CSC4820/6820 Interactive Computer Graphics

Fall 2014

Homework 4

Due date: 11:59 pm December 11, 2014 (Thursday)

The aim of this homework is to help improve your understanding the process of texture mapping. This homework is closely related to your project 4 coding.

1. Answer the following questions. Use the sample code attached to the project 4 requirements as an example. If possible, use code segments as examples to answer the questions.

* Briefly describe the steps of texture mapping in OpenGL/GLSL. Use code segments as examples if possible.

1. create an OpenGL texture object
2. The shader setup requires a uniform to hold the texture unit. The shader receives the texture unit, not the texture name. OpenGL provides multiple units so we can use more than one texture in a shader. Lets assume that the shader declares a uniform named textureMap0. Also assume that we only have one texture, which we are going to bind to texture unit 0.
3. In the OpenGL side of the application, in the setup step, we need to retrieve the location of the uniform textureUnit0. As far as OpenGL is concerned textureUnit0 is an integer. So, assuming that variable program stores the name of the GLSL program, the code to retrieve the location is as follows

Get the ID of the texture source uniform block in the shader program

textureUnit0 = glGetUniformLocation(program,"textureMap0");

1. Now that a texture image is transferred from main memory to GPU memory.

// glActiveTexture() indicates which texture unit the texture image will be sent to.

// A texture unit is used to hold an active texture image that is about to be accessed by shaders.

// There are multiple texture units on a GPU, allowing multi-texturing.

// Before "binding" a texture, you must indicate which texture unit to use.

// You also need to link the texture sampler variable in the fragment shader to the correct texture unit.

// In this case, we are using texture unit 0. So in renderScene() function, we call glUniform1i(texUnit, 0) to

// link texUnit (a handle of the "texUnit" variable in the fragment shader) to 0.

// glActiveTexture() function and glUniform1i(textUnit, ...) function calls must be consistent.

// If you change texture unit number in one function all, you must change the texture unit in the other function call.

if (textureObjectIDArray0[meshIndex] > 0) {

glActiveTexture(GL\_TEXTURE0);

glBindTexture(GL\_TEXTURE\_2D, textureObjectIDArray0[meshIndex]);

}

// Set texture filter parameters

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

Now for the last key to texture mapping, texture coordinates which are used to map the textures to the models

How is the job divided between the OpenGL part and the GLSL part? In the GLSL

we must declare the variable. This variable is declared as a sampler\*. Samplers are opaque types, i.e. they can not be accessed as a regular variable, and their usage is restricted to appropriate functions. Samplers allow access to textures defined in the OpenGL side of the application, through functions that will retrieve values from the texture data. Each texture type has an appropriate sampler type, for instance, for texture target GL\_TEXTURE\_2D the sampler type is sampler2D.

To access a texture we need two elements: the texture unit, and the texture coordinates. There are several functions to access texture data. Some of them are queries, others provide the texture data. Let’s start with some queries.

A generic shader can work with textures that may vary in size. Size information (width and height) can be provided to the shader through uniform variables.

Opengl side

textureUnit1 = glGetUniformLocation(program, "textureMap1");

Frag. Shader GLSL side

in vec2 textureCoord;

uniform sampler2D textureMap0;

Consider the following code:

* The key to texture mapping is to establish a mapping between each pixel on a 3D object and a pixel (sometimes called a texel) on an image. How is this mapping established and what kind of tools are used? Mapping is established by accessing the texture unit, and the texture coordinates by using the fragment shader
* What are texture coordinates? How are they related to texture mapping? How to obtain the texture coordinates for each pixel? Where are texture coordinates stored in an ASSIMP aiScene object? Texture coordinates are texture coordinates are in texture space. That is, they are relative to the location (0,0) in the texture. When a texture is applied to a primitive in 3D space, its texel addresses must be mapped into object coordinates. They must then be translated into screen coordinates, or pixel locations. A texture coord is stored in a 2D Array in Assimp..
* // mTextureCoords is different from mVertices or mNormals. It is a 2D array, not a 1D array.
* // So we need to copy it to a 1D texture coordinate array.
* // The first dimension of this array is the texture channel for this mesh.
* // The second dimension is the vertex index number.
* // The number of texture coordinates is always the same as the number of vertices.
* textureCoordArray = (float \*) malloc(sizeof(float) \* 2 \* currentMesh->mNumVertices);
* unsigned int k = 0;
* for (unsigned int j = 0; j < currentMesh->mNumVertices; j++) {
* textureCoordArray[k] = currentMesh->mTextureCoords[0][j].x;
* k++;
* textureCoordArray[k] = currentMesh->mTextureCoords[0][j].y;
* k++;
* }
* // Bind (transfer) the texture coordinate array to the VBO.
* glBufferData(GL\_ARRAY\_BUFFER, sizeof(float) \* 2 \* currentMesh->mNumVertices,
* textureCoordArray, GL\_STATIC\_DRAW);
* // Associate this VBO with the vTextureCoord variable in the vertex shader.
* // The vertex data and the vertex shader must be connected.
* glVertexAttribPointer( vTextureCoord, 2, GL\_FLOAT, GL\_FALSE, 0, BUFFER\_OFFSET(0) );
* glEnableVertexAttribArray( vTextureCoord );
* }
* textureObjectIDArray0 = (unsigned int\*) malloc(sizeof(unsigned int) \* scene->mNumMeshes);
* textureObjectIDArray1 = (unsigned int\*)malloc(sizeof(unsigned int)\* scene->mNumMeshes); //ADDED
* textureObjectIDArray2 = (unsigned int\*)malloc(sizeof(unsigned int)\* scene->mNumMeshes); //ADDED
* How are texture coordinates transferred to a GLSL shader?

// Texture mapping related variables.

float\* textureCoordArray = 0;

transferred to the GLSL shader here

vTextureCoord = glGetAttribLocation( program, "vTextureCoord" );

if (vTextureCoord == -1) {

cout << "There is an error getting the handle of GLSL variable vTextureCoord." << endl;

}

How does GLSL know to interpret the texture coordinates array as 2-component vectors, not a 3-component vectors (like vertex positions or vertex normals)? By taking the 2D Array and placing it in a 1D Array.

In the sample shader programs, why pass the texture coordinates to the vertex shader and then pass them to the fragment shader? The vertex shader is just a pass-through

in vec2 texCoord; // DataOut.texCoord = texCoord; to the Fragment Shader

in Data {

    vec3 normal;

    vec4 eye;

    vec2 texCoord;

Texture coordinates are used to retrieve a color

|  |
| --- |
|  |

* Why not transfer the texture coordinates directly to the fragment shader?
* If a 3D object file is loaded via ASSIMP, how do you find the texture file path information? Located in the MTL file.

After finding the texture file path, how do you load a texture image?

// Get the diffuse texture file path for this material.

aiReturn texFound = currentMaterial->GetTexture(aiTextureType\_DIFFUSE, texIndex, &path);

if (texFound == AI\_SUCCESS) {

string filename = path.data; // get filename

After the texture image is loaded, what OpenGL data structure do you use to store a texture image?

// Use SOIL to load texture image. SOIL will create a texture object for this texture

// image and return the texture object ID.

textureObjectIDArray0[i] = SOIL\_load\_OGL\_texture(filename.c\_str(), SOIL\_LOAD\_AUTO, SOIL\_CREATE\_NEW\_ID, SOIL\_FLAG\_MIPMAPS);

// If the returned texture ID > 0, it means the imaged is loaded successfully.

if (textureObjectIDArray0[i] <= 0) {

* How do you transfer a texture image to a GLSL shader variable?
* // "BindTexture" means that a texture image is transferred from main memory to GPU memory.
* // glActiveTexture() indicates which texture unit the texture image will be sent to.
* // A texture unit is used to hold an active texture image that is about to be accessed by shaders.
* // There are multiple texture units on a GPU, allowing multi-texturing.
* // Before "binding" a texture, you must indicate which texture unit to use.
* // You also need to link the texture sampler variable in the fragment shader to the correct texture unit.
* // In this case, we are using texture unit 0. So in renderScene() function, we call glUniform1i(texUnit, 0) to
* // link texUnit (a handle of the "texUnit" variable in the fragment shader) to 0.
* // glActiveTexture() function and glUniform1i(textUnit, ...) function calls must be consistent.
* // If you change texture unit number in one function all, you must change the texture unit in the other function call.
* if (textureObjectIDArray0[meshIndex] > 0) {
* glActiveTexture(GL\_TEXTURE0);
* glBindTexture(GL\_TEXTURE\_2D, textureObjectIDArray0[meshIndex]);
* }
* // Set texture filter parameters
* glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);
* glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);
* glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);
* glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

What is a texture unit? A texture unit is a storage unit What is an active texture unit? . A texture unit is used to hold an active texture image that is about to be accessed by shaders. What is special about texture unit 0?

// We only use texture unit 0. Here 0 means Texture Unit 0.

// This tells fragment shader to retrieve texture from Texture Unit 0.

glUniform1i(textureUnit0, 0);

* In a fragment shader, how do you use a pair of texture coordinates and a texture image to do texture mapping?

// retrieve color from each texture

vec4 textureColor1 = texture2D(textureMap0, textureCoord);

vec4 textureColor2 = texture2D(textureMap1, textureCoord);

vec4 textureColor3 = texture2D(textureMap2, textureCoord);

// Combine the two texture colors

// Depending on the texture colors, you may multiply, add,

// or mix the two colors.

if ((textureColor1.r <= 0.01f) && (textureColor1.g <= 0.01f)

&& (textureColor1.b <= 0.01f)) {

textureColor1.r = 1.0f;

}

vec4 textureColor4 = vec4(vec3(1.0, 1.0, 1.0) - textureColor3.rgb, 1.0);

fragColor = textureColor3 \* textureColor1 + textureColor4 \* textureColor2;

What does GLSL function texture() do? The texture() function takes texture coordinates between 0 and 1, or the fractional part if texture coordinates are repeated.

If you have multiple texture images, do you need multiple sets of texture coordinates? Yes.

What are the different ways to combine multiple textures?

// Combine the two texture colors

// Depending on the texture colors, you may multiply, add,

// or mix the two colors.

if ((textureColor1.r <= 0.01f) && (textureColor1.g <= 0.01f)

&& (textureColor1.b <= 0.01f)) {

textureColor1.r = 1.0f;

}

vec4 textureColor4 = vec4(vec3(1.0, 1.0, 1.0) - textureColor3.rgb, 1.0);

fragColor = textureColor3 \* textureColor1 + textureColor4 \* textureColor2;

or use the mix().

Give a few examples of multiple texture mapping?





Do you notice examples of multiple texture mapping in some of the 3D games you have played? Yes all.

* How do you combine texture mapping and lighting in a fragment shader?

texture2D(diffuseMap, gl\_TexCoord[0].st); // etc

gl\_FragColor = color \* lighting;

* What are texture filters? Why do you need them? Texture filtering or texture smoothing is the method used to determine the texture color for a texture mapped pixels using the colors of nearby pixels of the texture. Mathematically, texture filtering is a type of anti aliasing (AA), but it filters out high frequencies from the texture fill whereas other AA techniques generally focus on visual edges. It allows a texture to be applied at many different shapes, sizes and angles while minimizing blurriness, shimmering and blocking.

What is mip-mapping? A MIP map is a computer graphics technique used to achieve an illusion of depth in a two-dimensional representation of a three-dimensional ( 3D ) image.

1. What are the mistakes you made in the process of implementing project 4? If there are many, list the main ones. Accessing the image files, creating the arra , pasing the image files to the shader and figuring out the formula combine the textures correctly

Submit the report in text, Word, or PDF files to Desire2Learn under the dropbox “Homework4”. Write your name in the report.