CS 432 – Interactive Computer Graphics

Lecture 7 – Part 2

Texture Mapping in OpenGL

Topics

- Texture Mapping in OpenGL

Reading

- Angel
  - Chapter 7
- Red Book
  - Chapter 6
  - Chapter 8

Texture Mapping in OpenGL

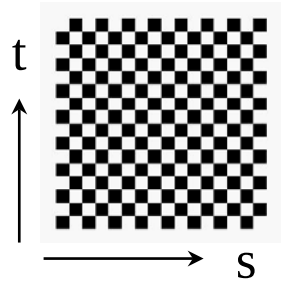
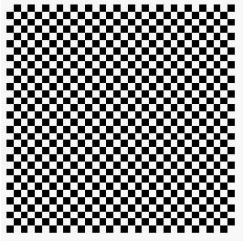
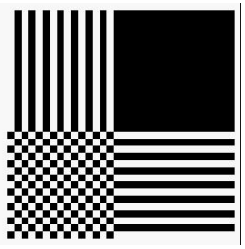
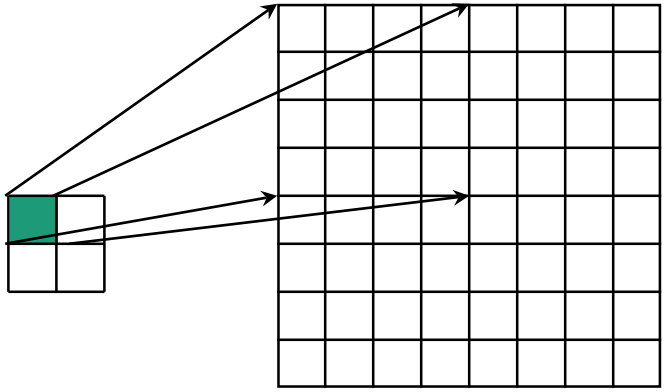
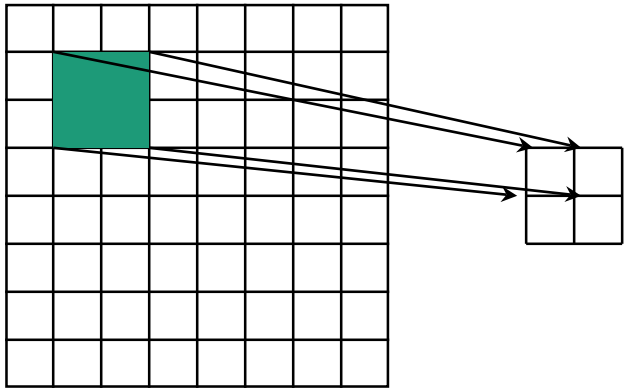
- Basic Strategy:
  1. Get the texture data
     - Read or generate image
  2. As GPU for a texture
  3. Make a texture active (bind it) and move data to that texture (put on GPU)
  4. Specify texture parameters
     - Wrapping, filtering
  5. Assign texture coordinates to vertices
     - Proper mapping function is left to application
  6. Draw the scene, making texture units active and associating textures with texture units as necessary.

Design Ideas

- You may want to include with the object (static or not?)
  - Texture coordinates
  - Texture

Create Texture Object

- Texel values can be up to 4D (RGBA)
- Loading textures is **expensive**
- Faster to bind (`glBindTexture`) then to load (`glTexImage*D`)
  - So load once, and reuse/bin when needed
- Textures are similar to VBOs and VAOs
  - Create texture names:
    - ```
      GLuint texName[4];
      glGenTextures(4, texName);
      ```
  - Enable/activate a texture object
    - `glBindTexture(GL_TEXTURE_2D, texName[0]);`
      ```

Multi-Texturing

- We can use *multi-texturing* to allow for several textures per object.
- This is done by assigning textures to *texture units*
- Think of texture locations a matrix:

GL_TEXTURE0	**GL_TEXTURE1**	**…**	**GL_TEXTUREN**
GL_TEXTURE_2D	GL_TEXTURE_2D	…	GL_TEXTURE_2D
GL_TEXTURE_CUBE_MAP	GL_TEXTURE_CUBE_MAP	…	GL_TEXTURE_CUBE_MAP

Multi-Texturing

- When drawing, we do the following:
  - We must first select/activate a texture unit:
    - `glActiveTexture(GL_TEXTURE0);`
  - Next we must make sure that the texture type we want within that unit is active
    - `glEnable(GL_TEXTURE_2D)`
  - Finally we just associate a texture with the texture type in that texture unit
    - `glBindTexture(GL_TEXTURE_2D, textures[0]);`

Step 1: Get Texture Data

- We define a texture image from an array of *texels* (texture elements) in CPU memory
  - `GLubyte my_texels[512][512][3]`
- We may populate this texture image by
  - Manually/systematically providing values
  - Loading from image

Step 1: Create Data

```
GLubyte image[64][64][3];


//Create a 64x64 checkerboard pattern
for(int i=0; i<64; i++){
	for(int j=0; j<64; j++){
		GLubyte c = (((i&(0x8)==0)^((j&0x8)==0))*255;
		image[i][j][0] = c;
		image[i][j][1] = c;
		image[i][j][2] = c;
	}
}
```

Step 1: Load Data

- For simplicity in this class to load texture from images let's
  1. Convert an image to a portable pix map (PPM) using an online utility:
     - http://ziin.pl/en/utilities/convert/
  2. Load the PPM into an unsigned byte array. Code provided for you in the Drawable class in the sample code:

```
static unsigned char* ppmRead(char* filename,
        int* width, int* height);
```

Step 2: Bind Data to Texture Unit

- We must first request from GPU a list of available textures
  - `getGenTextures(3,textures)`
- Then we must make active the texture we want
  - `glBindTexture(GL_TEXTURE_2D, textures[2]);`
- Next we must move the data into the texture
  - `glTexImage2D(target, level, components, w, h, border, format, type, texels)`
    - `target`: Type of texture, e.g. `GL_TEXTURE_2D`
    - `level`: Used for mipmapping (discussed later)
    - `components`: # of elements per texel (RGB, etc..)
    - `w`, `h`: Width and Height of texture in pixels
    - `border`: Use for smoothing (typically 0 or 1)
    - `format` and `type`: Describe texels
    - `texels`: Pointer to texel array
  - EX:

```
glTexImage2D(GL_TEXTURE_2D,0,GL_RGB,512,512,0,
                    GL_RGB,GL_UNSIGNED_BYTE,my_texels)
```

Step 3: Texture Parameters

- OpenGL has a variety of parameters that determine how texture is applied
  - Wrapping parameters determine what happens if *s* and *t* are outside the (0,1) range
  - Filter mode allows us to use area averaging instead of point samples
  - Mipmapping allows us to use textures at multiple resolutions
  - Environment parameters determine how texture mapping interacts with shading

Wrapping Mode

- Clamping: If $s, t > 1$ use 1, if $s, t < 0$ use 0
  - `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP;`
- Repeating: Use $s, t \; modulo \; 1$
  - `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);`



texture



GL_REPEAT wrapping



GL_CLAMP wrapping

Magnification and Minification

- One texel can cover more than one pixel (magnification) or one pixel can be covered by more than one texel (minification).
- To specify how to deal with this, for each situation (OpenGL detects if either situation occurs) we can specify:
  - Use point sampling (nearest texel)
  - Use linear filtering (2x2 filter) to obtain texture values



Texture Polygon

Magnification



Texture Polygon

Minification

Filter Modes

- Modes determined by
  - `glTexParameteri(target, type, mode)`
- Examples:
  - `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);`
  - `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);`
- Note: Linear filtering requires a border of an extra texel for filtering at edges (`border = 1`)

Mipmapped Textures

- Mipmapping allows for prefiltered texture maps of decreasing resolutions
- Lessens interpolation errors for smaller textured objects
  - Probably want higher resolution images for closer objects, lower resolution for further objects
- Declare mipmap level during texture definition
  - `glTexImage2D(GL_TEXTURE_*D, level, …)`
- Alternatively allow OpenGL to make the mipmaps
  - `glGenerateMipmaps(GL_TEXTURE_2D);`
  - ```
    glTexParameterf(GL_TEXTURE_2D,
                    GL_TEXTURE_MAG_FILTER,
                    GL_NEAREST_MIPMAP_NEAREST);
    ```
    ```

Step 4: Assign Texture Coordinates

- We talked about different ways in Part 1
- Let's look at the simple plane->plane mapping:

```
void Cube::quad2Triangles(vec4 v1, vec4 v2, vec4 v3, vec4 v4){
    //triangle 1
    vec3 N = normalize(cross(v2-v1,v3-v1));
    vertices[index]=v1;  normals[index] = N;    textures[index] = vec2(0.0,0.0);    index++;
    vertices[index]=v2;  normals[index] = N;    textures[index] = vec2(1.0,0.0);    index++;
    vertices[index]=v3;  normals[index] = N;    textures[index] = vec2(1.0,1.0);    index++;

    N = normalize(cross(v3-v1,v4-v1));
    vertices[index]=v1;  normals[index] = N;    textures[index] = vec2(0.0,0.0);    index++;
    vertices[index]=v3;  normals[index] = N;    textures[index] = vec2(1.0,1.0);    index++;
    vertices[index]=v4;  normals[index] = N;    textures[index] = vec2(0.0,1.0);    index++;
}
```

Step 5: Rendering with Textures

- Just need to
  - Make sure texture location attributes are linked and enabled
    - `vTexture = glGetAttribLocation(program, “vTexCoord”);`
    - `glEnableVertexAttribArray(vTexture);`
    - `glVertexAttribPointer(vTexture,2,GL_FLOAT, GL_FALSE,0, BUFFER_OFFSET(sizeof(vertices)+ sizeof(normals)));`
  - Make sure desired texture unit is active (for drawing)
    - `glActiveTexture(GL_TEXTURE0);`
    - `glEnable(GL_TEXTURE_2D)`
    - `glBindTexture(GL_TEXTURE_2D,texture);`
    - `glUniform1i(glGetUniformLocation(program,”texture”),0);`

Shaders

- Vertex Shader
  - Often just pass the texture attribute through to fragment shader
- Fragment Shader
  - Use a texture “sampler” to get the color from the current texture at the specified texture location

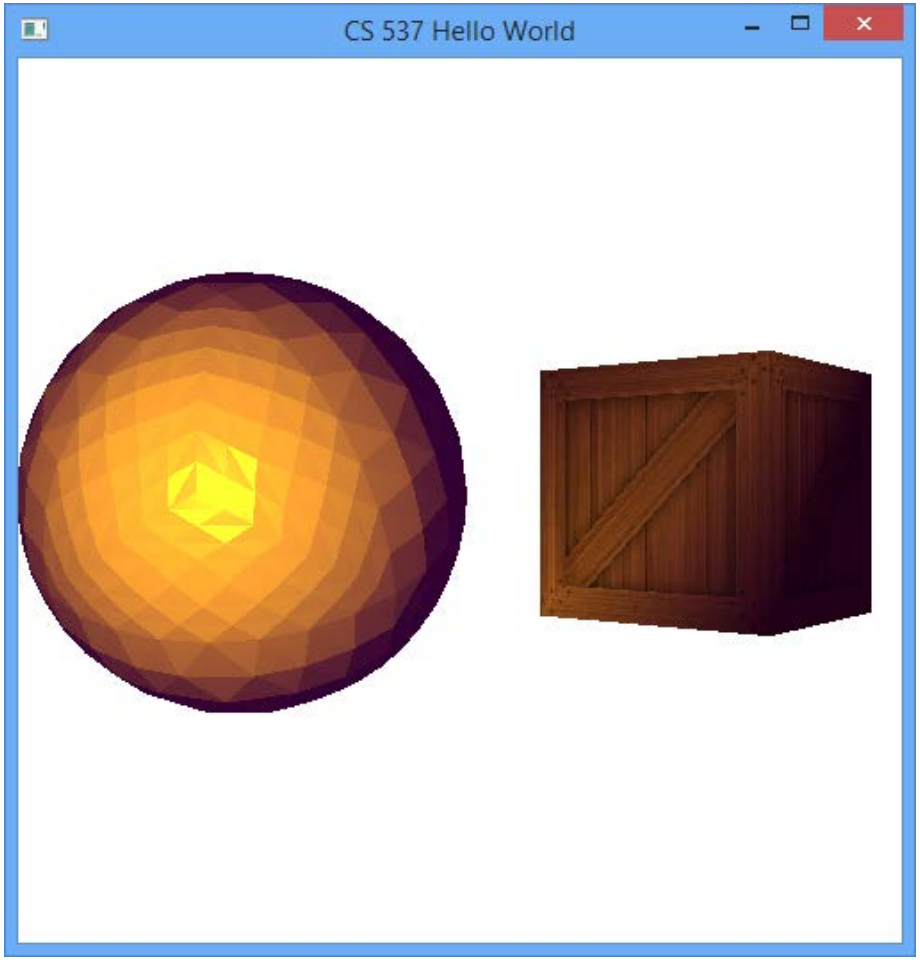
Vertex Shader

```
in vec4 vPosition;  //vertex position in object cords
in vec4 vColor; //vertex color from application
in vec2 vTexCoord; //texture coord from app

out vec4 color;  //output color to be interpolated
out vec2 texCoord; //output tex coordinate to be interpolated
```

Fragment Shader

```
in vec4 color;  //color from rasterizer
in vec2 texCoord; //texture coordinate from rasterizer
uniform sampler2D textureID; //texture object from application;
out vec4 fColor;


void main(){
        fColor = color*texture(textureID,texCoord);
}
```

Example: Crate = Lights + Texture



Crate Example: Building Object

```
CubeTextured::CubeTextured() {
    glGenBuffers(1, &VBO);
    glGenVertexArrays(1, &VAO);
    program = InitShader("../vshader09_texture_v150.glsl", "../fshader09_texture_v150.glsl");

    index = 0;
    TextureSize = 512;

    build();

    glBindVertexArray(VAO);
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertexLocations) + sizeof(vertexNormals) + sizeof(vertexTextureCoords),
        NULL, GL_STATIC_DRAW);
    glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(vertexLocations), vertexLocations);
    glBufferSubData(GL_ARRAY_BUFFER, sizeof(vertexLocations), sizeof(vertexNormals), vertexNormals);
    glBufferSubData(GL_ARRAY_BUFFER, sizeof(vertexLocations) + sizeof(vertexNormals), sizeof(vertexTextureCoords),
        vertexTextureCoords);

    GLuint vPosition = glGetAttribLocation(program, "vPosition");
    glEnableVertexAttribArray(vPosition);
    glVertexAttribPointer(vPosition, 4, GL_FLOAT, GL_FALSE, 0, BUFFER_OFFSET(0));

    GLuint vNormal = glGetAttribLocation(program, "vNormal");
    glEnableVertexAttribArray(vNormal);
    glVertexAttribPointer(vNormal, 3, GL_FLOAT, GL_FALSE, 0, BUFFER_OFFSET(sizeof(vertexLocations)));

    GLuint vTex = glGetAttribLocation(program, "vTexCoord");
    glEnableVertexAttribArray(vTex);
    glVertexAttribPointer(vTex, 2, GL_FLOAT, GL_FALSE, 0, BUFFER_OFFSET(sizeof(vertexLocations) + sizeof(vertexNormals)));
```

Crate Example : Building (cont)

- Continued…

```
    glGenTextures(1, &texture);
    GLubyte *image0 = ppmRead("../crate_texture.ppm", &TextureSize, &TextureSize);
    glBindTexture(GL_TEXTURE_2D, texture);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, TextureSize, TextureSize, 0, GL_RGB, GL_UNSIGNED_BYTE, image0);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);

}
```

Crate Example: Coords

```
void CubeTextured::build(){
        makeQuad(1,0,3,2);  //front
        makeQuad(2,3,7,6);  //right
        makeQuad(3,0,4,7);  //bottom
        makeQuad(6,5,1,2);  //top
        makeQuad(4,5,6,7);  //back
        makeQuad(5,4,0,1);  //left
}
```

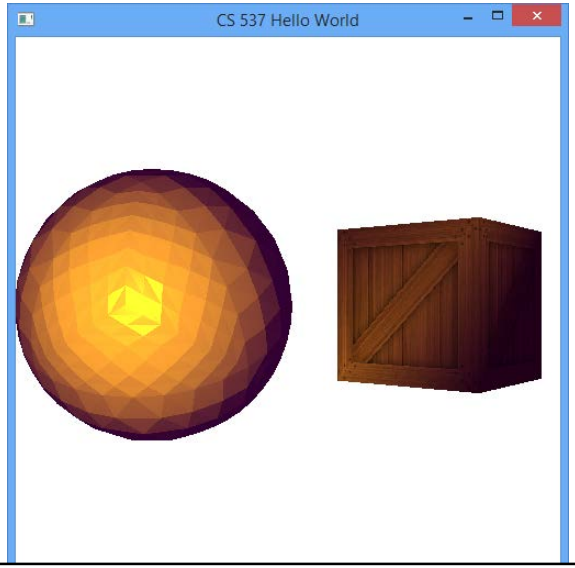
```
void CubeTextured::makeQuad(GLuint ind1, GLuint ind2, GLuint ind3, GLuint ind4){  //assume vertices in counter-clockwise order from top-left
    vec4 data[] = {vec4(-0.5,-0.5,0.5,1.0),vec4(-0.5,0.5,0.5,1.0),vec4(0.5,0.5,0.5,1.0),vec4(0.5,-0.5,0.5,1.0),
                   vec4(-0.5,-0.5,-0.5,1.0),vec4(-0.5,0.5,-0.5,1.0),vec4(0.5,0.5,-0.5,1.0),vec4(0.5,-0.5,-0.5,1.0)};

    //first triangle
    vec3 N = normalize(cross(data[ind2]-data[ind1],data[ind3]-data[ind1]));
    vertexLocations[index] = data[ind1]; vertexNormals[index] = N; vertexTextureCoords[index] = vec2(0,0); index++;
    vertexLocations[index] = data[ind2]; vertexNormals[index] = N; vertexTextureCoords[index] = vec2(1,0); index++;
    vertexLocations[index] = data[ind3]; vertexNormals[index] = N; vertexTextureCoords[index] = vec2(1,1); index++;

    //second triangle
    N = normalize(cross(data[ind3] - data[ind1],data[ind4]-data[ind1]));
    vertexLocations[index] = data[ind3]; vertexNormals[index] = N; vertexTextureCoords[index] = vec2(0,0); index++;
    vertexLocations[index] = data[ind4]; vertexNormals[index] = N; vertexTextureCoords[index] = vec2(1,1); index++;
    vertexLocations[index] = data[ind1]; vertexNormals[index] = N; vertexTextureCoords[index] = vec2(0,1); index++;

}
```

Crate Example: Draw Cube

```
void CubeTextured::draw(Camera cam, vector<Light> lights){
    glBindVertexArray(VAO);
    glUseProgram(program);

    GLuint light_loc = glGetUniformLocation(program, "lightPos");
    glUniform4fv(light_loc, 1, cam.getEye());

    GLuint diffuse_loc = glGetUniformLocation(program,"diffuseProduct");
    glUniform4fv(diffuse_loc,1,diffuse*lights[0].getDiffuse());

    GLuint spec_loc = glGetUniformLocation(program,"specularProduct");
    glUniform4fv(spec_loc,1,specular*lights[0].getSpecular());

    GLuint ambient_loc = glGetUniformLocation(program,"ambientProduct");
    glUniform4fv(ambient_loc,1,ambient*lights[0].getAmbient());

    GLuint alpha_loc = glGetUniformLocation(program,"alpha");
    glUniform1f(alpha_loc,shininess);

    GLuint model_loc = glGetUniformLocation(program,"model_matrix");
    glUniformMatrix4fv(model_loc,1,GL_TRUE,modelmatrix);

    GLuint view_loc = glGetUniformLocation(program, "camera_matrix");
    glUniformMatrix4fv(view_loc, 1, GL_TRUE, cam.getViewMatrix());

    GLuint proj_loc = glGetUniformLocation(program, "proj_matrix");
    glUniformMatrix4fv(proj_loc, 1, GL_TRUE, cam.getProjectionMatrix());

    glEnable(GL_TEXTURE_2D);
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, texture);
    glUniform1i(glGetUniformLocation(program,"textureID"),0);

    glDrawArrays(GL_TRIANGLES,0,numVertices);
}
```

Crate Example: Shaders

```
#version 150

in vec4 vPosition;
in vec3 vNormal;
in vec2 vTexCoord;

out vec4 color;
out vec2 texCoord;

uniform mat4 model_matrix;
uniform mat4 camera_matrix;
uniform mat4 proj_matrix;


uniform vec4 lightPos;
uniform vec4 ambientProduct,
      diffuseProduct, specularProduct;
uniform float alpha;


void main()
{
      texCoord = vTexCoord;
      //Then all the lighting effects
      // to compute output color…..
      //….
}
```




```
#version 150

in vec4 color;
in vec2 texCoord;

uniform sampler2D textureID;

out vec4 fColor;

void main()
{
    fColor = texture(textureID,texCoord)*color;

}
```