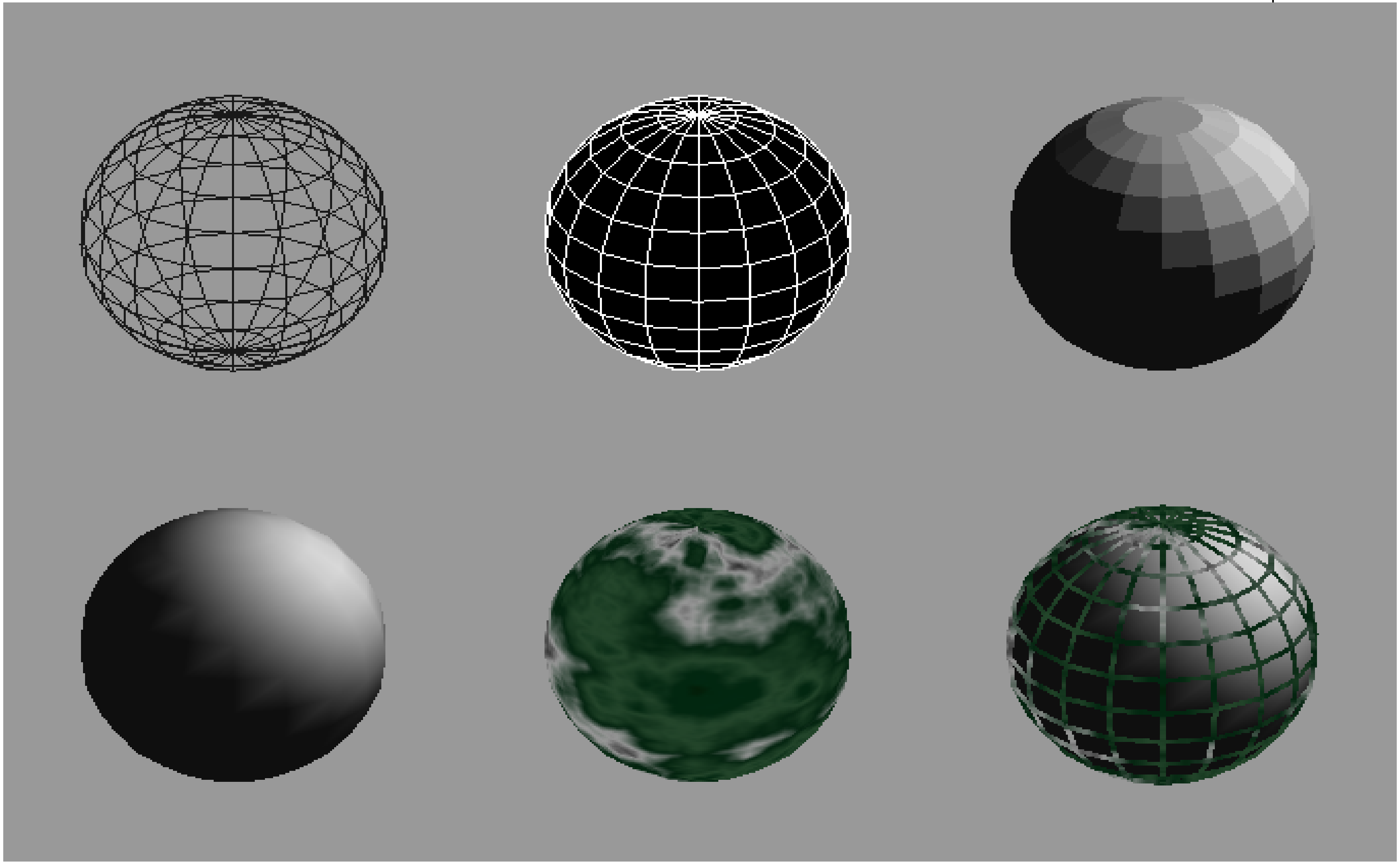
Texture Mapping

Lecture 13

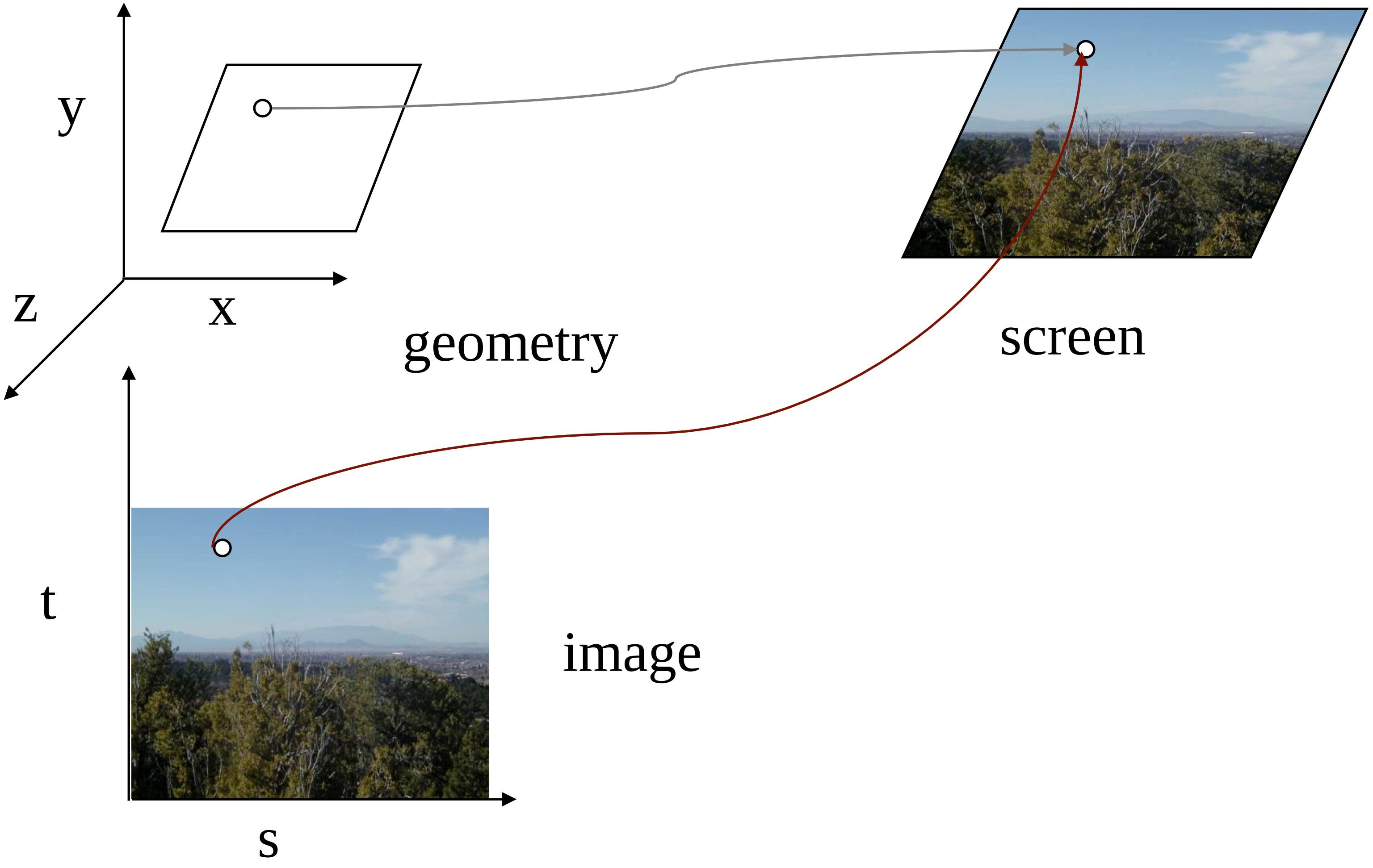
Outline

- Uses of texture mapping
- Advantages
- Applying Textures
- Mapping Textures
- Texture Application Mode
  - Filter Modes
  - Wrap Modes
  - Texture functions
- Texture example

Texture Mapping



Texture Mapping



Uses of texture mapping

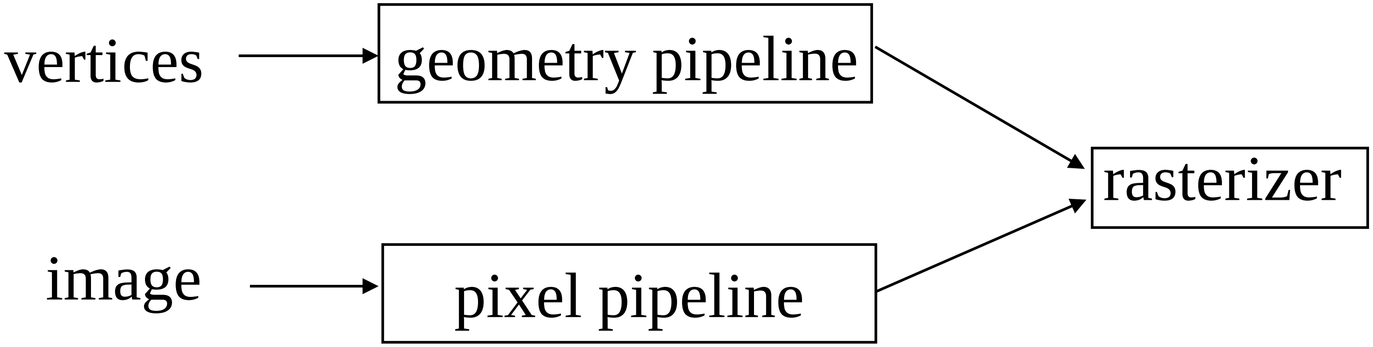
- Simulating materials like wood or bricks
- Reducing # of polygons of geometric object
- Image processing techniques like image warping, rotation and scaling
- Simulating reflective surfaces like mirrors or polished floors

Advantages

- Texture map can be reused for multiple objects
- Texture can be shared (consume less memory) and can be compressed
- Texture maps do not affect the geometry of the objects

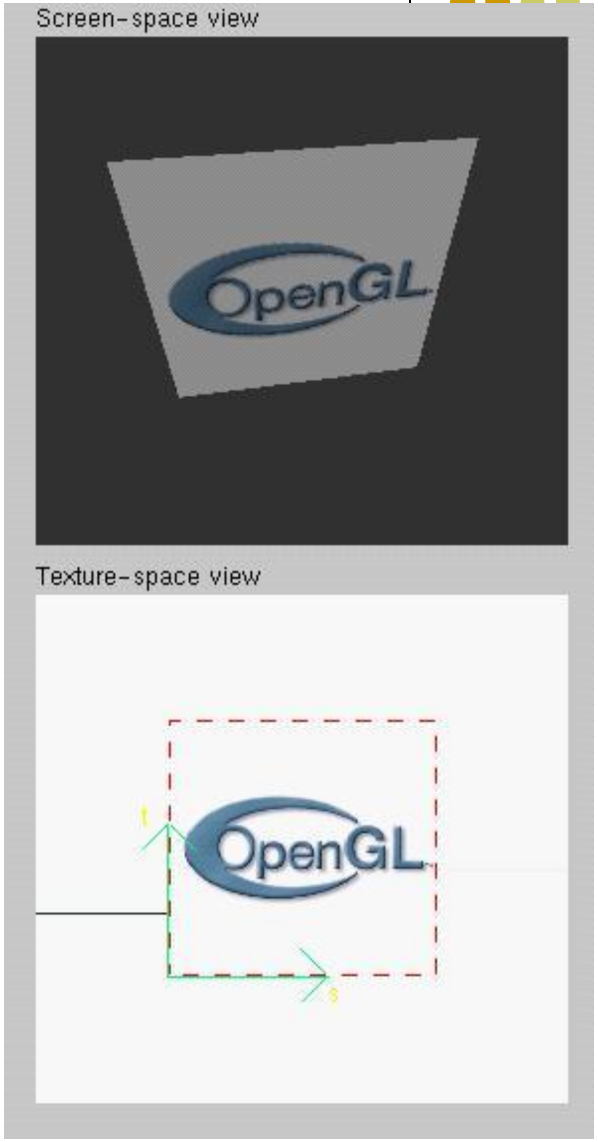
Texture Mapping and the OpenGL pipeline

- Images and geometry flow through separate pipelines that join at the rasterizer
  - “complex” textures do not affect geometric complexity



Texture Example

- The texture (below) is a 256 x 256 image that has been mapped to a rectangular polygon which is viewed in perspective



Applying Textures

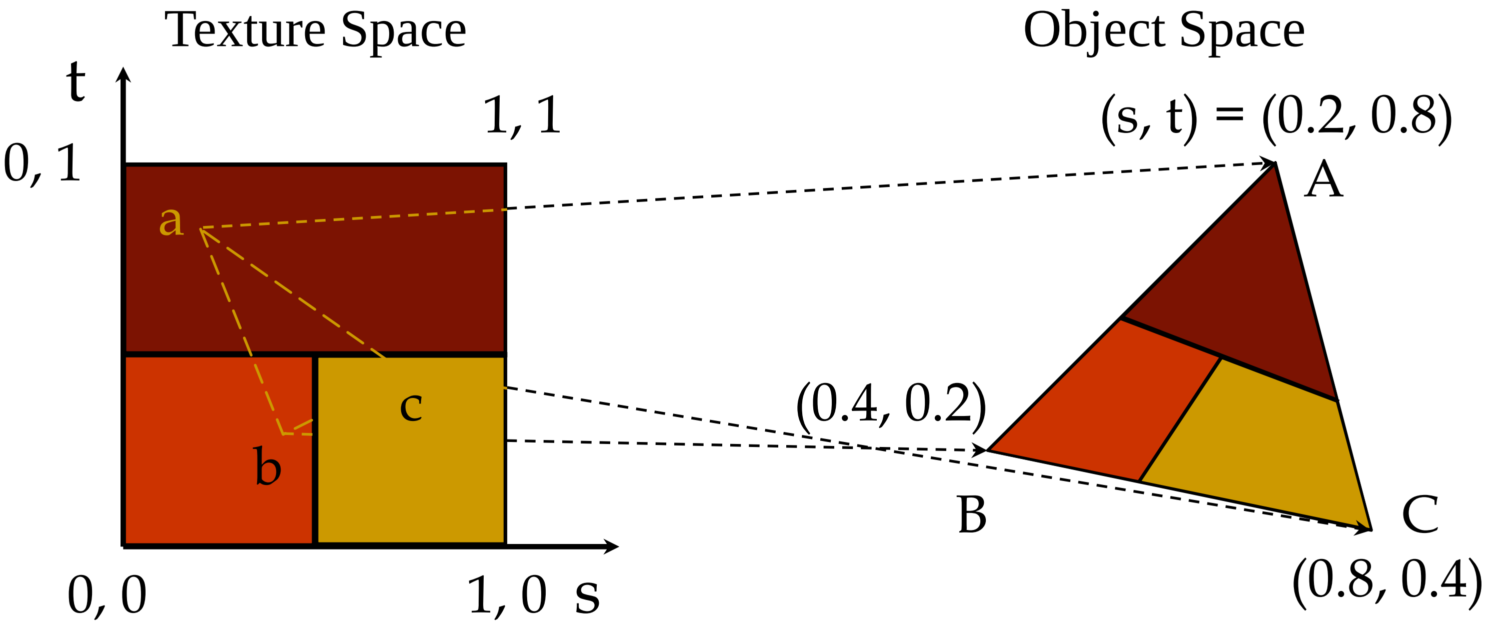
- Three steps
  - ① specify texture
    - read or generate image
    - assign to texture
    - enable texturing
  - ② assign texture coordinates to vertices
  - ③ specify texture parameters
    - wrapping, filtering

Applying Textures

- specify textures in texture objects
- set texture filter
- set texture function
- set texture wrap mode
- set optional perspective correction hint
- bind texture object
- enable texturing
- supply texture coordinates for vertex
  - coordinates can also be generated

Mapping a Texture

- Based on parametric texture coordinates
- **`glTexCoord*()`** specified at each vertex



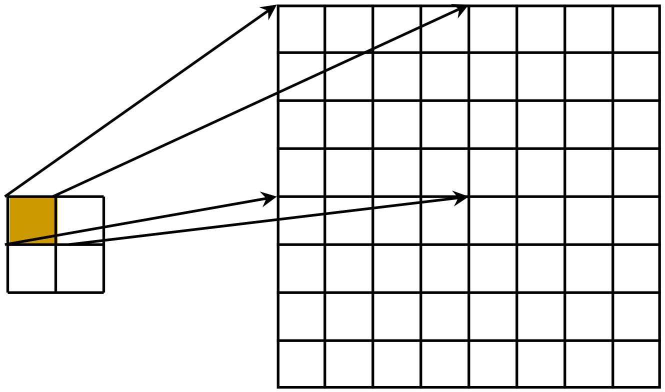
Texture Application Methods

- Filter Modes
  - minification or magnification
  - special mipmap minification filters
- Wrap Modes
  - clamping or repeating
- Texture Functions
  - how to mix primitive’s color with texture’s color
    - blend, modulate or replace texels

Filter Modes

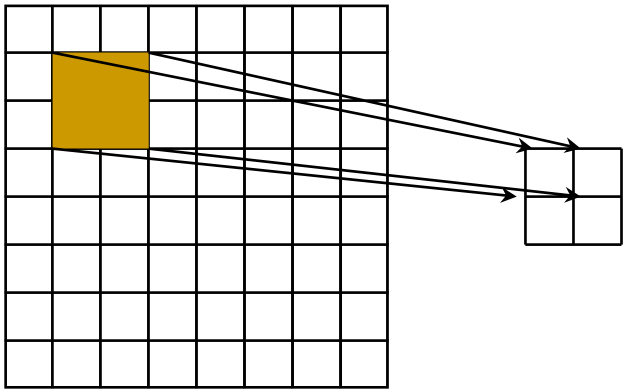
Example:

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



Texture Polygon

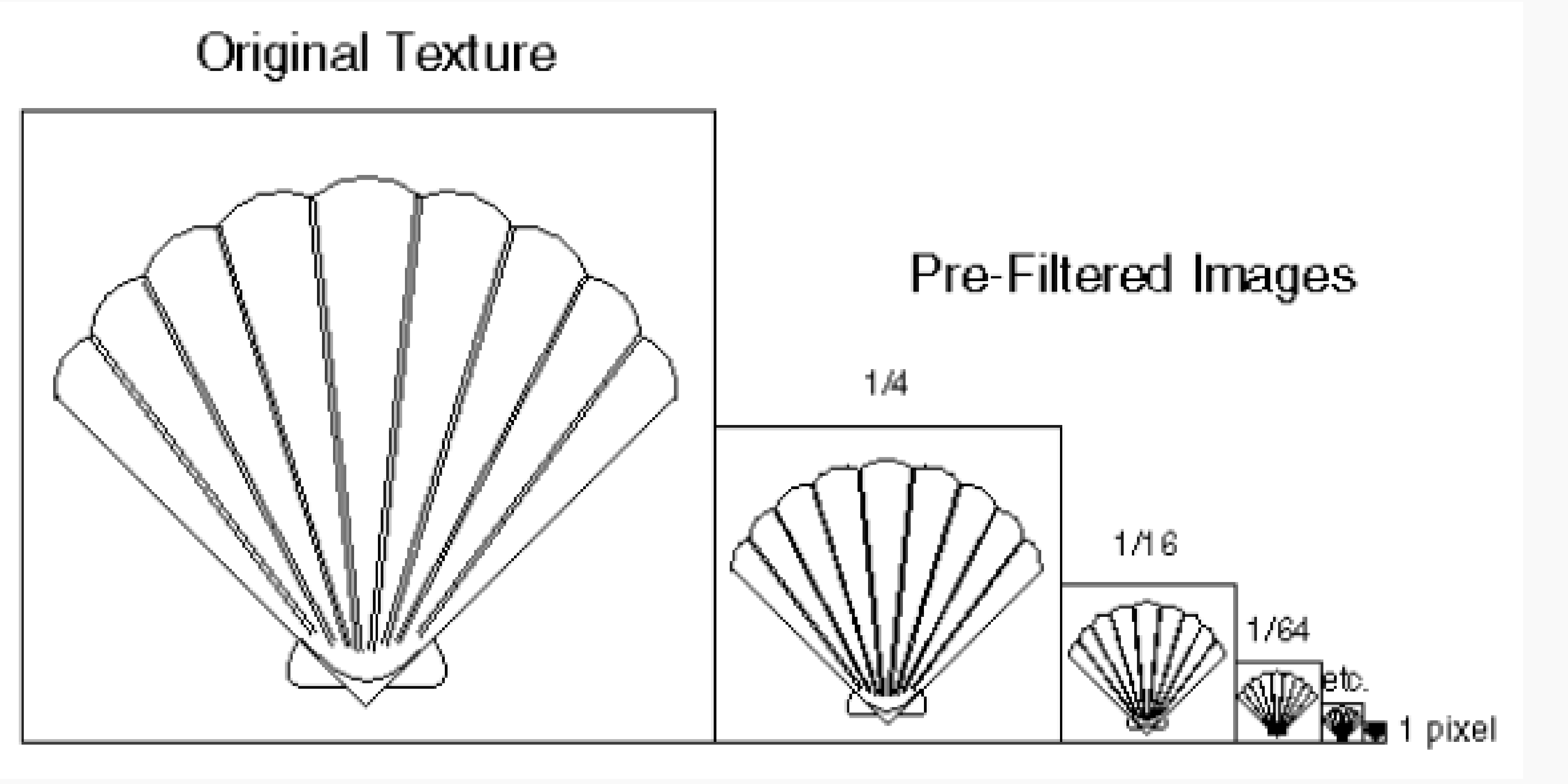
Magnification



Texture Polygon

Minification

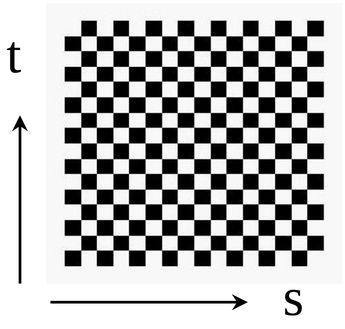
Mipmaps



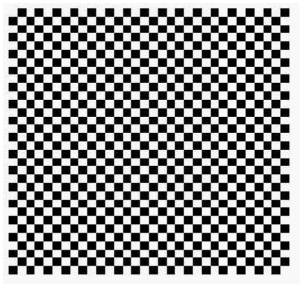
Wrapping Mode

- Example:

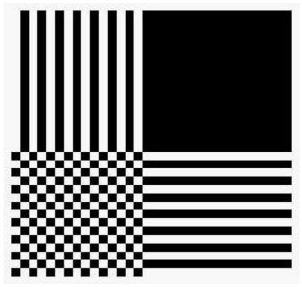
```
glTexParameteri( GL_TEXTURE_2D,
     GL_TEXTURE_WRAP_S, GL_CLAMP )
glTexParameteri( GL_TEXTURE_2D,
     GL_TEXTURE_WRAP_T, GL_REPEAT )
```



texture



GL_REPEAT wrapping



GL_CLAMP wrapping

Texture Functions

- Controls how texture is applied
- `glTexEnv{fi}[v](` ***`GL_TEXTURE_ENV, prop, param`*** `)`
- ***`GL_TEXTURE_ENV_MODE`*** modes
  - **`GL_MODULATE`**
  - **`GL_BLEND`**
  - **`GL_REPLACE`**
- Set blend color with ***`GL_TEXTURE_ENV_COLOR`***

Applying Textures

1. Load your image
2. Establish a texture content

```
glBindTexture(GL_TEXTURE_2D, 1);
```

3. Generate mipmaps and load the texture

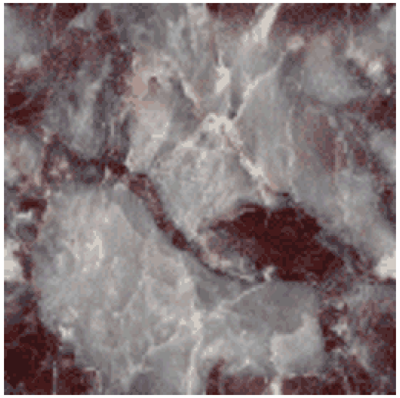
```
gluBuild2DMipmaps();
```

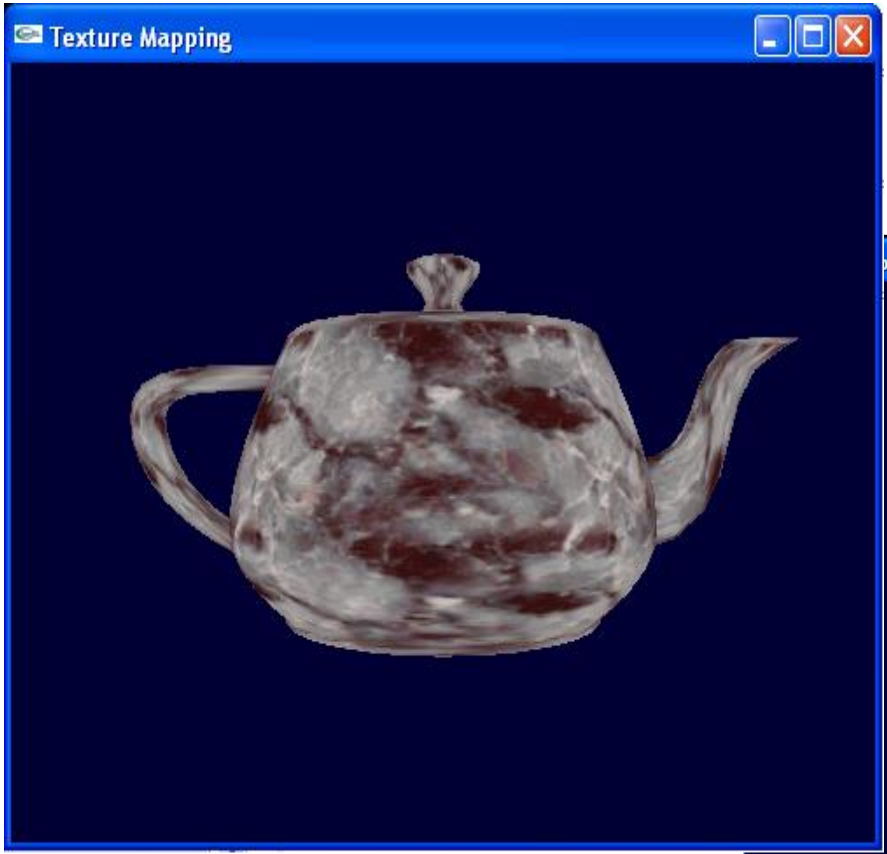
Remember that texture coordinates go [0,1] in both s and t.

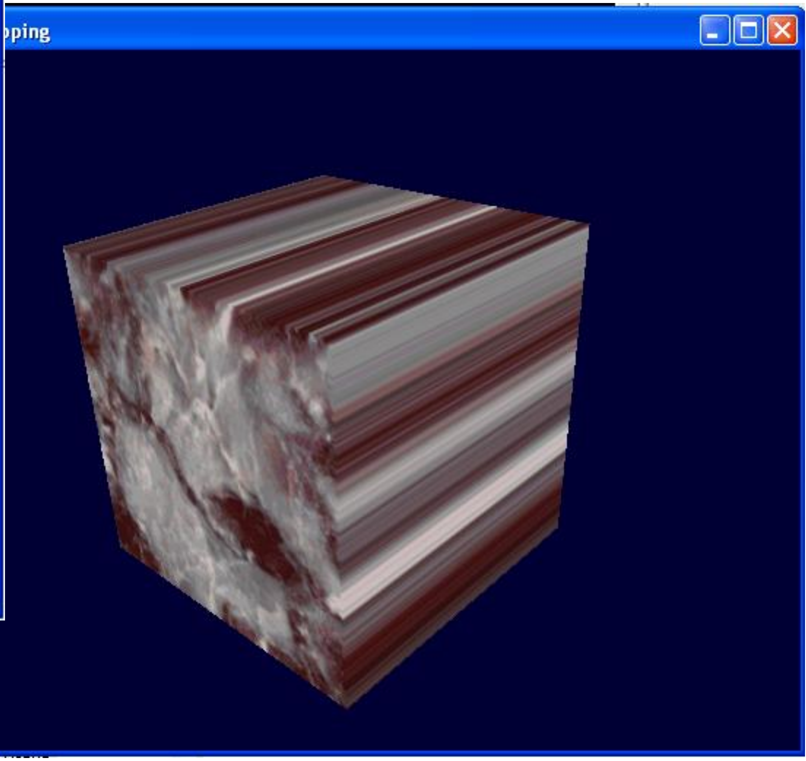
Setting the texture in OpenGL

```
gluBuild2DMipmaps(
     GL_TEXTURE_2D, //target
     3,             //components
     texwidth,      //dimensions of the
     texheight,     //texture
     GL_RGB,        //format
     GL_UNSIGNED_BYTE,  //type
     texdata);
     //pointer to the actual texture data
```

Texture Example

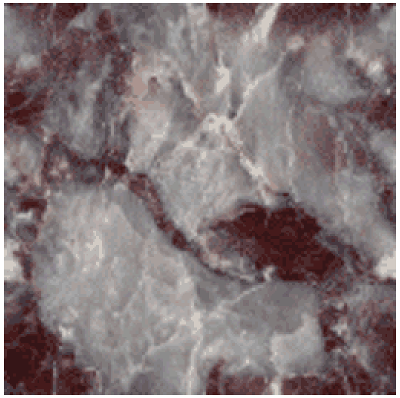


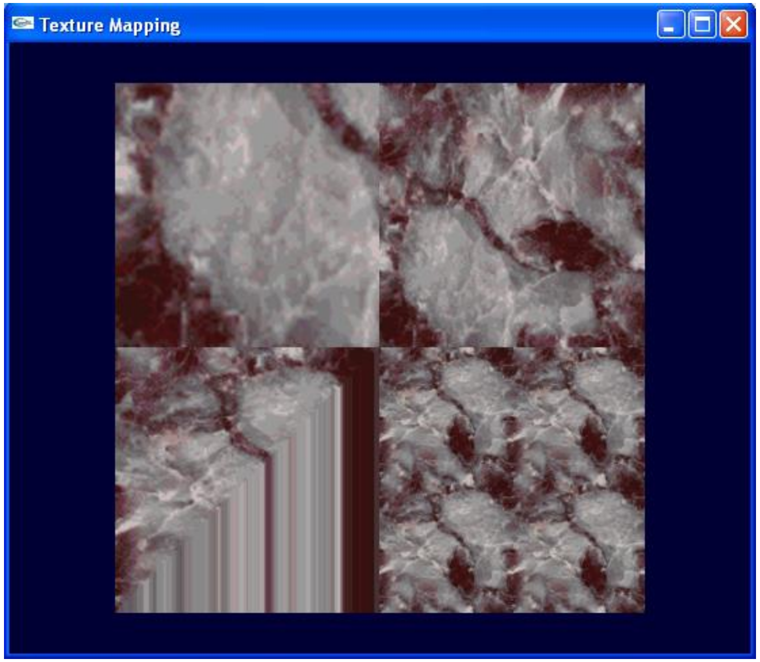




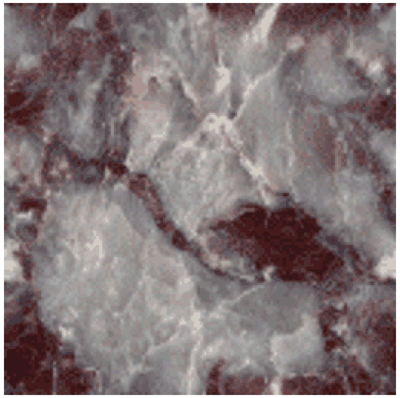
Texture Coordinates

```
glBegin(GL_POLYGON);

  glTexCoord2f(1.0,1.0);
  glVertex3f(10.0, 10.0, -20.0);

  glTexCoord2f(0.0,1.0);
  glVertex3f(0.0, 10.0, -20.0);

  glTexCoord2f(0.0,0.0);
  glVertex3f(0.0, 0.0, -20.0);

  glTexCoord2f(1.0,0.0);
  glVertex3f(10.0, 0.0, -20.0);

glEnd();
```

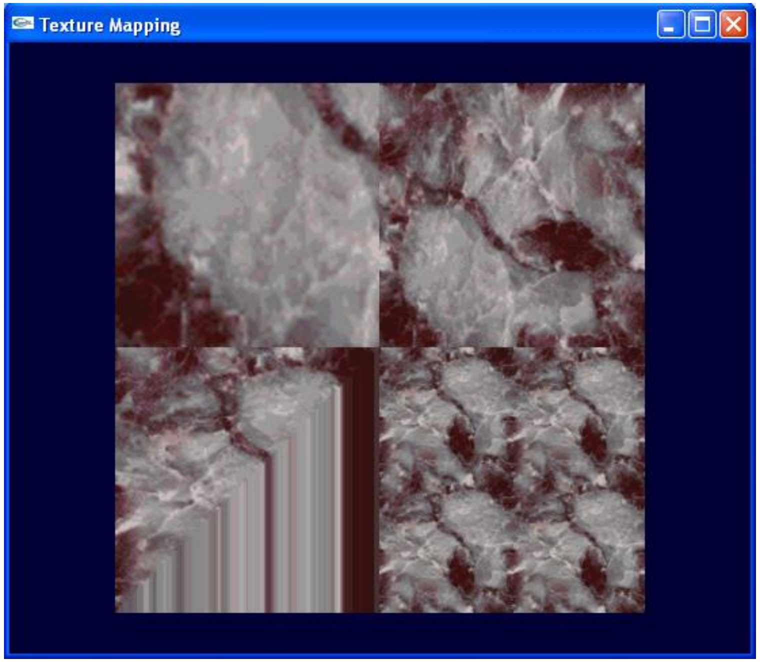




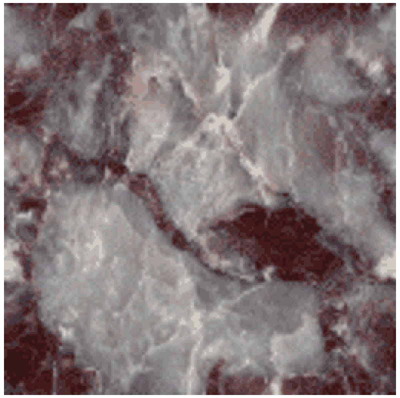
Texture Coordinates



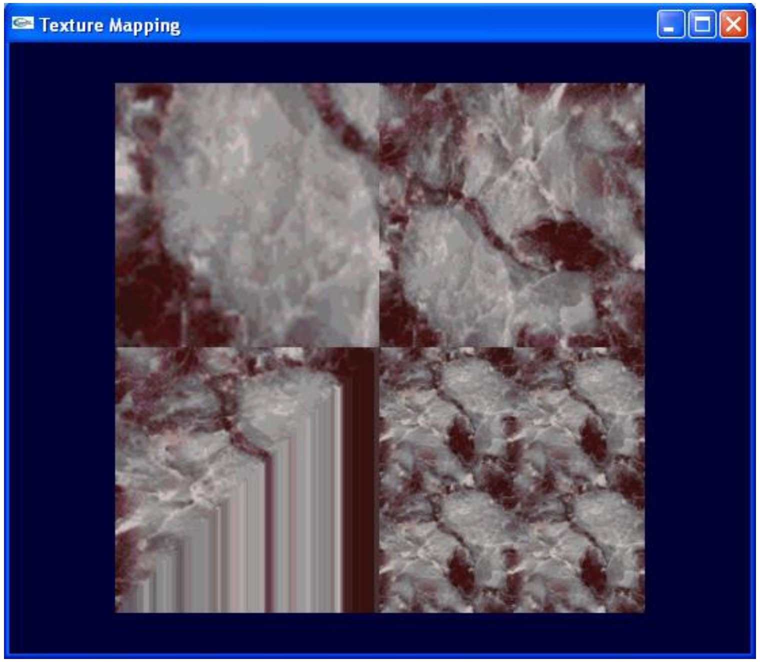
```
glBegin(GL_POLYGON);

   glTexCoord2f(0.5,0.5);
   glVertex3f(0.0, 10.0, -20.0);

   glTexCoord2f(0.0,0.5);
   glVertex3f(-10.0, 10.0, -20.0);

   glTexCoord2f(0.0,0.0);
   glVertex3f(-10.0, 0.0, -20.0);

   glTexCoord2f(0.5,0.0);
   glVertex3f(0.0, 0.0, -20.0);

glEnd();
```



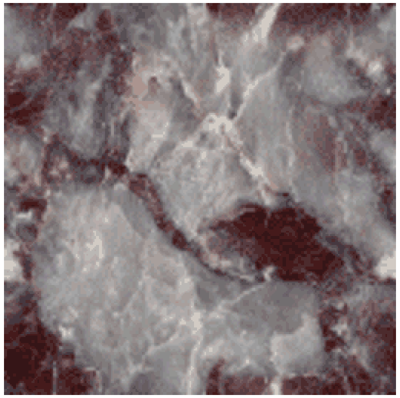
Texture Coordinates



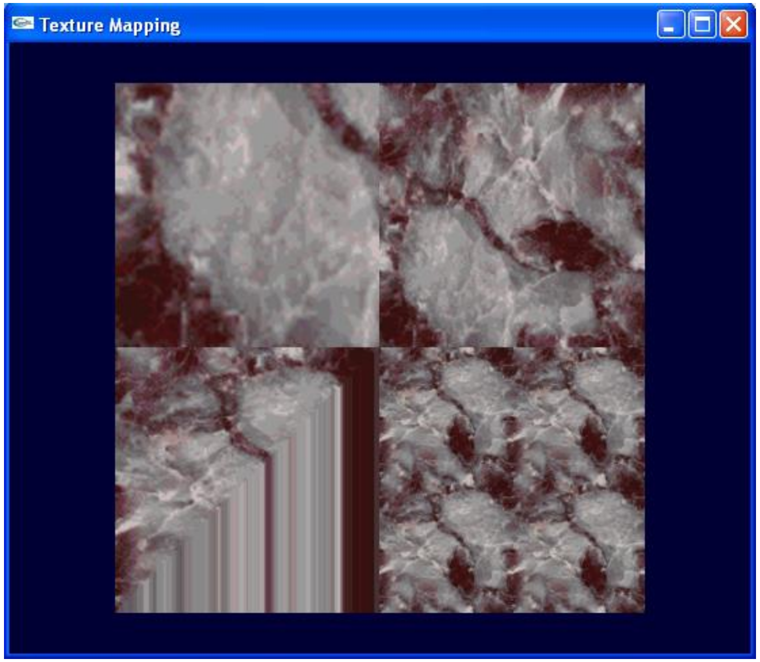
```
glBegin(GL_POLYGON);

  glTexCoord2f(0.0,0.0);
  glVertex3f(0.0, 0.0, -20.0);

  glTexCoord2f(0.0,1.0);
  glVertex3f(-10.0, 0.0, -20.0);

  glTexCoord2f(1.0,1.0);
  glVertex3f(-10.0, -10.0, -20.0);

  glTexCoord2f(0.0,0.0);
  glVertex3f(0.0, -10.0, -20.0);

glEnd();
```



Texture Coordinates



```
glBegin(GL_POLYGON);

  glTexCoord2f(0.0,0.0);
  glVertex3f(10.0, 0.0, -20.0);

  glTexCoord2f(0.0,2.0);
  glVertex3f(0.0, 0.0, -20.0);

  glTexCoord2f(2.0,2.0);
  glVertex3f(0.0, -10.0, -20.0);

  glTexCoord2f(2.0,0.0);
  glVertex3f(10.0, -10.0, -20.0);

glEnd();
```



Texture Example

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

int num_texture=-1;
//Counter to keep track of the last loaded texture
int id_texture;

int screen_width=640;
int screen_height=480;
int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB |
    GLUT_DEPTH);
    glutInitWindowSize(screen_width,screen_height);
    glutInitWindowPosition(0,0);
    glutCreateWindow("Texture Mapping");
    glutDisplayFunc(display);
    glutReshapeFunc (resize);
    init();
    glutMainLoop();

    return(0);
}
```

```
void resize (int width, int height)
{
    screen_width=width;
    screen_height=height;

    glClear (GL_COLOR_BUFFER_BIT |
    GL_DEPTH_BUFFER_BIT);
    glViewport(0,0,screen_width,screen_height);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60.0f,
          (GLfloat)screen_width/(GLfloat)screen_height,
          1.0f,100.0f);
    glutPostRedisplay ();
}
```

```
void  init(void) {
    // This clear the background color to dark blue
    glClearColor(0.0, 0.0, 0.2, 0.0);

    glShadeModel(GL_FLAT); // Type of shading for the polygons
    glViewport(0,0,screen_width,screen_height);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

     gluPerspective(45.0f,(GLfloat)screen_width/(GLfloat)screen_height,1.0f,1000.0f);

    glEnable(GL_DEPTH_TEST);
    glPolygonMode (GL_FRONT_AND_BACK, GL_FILL);

    glEnable(GL_TEXTURE_2D); // This Enable the Texture mapping

    id_texture=LoadBitmap("texture1.bmp");
    if (id_texture==-1)
    {
        MessageBox(NULL,"Image file: texture1.bmp not found",
                                            "Warning", MB_OK | MB_ICONERROR);
        exit (0);
    } }
```

```
void  display(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    glBindTexture(GL_TEXTURE_2D, id_texture);
    glBegin(GL_POLYGON);
            glTexCoord2f(1.0,1.0);
            glVertex3f(10.0, 10.0, -20.0);

            glTexCoord2f(0.0,1.0);
            glVertex3f(0.0, 10.0, -20.0);

            glTexCoord2f(0.0,0.0);
            glVertex3f(0.0, 0.0, -20.0);

            glTexCoord2f(1.0,0.0);
            glVertex3f(10.0, 0.0, -20.0);
     glEnd();
glFlush();
}
```

```
//This function loads a bitmap file and return the OpenGL
//reference ID to use that texture

int LoadBitmap(char *filename)
{
   int i, j=0; //Index variables
   FILE *l_file; //File pointer
   unsigned char *l_texture;
          //The pointer to the memory zone in which we will load the texture

   // windows.h gives us these types to work with the Bitmap files
   BITMAPFILEHEADER fileheader;
   BITMAPINFOHEADER infoheader;
   RGBTRIPLE rgb;

   num_texture++; //The counter of the current texture is increased

   if( (l_file = fopen(filename, "rb"))==NULL) return (-1);
          // Open the file for reading
```

```
    fread(&fileheader, sizeof(fileheader), 1, l_file); // Read the fileheader

    fseek(l_file, sizeof(fileheader), SEEK_SET); // Jump the fileheader
    fread(&infoheader, sizeof(infoheader), 1, l_file); // and read the infoheader

// Now we need to allocate the memory for our image (width * height * color deep)
    l_texture = (byte *) malloc(infoheader.biWidth * infoheader.biHeight * 4);

// And fill it with zeros
    memset(l_texture, 0, infoheader.biWidth * infoheader.biHeight * 4);

 // At this point we can read every pixel of the image
    for (i=0; i < infoheader.biWidth*infoheader.biHeight; i++)
    {
            // We load an RGB value from the file
            fread(&rgb, sizeof(rgb), 1, l_file);

            // And store it
            l_texture[j+0] = rgb.rgbtRed; // Red component
            l_texture[j+1] = rgb.rgbtGreen; // Green component
            l_texture[j+2] = rgb.rgbtBlue; // Blue component
            l_texture[j+3] = 255; // Alpha value
            j += 4; // Go to the next position
    }
```

```
fclose(l_file); // Closes the file stream

// Bind the ID texture specified by the 2nd parameter
glBindTexture(GL_TEXTURE_2D, num_texture);

// The next commands sets the texture parameters
// If the u,v coordinates overflow the range 0,1 the image is repeated
glTexParameterf(GL_TEXTURE_2D,
                              GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameterf(GL_TEXTURE_2D,
                              GL_TEXTURE_WRAP_T, GL_REPEAT);
 // The magnification function ("linear" produces better results)
glTexParameterf(GL_TEXTURE_2D,
                   GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameterf(GL_TEXTURE_2D,
                   GL_TEXTURE_MIN_FILTER,
                   GL_LINEAR_MIPMAP_NEAREST);
// We don't combine the color with the original surface color,
//use only the texture map.
glTexEnvf(GL_TEXTURE_ENV,
                   GL_TEXTURE_ENV_MODE, GL_REPLACE);
```

```
    // Finally we define the 2d texture
    glTexImage2D(GL_TEXTURE_2D, 0, 4, infoheader.biWidth,
     infoheader.biHeight, 0, GL_RGBA, GL_UNSIGNED_BYTE,
     l_texture);

    // And create 2d mipmaps for the minifying function
    gluBuild2DMipmaps(GL_TEXTURE_2D, 4, infoheader.biWidth,
     infoheader.biHeight, GL_RGBA, GL_UNSIGNED_BYTE,
     l_texture);

    free(l_texture);
            // Free the memory we used to load the texture

    return (num_texture);
            // Returns the current texture OpenGL ID
}
```