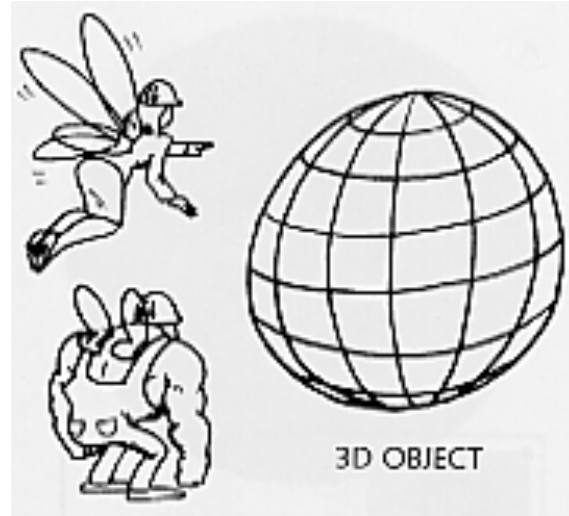
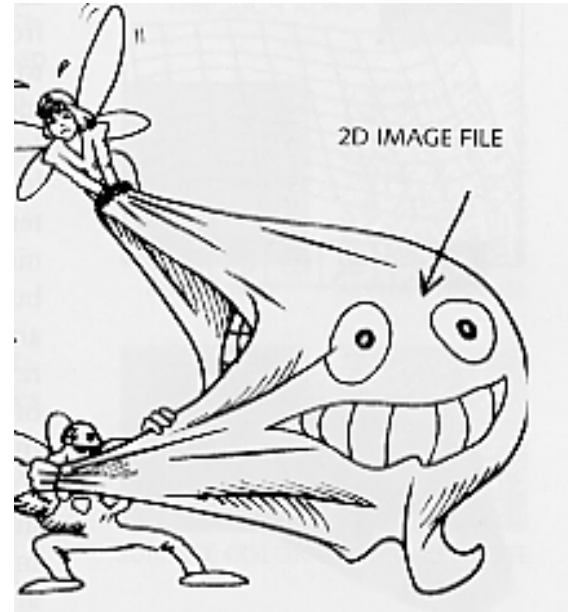
# Texture Map

# Texture

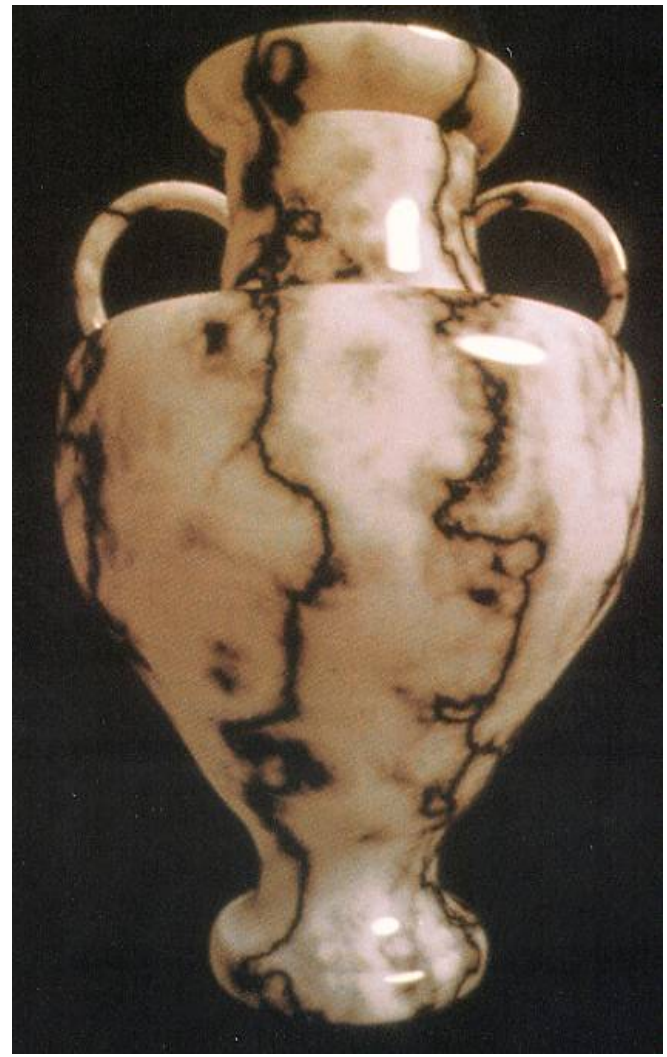
– Application of 2-D or 3-D texture onto surfaces of objects
– Give complexity without overhead of geometry
– Implemented in graphics cards



# Sources:

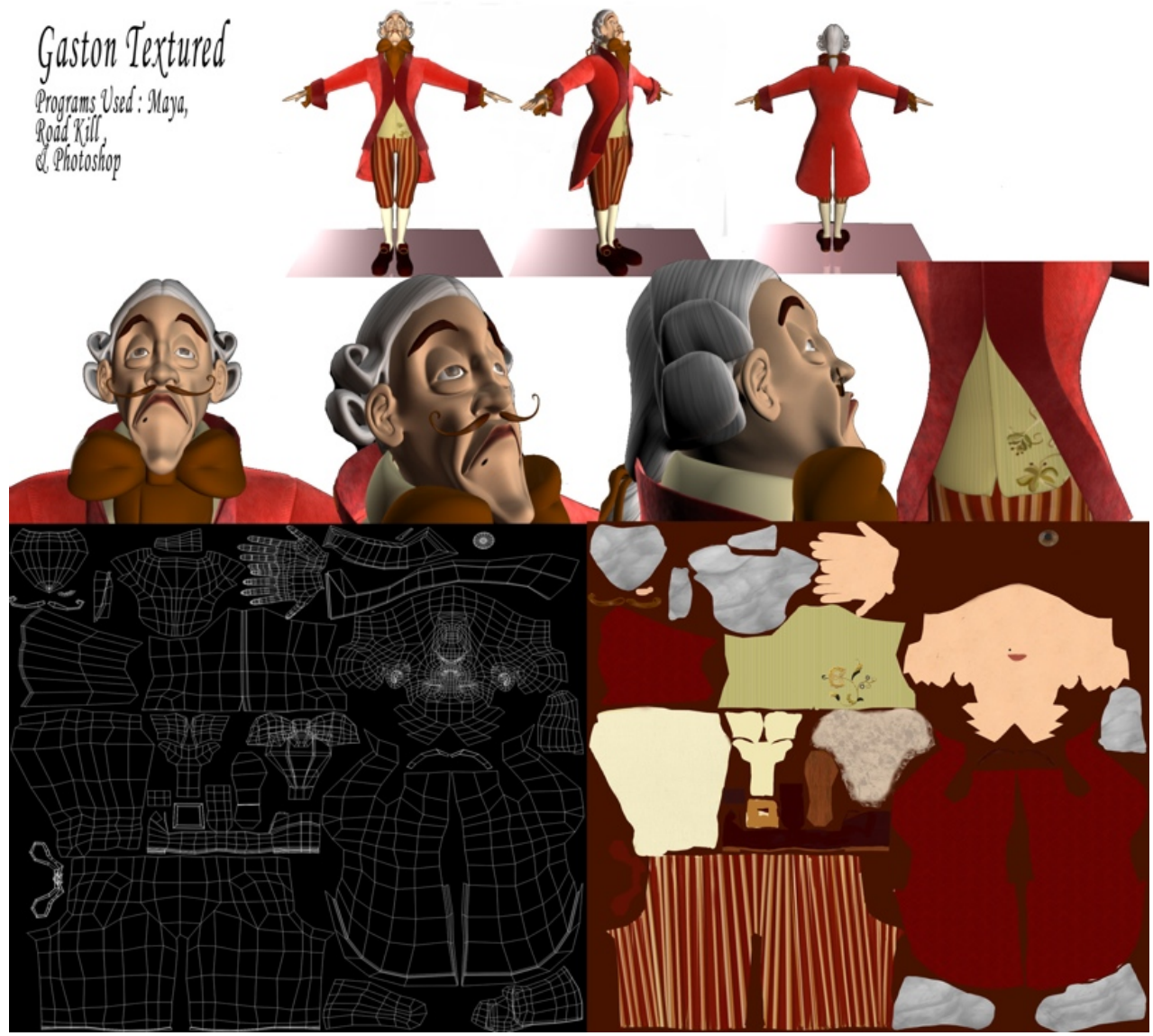
- Artwork
- Photos/Scanned Images
- Procedural (mathematical)
  - Image on right shows “Perlin noise” to generate synthetic marble texture
- Digitized from real 3-D object (CT, MRI, etc.)



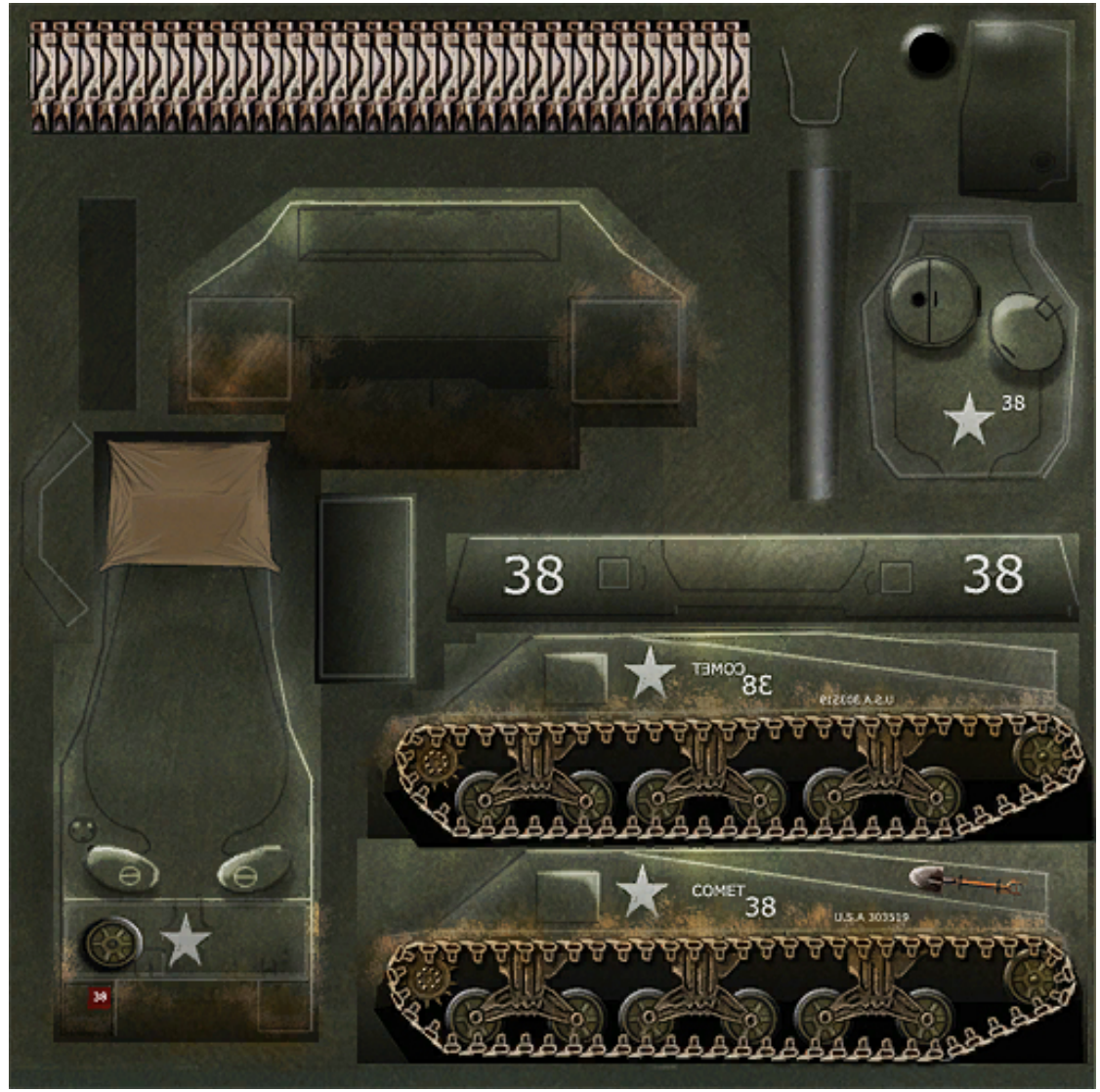
# Color Texture

Model:

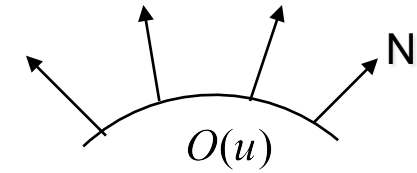
Texture:

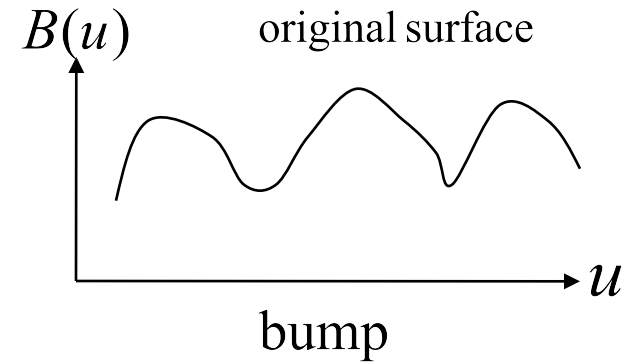



http://www.melmelsmith.com/

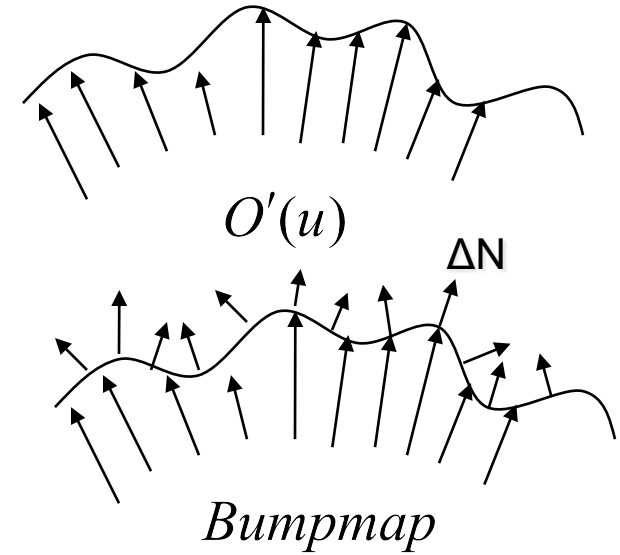


# Bump mapping




- Perturb surface normal without actually changing the surface
- Store “bumped” normals in a texture
- Problem at silhouette



# Displacement mapping

- Surface is actually displaced
- Triangles subdivided and vertices displaced according to map
- Self occlusion
- Bumpiness at the silhouette

# Combination

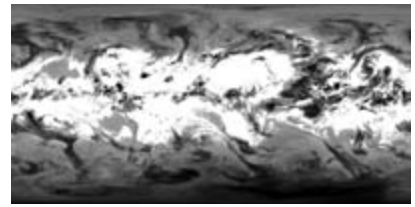
– Reflectivity (specular mapping)
– Transparency of the surface (transparency mapping)

Color map

Bump map

Specular map

Color map



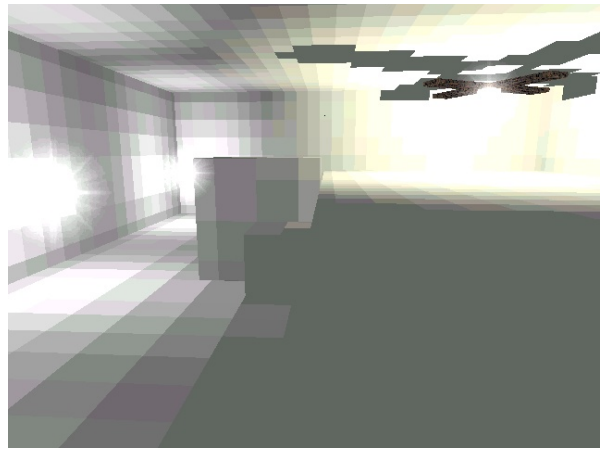
Transparency map

http://gw.marketingden.com

# Light/shadow map

- Used to modulate diffuse lighting on a surface
- Only works for static lighting
- If lighting changes, light map needs to change

Rendered with just texture map
(no lighting)

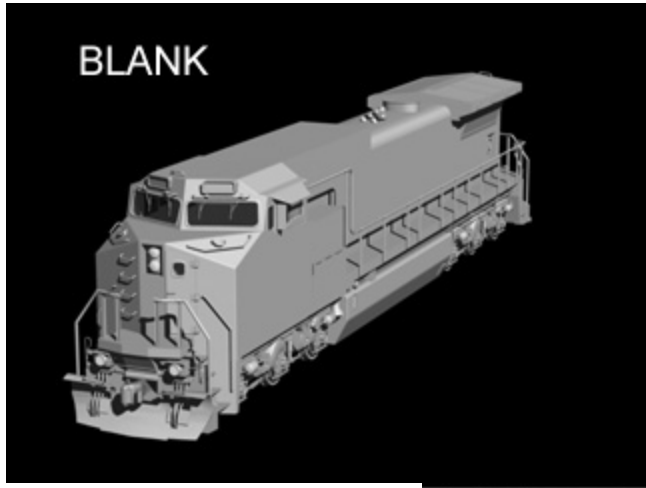


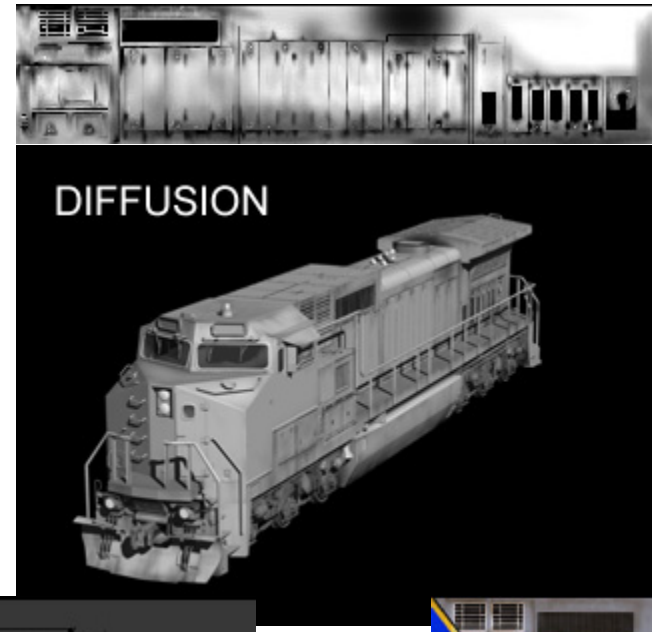
Pre-calculated light map

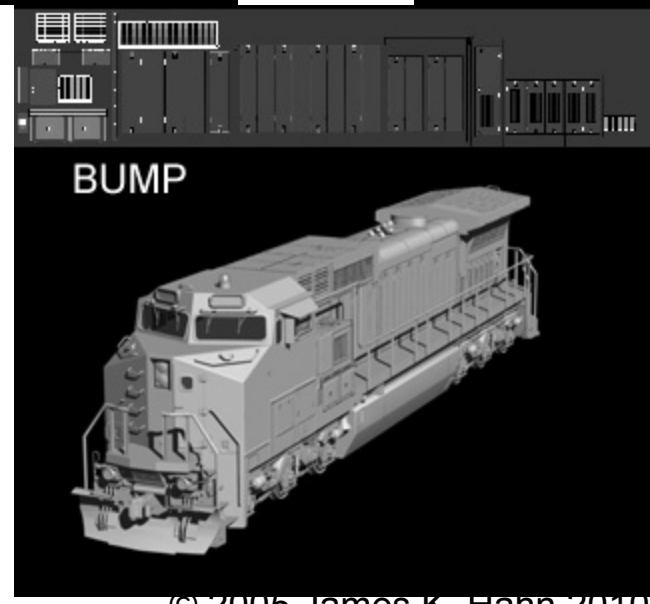
Texture map with light map
(multi-texturing)

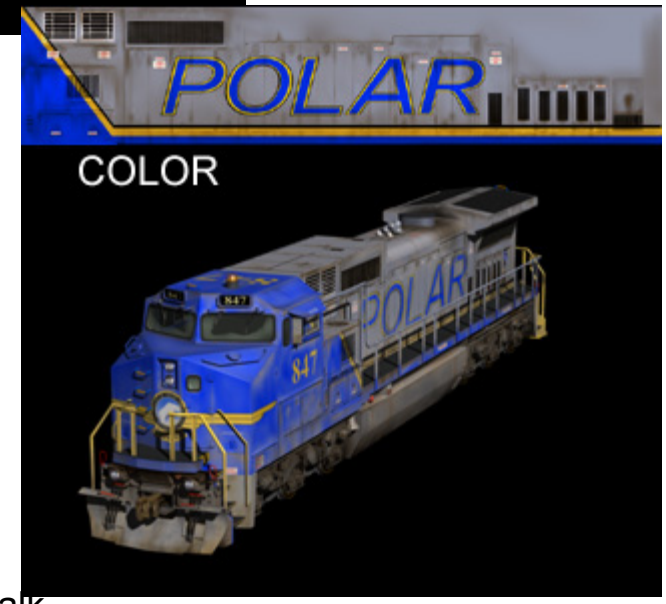
# Combination

http://www.arance.net/textures.htm



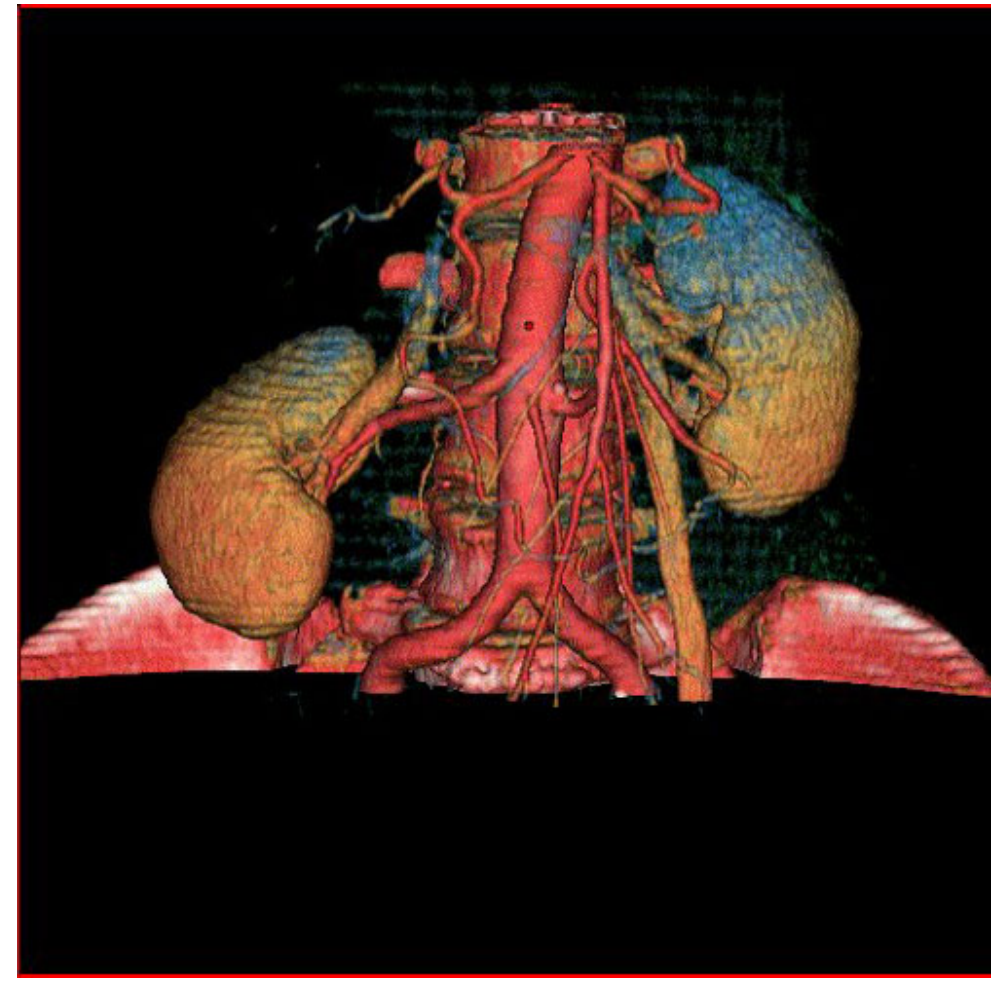



# Volume Texture

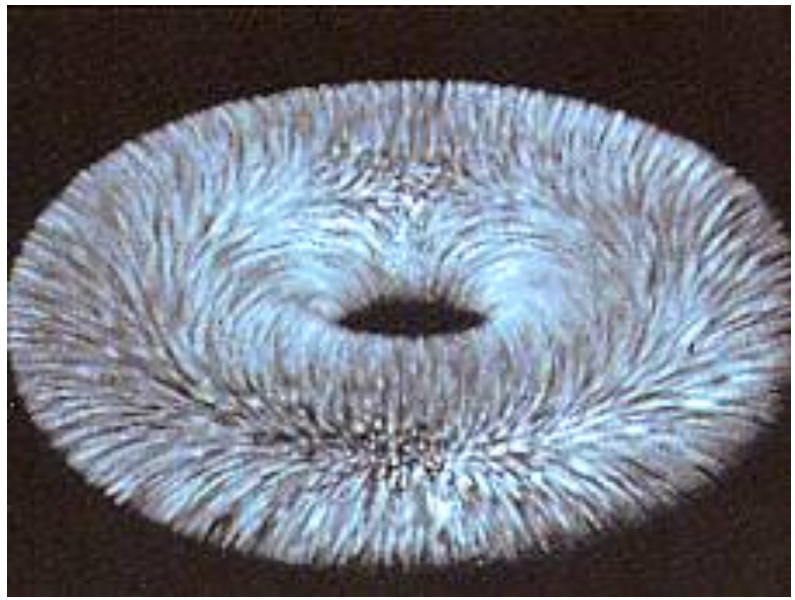
- A stack of 2D Images makes a 3D image
- Pixels (Voxels in 3D) have a "density" (0..1) that defines what is seen at what is not visible
- Pixels (Voxels) may have color too
- Visualize via Volume Rendering



http://www.gehealthcare.com

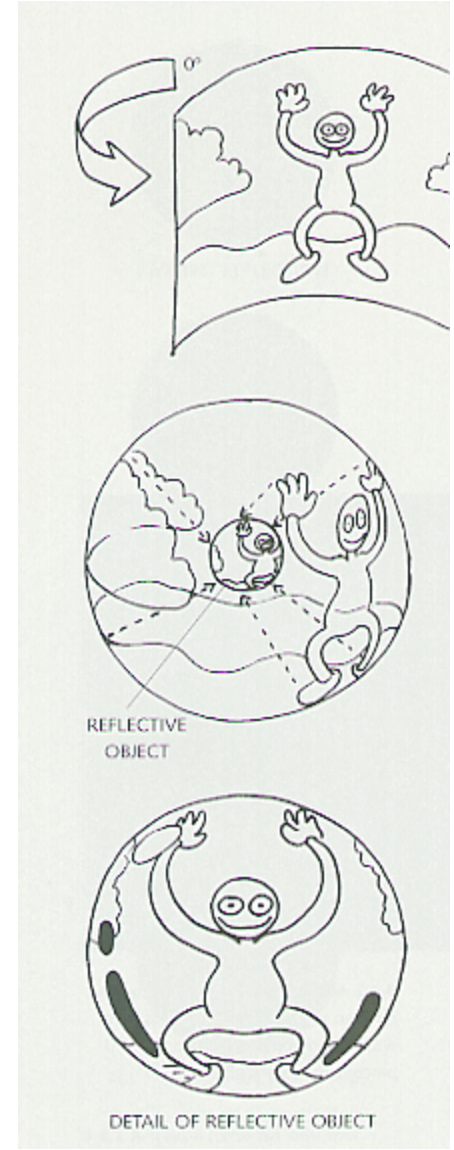
# Hypertexture

– No object shape
– A procedurally generated volume texture
– A 3D box defines the texture extent
– Can think of the texture map as a density function

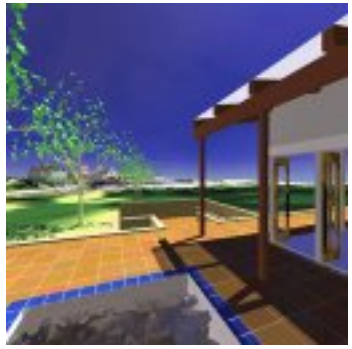


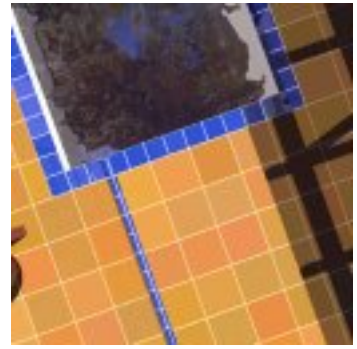
# Environment map

- Poor man's ray-tracing
  - Similar effect as ray-tracing
  - Computationally much cheaper
- Create environment map
  - Image of the environment from one viewpoint



- Map onto surface of objects using reflection direction
- 6 texture maps on cube - pick map based on largest x,yz component of reflection direction


back

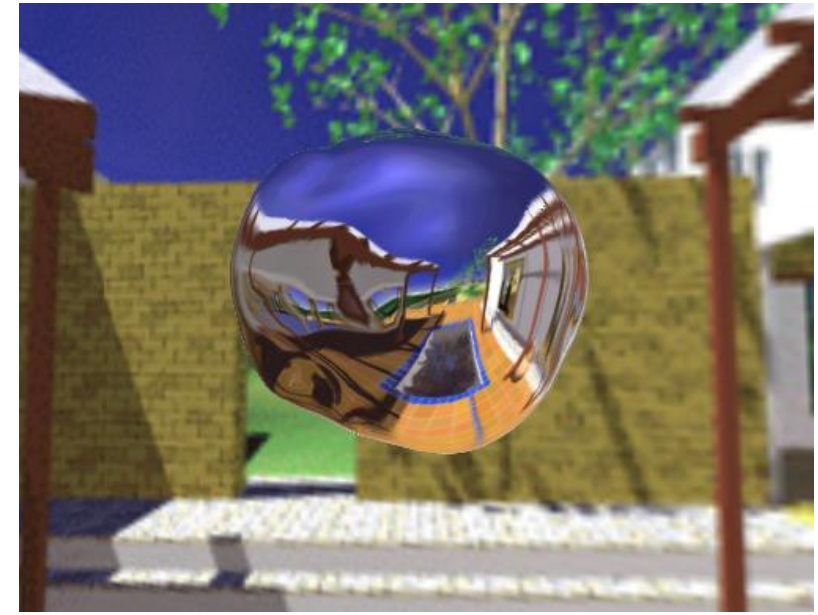

bottom


front

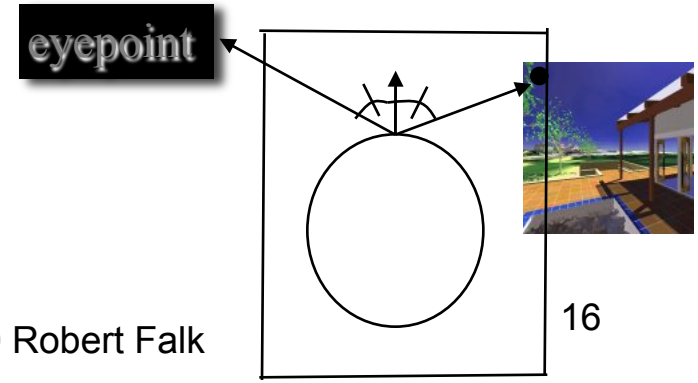

left


right

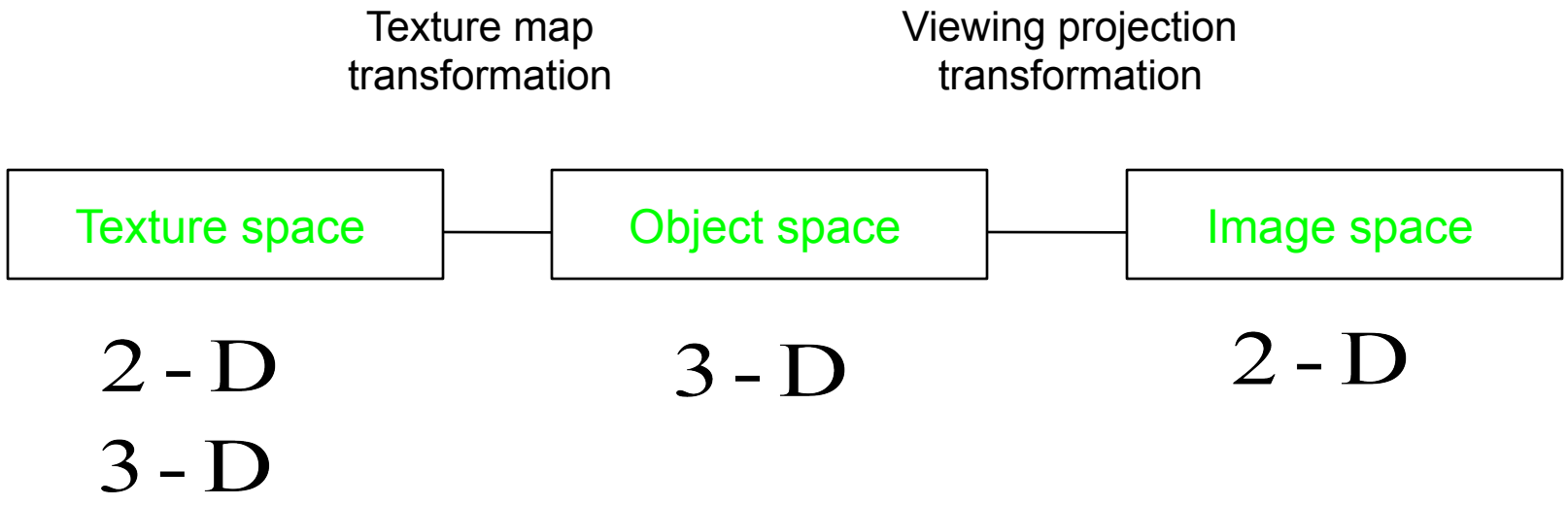

top



http://developer.nvidia.com/object/cube_map_ogl_tutorial.html

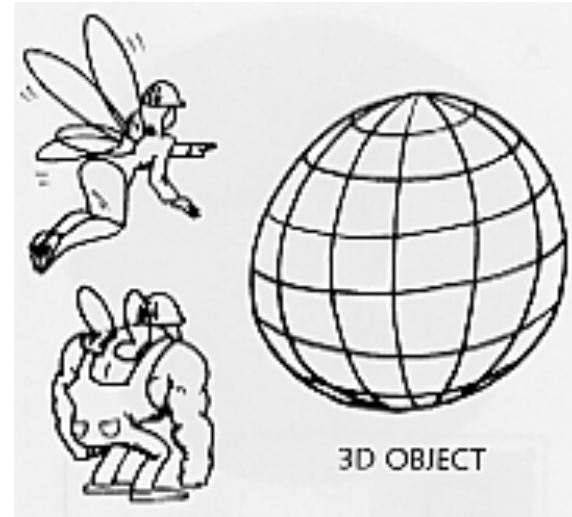


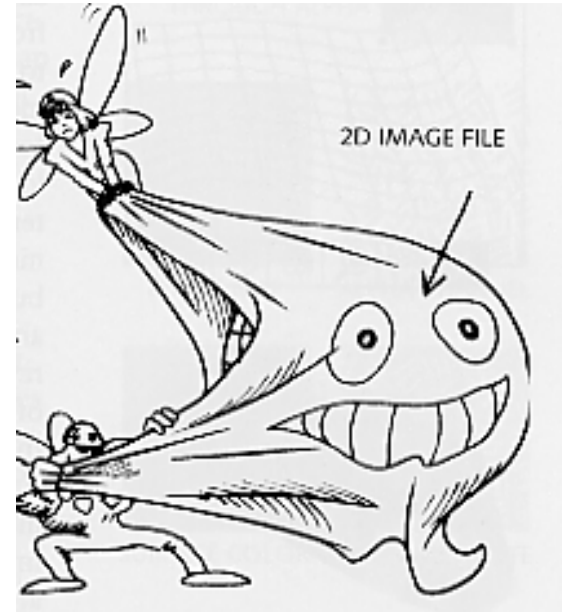
# Mapping



# Mapping 2-D texture onto polygons

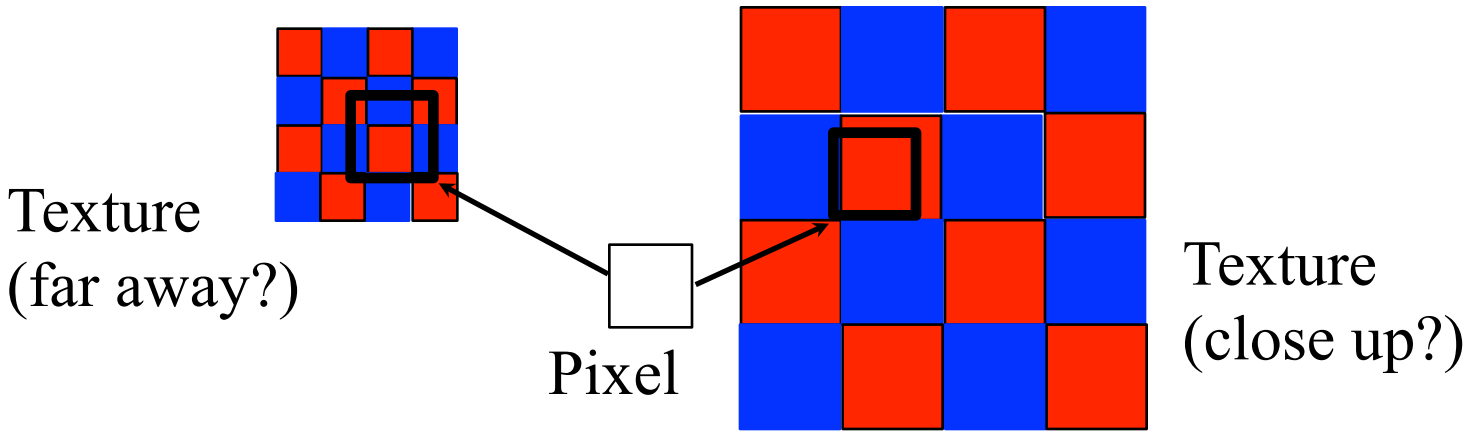
- Texture space $T(u,v)$, $u, v \in [0,1]$
- Define mapping (transformation function) from object space to texture space
  - $F(x, y, z) = (u, v)$
  - usually for vertices of polygons
- Interpolate to get $(u, v)$ for points interior to the polygon



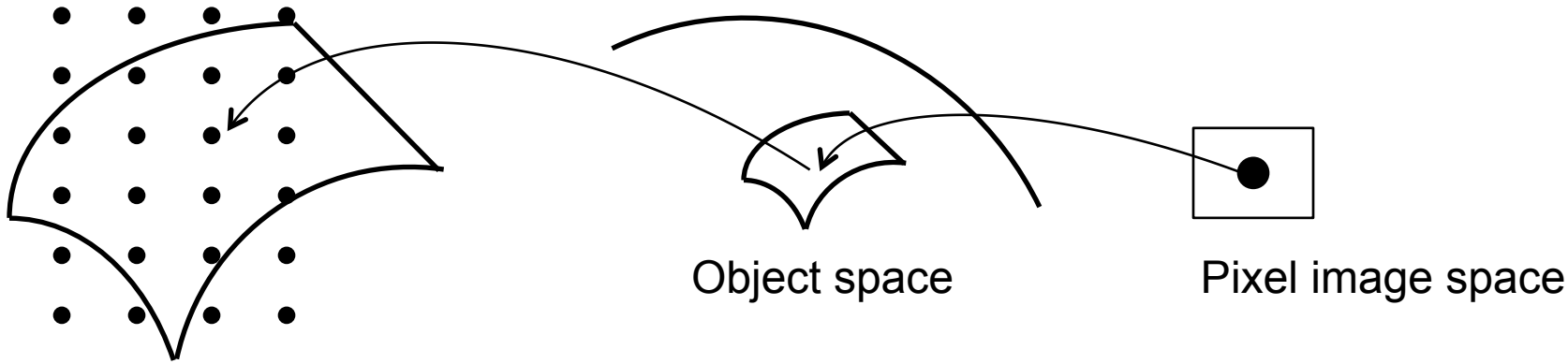



# Mapping between a pixel and a texel

- Pixel order scanning (image-order): For each pixel, find the “appropriate” texel value
  - A pixel may fall between texels
  - Pixel may cover many texels
  - Need transformation from image space to texture space

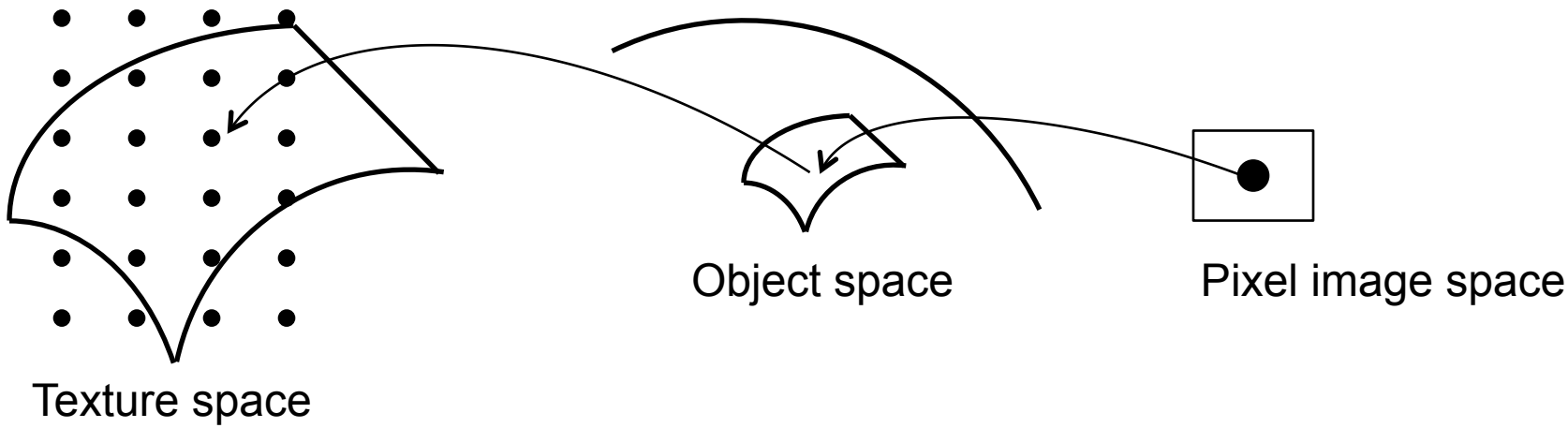


# Finding the appropriate texel




- Simplest: use the nearest texel: result in aliasing
- If pixel covers a number of texels, then average (above example)
- If pixel area smaller than texel area, then interpolate nearest texels

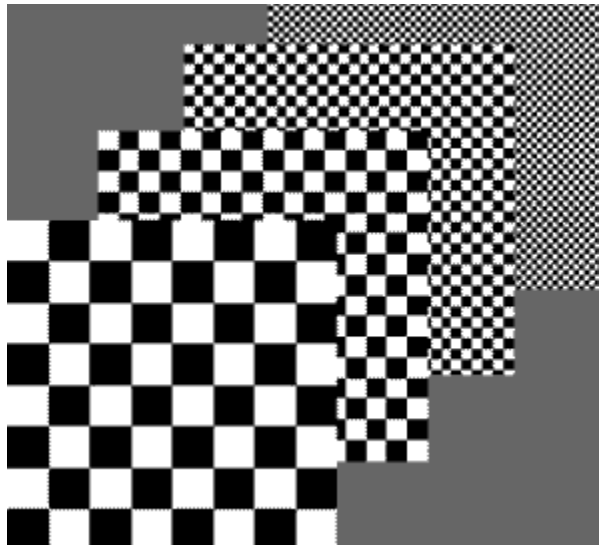
# Texture Mapping and Anti-Aliasing

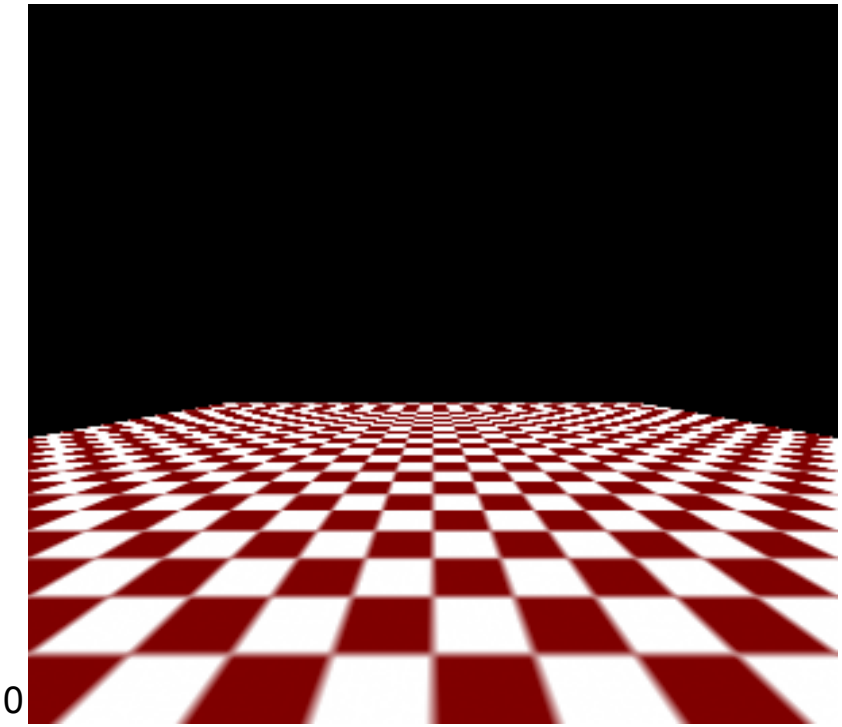



- To perform weighted area sampling transform pixel boundary (actually, it is the filter base) from image to texture space
- Two approachs
  - perform the operation for each pixel: too slow
  - pre-filtering: fast but may not be accurate (blurring or aliasing)

# Pre-filtering: Mip-Map

- When zoom in, texel size large compared to pixel size
  - May need to interpolate
- When zoom out, texel size small (must average many texels to get a pixel value)
  - If we do not average, aliasing





# Mip-Mapping

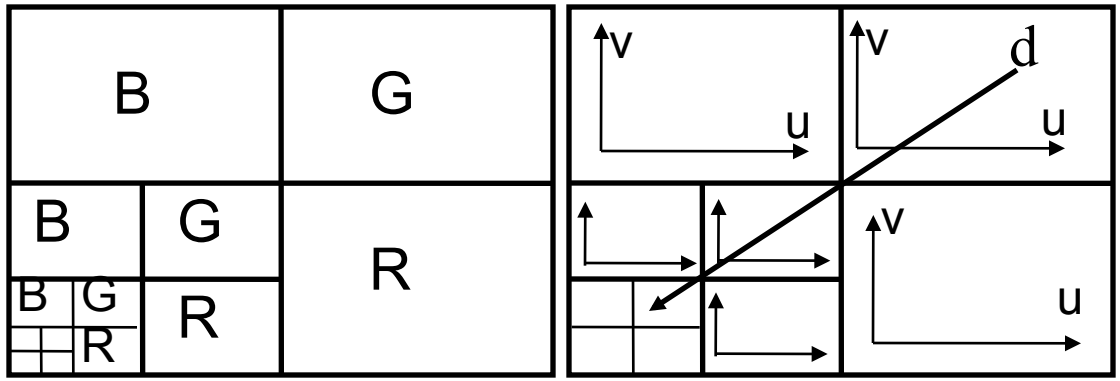
MIP stands for the Latin multim im parvo, meaning "many things in a small space”

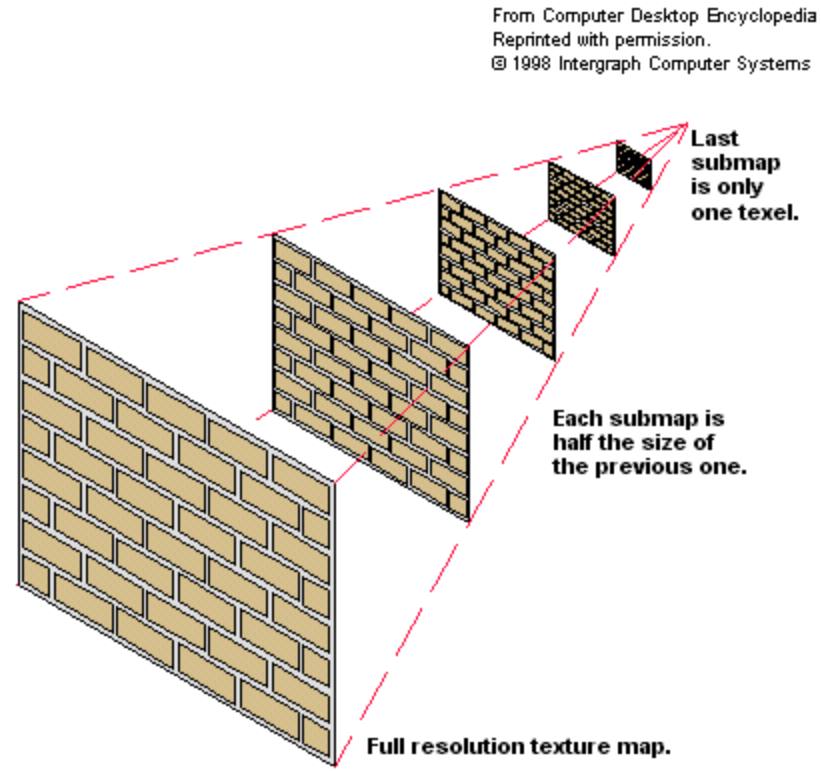
- Pre-average texels by combining 4 neighboring texels into one texel recursively
- “level of detail” of texture form a pyramid
- Each level ½ resolution of previews level

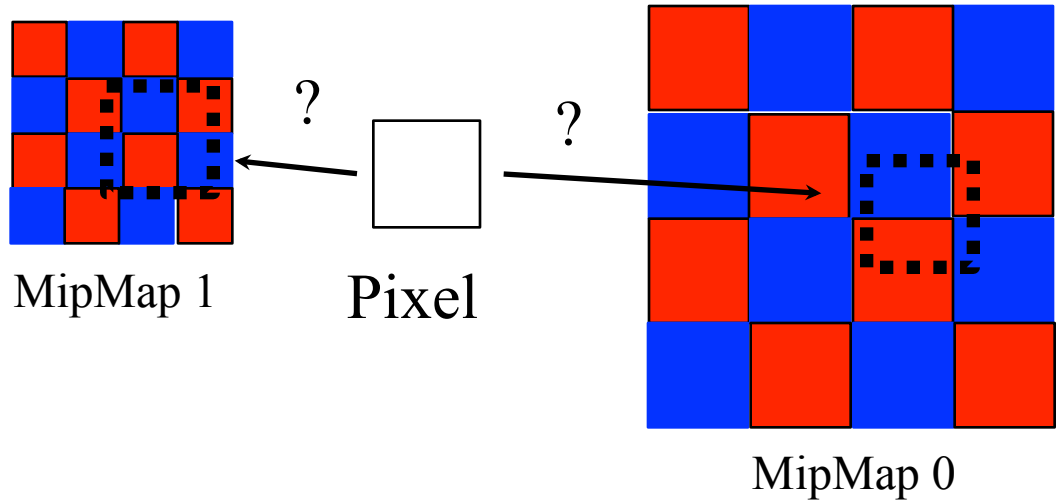

- Displayed at same resolution

# Storing Mip-Maps

- Can store levels efficiently by using ½ resolution at each level
- u, v varies from 0 to 1 at each level
- d gives the level of detail

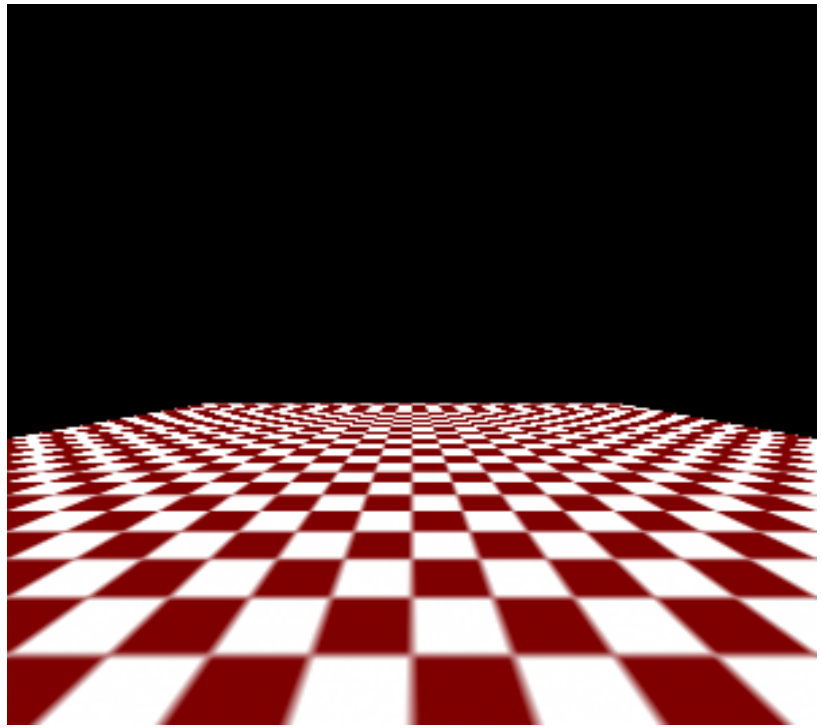


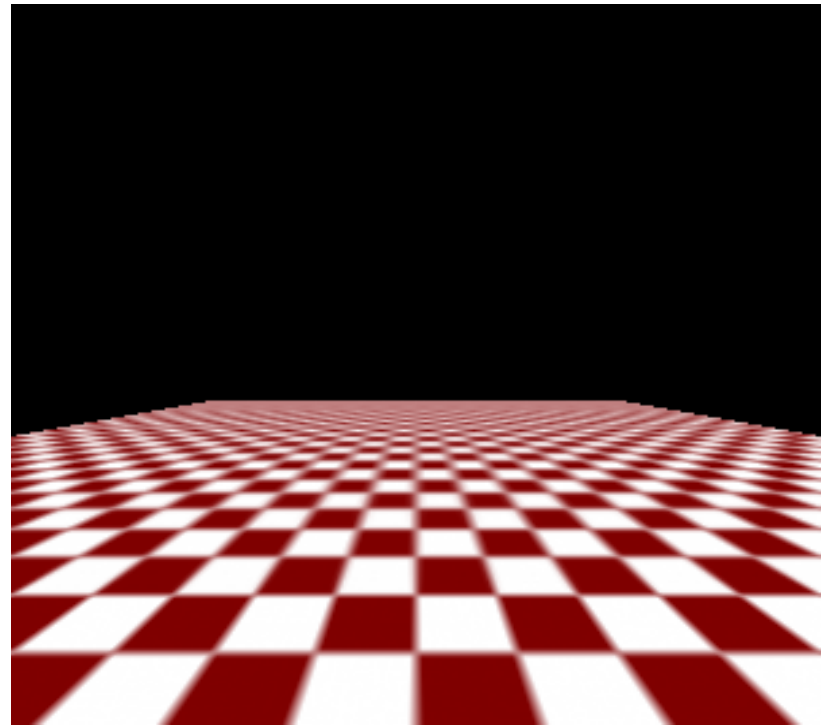






- For each pixel, determine the level of the texture map so that texel size is similar to pixel size
  - If the size of the pixel is in between the levels, can either choose the nearest level or
  - Interpolate between levels (interpolate the nearest texels between levels)
- Since pixels don't fall on the exact location of a texel
  - Choose the nearest texel within a level or
  - Interpolate nearest texels within a level
- OpenGL allows options to choose all 4 combinations

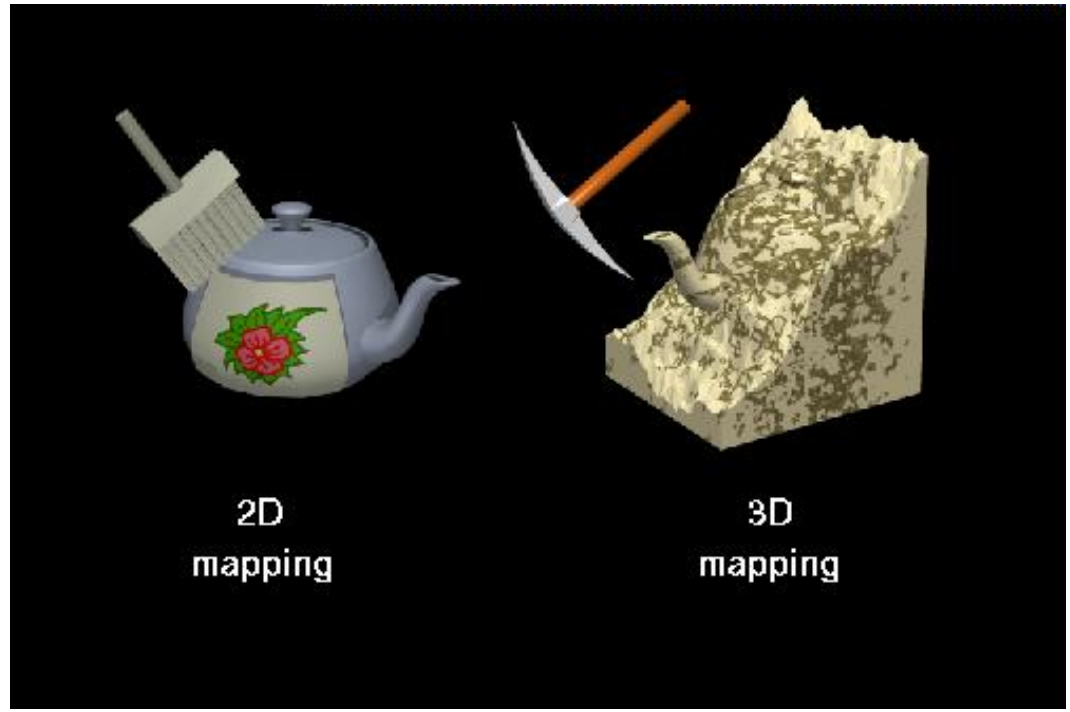
- Left image shows aliasing resulting from point sampling (pick one texel for each pixel)
- Right image shows the use of Mip Mapping to anti-alias
  - Note the blurring





# 3-D Texture Maps (volume texture or solid texture)

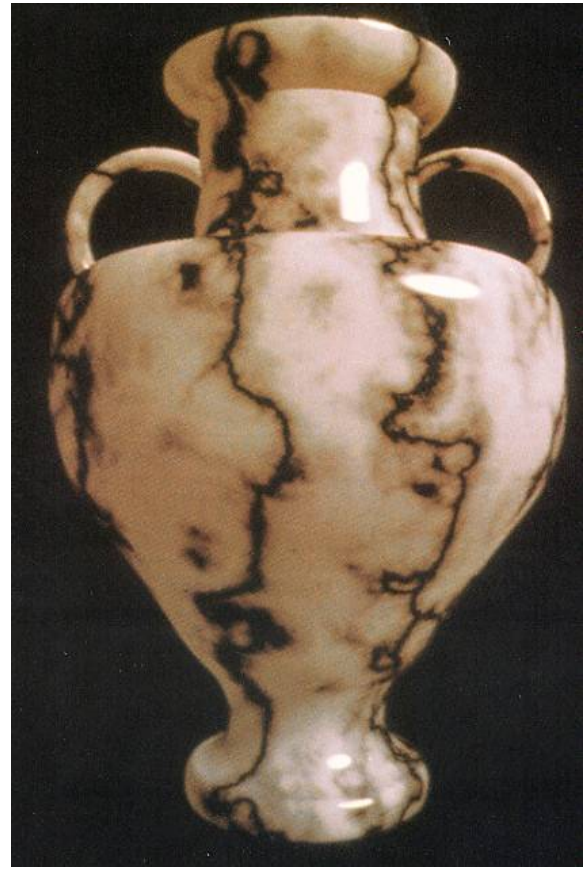
- Texture defined in 3-D
- Object embedded in texture map
- Surface of object intersecting the texture volume defines texture value
- Texture mapping transformation is simple




Kerlow

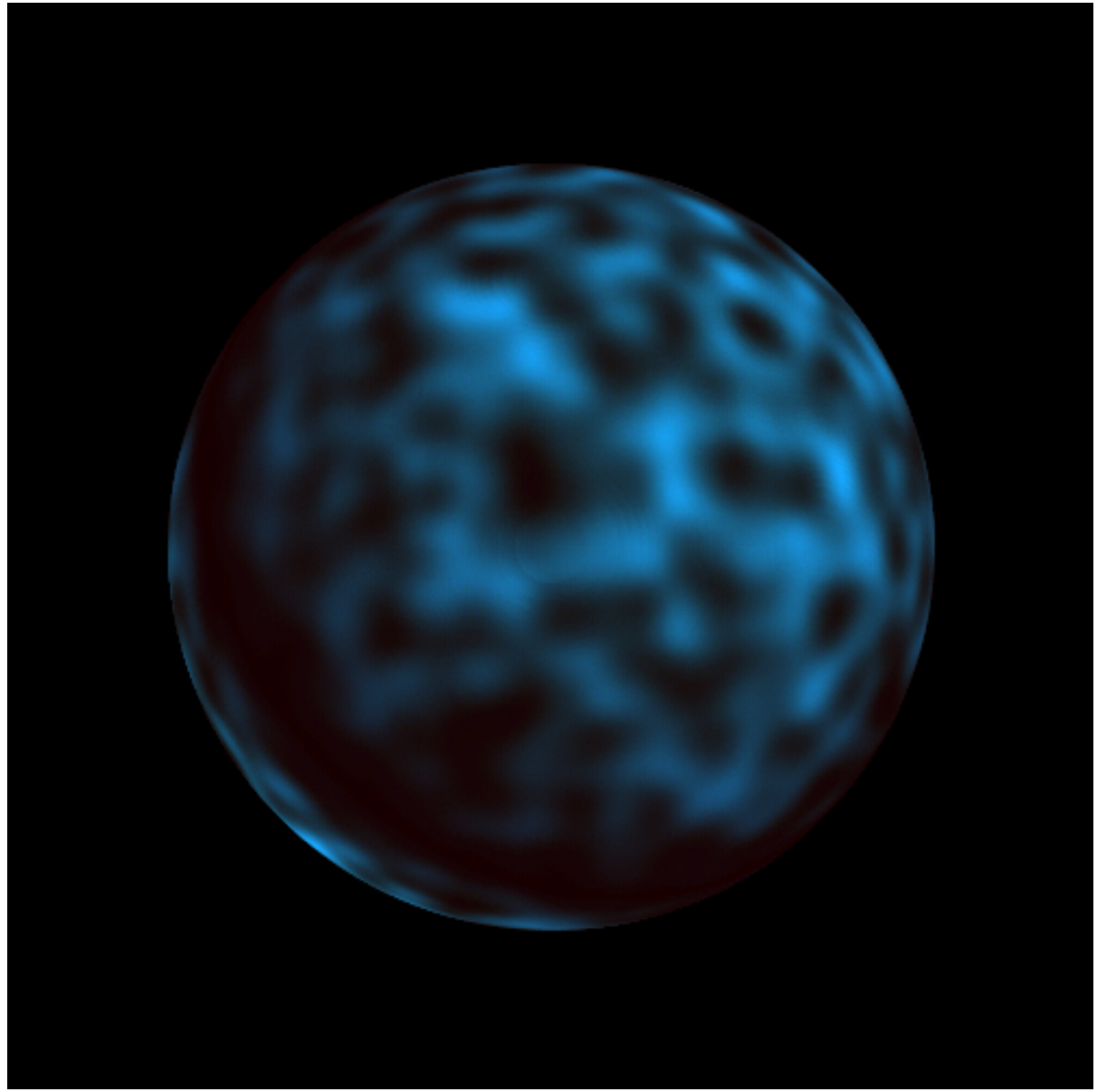
# Procedural textures

- Generate textures from procedures, such as Perlin noise
  - Based on using characteristics of noise occurring in nature (e.g. 1/f)
  - Effective use is based on creative use of the noise
- First (and most commonly) used in solid textures but need not be limited to solid textures



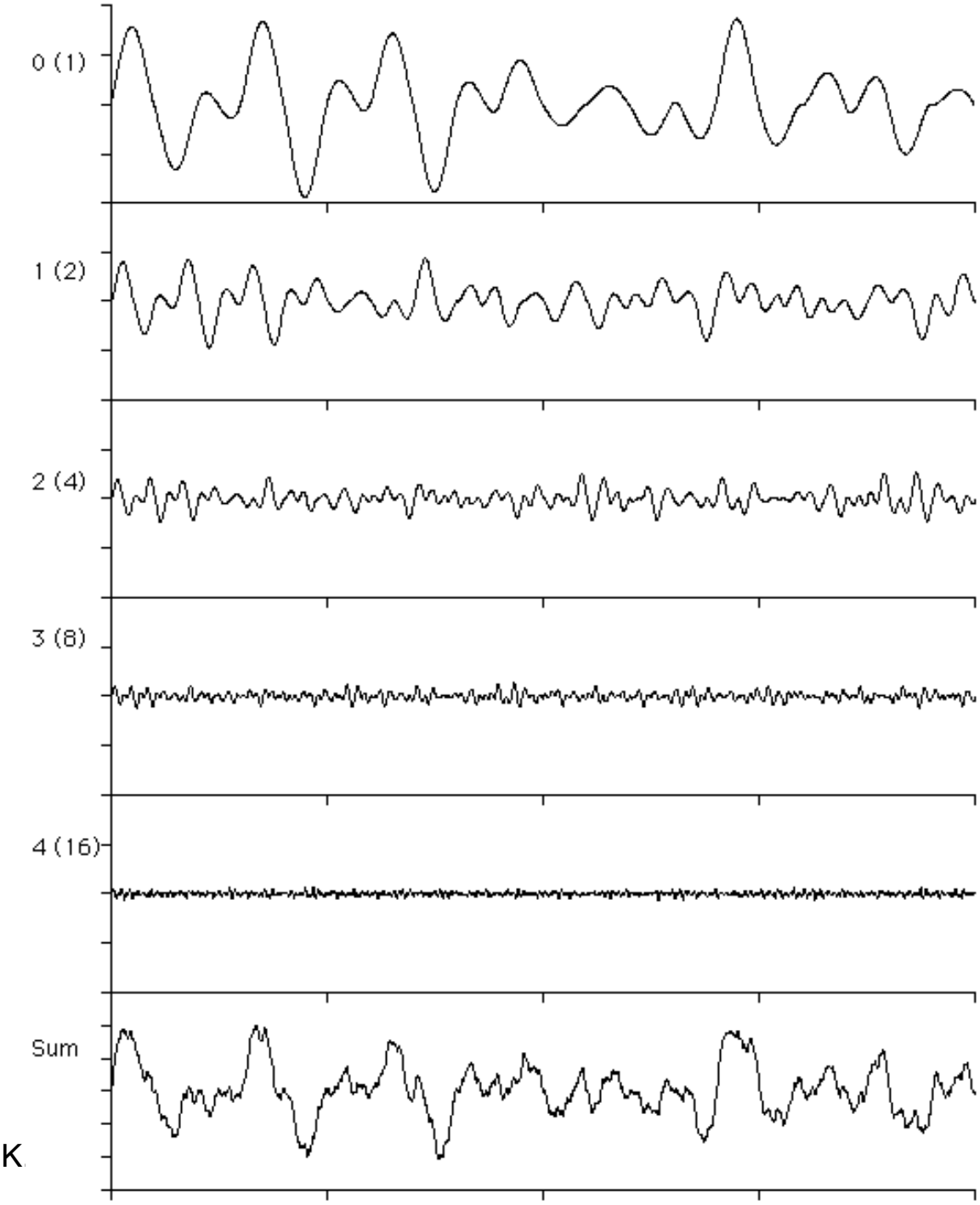
# Perlin noise

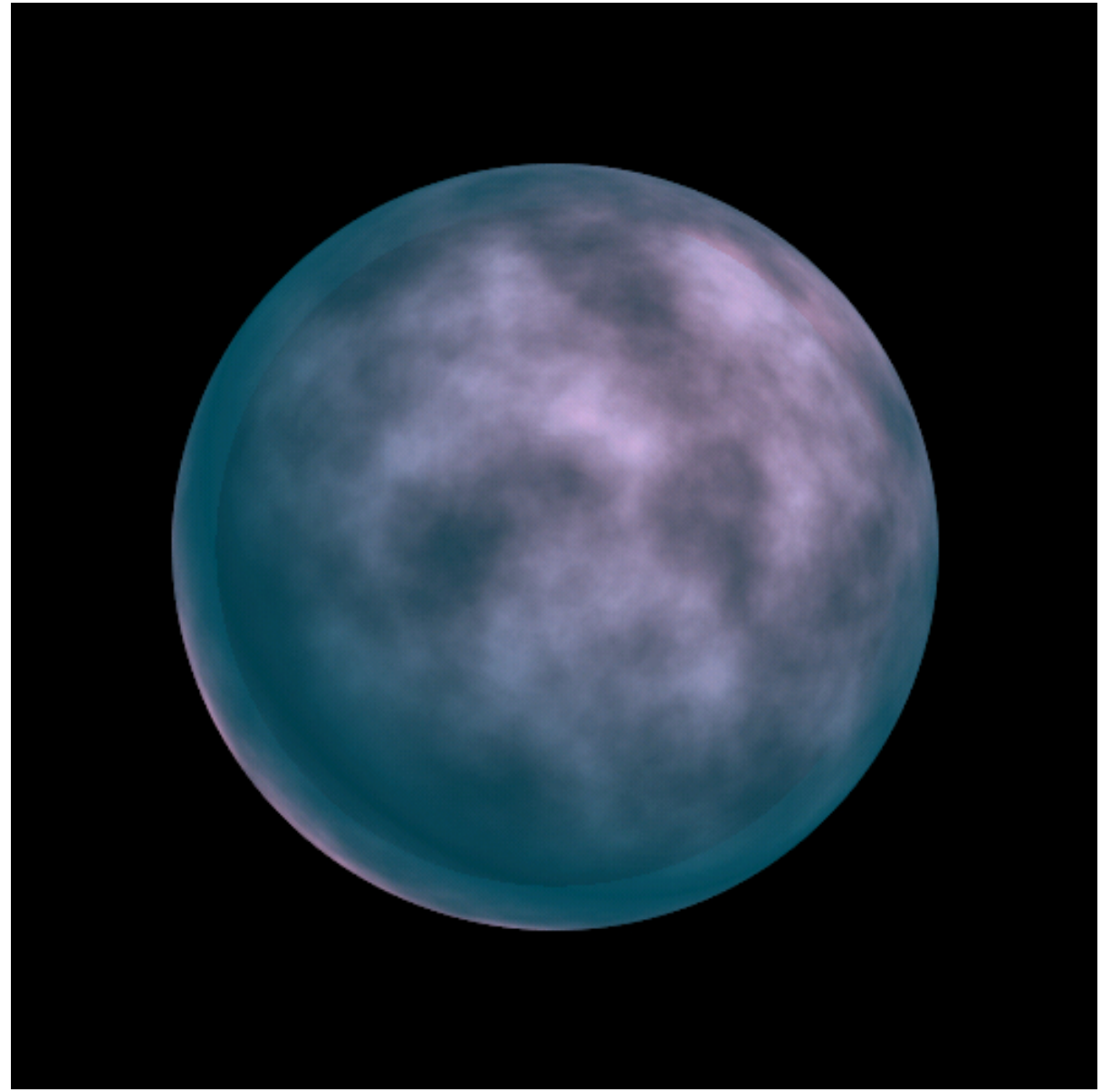
- Basis for all other noise
- Narrow band pass limit in frequency

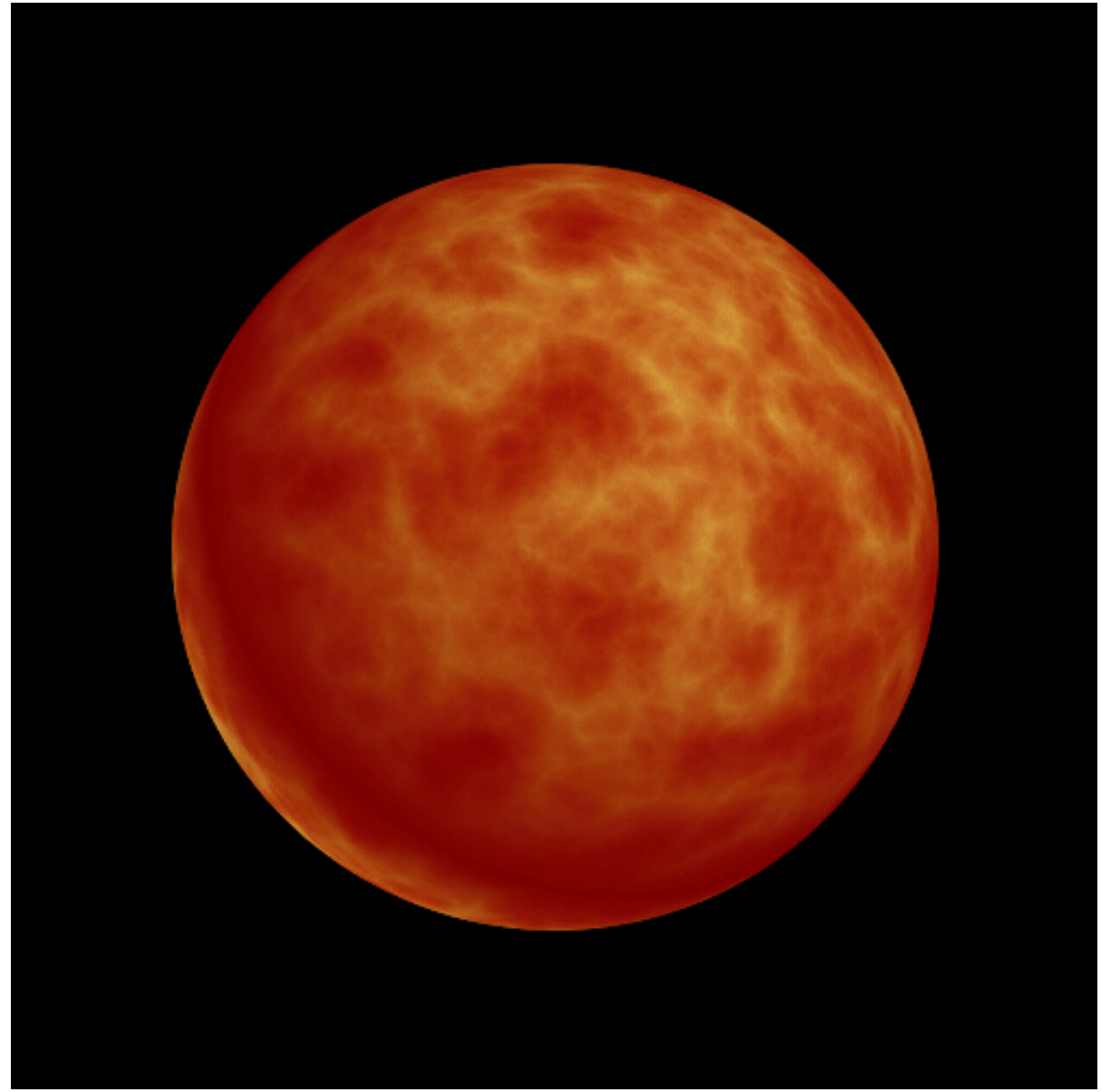


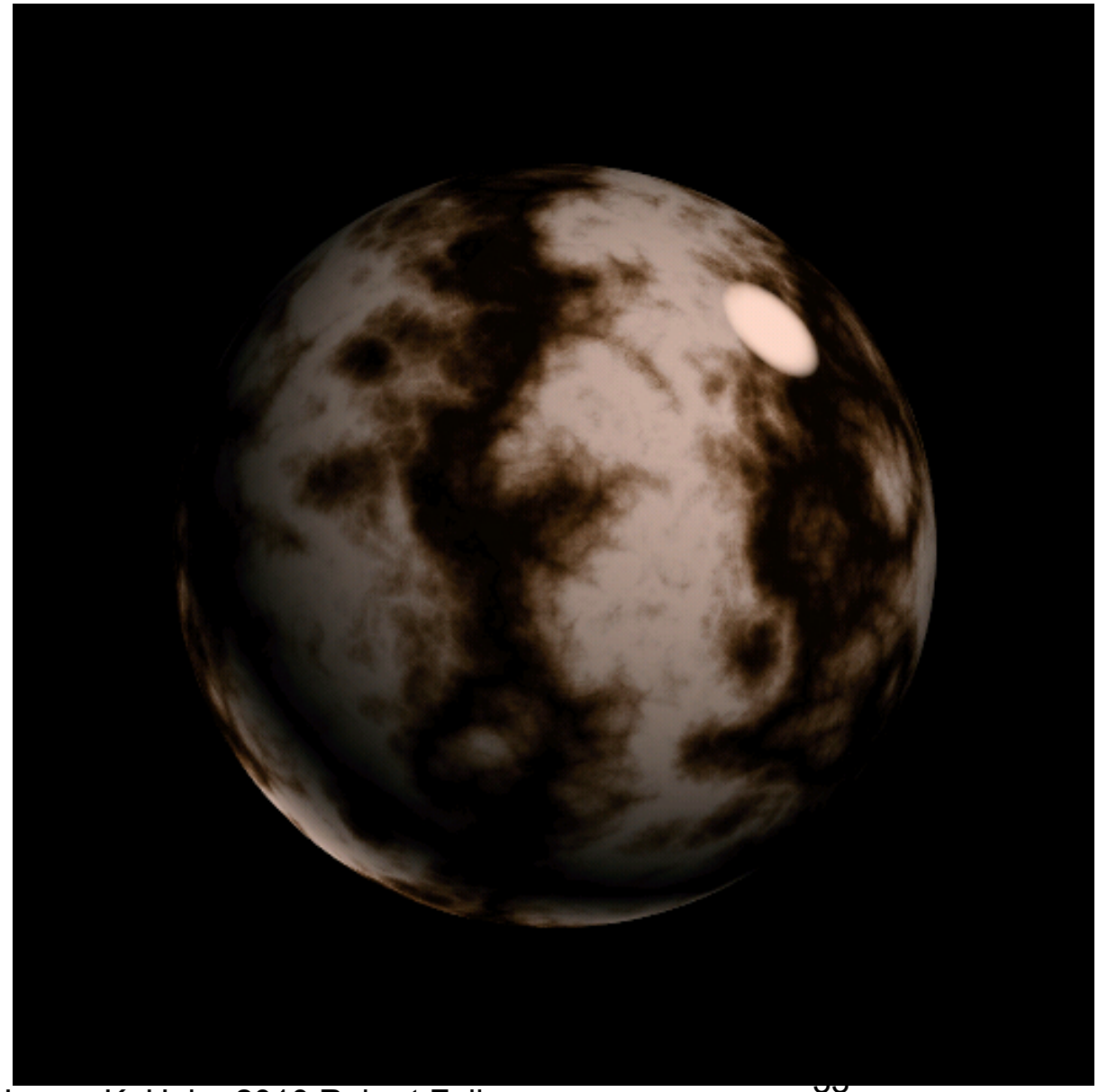
# Turbulence

- Add a number of copies of the noise
- Reduce amplitude and increase frequency
- Common choice: 1/2 amplitude and double frequency each time

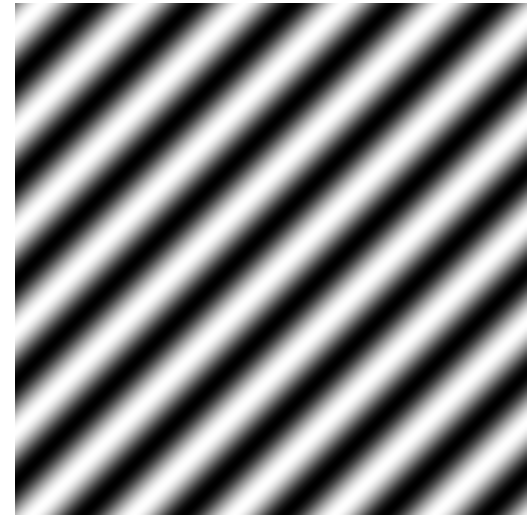








# Marble

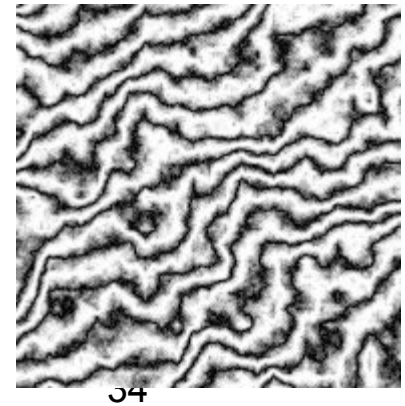


- Basic marble:

$$basic\_marble(x) = \text{marble\_color}(\sin(x))$$

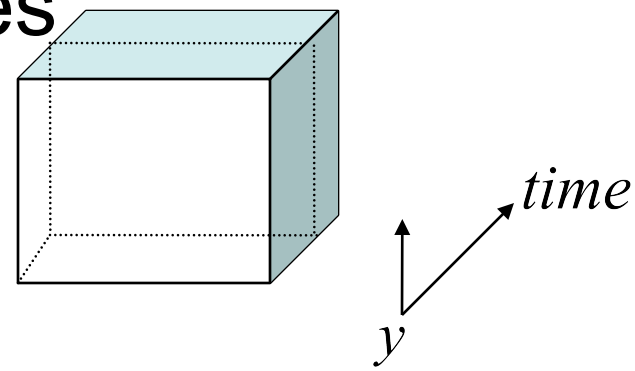
- Marble-color map scalar value to color (a palette) vector
- Marble:

$$marble(x) = \text{marble\_color}(\sin(x + \text{turbulence}(x)))$$



# Animating 3-D texture maps

- Moving plane in time axis causes general change in texture
- Moving plane in –y direction causes static texture to move upward
- Combination of two causes a varying texture to move upward



# OpenGL Textures

- Create image to be used as texture
- Name textures: `glGenTexture`
- Make a texture the current texture: `glBindTexture`
- Define a texture: `glTexImage2D`
- Map a texture onto an object: `glTexCoord*`
- Texture filter/wrap modes: `glTexParameteri`
- How the texture is used (combined): `glTexEnvi`
- Mip Mapping

# Naming Texture Objects

– Can create one or more named texture patterns then bind/activate it before being used on an object

```
GLuint texId;
glGenTexture( 1, &texId );
```

  - First argument is the number of textures to create
  - Second argument is the array of ids created by OpenGL

– Make a texture the current texture using

```
glBindTexture( GL_TEXTURE_2D, id );
```

# Define Texture Image

- `glTexImage2D( GL_TEXTURE_2D, level, GL_RGBA, w, h, border, format, type, *texels );`
  - `GL_TEXTURE_2D` : 2 dimensional texture
  - `level` : 0 indicates not a reduction of a larger texture array
  - `GL_RGBA` : indicates 4 elements for each texel (internal format)
  - `w` and `h` gives the width and height (power of 2 plus any borders)
  - `border` : 0 indicates no border around the texture image
  - `format` : how the pixel color value is stored (e.g. `GL_BGRA` indicate given in the order of B, G, R, A)
  - `type` : indicates pixel data type (e.g. `GL_UNSIGNED_BYTE`)
  - Last argument: pointer to image array data (e.g. data from image file). First pixel is lower-left corner of texture

# Converting a Texture Image

– If dimensions of image are not power of 2

```
gluScaleImage( format, w_in, h_in, type_in,
  *data_in, w_out, h_out, type_out,
  *data_out );
```

- *_in is for source image
- *_out is for destination image

– Image interpolated and filtered during scaling

# Mapping a Texture

- Based on parametric texture coordinates
- `glTexCoord*()` specified at each vertex
  - To assign full range of textures to a quadrilateral, assign parametric coordinates to each of the 4 vertices

```
glBindTexture( GL_TEXTURE_2D, id );

glBegin( GL_QUADS );
glTexCoord2f(0.0, 0.0);     glVertex3fv( p1 );
glTexCoord2f(1.0, 0.0);     glVertex3fv( p2 );
glTexCoord2f(1.0, 1.0);     glVertex3fv( p3 );
glTexCoord2f(0.0, 1.0);     glVertex3fv( p4 );
glEnd();
```

# Other ways to Specify Texture

- Use frame buffer as source of texture image

  `glCopyTexImage2D(…)`

  `glCopyTexImage1D(…)`

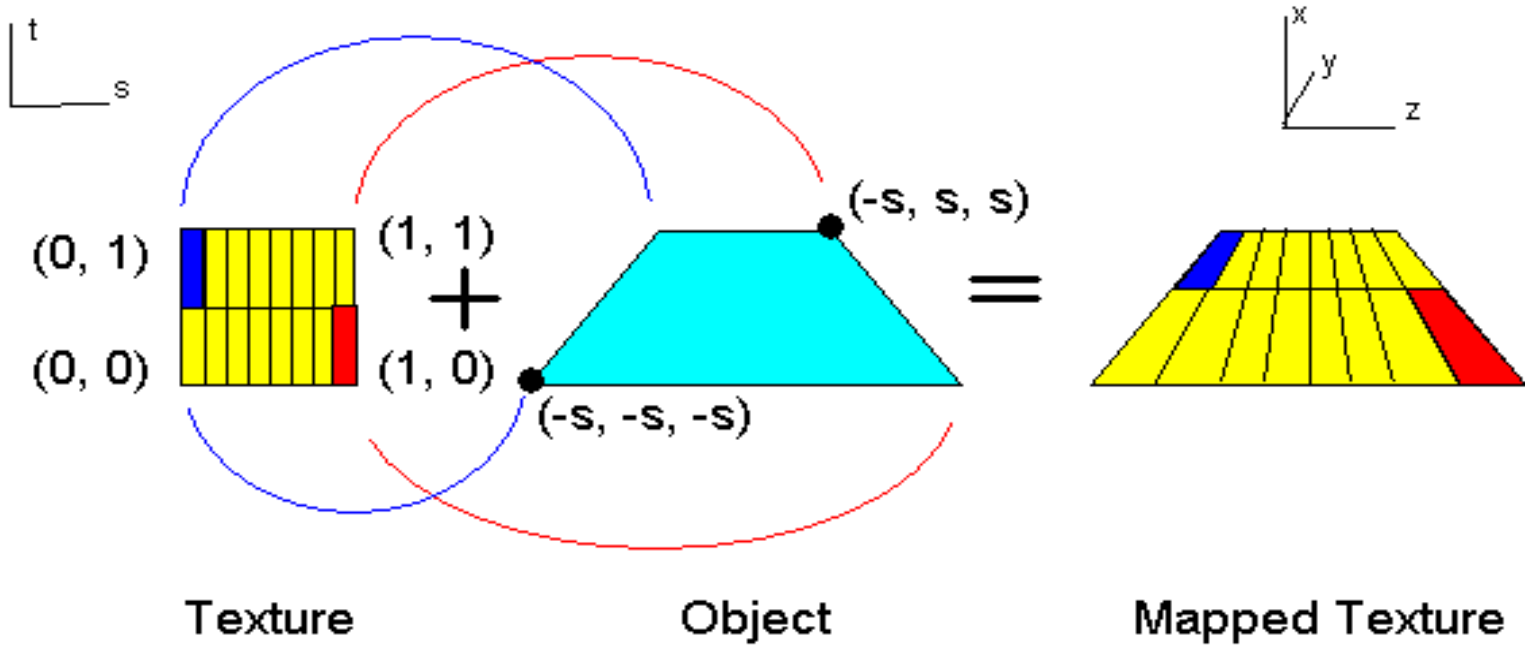
- Modify part of a defined texture

  `glTexSubImage2D(…)`

  `glTexSubImage1D(…)`

- Use frame buffer as source for part of texture image

  `glCopyTexSubImage2D(…)`, etc.




E.R. Bachmann & P.L. McDowell

# Texture Application Methods

- Wrap modes
  - clamping or repeating
- Texture mapping mode
  - how to mix primitive's color with texture's color:

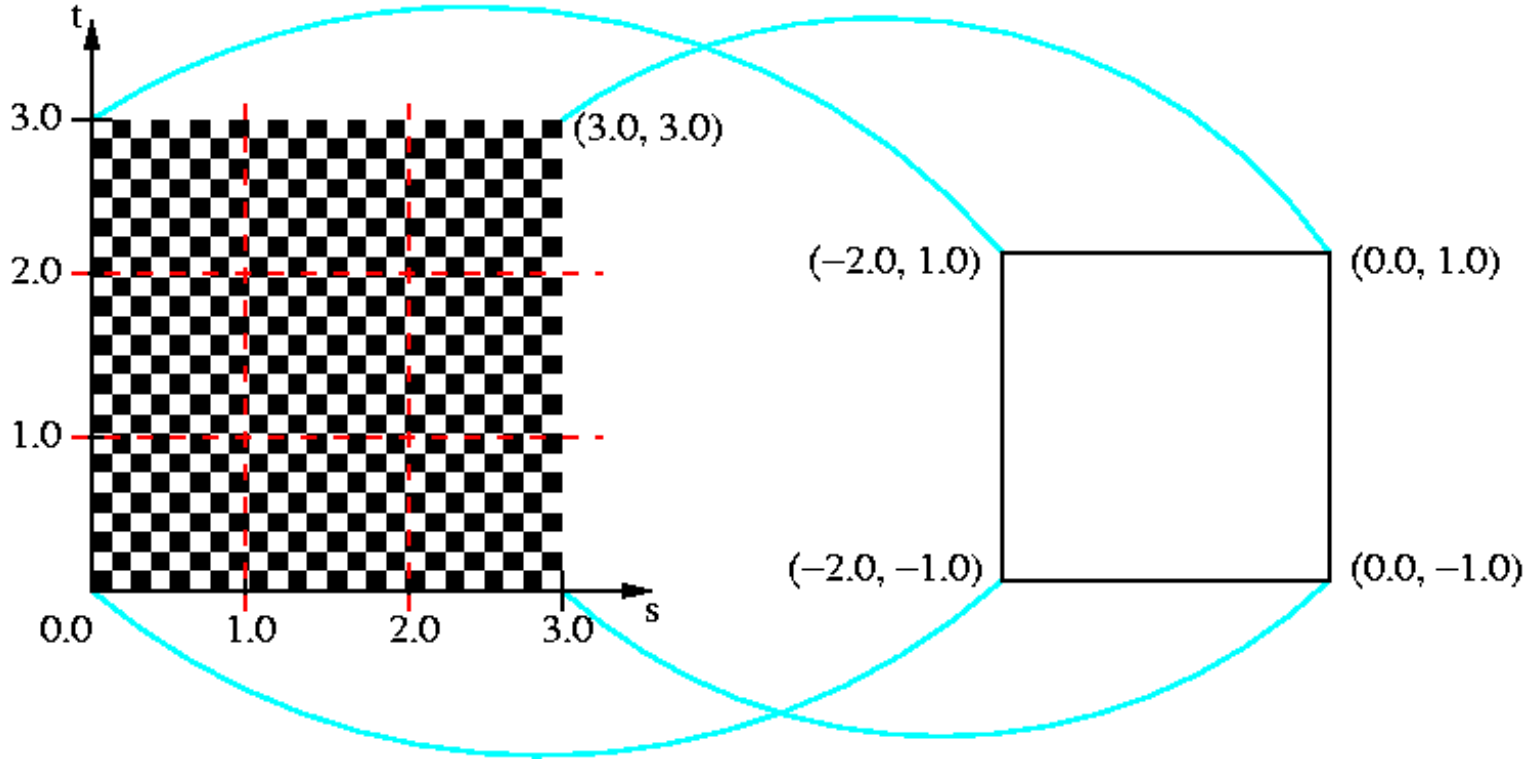
    blend, modulate, or replace texels
- Filter modes
  - Minification (zoom out) or magnification (zoom in)
  - special mipmap minification filters

# Wrapping Mode

– When coordinate values in texture space outside range from 0 to 1, replicate pattern by

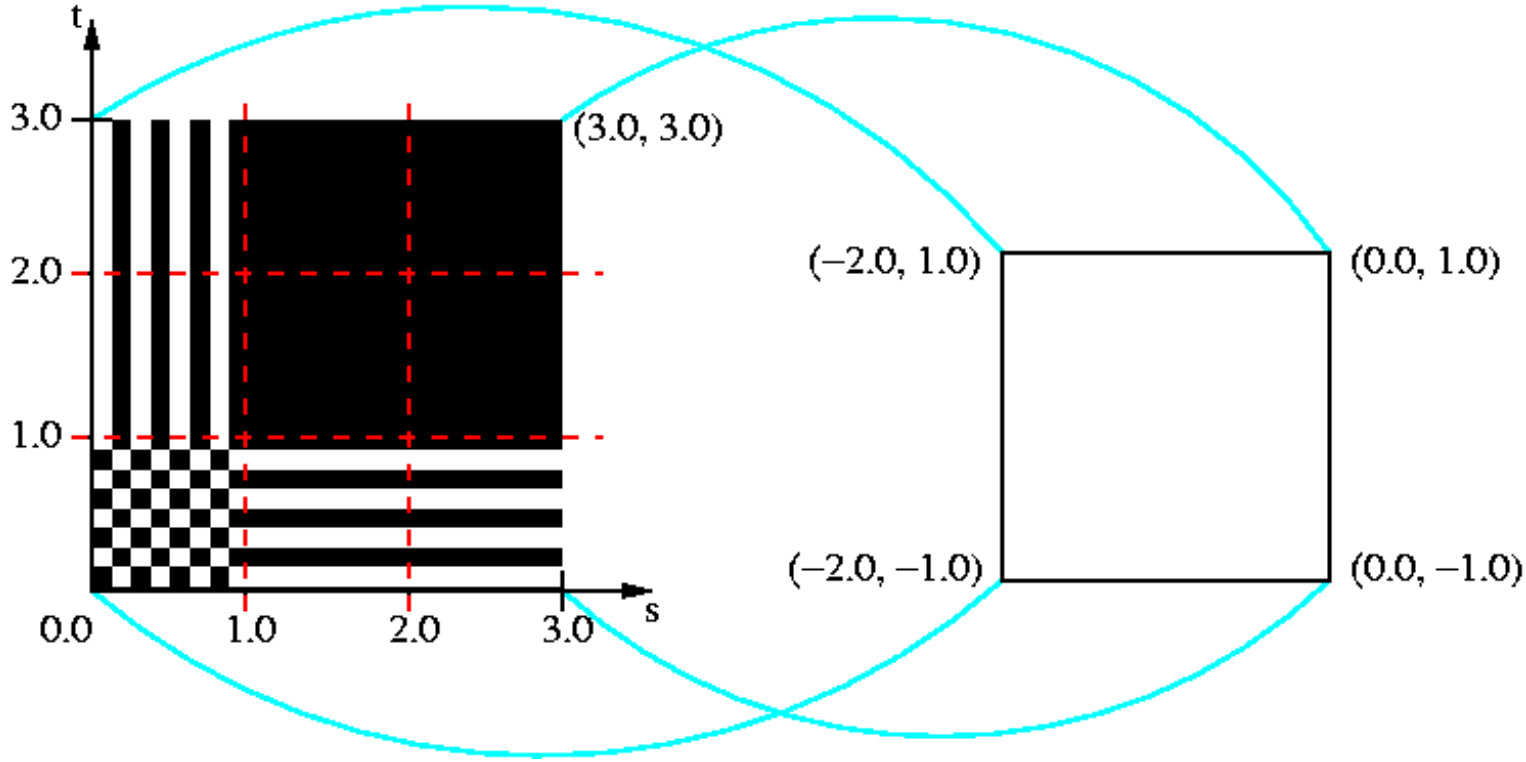
```
glTexParameteri ( GL_TEXTURE_2D, texWrapCoord,
  texWrapMode );
texWrapCoord:
  GL_TEXTURE_WRAP_S
  GL_TEXTURE_WRAP_T
texWrapMode:
  GL_REPEAT (default)
  GL_CLAMP_TO_EDGE
  GL_CLAMP_TO_BORDER
```

- Repeating texture (s and t parameters >1)




http://www.cs.uregina.ca

- Clamped texture (s and t parameters >1)




http://www.cs.uregina.ca

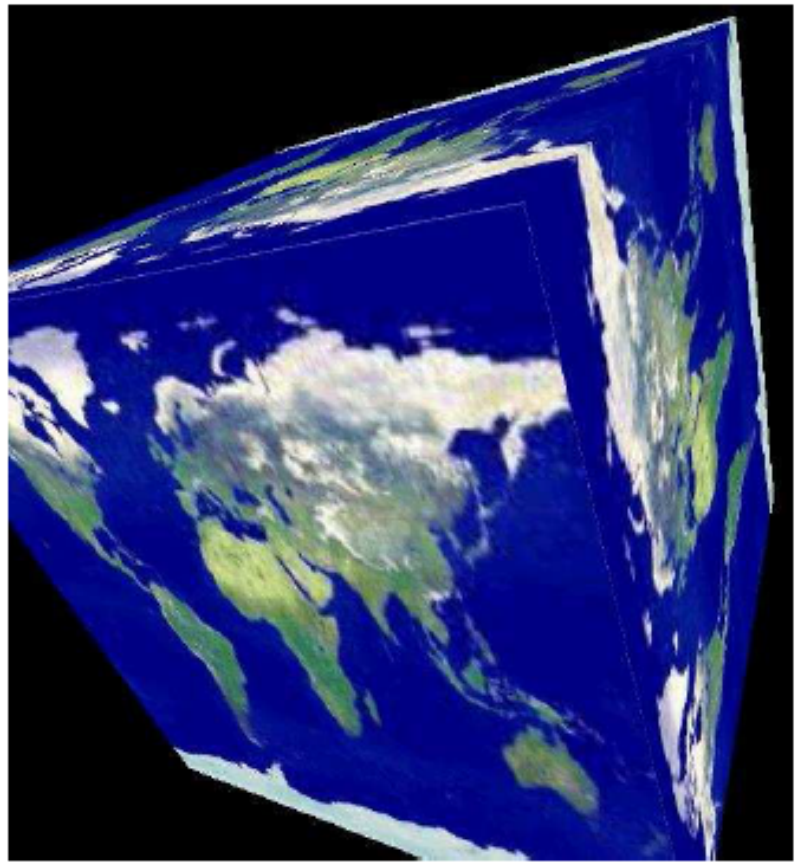
# Texture Mapping Mode

– Controls how texture is applied

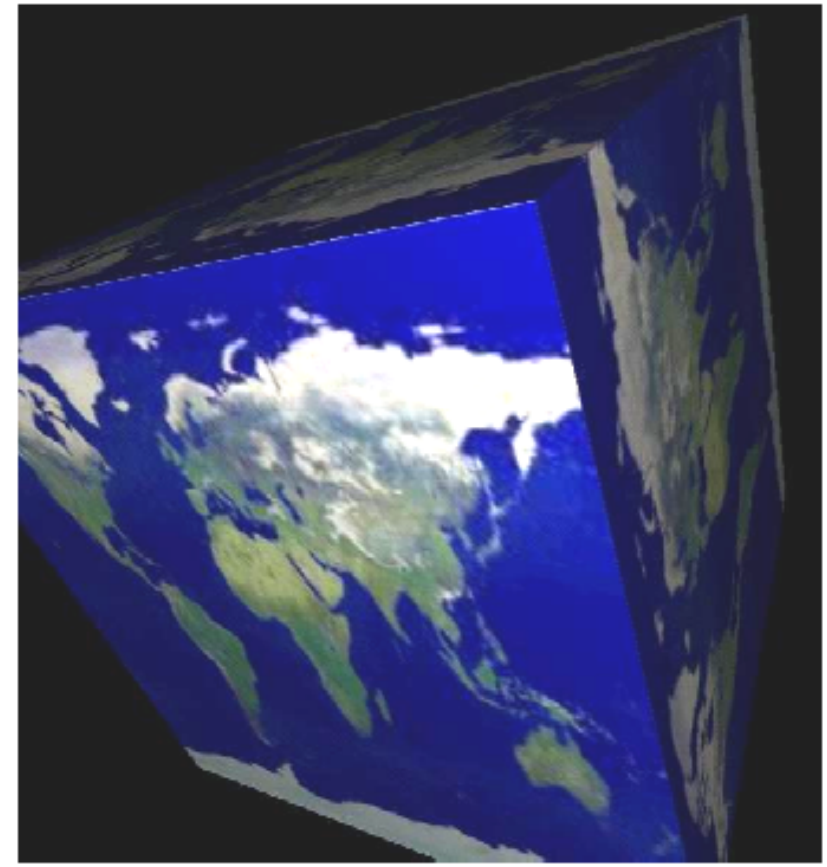
```
glTexEnvi( GL_TEXTURE_ENV,
GL_TEXTURE_ENV_MODE, applicationMethod );
```

– `applicationMethod`

- `GL_MODULATE`: current (lighting) value multiplied by texture map value (for default lighting - not shaders)
- `GL_REPLACE`: current value is replaced by texture map value (for default lighting - not shaders)



Replace



Modulate

# Mip mapped Textures

- Mipmap allows prefiltered texture maps of decreasing resolutions
- Declare mip map level during texture definition
  - Remember we used level 0 to indicate no reduction in image
  - Each level reduces the size of the texture image by a factor of 1/2

```
glTexImage*D( GL_TEXTURE_*D, level, … )
```

- Mipmap builder routines:
  - `gluBuild[2D,3D]Mipmaps( … );`
  - `glGenerateMipmap(GLenum target );`
  - `glTexParameteri(GL_TEXTURE_2D, GL_GENERATE_MIPMAP, GL_TRUE);`
- Filter mode used to determine how the mip map is used

# Filter Modes

Example:

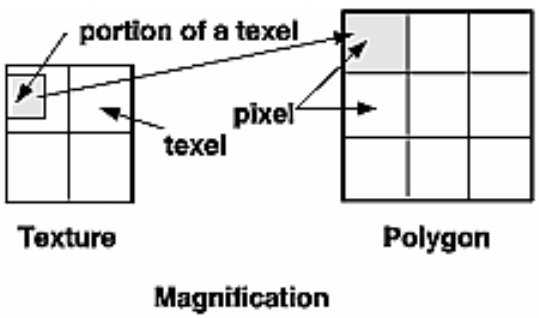
```
glTexParameteri( GL_TEXTURE_2D, type, mode );
```

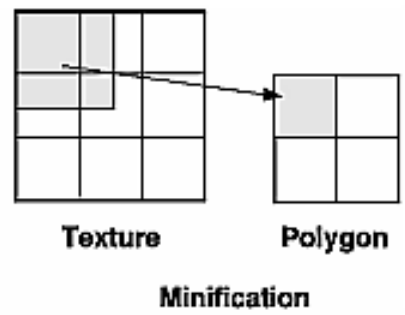
type:

- GL_TEXTURE_MAG_FILTER
- GL_TEXTURE_MIN_FILTER

mode:

- GL_NEAREST
- GL_LINEAR
- GL_NEAREST_MIPMAP_NEAREST
- GL_NEAREST_MIPMAP_LINEAR
- GL_LINEAR_MIPMAP_NEAREST
- GL_LINEAR_MIPMAP_LINEAR



- `GL_NEAREST`: point sampling
  - Pick texel nearest the pixel location
- `GL_LINEAR`: bilinear interpolation
  - Interpolation of 4 nearest texel (either within a level or between levels of mip maps)
- `GL_N/L_MIPMAP_N/L`: only for minification
  - First N/L indicates what to do within a level
  - Second N/L indicates what to do between levels
  - For example: `GL_NEAREST_MIPMAP_LINEAR`
    Interpolate nearest texels from two mipmap levels

# Next: Global Illumination