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

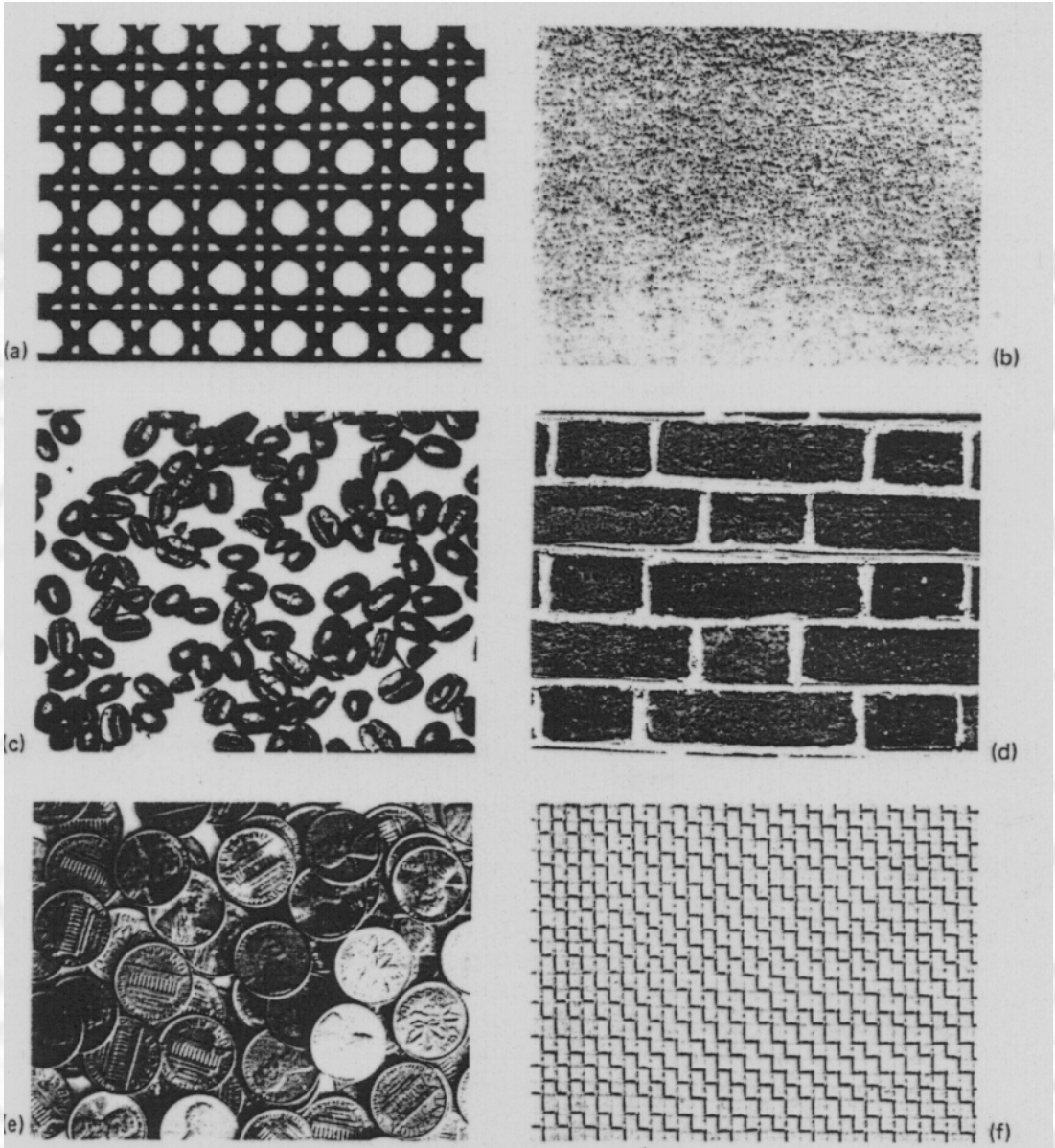
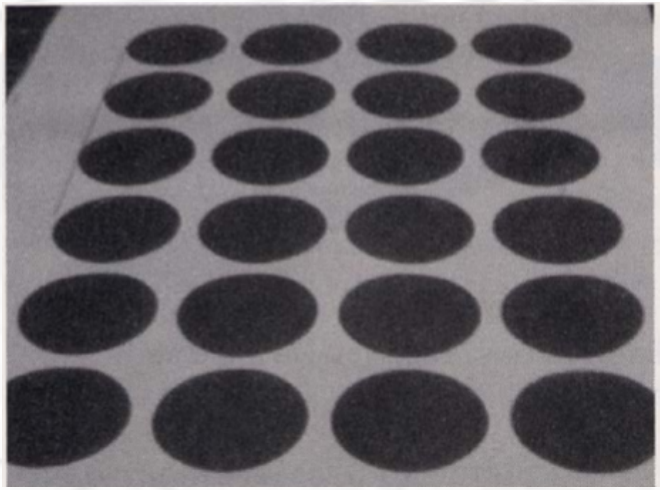


Fig. 6.1 Six examples of texture. (a) Cane. (b) Paper. (c) Coffee beans. (d) Brick wall. (e) Coins. (f) Wire braid.

Shape-from-texture

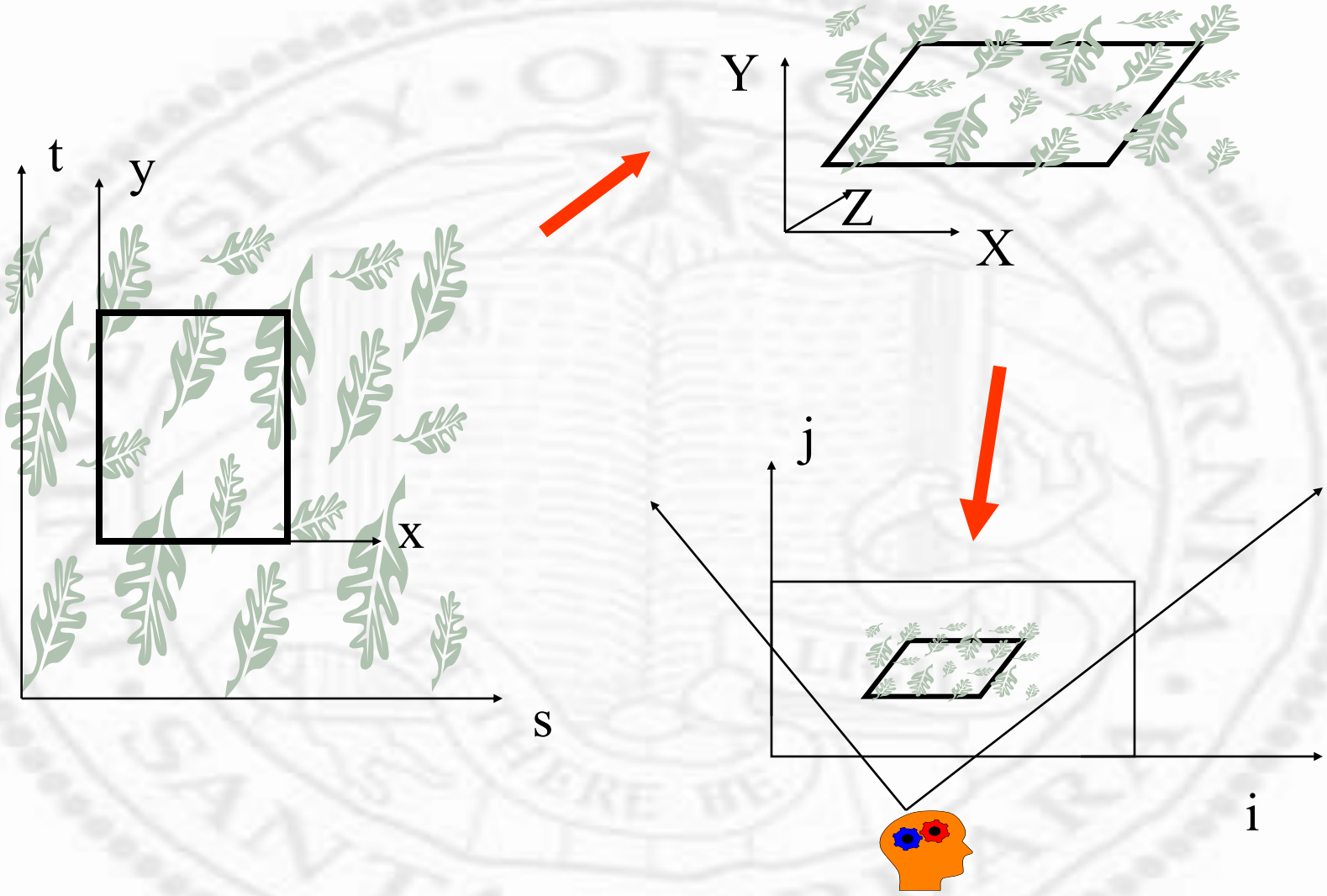


Theoretical Consideration
Texture mapping

- *Geometry* mapping
  - Where does the texture go physically?

- *Appearance* mapping
  - How will texture appear? As a decal? Modulate lighting? Modulate surface structure?

Geometry Mapping (2D)



Geometry Mapping (2D)

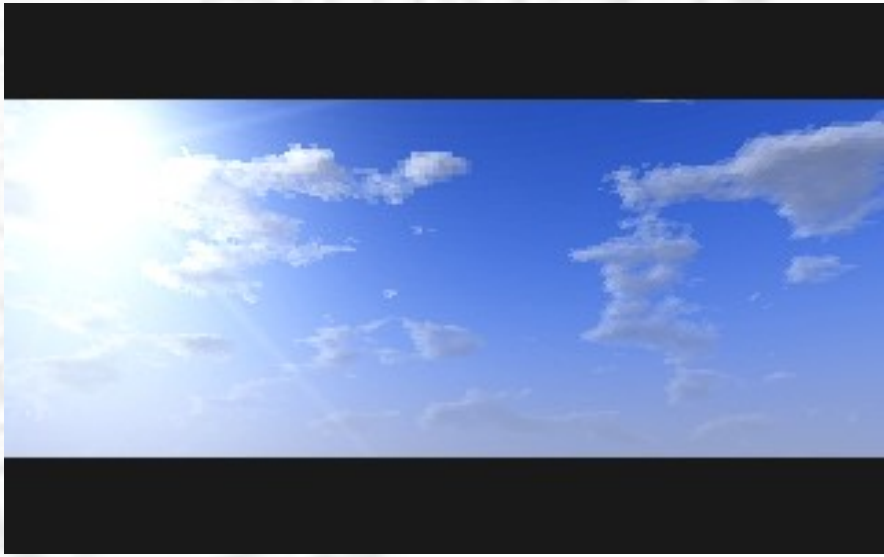
- Texture transforms the way the underlying object does, with an added transform to map from texture coordinates to object coordinates

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

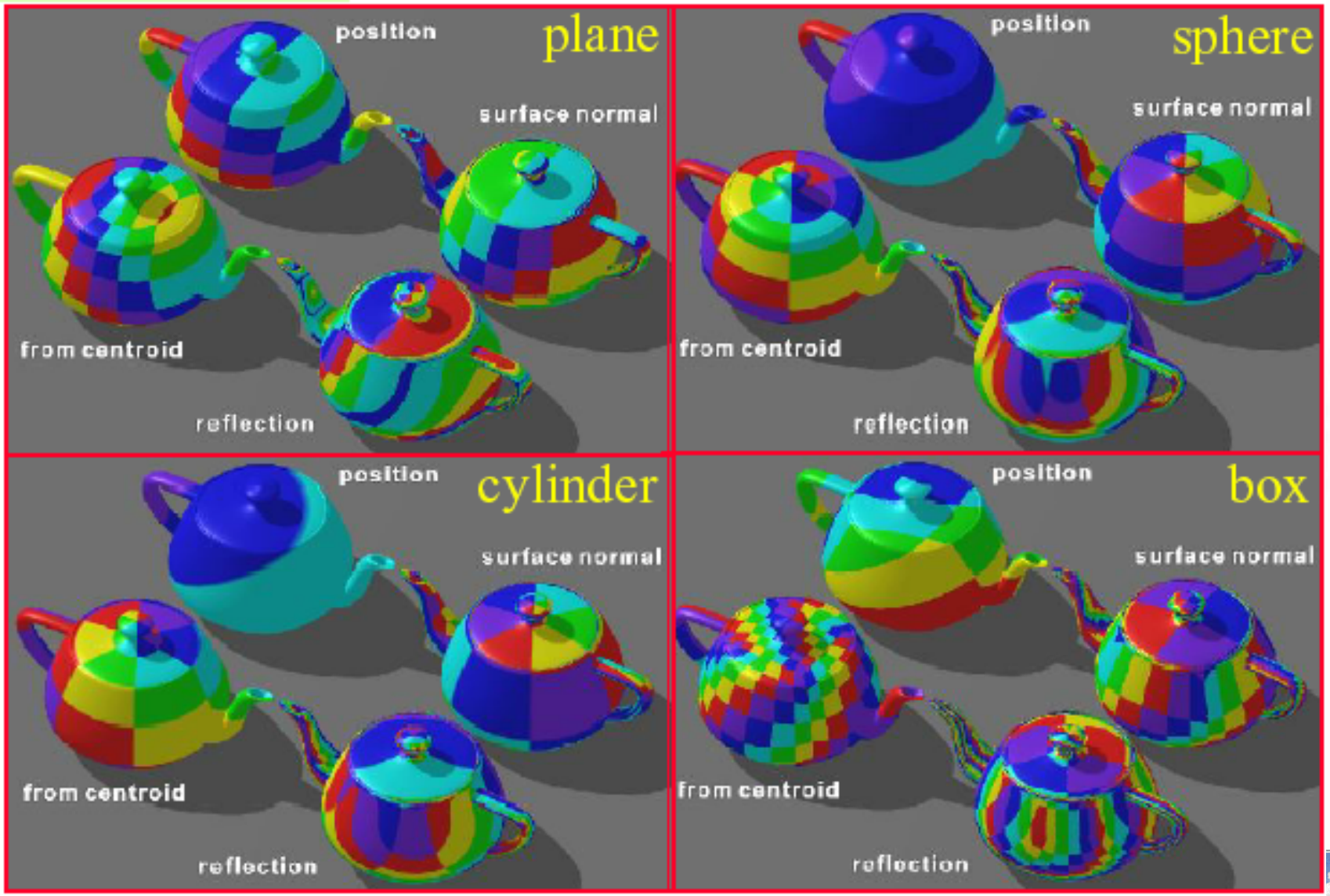
Image coordinate (i,j) Texture coordinate (s,t)

Appearance Mapping

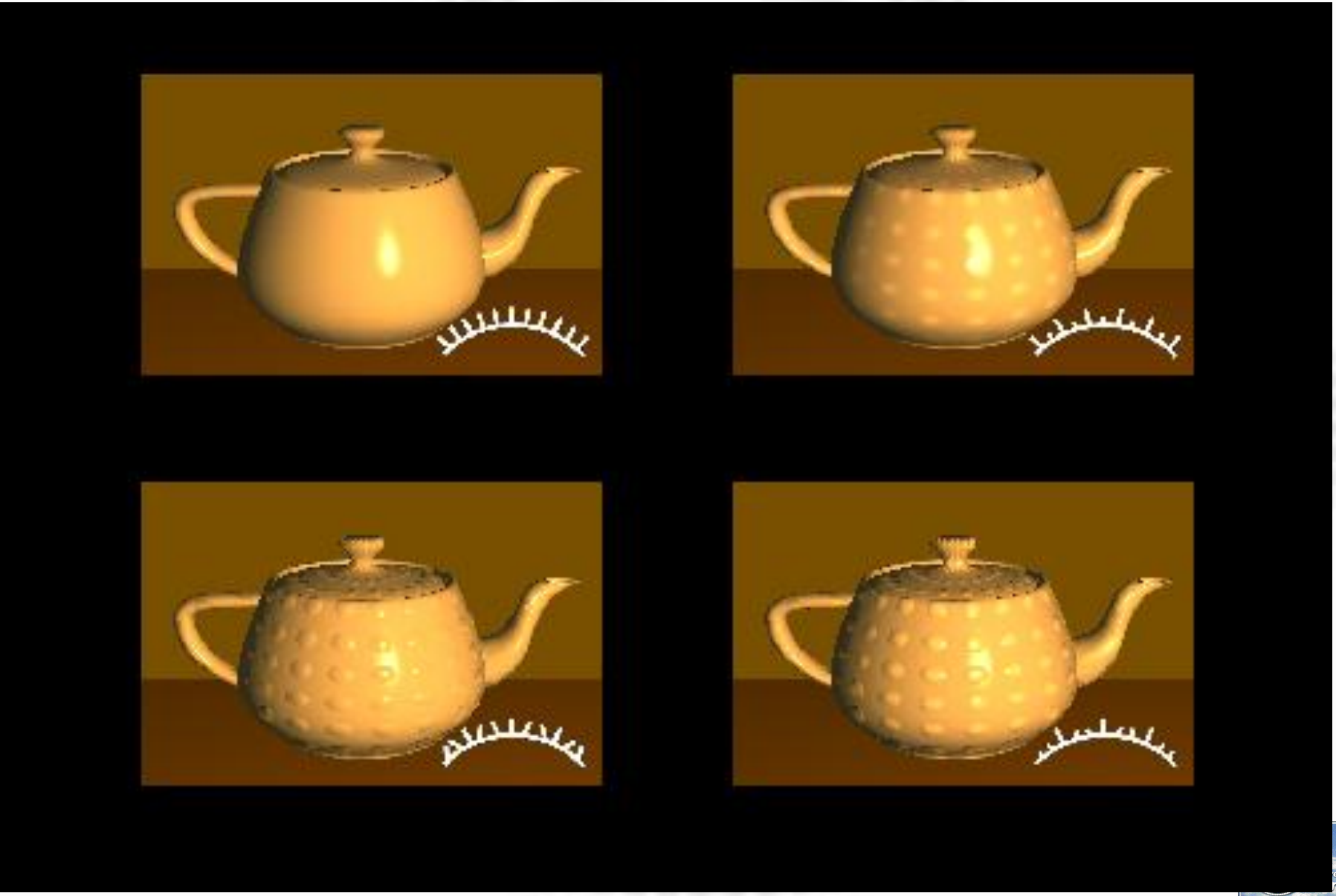
- Many choices
  - Use as a decal
    - Replace the object color
  - Use as a modulator
    - Modulate Alpha component
    - Modulate Luminance component
    - Modulate Color components
    - Module surface structure
    - Module surface orientation
    - Etc.

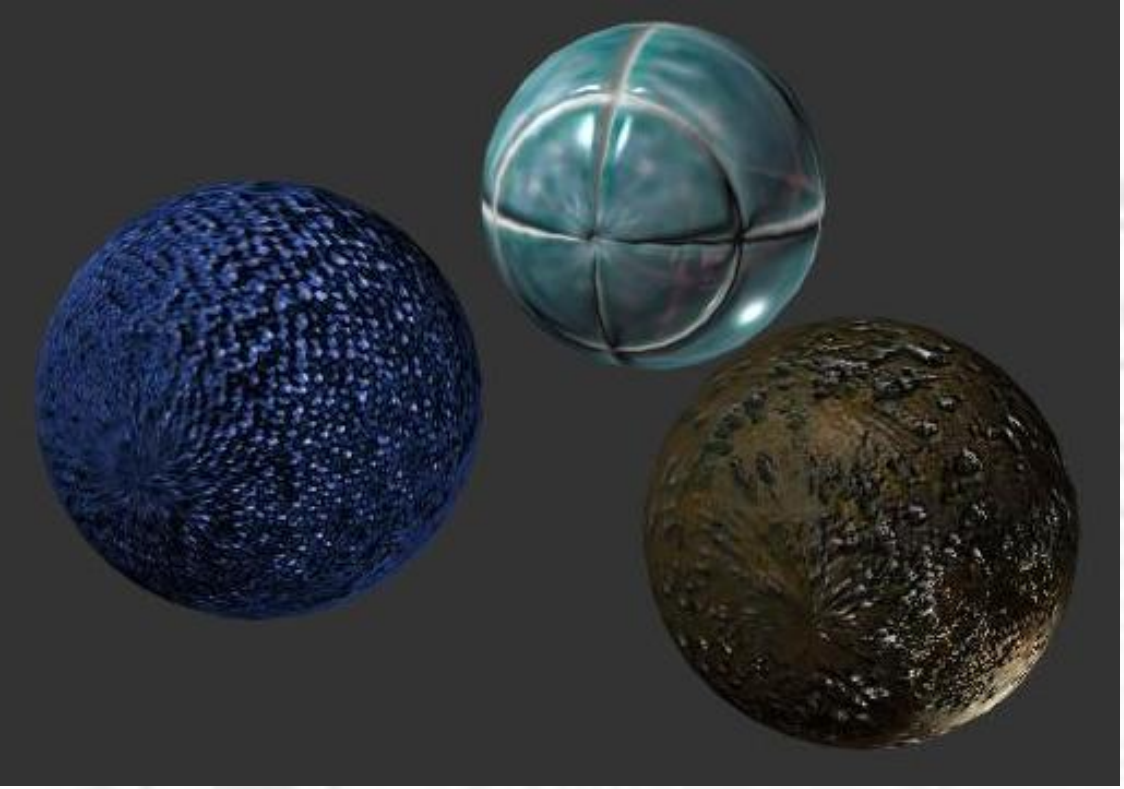


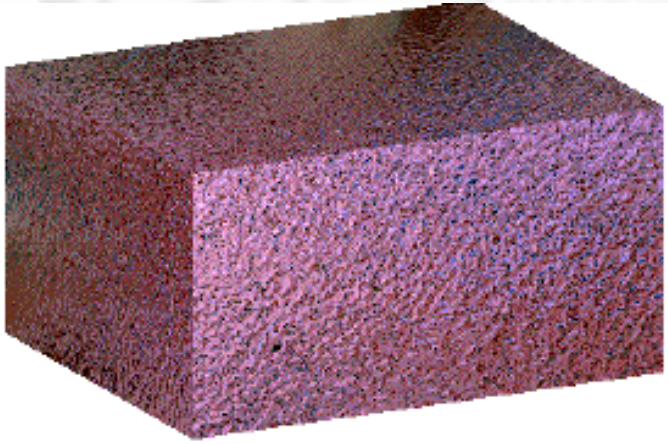
Use as Decals

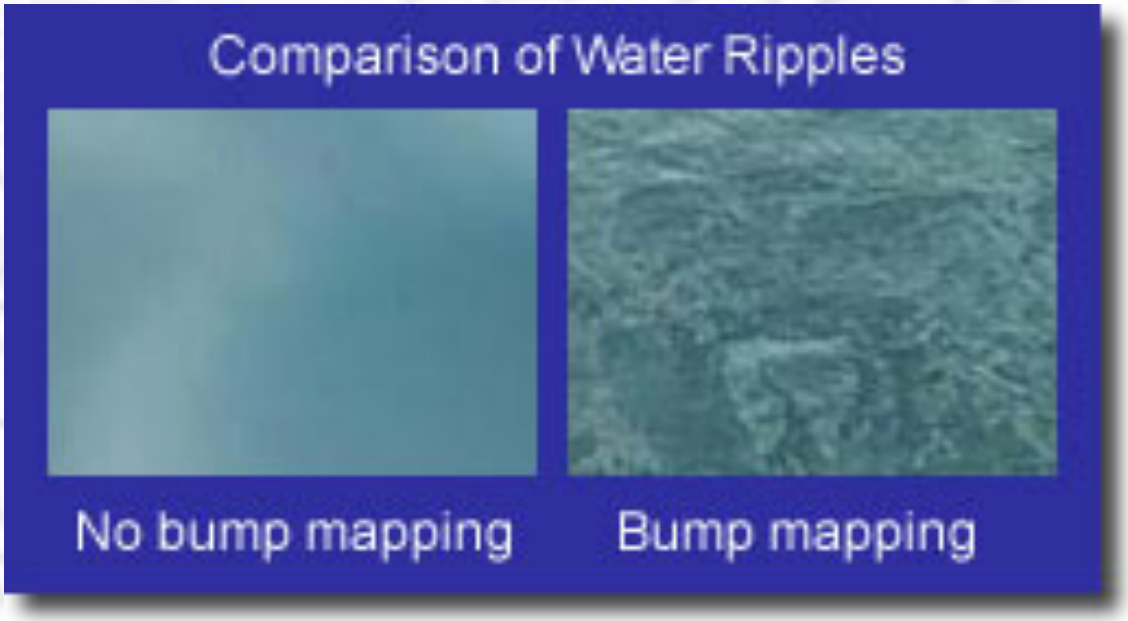


Bump & Displacement Mapping







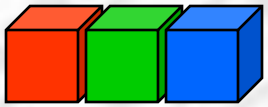

Comparison of Water Ripples
No bump mapping
Bump mapping

OpenGL Texture

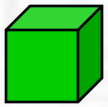
- Create texture objects – *placeholder, name only*
  - glGenTextures
- Bind a texture to the object – *create and make it active*
  - glBindTexture
- *Load the content*
  - glTexImage2D
- Enable texture mapping – *make it happen*
  - glEnable
- Indicate how texture should be applied –(*appearance* mapping)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

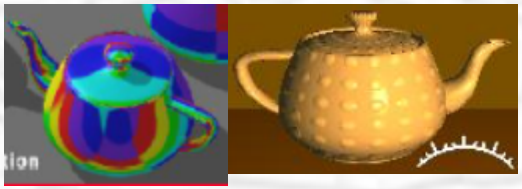


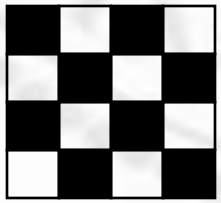
Load content (glTexImages2D)

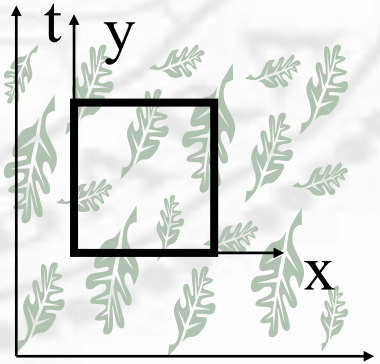
Enable the mapping (glEnable)

Appearance
(glTexEnv)

Geometry
(glTexCoor)







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
- *texels*: 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

Not 2^m by 2^n?

- gluScaleImage(

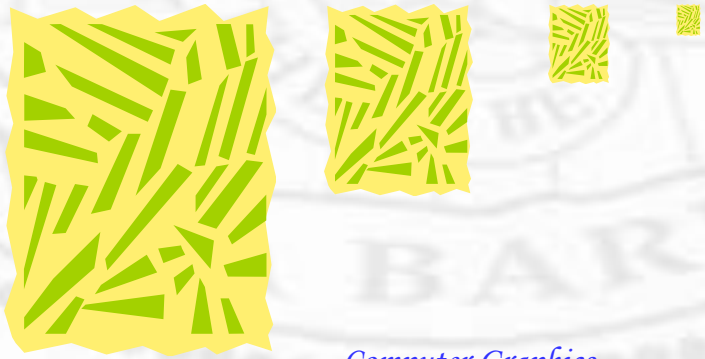
  format, widthin, heightin, typein,datain,

  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)



Example

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

…

```
glTexEnvf(GL_TEXTURE_ENV,
    GL_TEXTURE_ENV_MODE, GL_DECAL);
```

…

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

Geometry Mapping

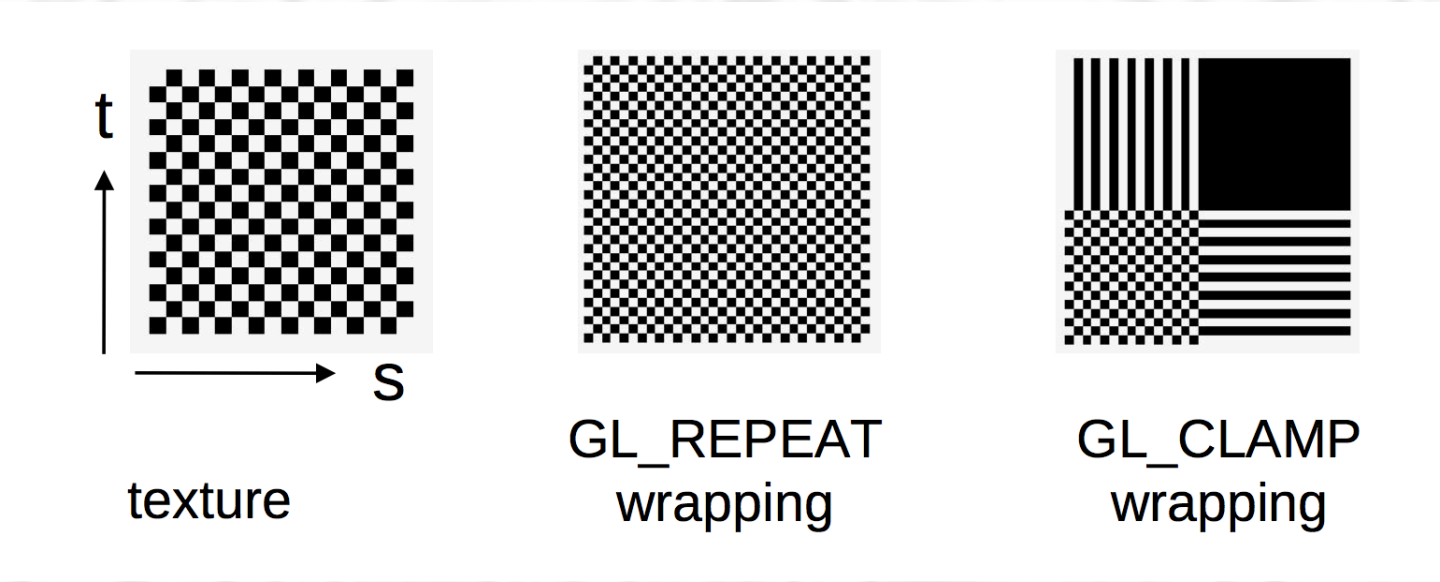
- Alternate texture coordinate and vertex coordinate bind one to the other
- Texture coordinates always go from 0 to 1 in *s* and 0 to 1 in *t*
- Vertex coordinates can be anything

```
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();
```

Wrapping

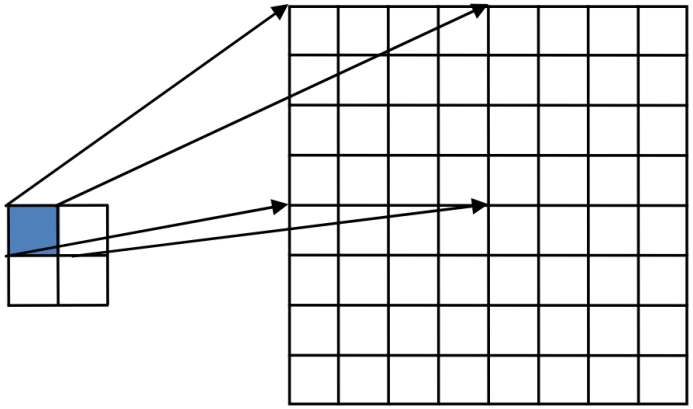
- void glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S (T), GL_REPEAT (GL_CLAMP));
  - What happen if one runs outside (0,1) range
    - E.g., if texture coordinates run from 0 to 10 in both s and t directions, 100 copies of textures are tiled
  - GL_REPEAT: integer part is ignored (1.1, 2.1, 3.1 …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

Wrapping (cont.)



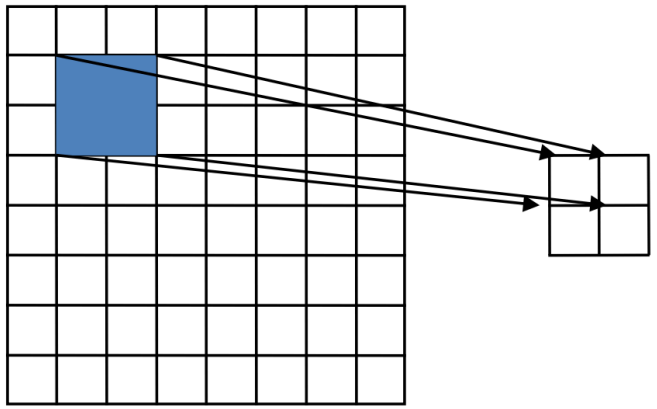
Filtering

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



Texture Polygon

Magnification



Texture Polygon

Minification

Putting it altogether

```
#include <GL/glut.h>

/*     Create checkerboard texture     */
#define checkImageWidth 64
#define checkImageHeight 64
GLubyte checkImage[checkImageWidth][checkImageHeight][3];
GLuint texName;
void makeCheckImage(void)
{
   int i, j, c;

   for (i = 0; i < checkImageWidth; i++) {
      for (j = 0; j < checkImageHeight; j++) {
         c = ((((i&0x8)==0)^((j&0x8))==0))*255;
         checkImage[i][j][0] = (GLubyte) c;
         checkImage[i][j][1] = (GLubyte) c;
         checkImage[i][j][2] = (GLubyte) c;
      }
   }
}
```

```
void myinit(void)
{
   glClearColor (0.0, 0.0, 0.0, 0.0);
   glEnable(GL_DEPTH_TEST);
   glDepthFunc(GL_LESS);

   makeCheckImage();
   glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
   glGenTextures(1, &texName);
   glBindTexture(GL_TEXTURE_2D, texName);
   glTexImage2D(GL_TEXTURE_2D, 0, 3, checkImageWidth,
      checkImageHeight, 0, GL_RGB, GL_UNSIGNED_BYTE,
      &checkImage[0][0][0]);
   glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
   glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
   glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
     GL_LINEAR);
   glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
     GL_LINEAR);
   glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
   glEnable(GL_TEXTURE_2D);
   glShadeModel(GL_FLAT);
}
```

```
void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   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);

   glTexCoord2f(0.0, 0.0); glVertex3f(1.0, -1.0, 0.0);
   glTexCoord2f(0.0, 1.0); glVertex3f(1.0, 1.0, 0.0);
   glTexCoord2f(1.0, 1.0); glVertex3f(2.41421, 1.0, -1.41421);
   glTexCoord2f(1.0, 0.0); glVertex3f(2.41421, -1.0, -1.41421);
   glEnd();
   glutSwapBuffers();
}
```

```
int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
   glutCreateWindow("checker");
   myinit();
   glutReshapeFunc (myReshape);
   glutDisplayFunc(display);
   glutMainLoop();
   return 0;            /* ANSI C requires main to return int. */
}
```

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

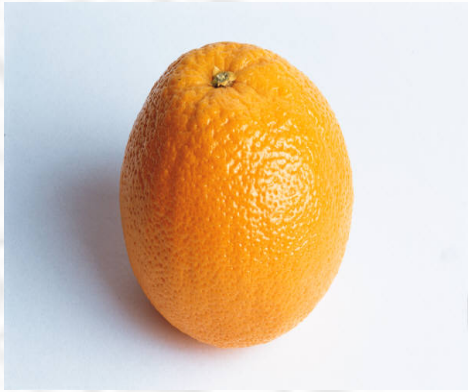
Storing a Texture Image

- *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

  glPixelStorei(GL_UNPACK_ALIGNMENT,1);

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
- Primary method
  - Store heightmap in the texture
  - Calculate the normal of each pixel based on the heightmap

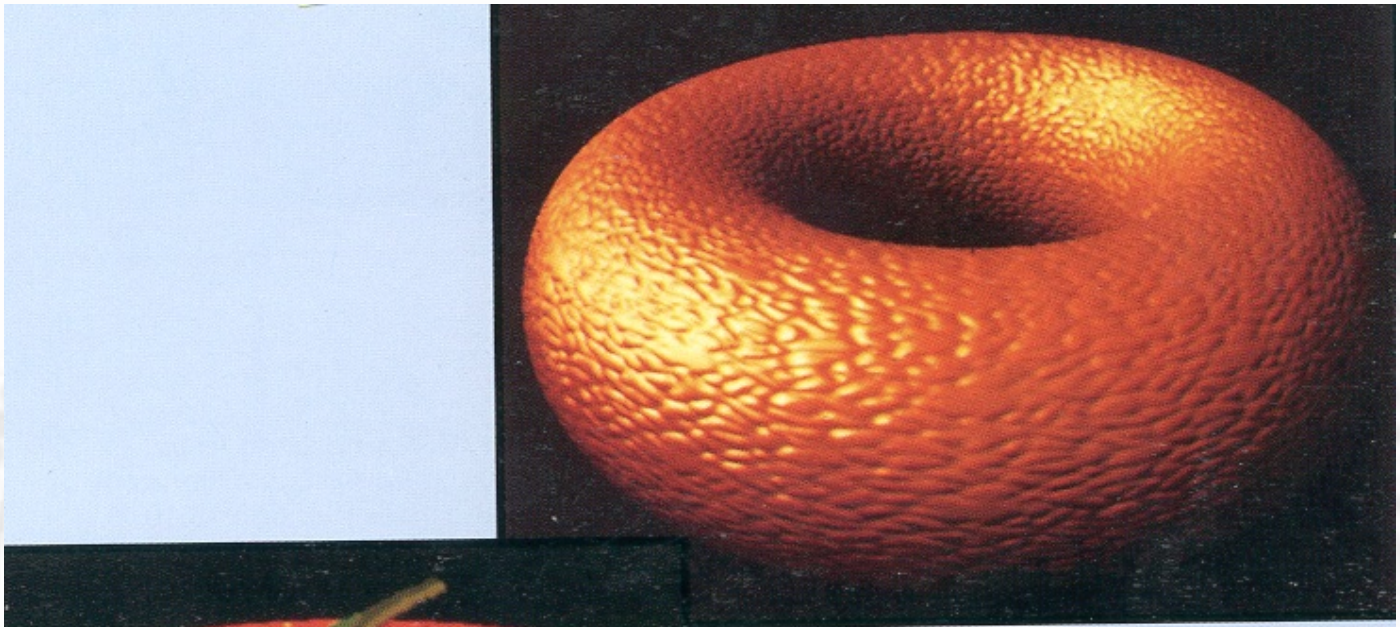


Plate III.3 A torus bump mapped with a hand-generated bump function (Section 16.3.3). (By Jim Blinn. Courtesy of University of Utah.)

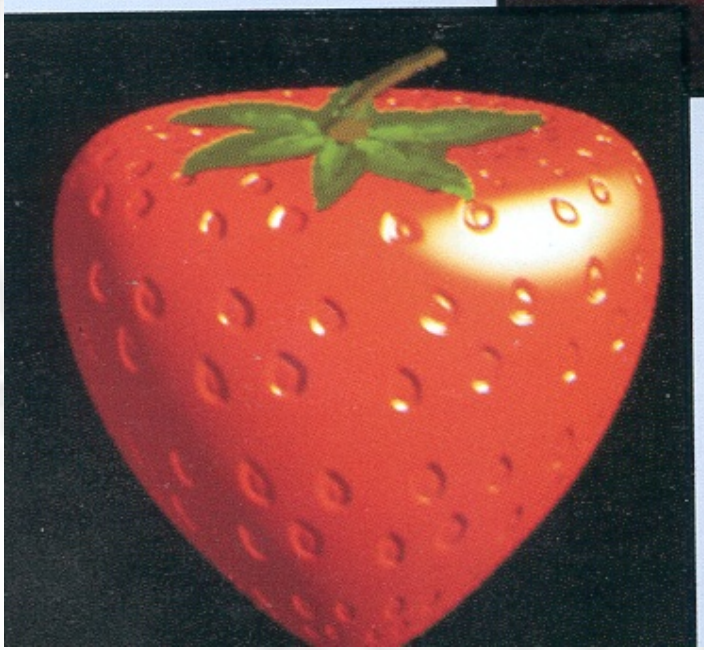


Plate III.4 A strawberry bump mapped with a hand-generated bump function (Section 16.3.3). (By Jim

Q & A

(Don’t forget to introduce Image Class)