Phong Shading and Texture Mapping

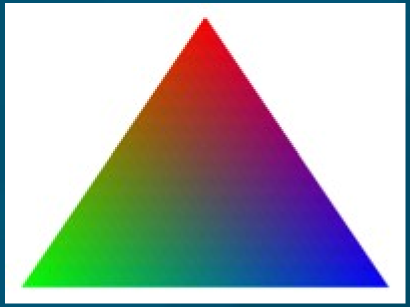
10 March 2009
CMPT370
Dr. Sean Ho
Trinity Western University

What's on for today

- Shading polygons
  - Flat shading
  - Gouraud shading
  - Phong shading
- Texture mapping
  - Coordinate transforms
  - Cylinder, sphere, cube maps
  - Bump mapping
  - Environment mapping

Shading polygons

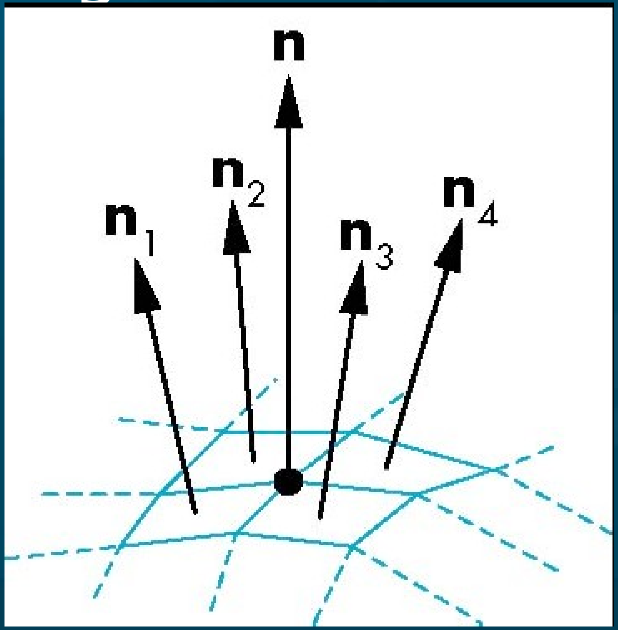
- We specify in our model for each vertex:
  - Vertex coordinates
  - Vertex colours
  - Vertex normal



- Use lighting model to calculate vertex shades
- Smooth shading: vertex shades are interpolated across the polygon
    - glShadeModel(GL_SMOOTH);
- Flat-shading uses the colour of the first vertex:
    - glShadeModel(GL_FLAT);

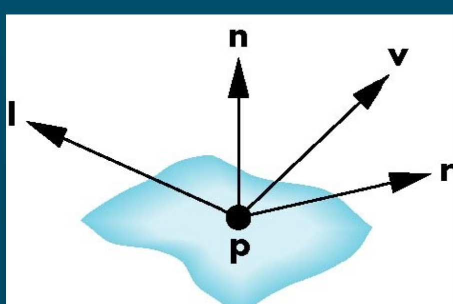
Calculating vertex normals

- Each polygon is flat: we can find normal vector
- One strategy: average the normals of the faces surrounding that vertex
- For triangular faces ABC: AB x AC gives normal
  - Magnitude is area of parallelogram
- Sum these cross-products:
  - Get a weighted average of face normals
  - Weighted by area

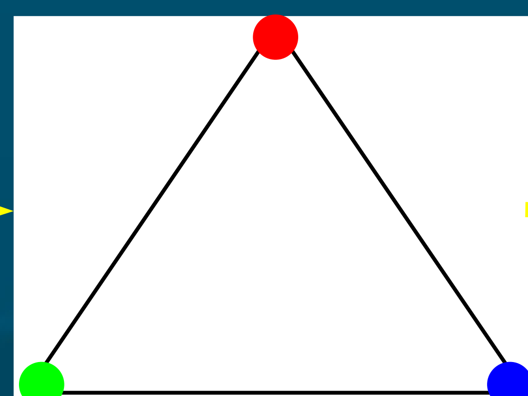


Gouraud shading

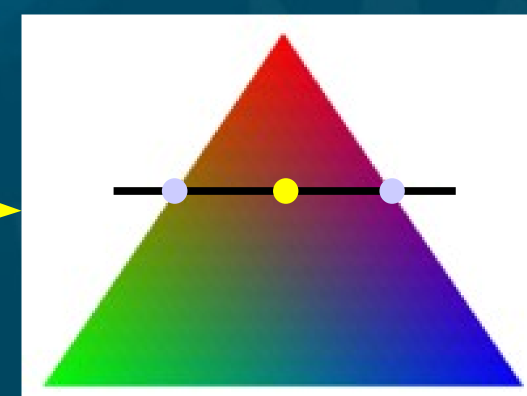
- Specify vertex normals
- Apply lighting model (ambient, diffuse, specular) to each vertex to get vertex shades
- Interpolate vertex shades across polygon
  - Interpolate along edges first
  - Then along each scan line (done in hardware)



vertex normals



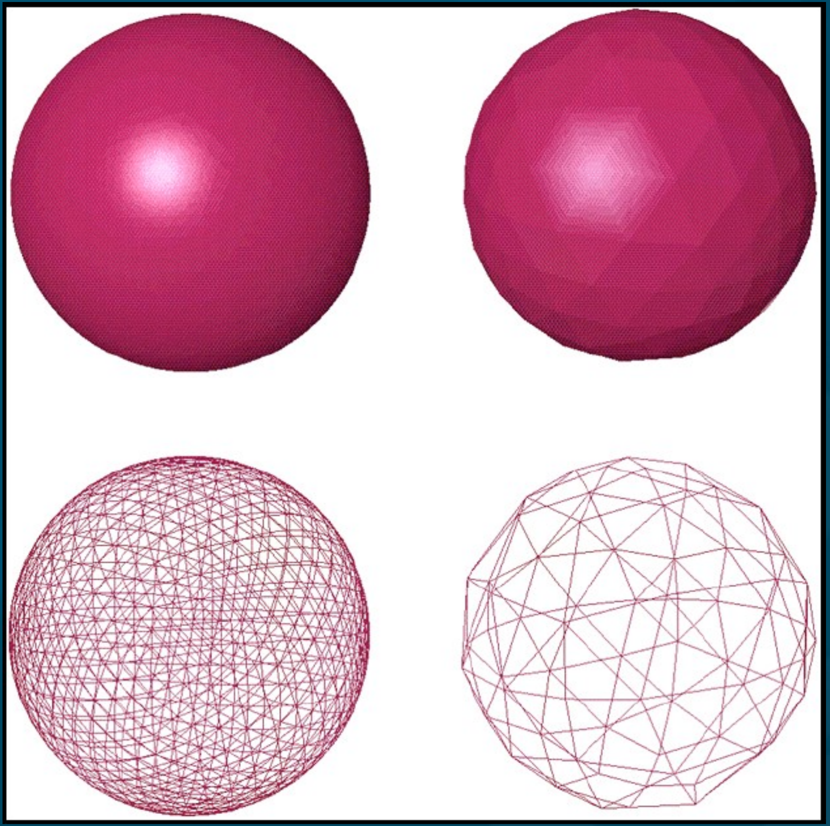
vertex shades



filled fragment

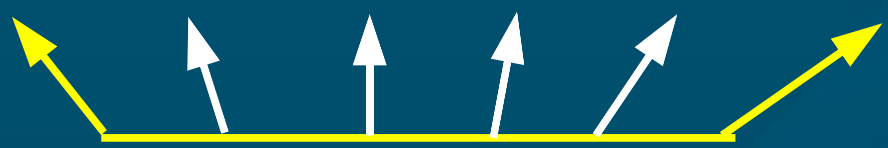
Gouraud shading: quality

- Depends on how big each polygon appears on screen, compared to pixel size
  - Fewer polygons => bigger on screen => worse quality



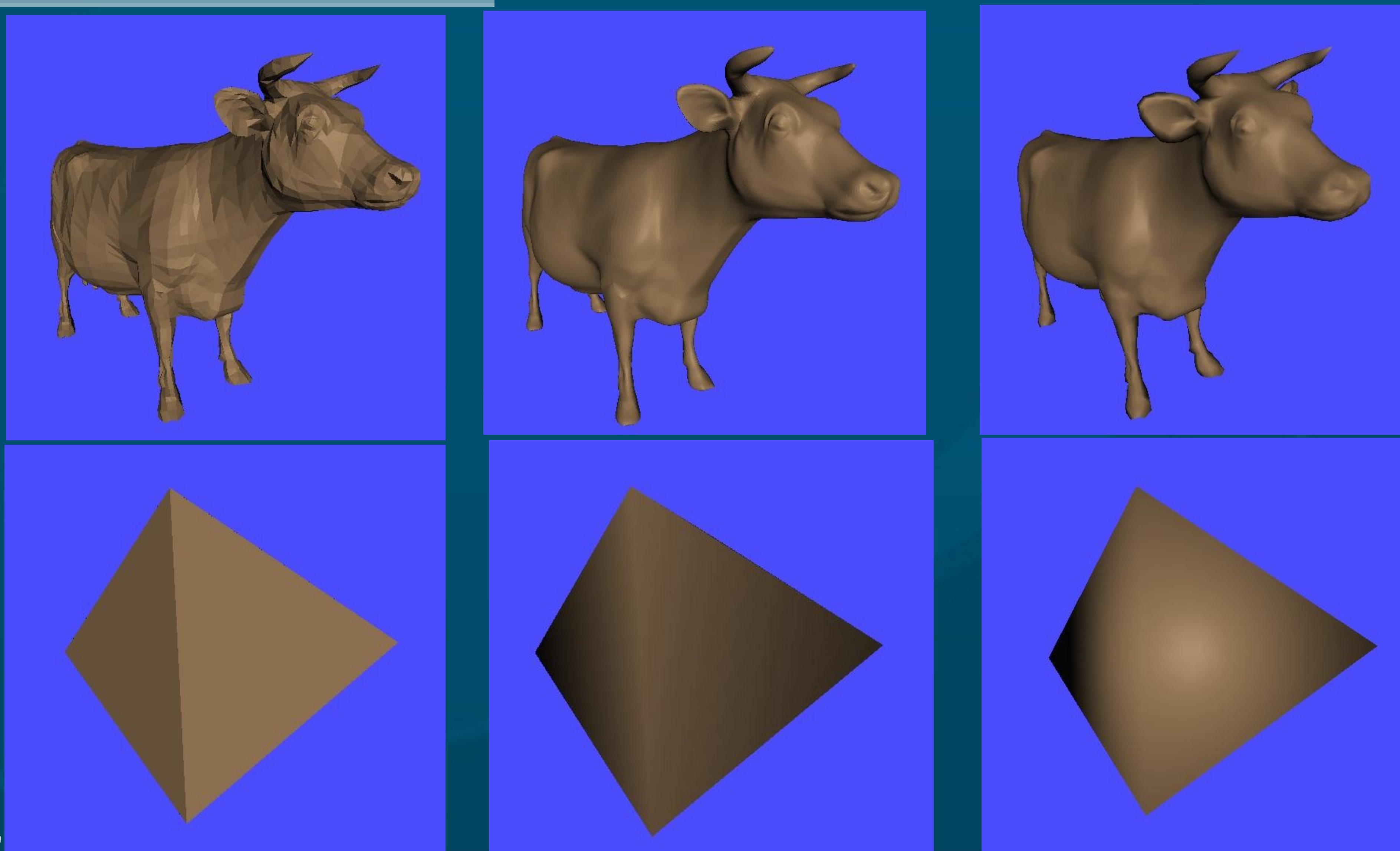
Phong shading

- Specify vertex normals
- Interpolate vertex normals across polygon
    - Interpolating vectors, not intensities!
- Apply lighting model at each pixel to get shades



    - Gouraud may miss small specular highlights
- OpenGL implements Gouraud but not Phong
    - Work to calculate lighting model at each pixel
    - Or use programmable shaders!

Flat vs. Gouraud vs. Phong



Texture mapping

- Complex objects with many varying shades:
  - Could use a new polygon for every shade
  - Or use an image pasted on top of the surface
- E.g., modeling the earth:
  - Blue sphere is too simple
  - Modeling every continent and mountain range with little polygons is too much
  - Texture-map a picture onto the sphere

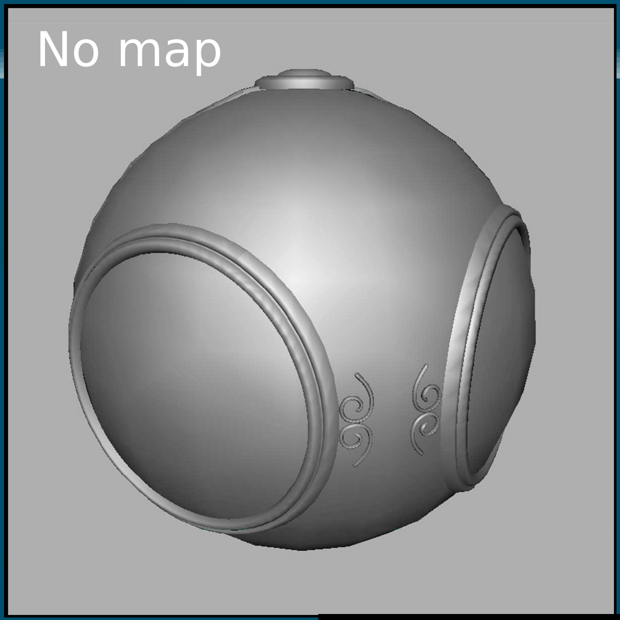
Bump mapping

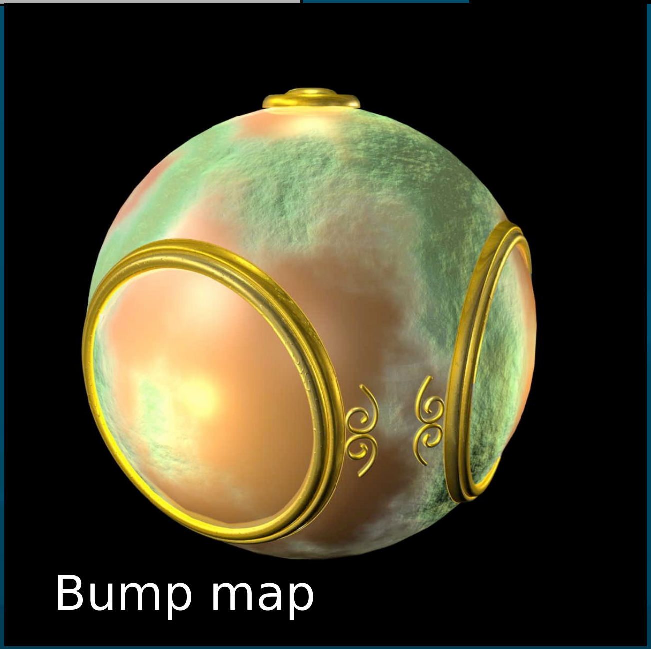
- e.g., modeling an orange:
  - Geometry is just a simple sphere
  - Texture map colours, striations, etc.
  - But surface is still smooth: what about small dimples?
    - Shading should change as light and view directions change
- Bump mapping tweaks the normal vectors to simulate dimples or bumpiness
  - Silhouette still reflects underlying geometry

Kinds of maps

- Texture map:
  - Paste an image onto a surface
- Bump map:
  - Perturbs normal vectors in lighting model to simulate small changes in surface orientation
- Environment (reflection) map:
  - Use a picture of the surrounding room/sky for a texture map
  - Simulates reflections in very specular surfaces
- Only texture maps are built-in to OpenGL

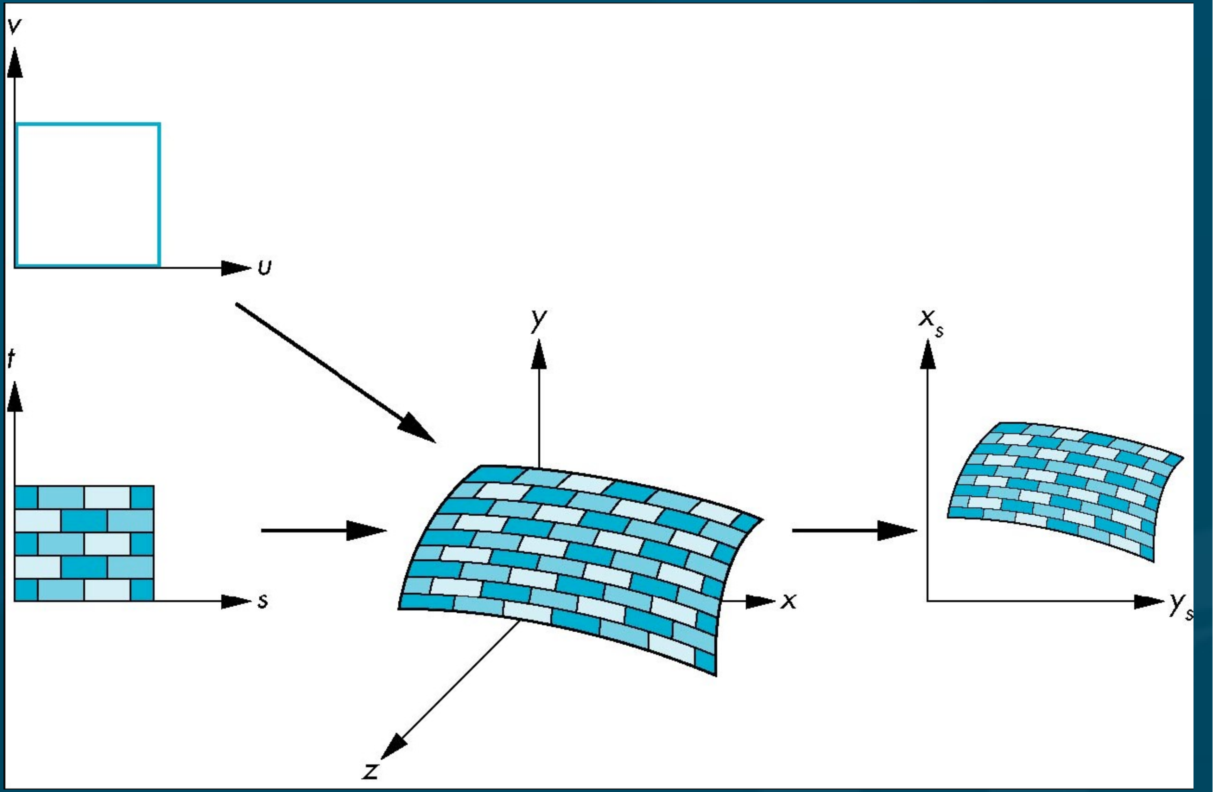
Texture/bump/environ maps





Mapping: coordinate systems

- Essential question for maps: how to map coordinate systems?
  - Parametric coords (u,v) describing the surface
  - Texture coords (s,t)
  - World coords (x,y,z)
  - Window coords (x_s, y_t)

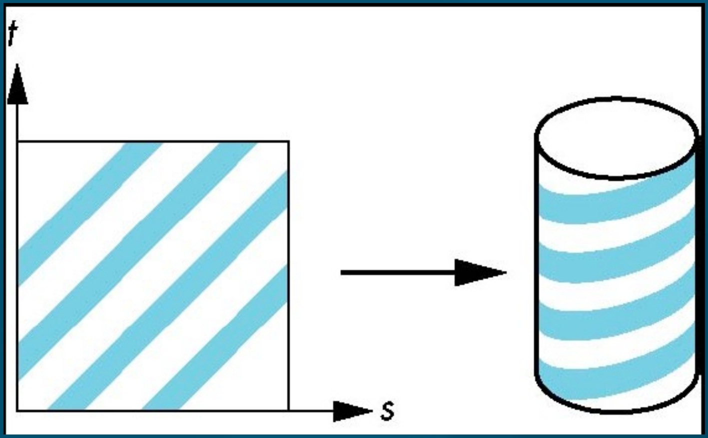


Backward mapping

- For each point (x,y,z) on the surface in world coords, we want to go backwards to find which pixel (s,t) in the texture we should paste:
  - s = s(x,y,z);
  - t = t(x,y,z);
- Two-part mapping:
  - First map texture onto a simple intermediate shape
    - Cylinder
    - Sphere
    - Cube

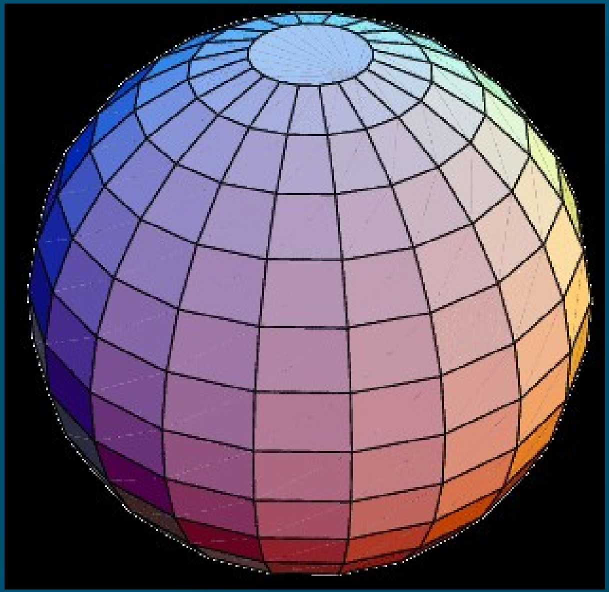
Cylindrical mapping

- Parametric cylinder:
  - $x = r\cos(2\pi s)$
  - $y = r\sin(2\pi s)$
  - $z = t/h$

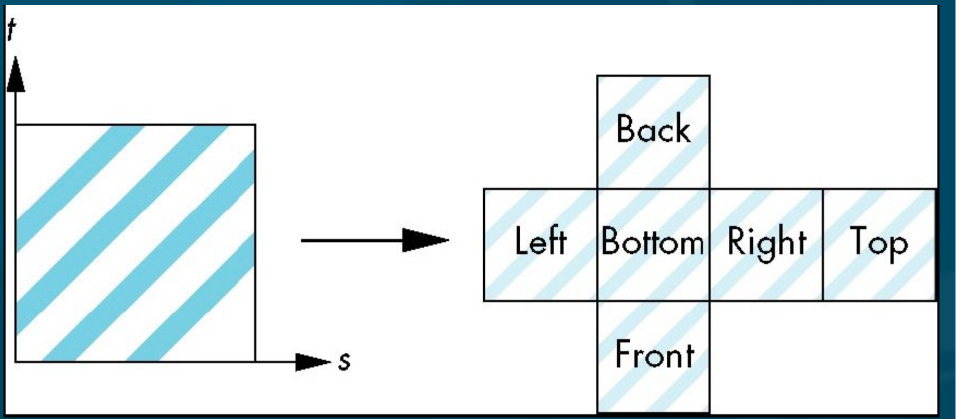


- Map from
  - Square [0,1] x [0,1] in (s,t) texture space to
  - Cylinder of radius r, height h in (x,y,z) world coordinates

Spherical maps, cube maps



- Parametric sphere:
  - $x = r\cos(2\pi s)$
  - $y = r\sin(2\pi s)\cos(2\pi t)$
  - $z = r\sin(2\pi s)\sin(2\pi t)$
  - Bad distortions at the poles
- Cube/box mapping:
  - Easy with orthographic projection



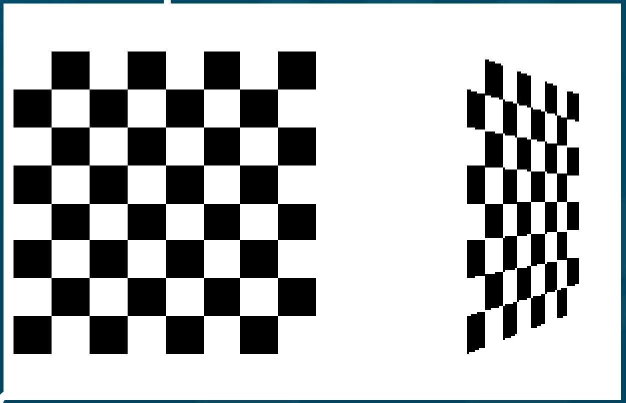
- Both are widely used for environment maps

Implementing bump mapping

- Parameterized surface:
  - $p(u,v) = (x(u,v), y(u,v), z(u,v))$
  - Tangent vectors: $p_u = \partial p/\partial u$, $p_v = \partial p/\partial v$
  - Normal vector: $n = p_u \times p_v$
- Perturbed surface: $p'(u,v) = p(u,v) + d(u,v)\, n(u,v)$
  - $d(u,v)$ is the displacement function/map
- Perturbed normal: $n' = p'_u \times p'_v$
  - $n' \approx (\partial d/\partial u)(n \times p_v) + (\partial d/\partial v)(n \times p_u)$

Texture mapping in OpenGL

- Bump mapping / environment mapping are not provided in OpenGL
  - Can be done with fragment programs (GLSL)
- Using texture mapping in OpenGL:
  - Create texture and bind to object
  - Select how texture will affect each pixel
  - Enable texture mapping
  - Draw object, specifying texture coordinates



- See Redbook examples, checker.c

Creating a texture

- The following steps should be done once during initialization, not on every display refresh:

- Read in an image: 3D array (rows, cols, RGBA)
  - Programmatically generate (checker.c), or
  - Read from file (Fl_JPEG_Image->data())
- Bind new texture object: glBindTexture()
- Specify parameters: wrapping, filtering
- Load image data to texture: glTexImage2D()

Texture objects (OpenGL 1.1)

- Akin to display lists, but for textures
- Allows us to reuse textures, bind to objects
  - Request a new texture object id
    - glGenTextures(1, &texName);
  - Can also request several texture object ids
  - Bind this new texture object
    - glBindTexture(GL_TEXTURE_2D, texName);
- All subsequent texture commands are stored in this texture object
- Use glBindTexture() to switch texture objects

Loading image data to a texture

- glTexImage2D(GL_TEXTURE_2D, *level*, *intFmt*, *width*, *height*, 0, *format*, *type*, *pixels*)
  - level: mip-mapping level, usually 0
  - intFmt: GL_RGB, GL_RGBA, etc.
  - width, height: must be power of 2, ≥64
      - *(border: most hardware only supports '0')*
  - format, type: describe incoming pixels:
    - e.g., GL_RGB, GL_UNSIGNED_BYTE
    - Affected by glPixelStore(), similar to glDrawPixels()
  - pixels: pointer to the actual image data

Texture size must be 2^n

- OpenGL requires the width and height of textures to be powers of 2
  - But need not be square
- GLU provides a helper function to scale:
  - gluScaleImage(fmtIn, wIn, hIn, typeIn, *pixelsIn, wOut, hOut, typeOut, *pixelsOut)