Texture Mapping

Texture

- So far, surfaces are drawn either with
  - Uniform color
  - Varying shades of the same color
  - Dull (diffuse) or shining (specular)
- Real surfaces have *colors* and *patterns*
  - Wood
  - Brick wall
  - Book cover
  - Grass, etc.

Texture (cont.)

- Repetitive patterns obeying certain rules
- Man-made texture
  - Texels + placement rules
  - Checkboard, brickwall, wrapping paper
- Natural texture
  - Statistical properties
  - Water surface, grass, sky

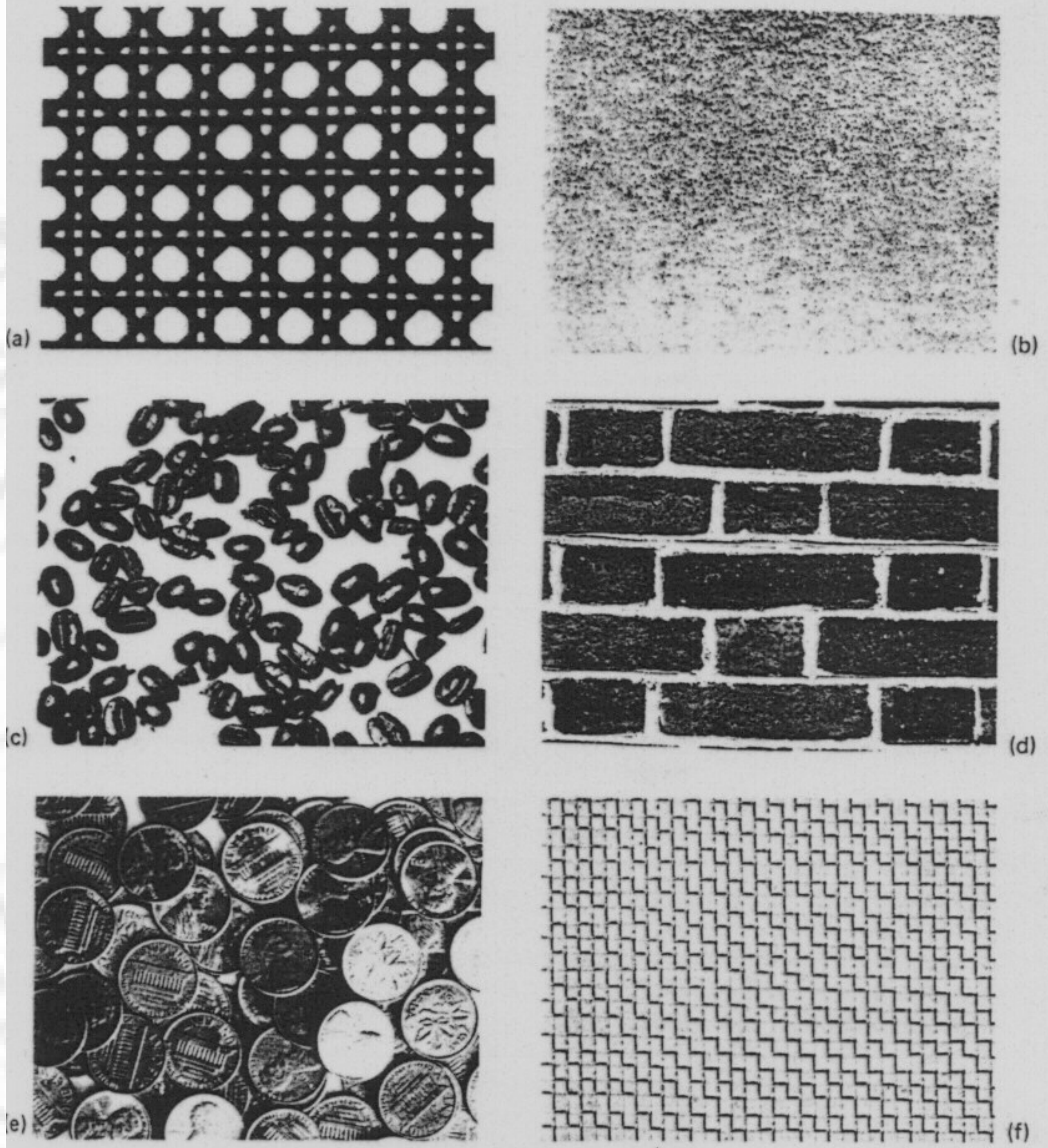
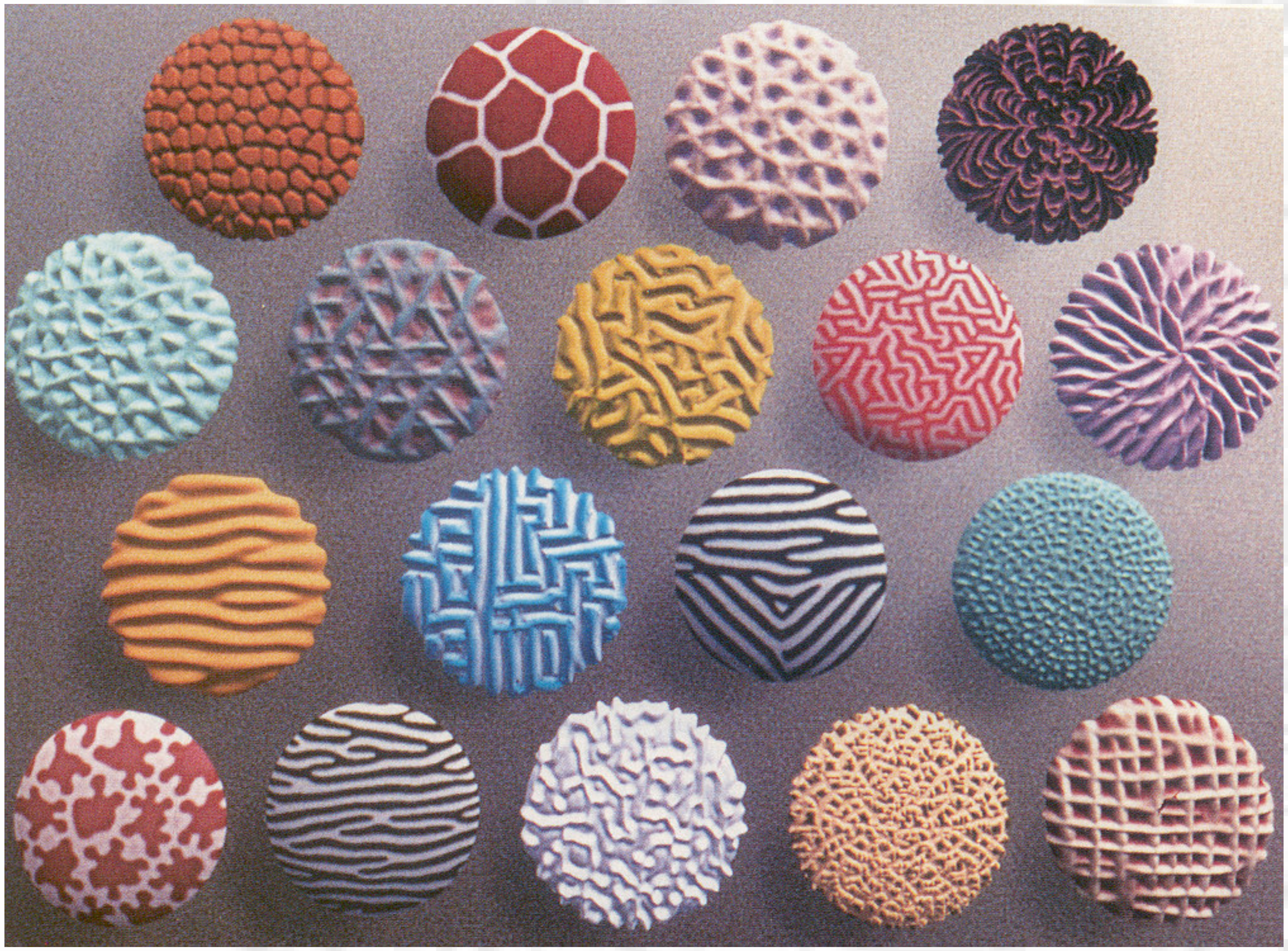


Fig. 6.1 Six examples of texture. (a) Cane. (b) Paper. (c) Coffee beans. (d) Brick wall. (e) Coins. (f) Wire braid.

Texture (cont.)

- Generation
  - Particularly useful in 3D
  - Mathematical properties understood (e.g., stripes on Zebra, Giraffe)
- Analysis
  - What type of texture is this?
  - How is the textured surface oriented?
- Mapping
  - 1D, 2D, 3D patterns, digitized/synthesized
  - To improve rendering realism

2D Procedural Textures



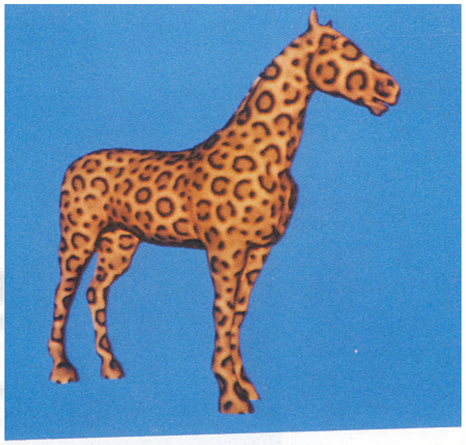


Figure 4: Leopard-Horse

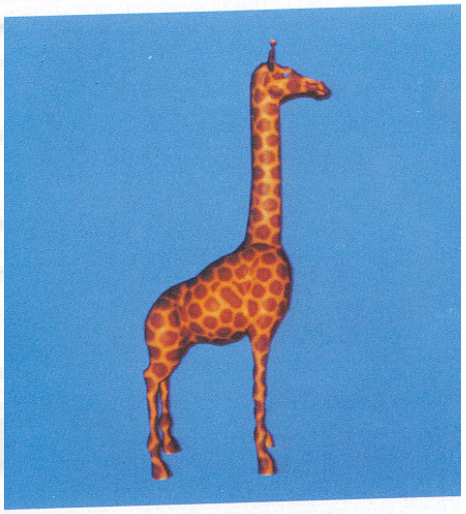
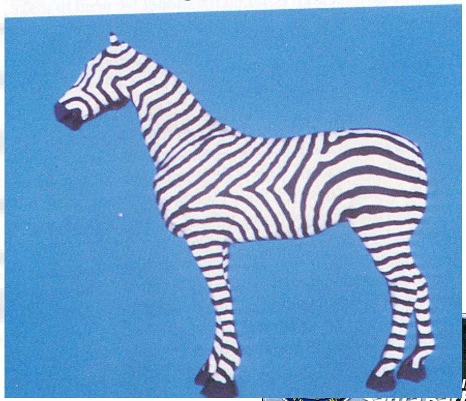
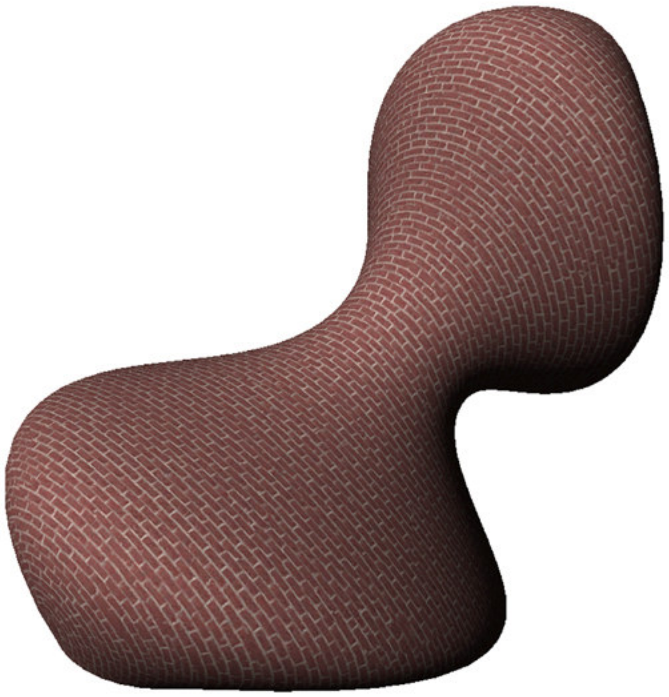


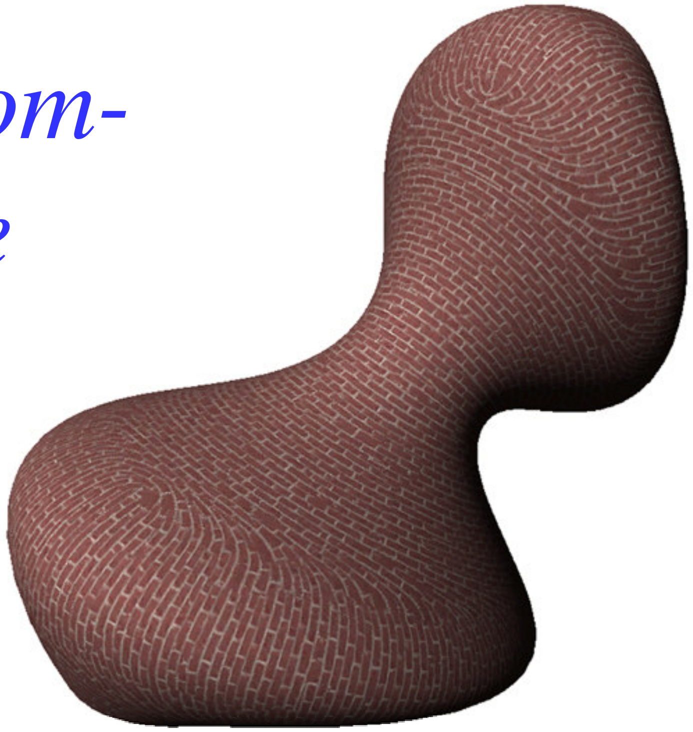
Figure 5: Giraffe

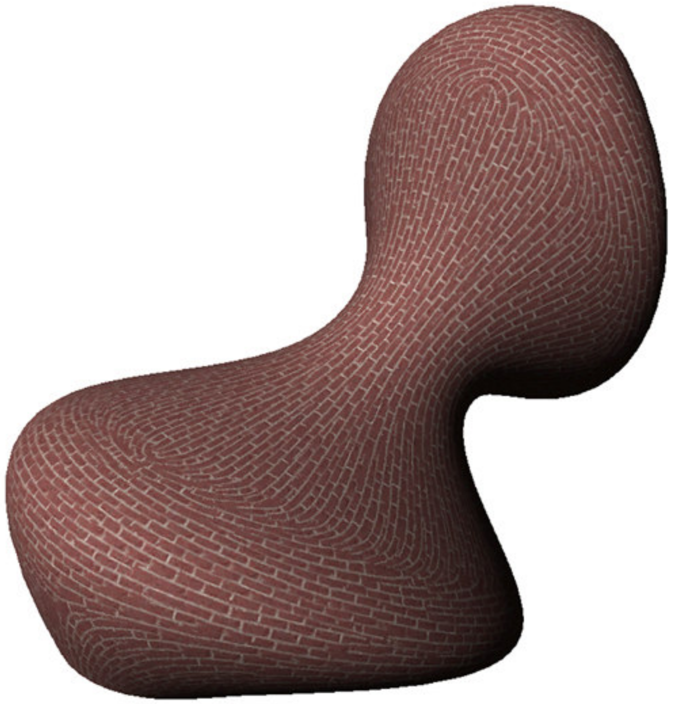


3D Procedural Textures

Shape-from-texture



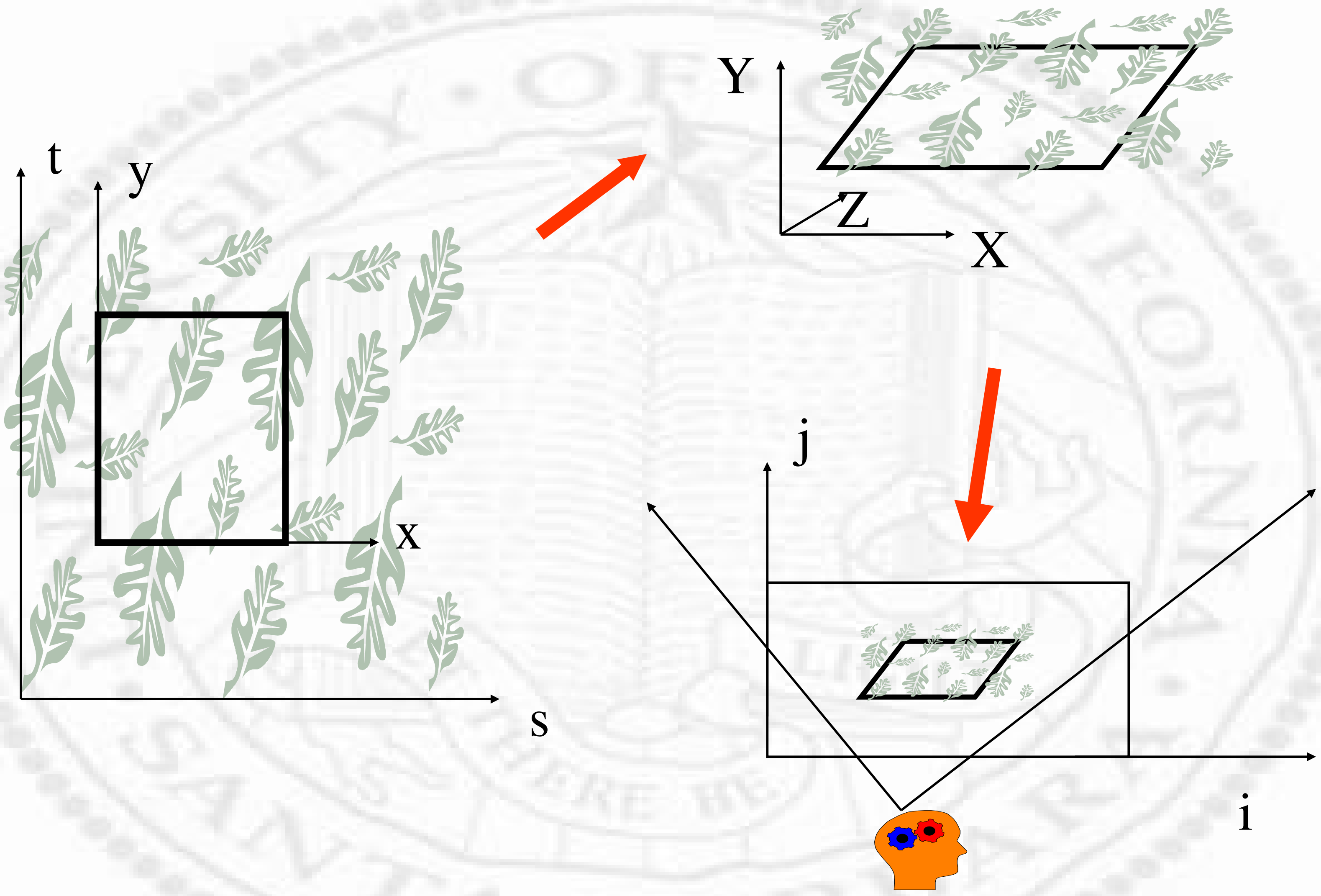




Theoretical Consideration Texture mapping

- *Geometry* mapping
  - Where does the texture go physically?
  - “cookie-cutter” problem
- *Appearance* mapping
  - How will texture appear? As a decal? Modulate lighting? Modulate surface structure?

Geometry Mapping (2D)



Geometry Mapping (2D)

- There are a number of coordinates
  - Object coordinate (*x*,*y*)
  - Texture coordinate (*s*,*t*)
  - World coordinate (X,Y,Z)
  - Image coordinate (*i*,*j*)

Geometry Mapping (2D)

- Object (x,y) & Texture (s,t) coordinates are related by:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} s \\ t \end{bmatrix} + \begin{bmatrix} T_s \\ T_t \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & 0 & T_s \\ c & d & 0 & T_t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s \\ t \\ 0 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix} = M_{obj \leftarrow texture} \begin{bmatrix} s \\ t \\ 0 \\ 1 \end{bmatrix}$$

Geometry Mapping (2D)

- Object & World coordinates are related by:

$$\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = M_{world \leftarrow obj} \begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix}$$

Geometry Mapping (2D)

- World & Image coordinates are related by:

$$\begin{bmatrix} i \\ j \\ 0 \\ 1 \end{bmatrix} = M_{image \leftarrow viewer} M_{viewer \leftarrow world} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} M_{image \leftarrow world} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} i \\ j \\ 0 \\ 1 \end{bmatrix} = M_{image \leftarrow world} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Geometry Mapping (2D)

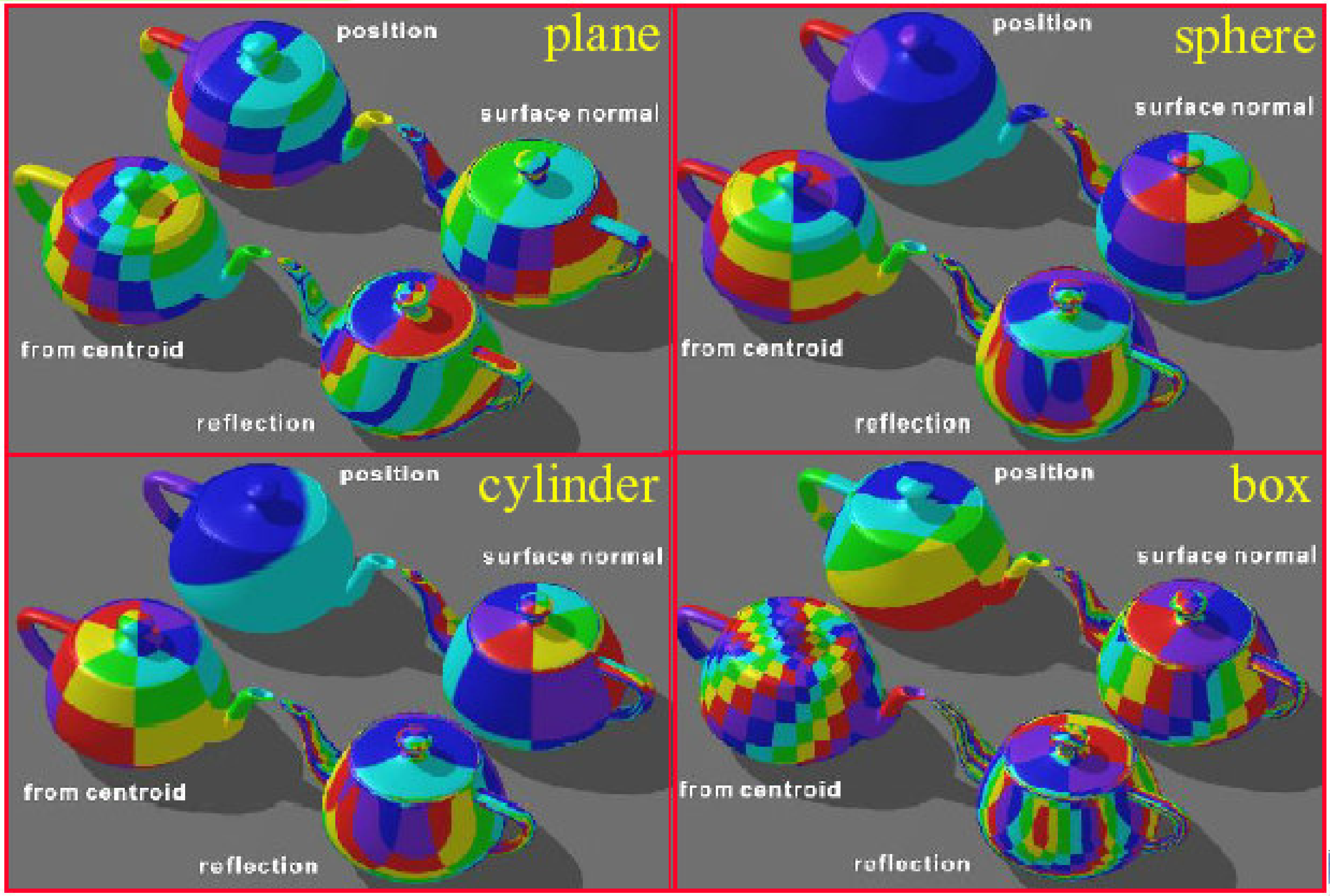
- Putting it all together:
  - Texture transforms the way the underlying object does, with an added transform to map from texture coordinates to object coordinates

$$\begin{bmatrix} i \\ j \\ 0 \\ 1 \end{bmatrix} = M_{image \leftarrow eye} M_{eye \leftarrow world} M_{world \leftarrow obj} M_{obj \leftarrow texture} \begin{bmatrix} s \\ t \\ 0 \\ 1 \end{bmatrix}$$

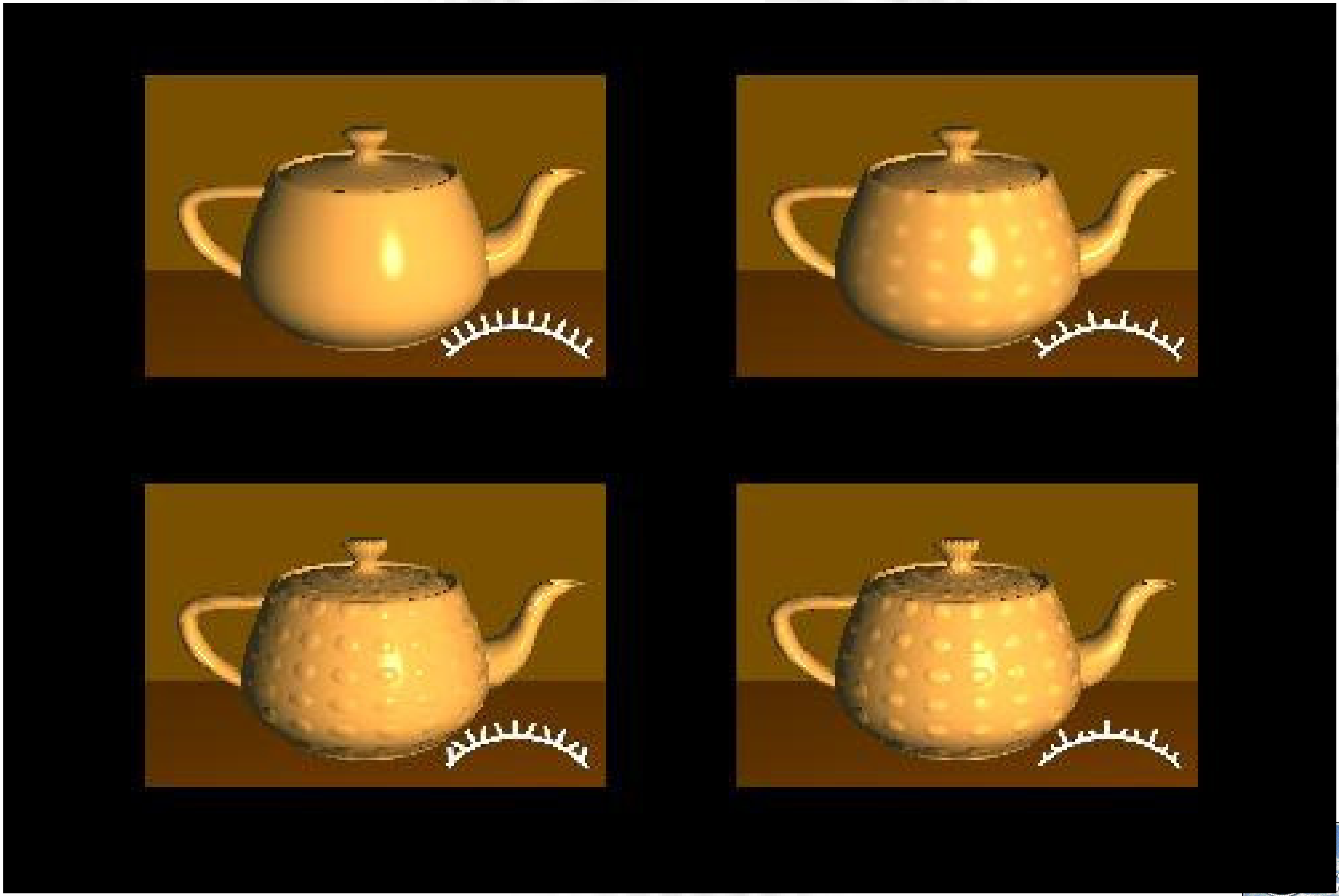
Appearance Mapping

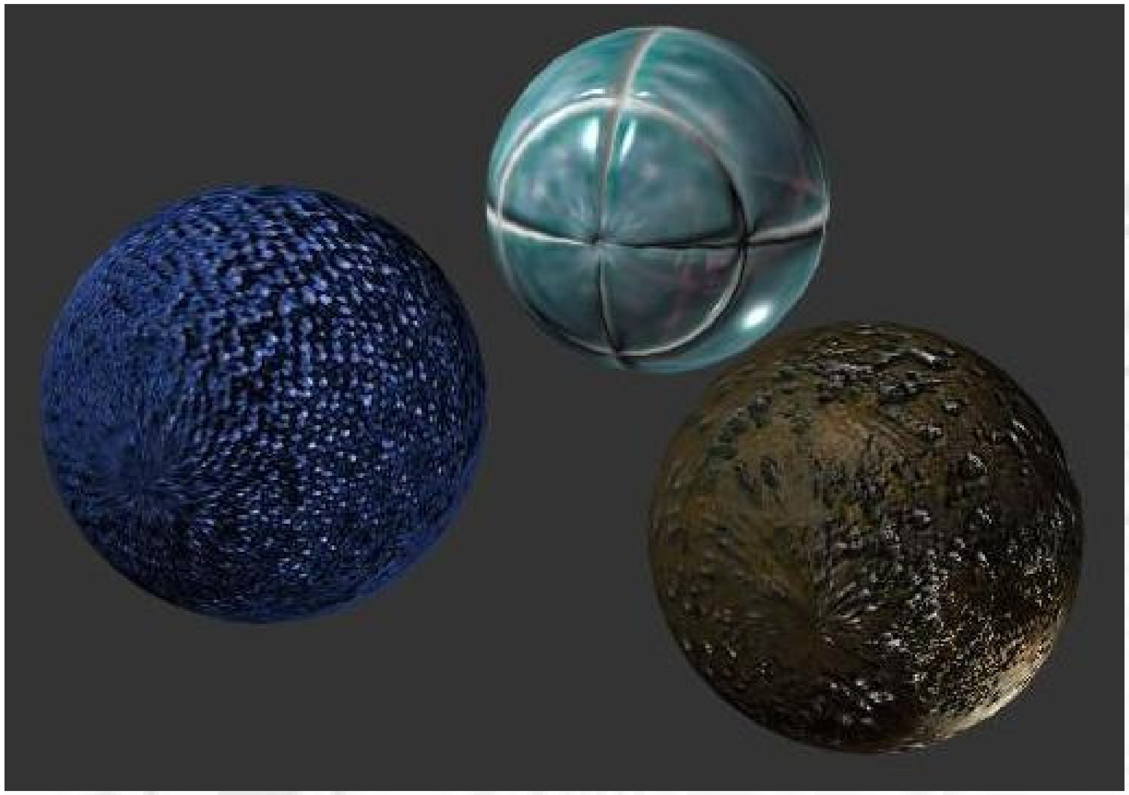
- Many choices
  - Use as a decal
    - Replace the object color
  - Use as a modulator
    - Modulate Alpha component
    - Modulate Luminance component
    - Modulate Color components
    - Module surface structure
    - Module surface orientation
    - Etc.

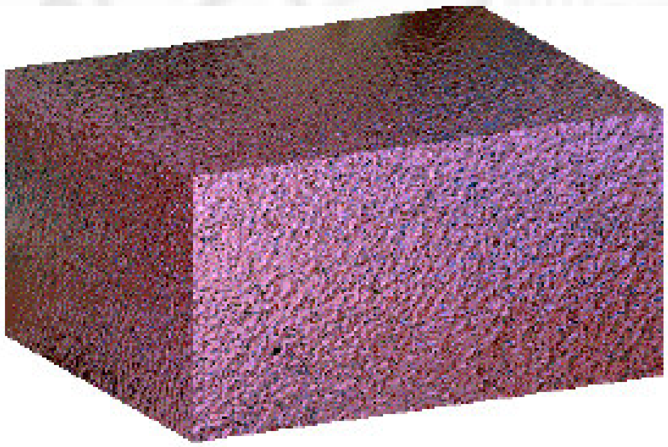
Use as Decals

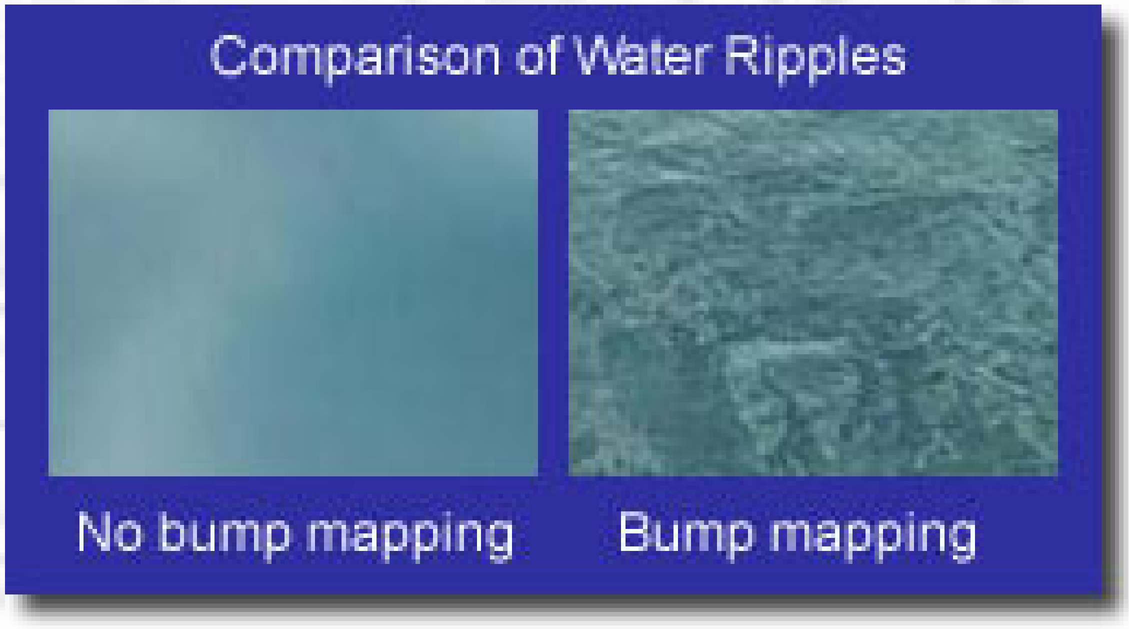


Bump & Displacement Mapping





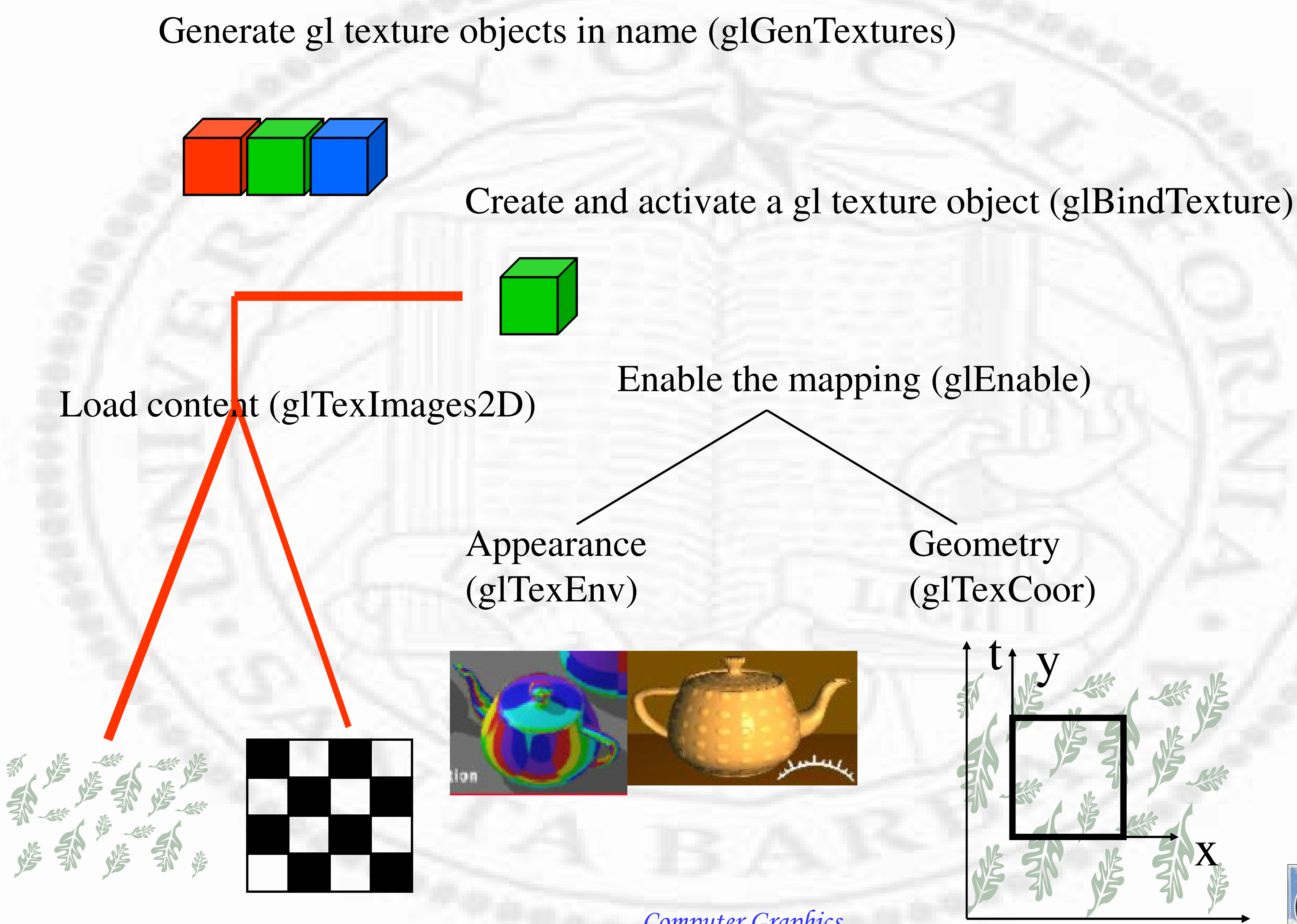



Comparison of Water Ripples
No bump mapping
Bump mapping

OpenGL Texture

- Create texture objects – *placeholder, name only*
  - glGenTextures
- Bind a texture to the object – *create and make it active*
  - glBindTexture
- *Load the content*
  - glTexImage2D
- Enable texture mapping – *make it happen*
  - glEnable
- Indicate how texture should be applied –(*appearance* mapping)
  - glTexenv
- Draw w. both object and texture coordinates – (*geometry* mapping)
  - glTexCoor

OpenGL Texture (cont.)


Generate gl texture objects in name (glGenTextures)
Create and activate a gl texture object (glBindTexture)
Load content (glTexImages2D)
Enable the mapping (glEnable)
Appearance (glTexEnv)
Geometry (glTexCoor)
t
y
x

Create Texture Objects

- glGenTextures(GLsizei *n*, GLuint **texturenames*)
  - Generate *n* OpenGL texture objects and return the indices in the supplied array
  - Texture objects have default properties (e.g., min & max filters, wrapping modes, border color) that can be assumed
  - Multiple texture objects can be put in a working set as resident for more efficient operations

Create Texture Objects (cont.)

```
…
Gluint texName;
glCenTextures(1,&texName);
…


…
GLuint texName[3];
glGenTextures(3,texName);
…
```

Bind Texture Objects

- glBindTexture(GLenum target, GLuint texturename)
  - Target: GL_TEXTURE_1D or GL_TEXTURE_2D
  - The first time: particular texture object is created
    - Create an empty texture object
      - Subsequent glTexImage*() refers to this one
    - Note that the real texture image data is missing
      - To be filled by, e.g., glTexImage*()
  - Later: particular texture object becomes active

```
…
glBindTexture (GL_TEXTURE_2D, texName[0]);
…
```

Load Texture Content

- void glTexImage2D(GLenum *target*, Glint *level*, Glint *internalFormat*, GLsizei *width*, GLsizei *height*, GLint *border*, GLenum *format*, GLenum *type*, const GLvoid **texels*)
  - *target*: GL_TEXTURE_2D
  - *level*: 0 (usually), >=0 (mipmap)
  - *internalFormat*: GL_ALPHA, GL_LUMINANCE (1), GL_LIMINANCE_ALPHA (2), GL_INTENSITY, GL_RGB (3), GL_RGBA (4)
  - *width, height*: 2^m x 2^n
  - *border*: 0 or 1

Load Texture Content (cont.)

- *format*: GL_COLOR_INDEX, GL_RGB, GL_RGBA, GL_RED, GL_GREEN, GL_BLUE, GL_ALPHA, GL_LUMINANCE, GL_LUMINANCE_ALPHA
- *type*: GL_BYTE, GL_UNSIGNED_BYTE, GL_SHORT, GL_UNSIGNED_SHORT, GL_INT, GL_UNSIGNED_INT, GL_FLOAT
- *pixels*: pointers to data
- Difference between (external)*format* and *internalformat*
  - *Format* specifies how images are *stored*
  - *Internalformat* specifies how images should be *used*
  - E.g., you can have a RGB format images and use only the R component

Enable Texture Mapping

- glEnable(GL_TEXTURE_2D)
  - Allow texture mapping computation
  - Affect later primitives until turned off with glDisable()

Appearance Mapping

- Void glTexEnv{if}(GLenum target, GLenum pname, TYPE param)
  - target: GL_TEXTURE_ENV
  - pname: GL_TEXTURE_ENV_MODE
  - param: GL_DECAL, GL_REPLACE, GL_MODULATE, GL_BLEND

```
…
glTexEn□□(GL_TEXTURE_EN□,
    GL_TEXTURE_EN□_□ □DE, GL_DEC□L);
…
```

Geometry Mapping

- glBegin(GL_QUADS);
- glTexCoord2f(0.0,0.0); glVertex3f(-2.0,-1.0,0.0);
- glTexCoord2f(0.0,1.0); glVertex3f(-2.0, 1.0,0.0);
- glTexCoord2f(1.0,1.0); glVertex3f(0.0, 1.0,0.0);
- glTexCoord2f(1.0,0.0); glVertex3f(0.0,-1.0,0.0);
- glEnd();

Geometry Mapping

- Alternate texture coordinate and vertex coordinate bind one to the other
- Texture coordinates always go from 0 to 1 in *s* and 0 to 1 in *t*
- Vertex coordinates can be anything

Putting it altogether

```
□in□lude □GL□glut□□□

□□    Create □□e□□er□□ard texture    □□
□de□ine □□e□□□mage□ idt□ □□
□de□ine □□e□□□mage□ eig□t □□
GLu□□te □□e□□□mage[□□e□□□mage□ idt□][□□e□□□mage□ eig□t][3];
GLuint texName;
□□id ma□eC□e□□□mage(□□id)
□
   int i, □, □;

   □□r (i □ 0; i □ □□e□□□mage□ idt□; i□□) □
      □□r (□□ 0; □□ □□e□□□mage□ eig□t; □□□) □
         □ □ ((((i&0x□)□□0)□((□&0x□))□□0))□2□□;
         □□e□□□mage[i][□][0] □ (GLu□□te) □;
         □□e□□□mage[i][□][1] □ (GLu□□te) □;
         □□e□□□mage[i][□][2] □ (GLu□□te) □;
      □
   □
□
```

```
□□id m□init(□□id)
□
   glClearC□l□r (0□0, 0□0, 0□0, 0□0);
   glEna□le(GL_DE□T□_TE□T);
   glDe□t□□un□(GL_LE□□);

   ma□eC□e□□□mage();
   gl□ixel□t□rei(GL_UN□□C□_□L□GN□ ENT, 1);
   glGenTextures(1, &texName);
   glBindTexture(GL_TEXTURE_2D, texName);
   glTex□mage2D(GL_TEXTURE_2D, 0, 3, □□e□□□mage□ idt□,
      □□e□□□mage□ eig□t, 0, GL_RGB, GL_UN□□GNED_B□TE,
      &□□e□□□mage[0][0][0]);
   glTex□arameter□(GL_TEXTURE_2D, GL_TEXTURE_□ R□□_□, GL_CL□□ □);
   glTex□arameter□(GL_TEXTURE_2D, GL_TEXTURE_□ R□□_T, GL_CL□□ □);
   glTex□arameter□(GL_TEXTURE_2D, GL_TEXTURE_□ □G_□□LTER,
    GL_L□NE□R);
   glTex□arameter□(GL_TEXTURE_2D, GL_TEXTURE_□ □N_□□LTER,
    GL_L□NE□R);
   glTexEn□□(GL_TEXTURE_EN□, GL_TEXTURE_EN□_□ □DE, GL_DEC□L);
   glEna□le(GL_TEXTURE_2D);
   gl□□ade□ □del(GL_□L□T);
```

```
□□id dis□la□(□□id)
□
    glClear(GL_C□L□R_BU□□ER_B□T □GL_DE□T□_BU□□ER_B□T);
    glBegin(GL_□U□D□);
    glTexC□□rd2□(0□0, 0□0); gl□ertex3□(□2□0, □1□0, 0□0);
    glTexC□□rd2□(0□0, 1□0); gl□ertex3□(□2□0, 1□0, 0□0);
    glTexC□□rd2□(1□0, 1□0); gl□ertex3□(0□0, 1□0, 0□0);
    glTexC□□rd2□(1□0, 0□0); gl□ertex3□(0□0, □1□0, 0□0);

    glTexC□□rd2□(0□0, 0□0); gl□ertex3□(1□0, □1□0, 0□0);
    glTexC□□rd2□(0□0, 1□0); gl□ertex3□(1□0, 1□0, 0□0);
    glTexC□□rd2□(1□0, 1□0); gl□ertex3□(2□□1□21, 1□0, □1□□1□21);
    glTexC□□rd2□(1□0, 0□0); gl□ertex3□(2□□1□21, □1□0, □1□□1□21);
    glEnd();
    glut□□a□Bu□□ers();
□
```

```
int
main(int arg□, □□ar□□arg□)
□
    glut□nit(&arg□, arg□);
    glut□nitDis□la□□ □de(GLUT_D□UBLE □GLUT_RGB □GLUT_DE□T□);
    glutCreate□ ind□□ (□□□e□□er□);
    m□init();
    glutRes□a□e□un□(m□Res□a□e);
    glutDis□la□□un□(dis□la□);
    glut□ ainL□□□();
    return 0;          □□□N□□C re□uires main t□return int□□□
□
```

Practical Consideration in OpenGL

- Generating a texture image
- Storing a texture image
- Mapping from external format to internal format
- Texture border (repeat or clamp)
- Multiple levels of details
- Filtering (anti-aliasing)
- Efficiency consideration

Texture Generation

- As you might have suspected, OpenGL, being a pure graphics package, does not provide
  - Routines to read image files
    - Public domain software, e.g. xv, acdsee, with source codes for reading a variety of image formats
    - Try pbm, pgm, ppm
  - Routines to generate texture
    - Simple ones such as checkboard patterns are easily generated
    - More complicated ones such as Zebra pattern using reaction-diffusion PDE

Storing a Texture Image

- Texture images can be
  - 1 bit/pixel (e.g., a bitmap)
  - 8 bits/pixel (e.g., a grayscale image)
  - 24 bits/pixel (e.g., a color image with RGB)
  - 32 bits/pixel (e.g., a color image with RGBA)
- Hardware may dictate data storage on 2- 4- 8- byte boundary
- Explicit specification of storage format

Storing a Texture Image

- void glPixelStore{if}(GLenum pname, TYPE param)
  - pname: GL_UNPACK_*, GL_PACK_*
    - GL_PACK_* controls how data is packed into memory
    - GL_UNPACK_* controls how data is unpacked from memory
  - param: valid values for pname

Storing a Texture Image

- *SWAP_BYTES (false)
  - Whether multiple byte elements (e.g., int) should be swapped
- *LSB_FIRST (false)
  - For 1-bit images (bitmaps)
  - 0x31 {0,0,1,1,0,0,0,1} (false)
  - 0x31 {$\underline{1,0,0,0}$,$\underline{1,1,0,0}$) (true)
  - RGB is always R, then G, then B
  - Only when multiple bytes/color are swapped

Storing a Texture Image

- *ALIGNMENT (1,2,4,8)
  - Data should be aligned properly to facilitate hardware retrieval operations
  - 1: next byte is read
  - 2: every row lines up at 2 byte boundary
  - 4: every row lines up at 4 byte boundary
- Hint: if you don’t care about specific hardware and store image data consecutively without gap, do

  gl□ixel□t□rei(GL_UN□□C□_□L□GN□ ENT,1);

Mapping

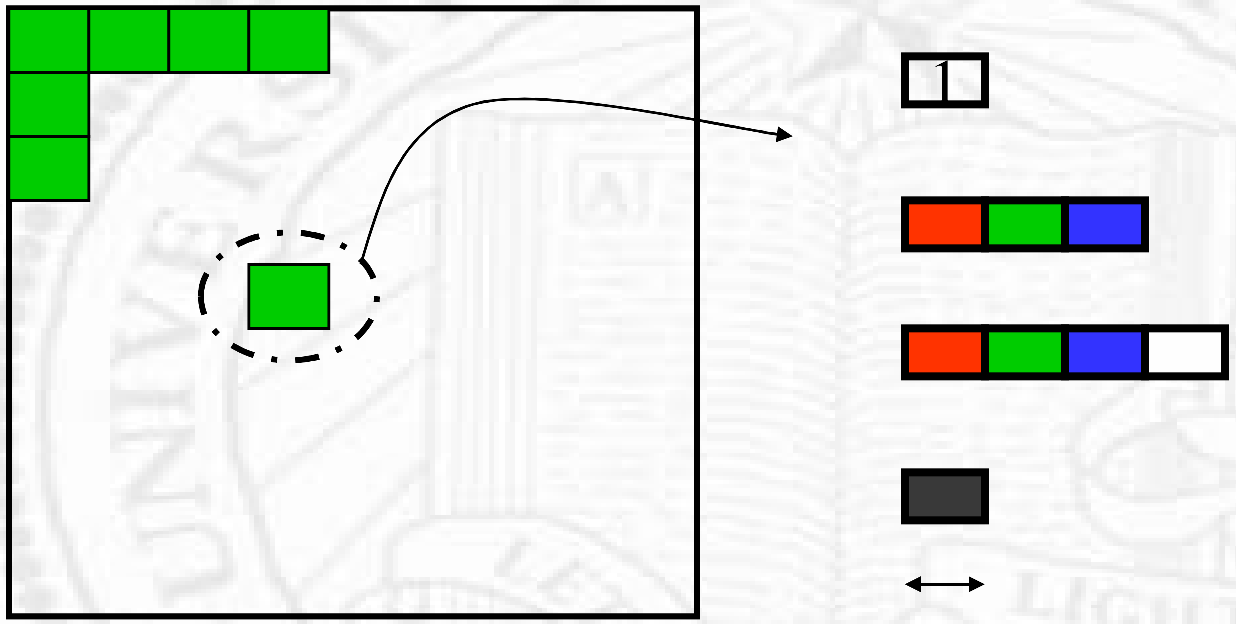
- void glTexImage2D(…)
  - target: GL_TEXTURE_2D
  - level: 0 (usually), >=0 (mipmap)
  - internalFormat: GL_ALPHA, GL_LUMINANCE (1), GL_LIMINANCE_ALPHA (2), GL_INTENSITY, GL_RGB (3), GL_RGBA (4)
  - width, height: 2^m x 2^n
  - border: 0 or 1
  - 2^mx2^n or 2^m+b x 2^n+b (64x64, 66x66 –w. one pixel border)

Mapping

- format: GL_COLOR_INDEX, GL_RGB, GL_RGBA, GL_RED, GL_GREEN, GL_BLUE, GL_ALPHA, GL_LUMINANCE, GL_LUMINANCE_ALPHA
- type: GL_BYTE, GL_UNSIGNED_BYTE, GL_SHORT, GL_UNSIGNED_SHORT, GL_INT, GL_UNSIGNED_INT, GL_FLOAT
- pixels: pointers to data

Mapping

- Format & type



Index

RGB

RGBA

Luminance

BYTE, SHORT, INT, FLOAT (signed or unsigned)

Mapping

- internalFormat: Which of the R, G, B and A or luminance values are selected for use in texture mapping
  - Improved flexibility
- Hint: most of the time, you read a RGB image and use all three components, hence, both *format* and *internalFormat* should be GL_RGB

Not 2^m by 2^n?

- gluScaleImage(

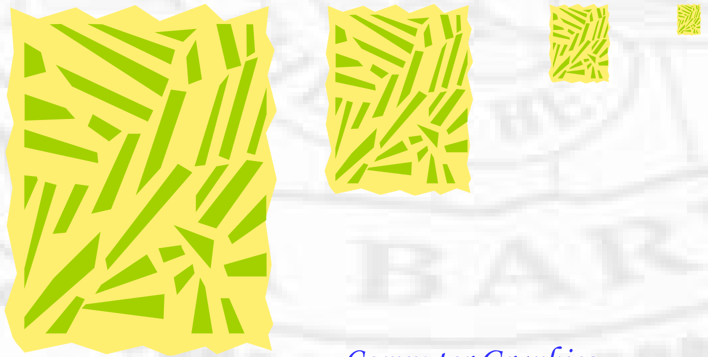
  format, widthin, heightin, typein,datain,

  Widthout, heightout, typoeout, dataout)

  - Interpolate with linear interpolation and box filtering

Multiple Levels of Detail

- Texture objects can be viewed from different distances or viewpoints
- Enlargement and shrinkage are common
- Let user create a pyramidal map (mipmap) to describe textures at different resolutions
- glTexImage2D() are called multiple times with different mipmap images (original at level 0)

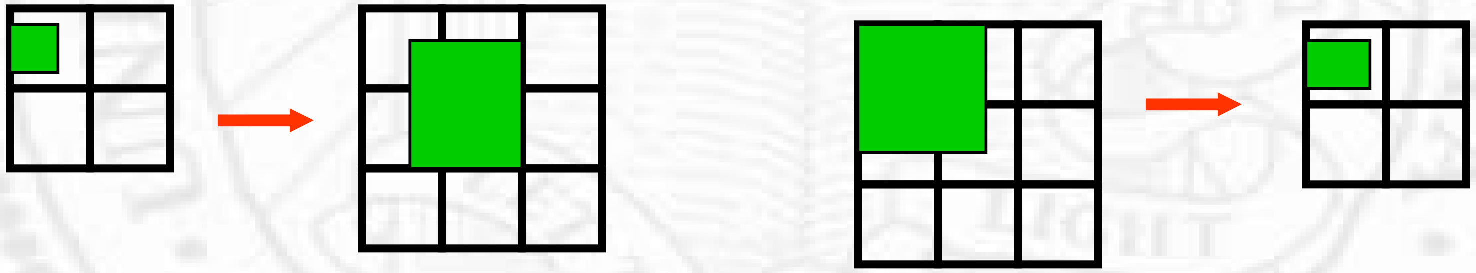


Filtering and Repetition

- void glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S (T), GL_REPEAT (GL_CLAMP));
  - What happen if one runs outside (0,1) range
    - E.g., if texture coordinates run from 0 to 10 in both s and t directions, 100 copies of textures are tiled
  - GL_REPEAT: integer part is ignored (1.1, 2.1, 3.1 … are equivalent to 0.1)
  - GL_CLAMP: >1.0 set to 1.0, <0.0 set to 0.0

Filtering and Repetition

- void glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN(MAG)_FILTER, GL_NEAREST (GL_LINEAR));



Filtering and Repetition

- For each pixel on display
  - GL_NEAREST: the color (luminance, alpha) of the texel closet to the center of that pixel is used
  - GL_LINEAR: a weighted linear average of 2x2 array of texels that lie nearest to the center of that pixel is used

Appearance Mapping Revisited

- First, texture mapping is not guaranteed to work in color index mode
- Otherwise, there are four ways supported by OpenGL

internalformat	Replace	Modulate
GL_ALPHA	$C = C_f$	$C = C_f$
	$A = A_t$	$A = A_f A_t$
GL_LUMINANCE	$C = L_t$	$C = C_f L_t$
	$A = A_f$	$A = A_f$
GL_LUMINANCE_ALPHA	$C = L_t$	$C = C_f L_t$
	$A = A_t$	$A = A_f A_t$
GL_INTENSITY	$C = I_t$	$C = C_f I_t$
	$A = I_t$	$A = A_f I_t$
GL_RGB	$C = C_t$	$C = C_f C_t$
	$A = A_f$	$A = A_f$
GL_RGBA	$C = C_t$	$C = C_f C_t$
	$A = A_t$	$A = A_f A_t$

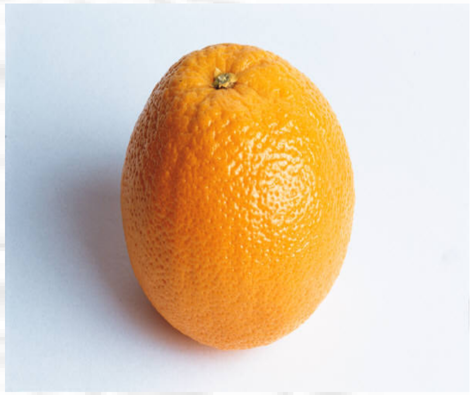
t: texture, *f*: incoming fragment

internalformat	Decal	Blend
GL_ALPHA	*undefined*	$C = C_f$ $A = A_f A_t$
$GL\_LUMINANCE$	*undefined*	$C = C_f(1 - L_t) + C_c L_t$ $A = A_f$
$GL\_LUMINANCE\_ALPHA$	*undefined*	$C = C_f(1 - L_t) + C_c L_{tt}$ $A = A_f A_t$
$GL\_INTENSITY$	*undefined*	$C = C_f(1 - L_t) + C_c L_t$ $A = A_f(1 - L_t) + A_t I_t$
GL_RGB	$C = C_t$ $A = A_f$	$C = C_f(1 - L_t) + C_c L_t$ $A = A_f$
GL_RGBA	$C = C_f(1 - A_t) + C_t A_t$ $A = A_f$	$C = C_f(1 - L_t) + C_c L_t$ $A = A_f A_t$

t: texture, *f*: incoming fragment

Bump Mapping

- Consider the scenario where appearance of texture is *viewpoint dependent*
  - E.g. surface pattern of an orange (with highlight)
  - Digitize an image of an orange
  - Apply that as texture
  - Problem: the highlight will not move no matter how you change the light and view point!



Bump Mapping

- How can texture mapping responds to change of viewpoint and light source?
  - Cannot be applied as decal
  - Modulation usually do not work well (e.g. highlight)
- Solutions:
  - Either
    - Surface position
    - Surface orientation

  Have to change

Bump Mapping

- To model the perturbation of a rough surface, we can do $P' = P + T(u,v)n$
- How do you render such a structure?
  - Not a single polygon or a smooth surface anymore
  - Holes may appear

Bump Mapping

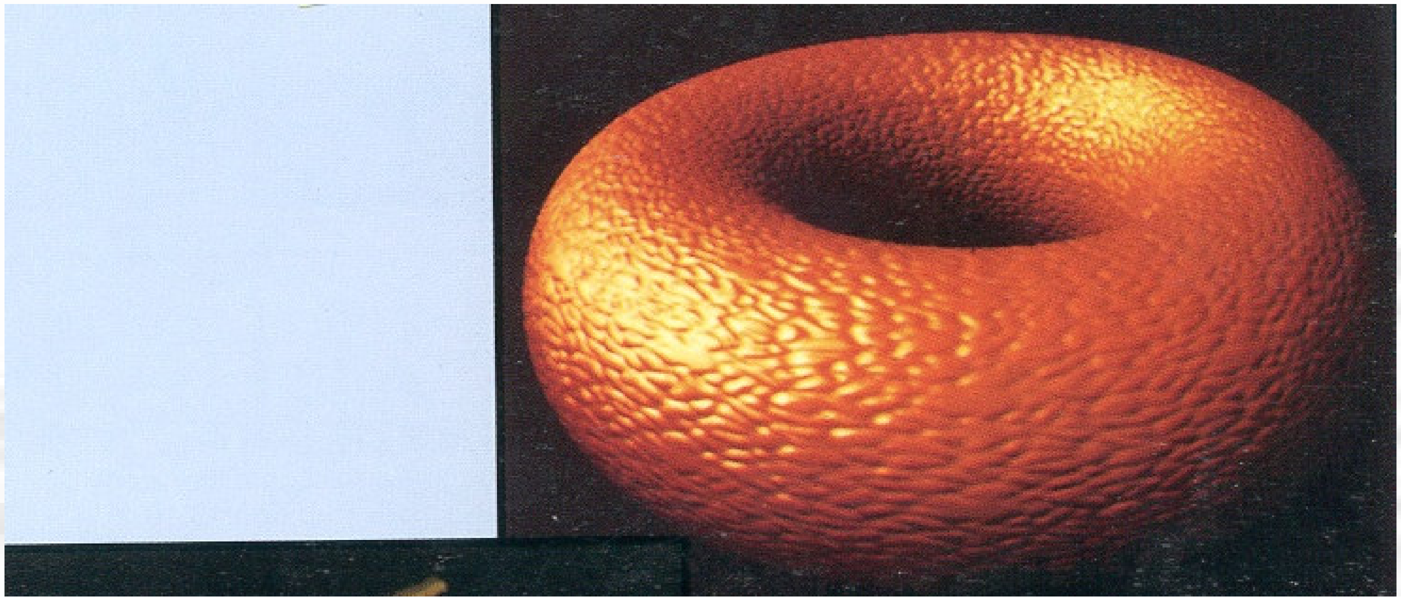
- Observation
  - The perception of depth does not necessarily require true variation of the depth
  - A change in normal vector and lighting can simulate that effect very well

$$P' = P + T(u,v)n$$

$$\frac{\partial P'}{\partial u} = \frac{\partial P}{\partial u} + \frac{\partial T}{\partial u} n + T \frac{\partial n}{\partial u}$$

$$\frac{\partial P'}{\partial v} = \frac{\partial P}{\partial v} + \frac{\partial T}{\partial v} n + T \frac{\partial n}{\partial v}$$

$$\Rightarrow \quad n' = n + \frac{\partial T}{\partial u} n \times \frac{\partial P}{\partial v} + \frac{\partial T}{\partial v} n \times \frac{\partial P}{\partial u}$$



```
ERROR: undefined
OFFENDING COMMAND: f‘~

STACK:
```