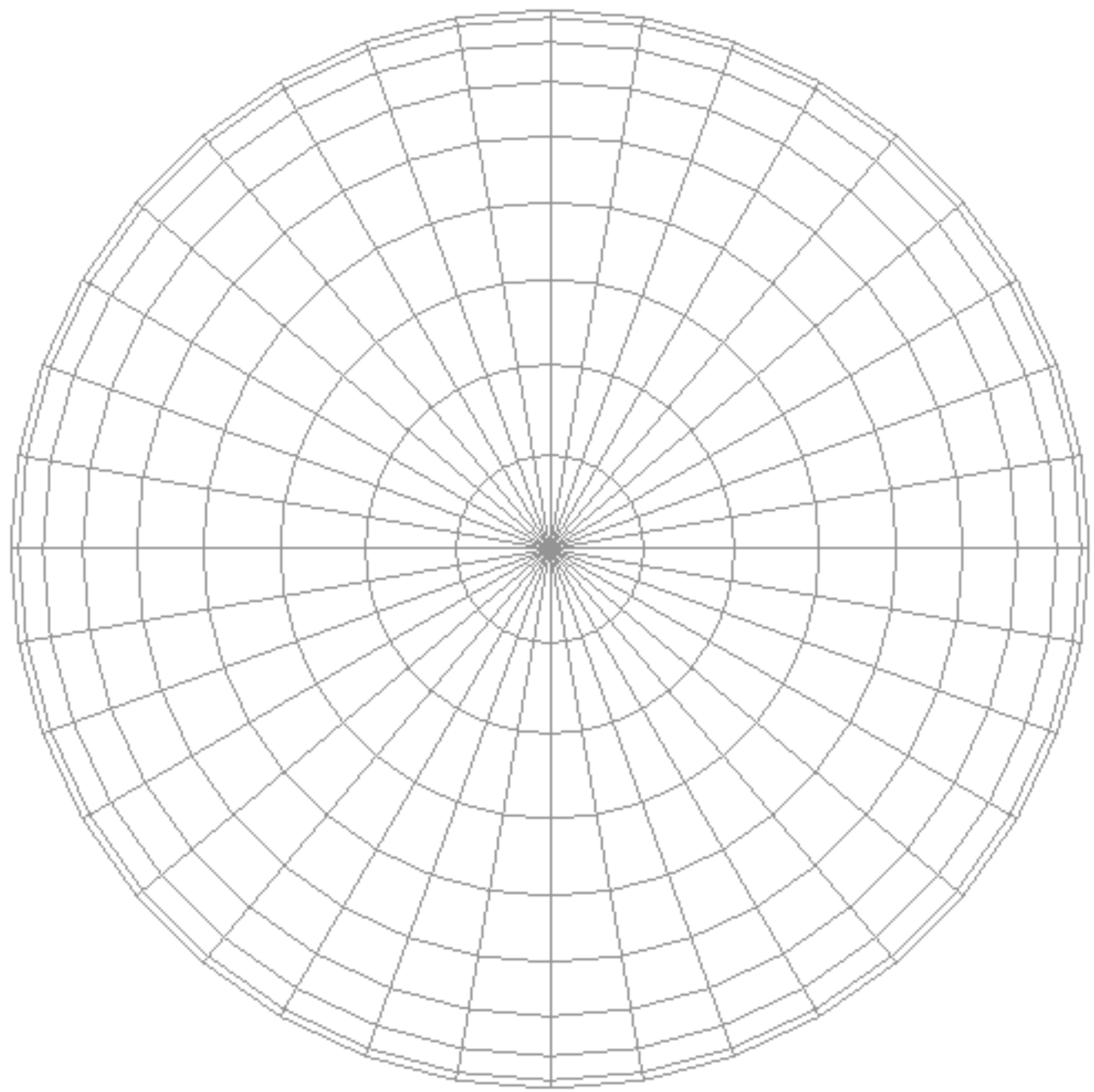
Surface Parametrization

- 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:

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

And:

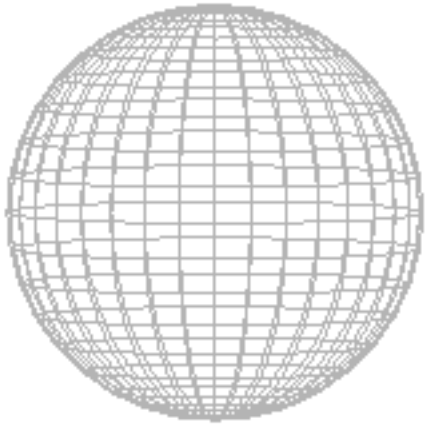
$$z = r\sin\phi$$

so

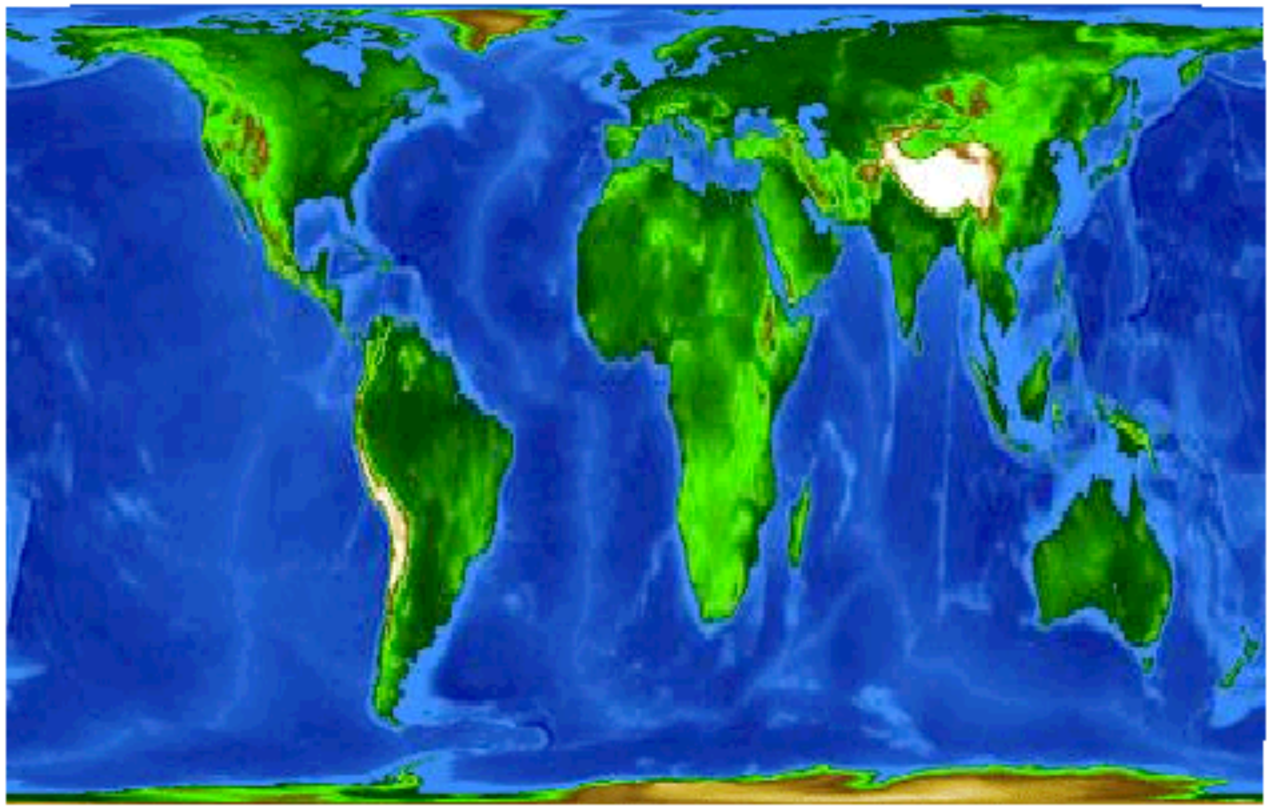
$$\phi = \arcsin\frac{z}{r}$$

Cylindrical Projection

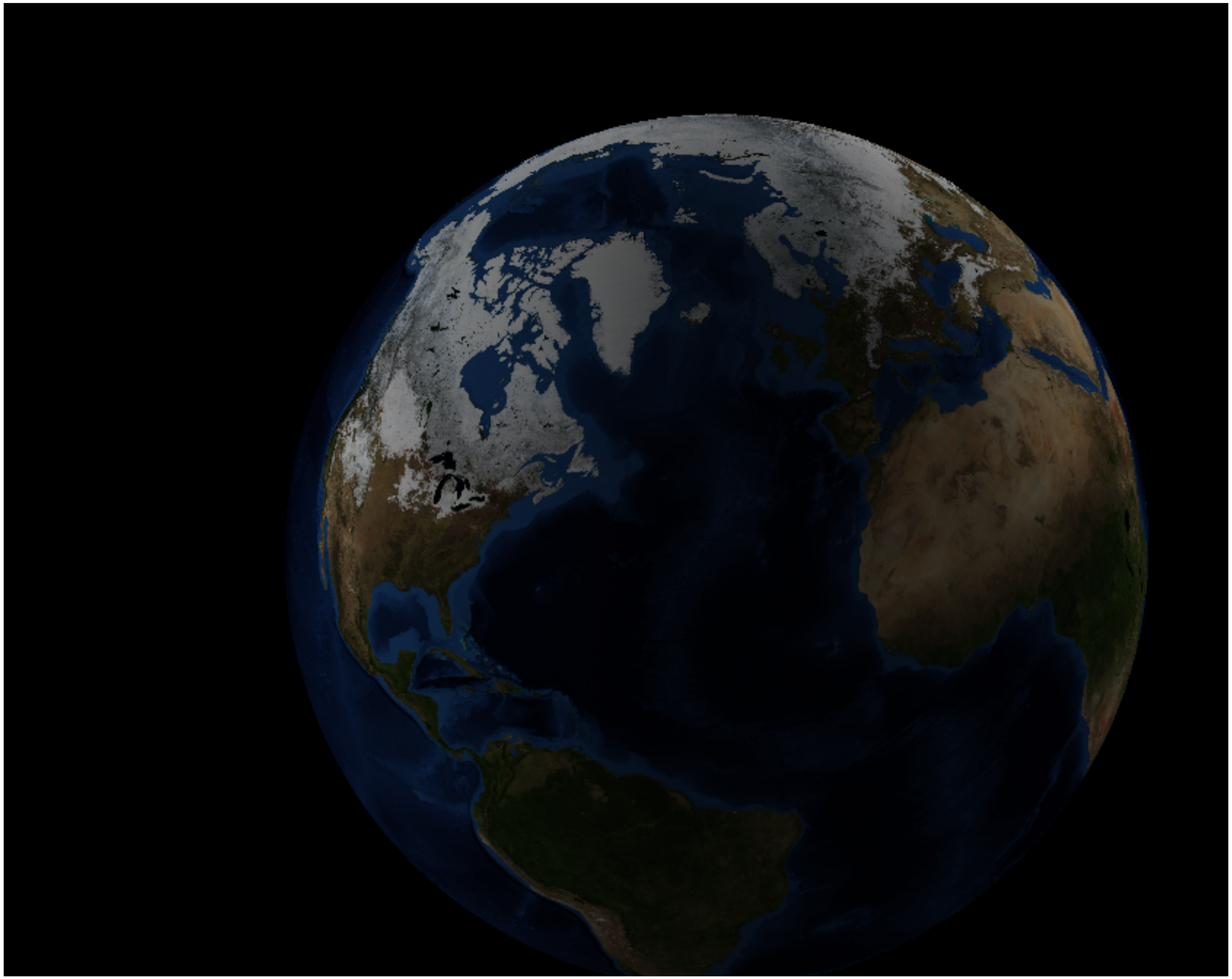
- Simplest map projection: lat. vs. long.



From Map to Globe



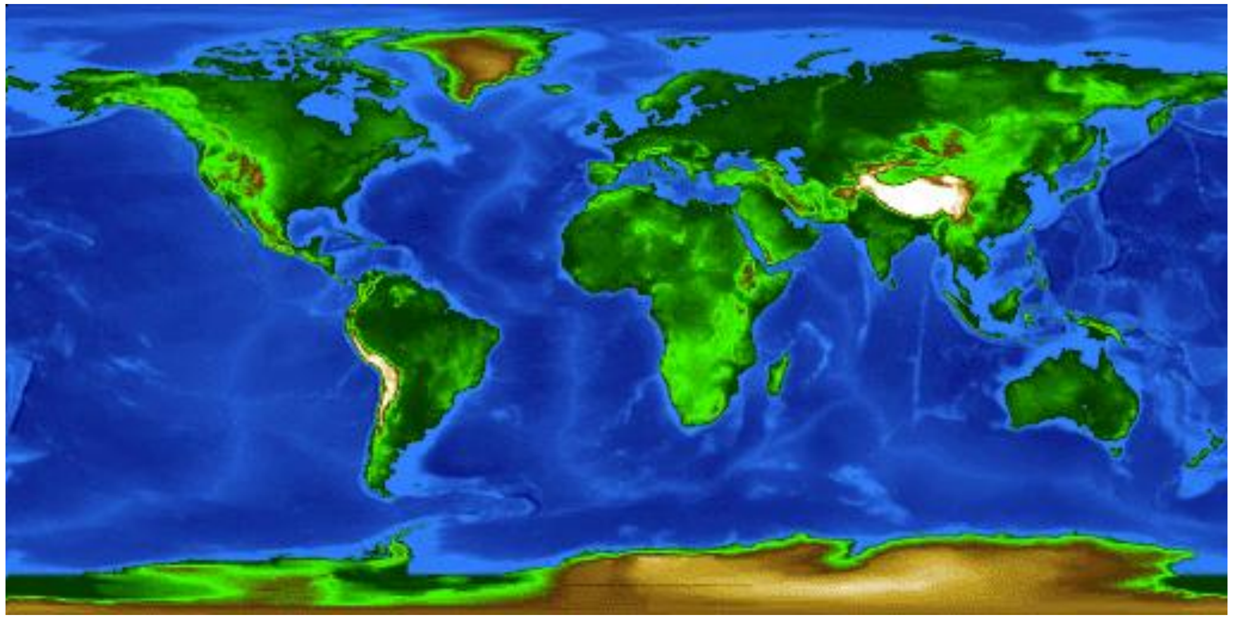
Code in your Assignment 3



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:

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

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

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

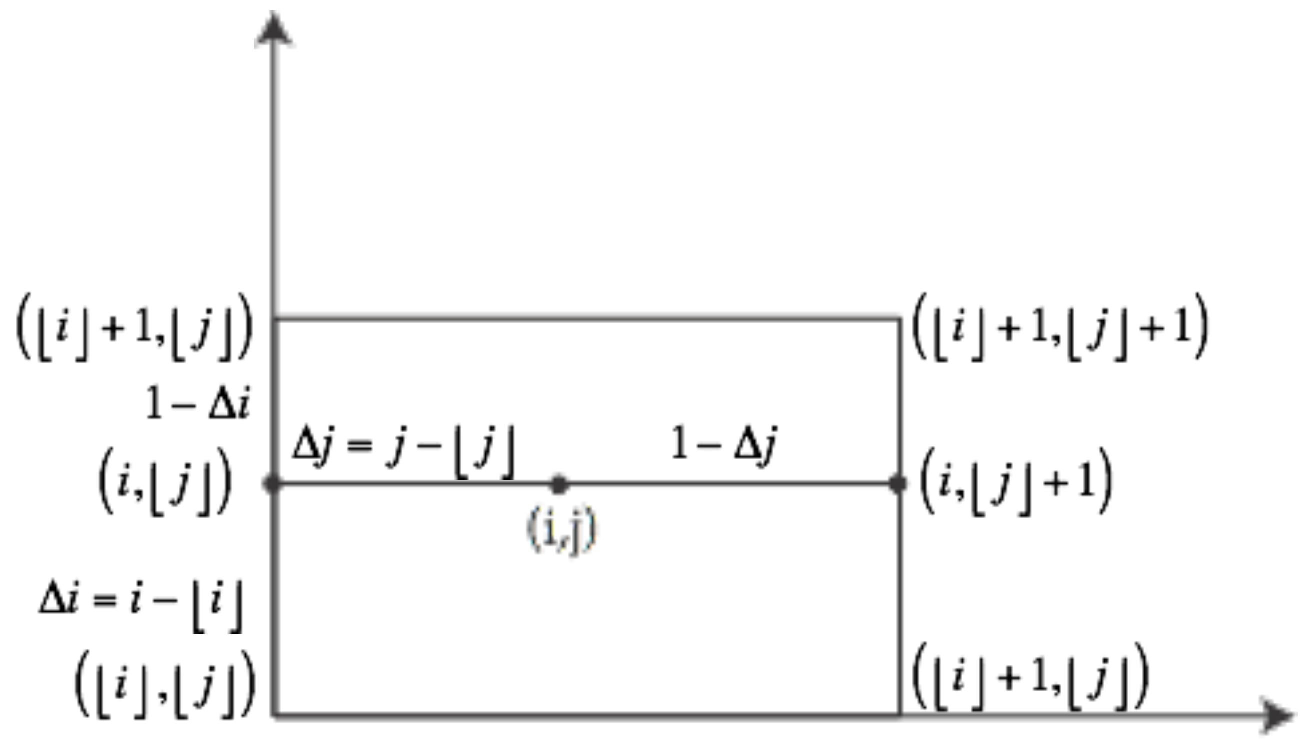
- Texture coordinates are rarely exact
    - land between the texels (texture pixels)
- So interpolate texel values:
    - nearest neighbour
    - bilinear interpolation
    - trilinear interpolation

Nearest Neighbour

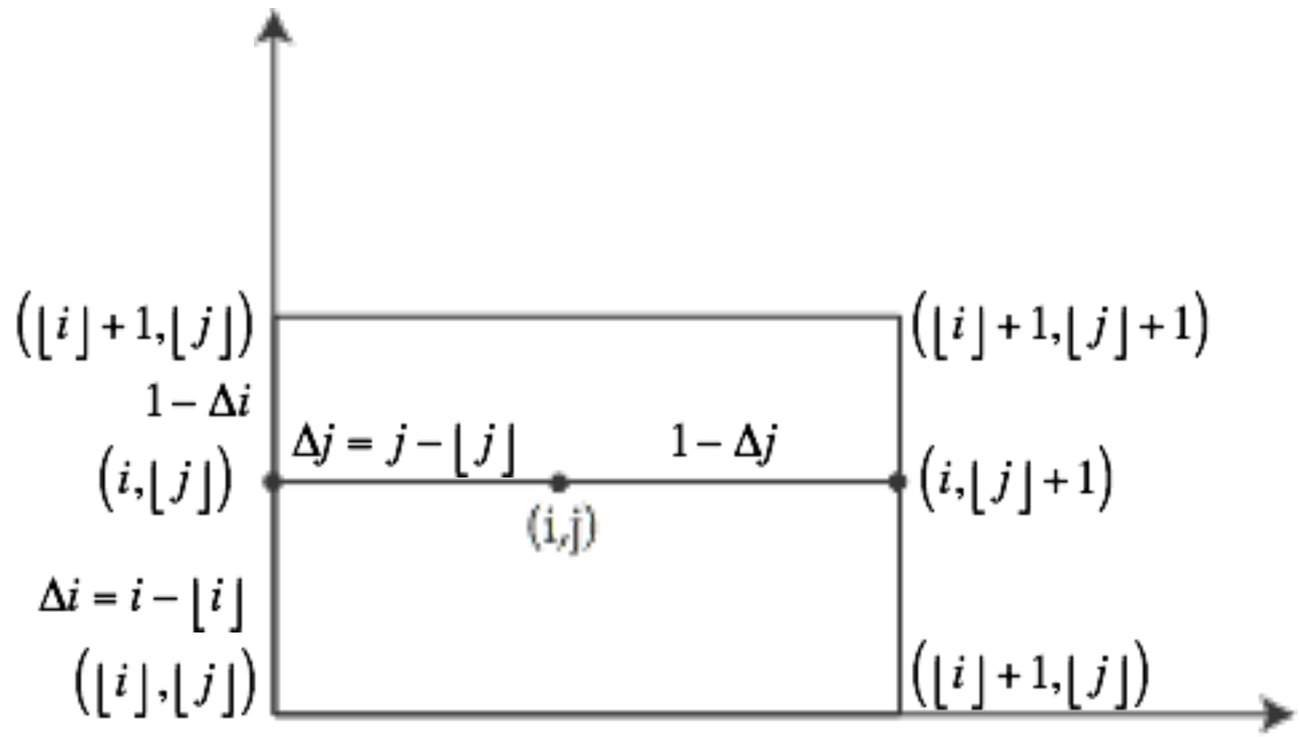
- Simplest form of interpolation:
  - take the nearest available texel
  - e.g. (2.7, 1.38765) maps to (3, 1)
  - preserves sharp edges
  - good for geometric patterns

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



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

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

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

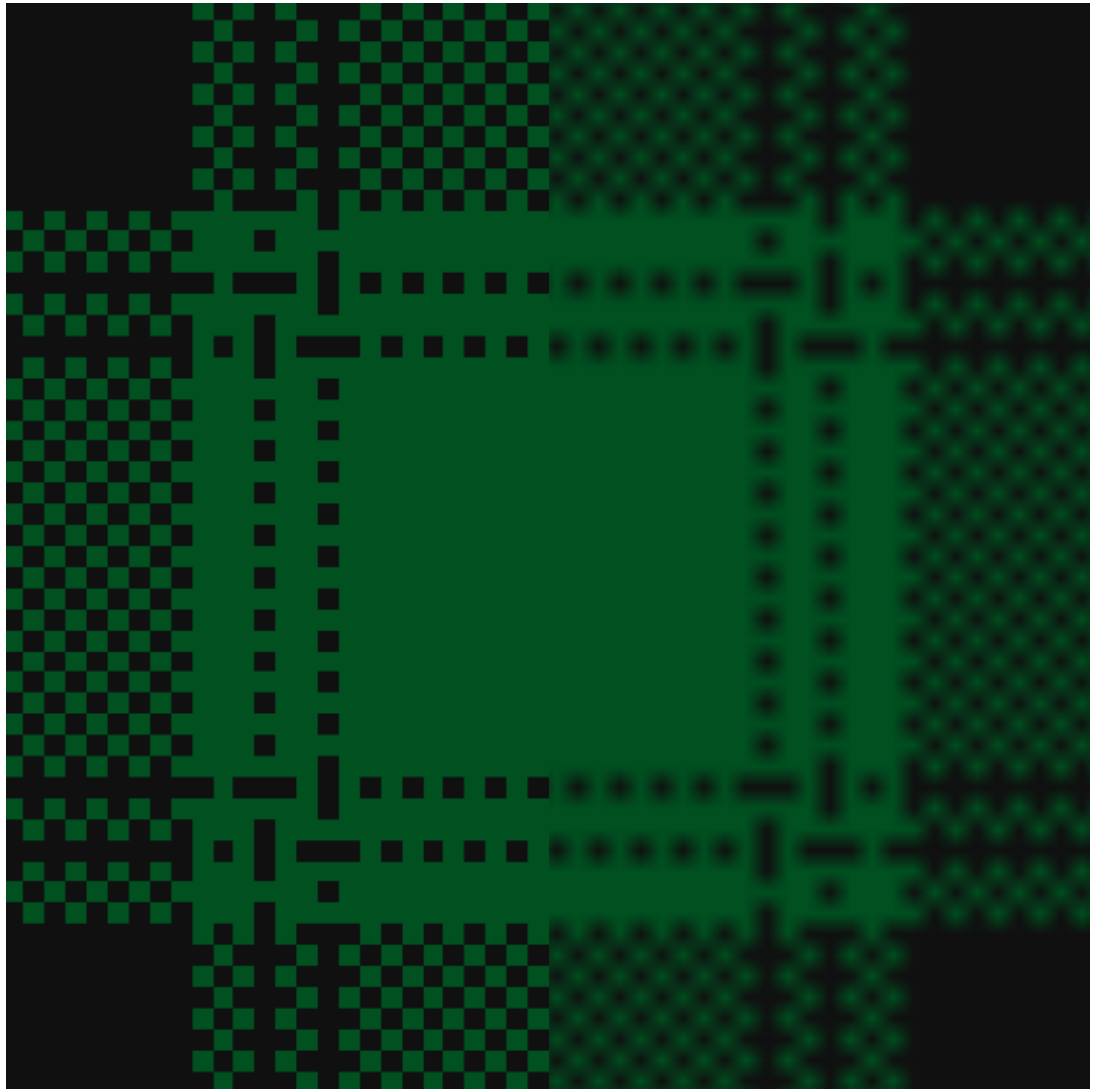
Pseudo Code

```
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()
```

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"

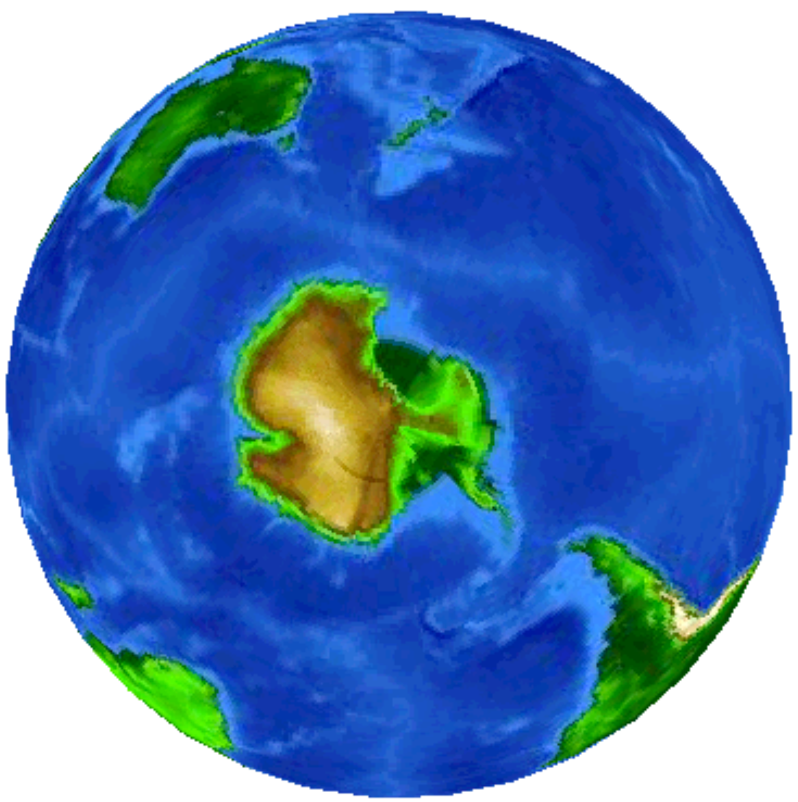
Geometric Texture



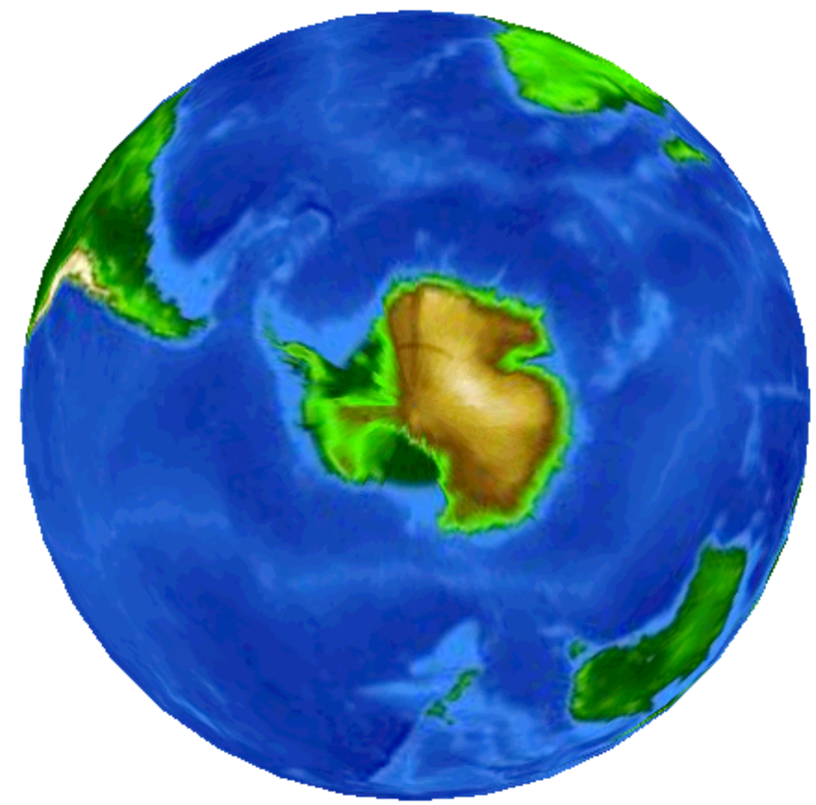
Nearest Neighbour

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
  - **File to RAM**: Image or texture data is first loaded from a file (e.g., PNG, JPEG) into the main memory (RAM).
  - **RAM to VRAM**: The data is then transferred from RAM to the graphics memory (VRAM), which the GPU can access quickly for rendering operations.
- *Pixel Operations:*
  - *scaling, biasing, & mapping*
  - *clamping*
  - *rasterization*

Texture Operations

Pixel Operations

- **Scaling, Biasing, and Mapping**: These are fundamental operations in adjusting pixel values:
  - **Scaling**: Adjusts the brightness or color by multiplying pixel values.
  - **Biasing**: Adds a constant to pixel values, adjusting brightness in a linear fashion.
  - **Mapping**: Transforms pixel values into a desired range, often between [0, 1].
- **Clamping**: Ensures pixel values stay within a specific range (e.g., [0, 1]). If a value falls outside, it's clipped to the nearest limit.
- **Rasterization**: Converts vector graphics or shapes into a raster format (pixel-based) that can be displayed on a screen.

Texture Image Coordinates

- OpenGL indexes texels with [0..1] rather than absolute pixel positions. This makes textures resolution-independent, as coordinates are normalized assumes image size is 2^m x 2^n
- OpenGL supports the use of up to 3 texture coordinates (s, t, r) for advanced effects, although 2D textures only need s and t.Surface parameters don't always match

Image Coordinates

- Normalized coordinate system in OpenGL, where both surface parameters and texture coordinates are indexed within the [0..1] range, is highly beneficial for mapping surfaces to textures.
    - Resolution Independence
    - Seamless Mapping of Surface to Texture
    - Texture Wrapping
    - Easier Transformations

Texture Transformations with the Texture Matrix

- **Surface Parameter Mismatch**: Sometimes, texture coordinates and surface parameters don't align perfectly due to model transformations or differing coordinate systems.
- **Texture Matrix**: OpenGL provides a texture matrix to apply transformations to texture coordinates, aligning them to match the model or world space.
- **By default**, this matrix is an identity matrix, meaning it applies no transformation until modified.

Specifying Parameters

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