XIAMEN UNIVERSITY

COMPUTER GRAPHICS

OpenGL Texture Mapping

Dr. Zhonggui Chen
School of Informatics, Xiamen University

http://graphics.xmu.edu.cn

Texture Matrix

- Texture mode is entered with a call to glMatrixMode(GL TEXTURE)
- The texture matrix stack and current texture matrix can be manipulated with glPushMatrix() , glLoadIdentity() , glTranslatef()
- Texture coordinates are transformed by multiplication from the left by the current texture matrix

- Example: \Chapter12\FieldAndSkyTextureAnimated

Texture Matrix

- Example: \Chapter12\FieldAndSkyTextureAnimated

```
// Enter texture mode and load identity.
glMatrixMode(GL_TEXTURE);
glLoadIdentity();
...
glTranslatef(0.1 * cos(angle), 0.1 * sin(angle), 0.0);
// Map the sky texture onto a rectangle parallel to the xy-plane.
glBindTexture(GL_TEXTURE_2D, texture[1]);
...
// Reenter modelview mode.
glMatrixMode(GL_MODELVIEW);
```

Lighting Textures

- glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, parameter)
- parameter:
  - GL_REPLACE,
    - the texture colors overwrite the current primitive pixel colors
  - GL_MODULATE
    1. Computes RGB values at a primitive's vertices using OpenGL's lighting equation and interpolates these through its interior to determine the RGB values at each of its pixels
    2. Uses the texture map to obtain RGB values from the texture at each of the primitive's pixels.
    3. Determines the final RGB values at each pixel as the product of the corresponding values from the preceding two steps.

Lighting Textures

- Example: If the RGB tuple at a pixel P is (0.5, 0.75 , 0.1) as obtained by interpolation from vertex RGB values computed after lighting, while that determined at P from the texture via the texture map is (0 . 4 , 0 . 5 , 1 . 0), then the final color applied to P using the GL_MODULATE option is:

  (0.5 × 0.4, 0.75 × 0.5, 0 .1 × 1.0) = (0.2 , 0.375 , 0.1).

- Example:
  \Chapter12\FieldAndSkyLit

Lighting Textures

- Run \Chapter12\LitTexturedCylinder\litTexturedCylinder.cpp
- The specular color components are separated and not multiplied with the corresponding texture color components, but added in after.

```
glLightModeli(GL_LIGHT_MODEL_COLOR_CONTROL,
GL_SEPARATE_SPECULAR_COLOR);
// Enable separate specular light calculation.
```

Multitexturing

- OpenGL allows more than one texture to be applied to a polygon
- Multitexturing requires more than one texture unit, each with id of the form GL_TEXTUREi

- Initialize GL_TEXTURE0 and bind it to texture[0]

```
// Select texture unit 0 as the currently active texture unit and
specify its texture states.
glActiveTexture(GL_TEXTURE0);
glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, texture[0]);
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
```

Multitexturing

- Initialize GL_TEXTURE1 and bind it to texture[1]

```
// Select texture unit 1 as the currently active texture unit and
specify its texture states.
glActiveTexture(GL_TEXTURE1);
glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, texture[1]);

// Unit 1 COMBINES with unit 0 in a manner
// specified by the combiner function.
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_COMBINE);
```

Multitexturing

- Combiner function:

```
// The COMBINER FUNCTION is specified to be interpolation
between RGB values of Arg0 and Arg1
// according to the formula: Arg0 * Arg2 + Arg1 * (1-Arg2)
glTexEnvi(GL_TEXTURE_ENV, GL_COMBINE_RGB, GL_INTERPOLATE);
```

- The interpolation is given by

$$Arg0 * Arg2 + Arg1 * (1 - Arg2)$$

which interpolates between Arg0 and Arg1, the interpolation parameter being Arg2

Multitexturing

- Specify the combiner's three arguments

```
// Texture combiner's zeroth source's RGB are from texture unit 0.
glTexEnvi(GL_TEXTURE_ENV, GL_SRC0_RGB, GL_TEXTURE0);
// Arg0's RGB values are zeroth source's color.
glTexEnvi(GL_TEXTURE_ENV, GL_OPERAND0_RGB, GL_SRC_COLOR);

// Texture combiner's first source's RGB are from texture unit 1.
glTexEnvi(GL_TEXTURE_ENV, GL_SRC1_RGB, GL_TEXTURE1);
// Arg1's RGB values are first source's color.
glTexEnvi(GL_TEXTURE_ENV, GL_OPERAND1_RGB, GL_SRC_COLOR);

// Texture combiner's second source's alpha is from
// GL_TEXTURE_ENV_COLOR.
glTexEnvi(GL_TEXTURE_ENV, GL_SRC2_ALPHA, GL_CONSTANT);
// Arg2 is second source's alpha.
glTexEnvi(GL_TEXTURE_ENV, GL_OPERAND2_ALPHA, GL_SRC_ALPHA);
```

Multitexturing

- Texture coordinates for the two texture units

```
glBegin(GL_POLYGON);
// glMultiTexCoord2f(GL_TEXTUREi, *, *) specifies
// the texture coordinates of texture unit i.
glMultiTexCoord2f(GL_TEXTURE0, 0.0, 0.0);
glMultiTexCoord2f(GL_TEXTURE1, 0.0, 0.0);
glVertex3f(-20.0, -20.0, 0.0);

…

glEnd();
```

Multitexturing

- \Chapter12\Multitexture\multitexture.cpp

Figure 12.42: Screenshots of `multitexture.cpp`: (a) Mid-day (b) Late evening (c) Night.

Multitexturing

- Set GL_TEXTURE_ENV_COLOR values

```
// Specify the texture environment variable.
glTexEnvfv(GL_TEXTURE_ENV, GL_TEXTURE_ENV_COLOR, constColor);
// Set the interpolation parameter, which is the alpha of
// environment color (i.e., Arg2 in setup).
constColor[3] = alpha;
```

- Callback routine for non-ASCII key entry

```
void specialKeyInput(int key, int x, int y)
{
    if (key == GLUT_KEY_RIGHT){
        if (alpha < 1.0) alpha += 0.01;}
    if (key == GLUT_KEY_LEFT)
    {   if (alpha > 0.0) alpha -= 0.01; }
    glutPostRedisplay();
}
```

Cube Mapping

- The environment is projected onto the faces of an equal-sided cube, or skybox as it's often called

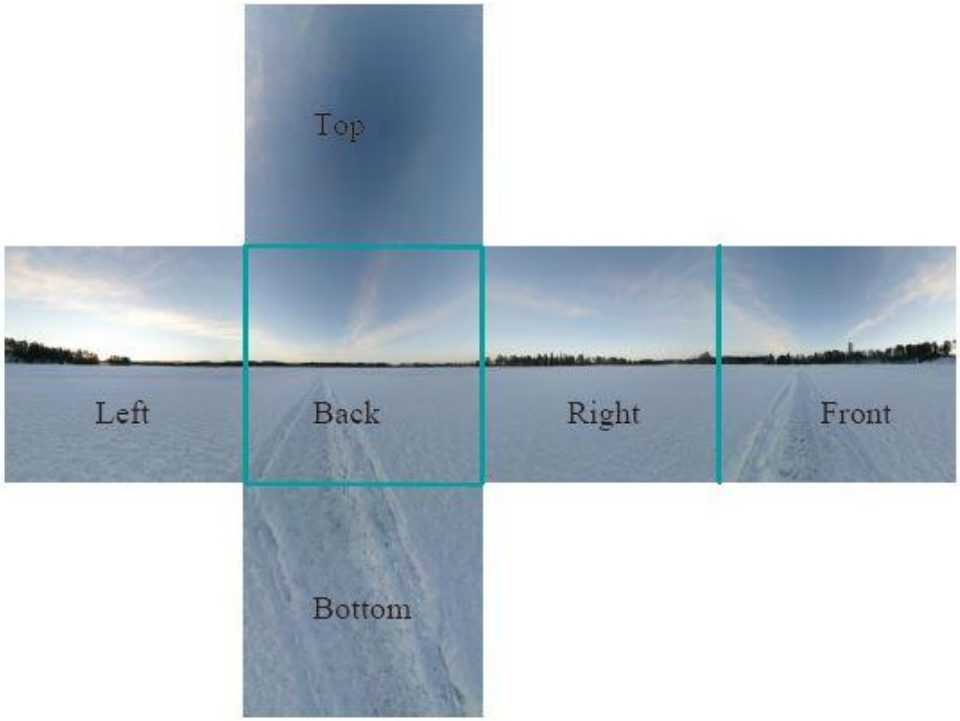


Figure 13.18: Cube map: images are rotated around the green edges to make a skybox. (Thanks Emil Persson, aka Humus, for a Creative Commons license.)

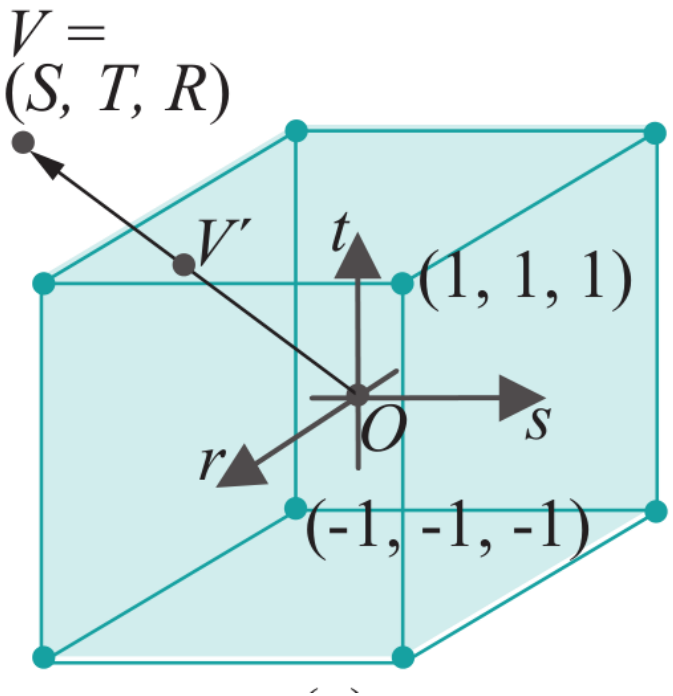
Cube Mapping

- Example: \Chapter13\Skybox\skybox.cpp

Figure 13.19:
Screenshot of `skybox.cpp`.

Cube Mapping

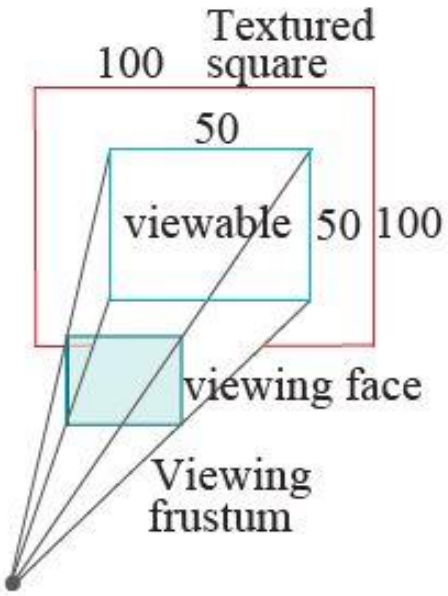
- 3D texture coordinates: (S,T,R), defining a direction V from the center O of a canonical skybox of side lengths 2 with vertex coordinates all ± 1
- The line in the direction of V, intersects the skybox at a point V', which specifies a point of a texture.



Cube Mapping

- Draw one textured square by applying 3D texture coordinates:

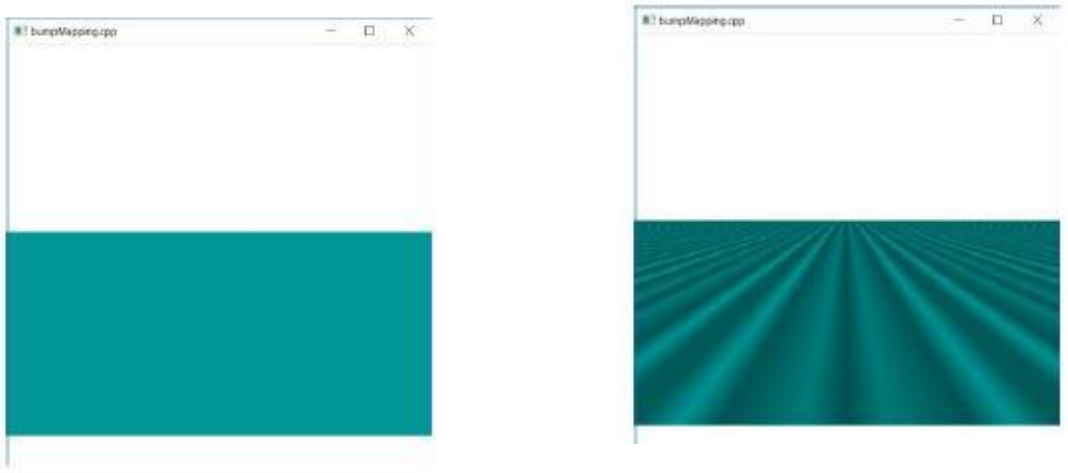
```
glBegin(GL_POLYGON);
glTexCoord3f(-1.0, 1.0, 1.0); glVertex3f(-50.0, -50.0, -25.0);
glTexCoord3f(1.0, 1.0, 1.0); glVertex3f(50.0, -50.0, -25.0);
glTexCoord3f(1.0, -1.0, 1.0); glVertex3f(50.0, 50.0, -25.0);
glTexCoord3f(-1.0, -1.0, 1.0); glVertex3f(-50.0, 50.0, -25.0);
glEnd();
```



Bump Mapping

- Make the surface appear ridged or dimpled, by means of perturbing the surface normals, but without actually changing any geometry.

- Example: \Chapter13\BumpMapping\bumpMapping.cpp



Bump mapping (left) off (right) on. The underlying geometry is a flat plane in both cases.