# Lab 03 Model, Camera and Texture Mapping

## Wavefront Obj loading

Here is an excerpt from a typical obj file:

```
# cube
v 1.000000 -1.000000 -1.000000
v 1.000000 -1.000000 1.000000
v -1.000000 -1.000000 1.000000
v -1.000000 -1.000000 -1.000000
v 1.000000 1.000000 -1.000000
v 1.000000 1.000000 1.000001
v -1.000000 1.000000 1.000000
v -1.000000 1.000000 -1.000000
vt 0.748573 0.750412
vt 0.749279 0.501284
vt 0.999110 0.501077
vt 0.999455 0.750380
vt 0.250471 0.500702
vt 0.249682 0.749677
vt 0.001085 0.750380
vt 0.001517 0.499994
vt 0.499422 0.500239
vt 0.500149 0.750166
vt 0.748355 0.998230
vt 0.500193 0.998728
vt 0.498993 0.250415
vt 0.748953 0.250920
vn 0.000000 0.000000 -1.000000
vn -1.000000 -0.000000 -0.000000
vn -0.000000 -0.000000 1.000000
vn -0.000001 0.000000 1.000000
vn 1.000000 -0.000000 0.000000
vn 1.000000 0.000000 0.000001
vn 0.000000 1.000000 -0.000000
vn -0.000000 -1.000000 0.000000
f 5/1/1 1/2/1 4/3/1
f 5/1/1 4/3/1 8/4/1
f 3/5/2 7/6/2 8/7/2
f 3/5/2 8/7/2 4/8/2
f 2/9/3 6/10/3 3/5/3
f 6/10/4 7/6/4 3/5/4
f 1/2/5 5/1/5 2/9/5
f 5/1/6 6/10/6 2/9/6
f 5/1/7 8/11/7 6/10/7
f 8/11/7 7/12/7 6/10/7
f 1/2/8 2/9/8 3/13/8
f 1/2/8 3/13/8 4/14/8
```

- # is a comment, just like // in C++
- v is a vertex
- vt is the texture coordinate of one vertex
- vn is the normal of a vertex
- f is a face

v, vt and vn are simple to understand. For example, “f 8/11/7 7/12/7 6/10/7”:

- 8/11/7 describes the first vertex of the triangle
- 7/12/7 describes the second vertex of the triangle
- 6/10/7 describes the third vertex of the triangle
- For the first vertex, 8 denotes the vertex index, which in this case has the coordinates -1.000000 1.000000 -1.000000 (indexing is 1-based not 0-based like in C++)
- 11 denotes which texture coordinate to use, in this case 0.748355 0.998230
- 7 denotes which normal to use, in this case 0.000000 1.000000 -0.000000

## Projection

```
// Task 2: perspective projection
//*/
// Projection matrix: 45° Field of View, 4:3 ratio, display range: 0.1 unit <->
100 units
projectionMatrix = perspective(radians(45.0f), 4.0f / 3.0f, 0.1f, 100.0f);
// Subtask 2.1: change z translation and define MVP
modelMatrix = translate(mat4(), vec3(0.0, 0.0, -10));
MVP = ...
//*/
```

***What is the role of the last two arguments of the `perspective function`?***

***Question 1 How can you achieve the zoom in/out effect?***

```
// Task 3: ortho projection
//*/
// Subtask 3.1: change z
projectionMatrix = ortho(-5.0f, 5.0f, -5.0f, 5.0f, 0.0f, 10.0f); // In world
coordinates
// Subtask 3.2: change z translation
modelMatrix = translate(mat4(), vec3(0.0, 0.0, -10));
MVP = ...
//*/
```

***Question 2 What are the differences between ortho and perspective functions?***

***Question 3 Translate the object in z-direction using ortho and perspective functions, what do you observe?***

```
// Task 4: view
//*/
// Task 4.1: make the camera move linearly in time
// Task 4.2: make the camera move periodically around the
Object
viewMatrix = lookAt(
    vec3(3, 3, 10), // Camera position, in World Space
    vec3(0, 0, 0), // and looks at the origin
    vec3(0, 1, 0)  // Head is up (set to 0, -1, 0 to look upside-down)
);
modelMatrix = mat4(1.0);
MVP = ...;
```

```
//*/
```

***What is the correct order of multiplication for MVP?***

## Camera

```
// Task 5: camera
//*/
camera->update();
projectionMatrix = camera->projectionMatrix;
viewMatrix = camera->viewMatrix;
modelMatrix = glm::mat4(1.0);
MVP = ...;
//*/
```

Camera parameters:

- position
- horizontalAngle
- verticalAngle
- FoV
- speed
- mouseSpeed
- fovSpeed

Faster computers (better GPU) will have larger frame rates, while slower GPU will render the scene with less FPS (frames per second). The sensitivity of the camera is affected by the FPS.

```
// Compute time difference between current and last frame
double currentTime = glfwGetTime();
float deltaTime = float(currentTime - lastTime);
```

Get mouse position

```
// Get mouse position
double xPos, yPos;
glfwGetCursorPos(window, &xPos, &yPos);
```

Get window size

```
int width, height;
glfwGetWindowSize(window, &width, &height);
```

Keep the cursor in the middle of the screen

```
// Reset mouse position for next frame
glfwSetCursorPos(window, width / 2, height / 2);
```

```
// Task 5.1: simple camera movement that moves in +-z and +-x axes
//*/
// Move forward
if (glfwGetKey(window, GLFW_KEY_W) == GLFW_PRESS) {
    position -= ?? * deltaTime * speed;
}

...
```

```
// Task 5.2: update view matrix so it always looks at the origin
```

```
projectionMatrix = perspective(radians(FoV), 4.0f / 3.0f, 0.1f, 100.0f);
viewMatrix = lookAt(
     ...
);.
```

***Implement a movement of the camera in +-z and +-x axes using the WSDA keys.***

```
// Task 5.3
// Compute new horizontal and vertical angles, given windows size
// and cursor position
```

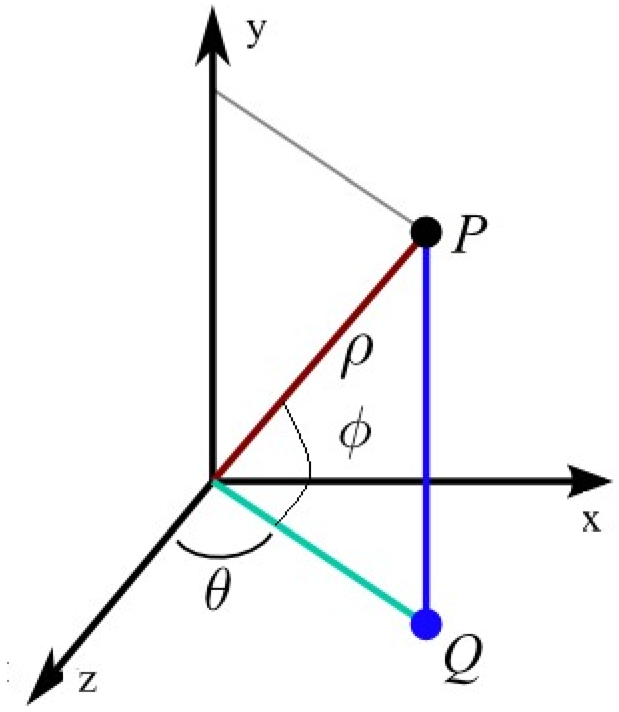
***How would you compute the horizontal and vertical angles given the mouse position and window size?***

```
// Task 5.4: compute direction, right and up vectors of the camera coordinate
system
// use spherical coordinates

vec3 direction(

vec3 right(

vec3 up =
```



***Compute the camera orientation for the given horizontal and vertical angles.***

```
// Task 5.5: update camera position using the direction/right vectors
// Move forward
if (glfwGetKey(window, GLFW_KEY_W) == GLFW_PRESS) {
    position += ...;
}
```

…

***Handle zoom in/out effects using UP/DOWN keys.***

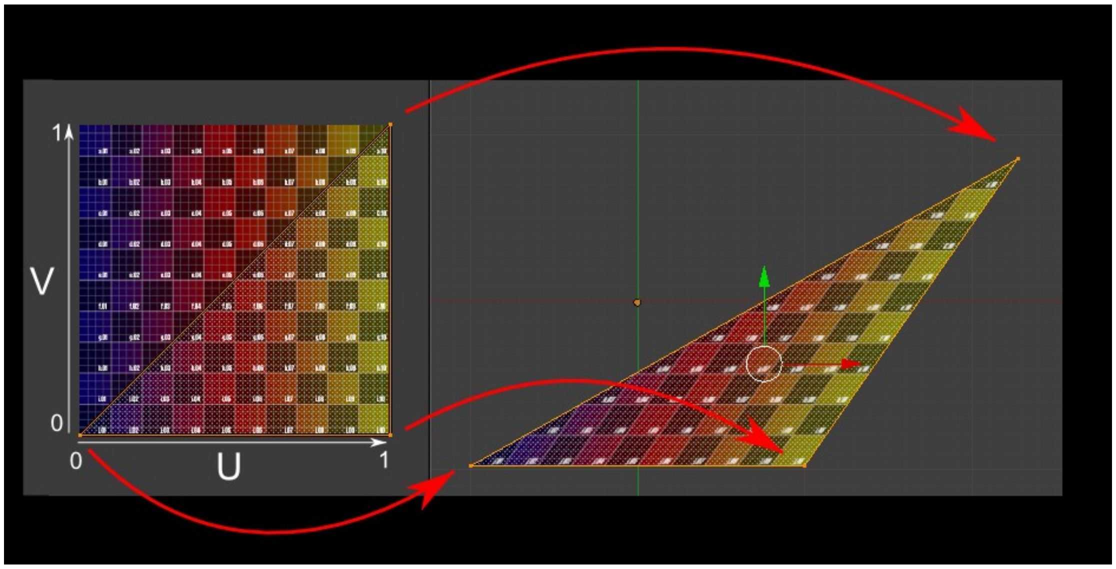
```
// Task 5.7: construct projection and view matrices

projectionMatrix =

viewMatrix =
```

## Texture Mapping

When texturing a mesh, we need to specify the mapping between the texture and the mesh. This is done using UV (or ST, they are the same) coordinates. The obj may specify U, V coordinates for each vertex. These coordinates are used to access the texture in the following way:



***Load a BMP image and the UV coordinates to attribute 1.***

```
// Task 6: texture loading
// uncomment Task 6 in main loop
//*/
// Get a handle for our "textureSampler" uniform
textureSampler = glGetUniformLocation(shaderProgram, "textureSampler");

// load BMP
texture = loadBMP("uvtemplate.bmp");

// uvs VBO
glGenBuffers(1, &suzanneUVVBO);
glBindBuffer(GL_ARRAY_BUFFER, suzanneUVVBO);
glBufferData(GL_ARRAY_BUFFER, suzanneUVs.size() * sizeof(glm::vec2),
    &suzanneUVs[0], GL_STATIC_DRAW);
glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, 0, NULL);
glEnableVertexAttribArray(1);
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
//*/
```

Fragment shader.

```
#version 330 core

// Interpolated values from the vertex shaders
in vec2 UV;

// Output data
out vec3 color;

// Values that stay constant for the whole mesh.
uniform sampler2D textureSampler;

void main(){

    // Output color = color of the texture at the specified UV
    color = texture(textureSampler, UV ).rgb;
}
```

- The fragment shader accepts the UV coordinates of the fragment.
- "sampler2D" is the data type GLSL uses to represent a texture. It can be a single texture or multiples of it with varying scale (mipmaps).
- Getting the color of the fragment from the from the texture based on the UV coordinates is done by calling the "texture" function.

***Bind the texture before drawing the object.***

Binding and sending the data to OpenGL follows almost the same procedure as every other uniform variable.

In this case the steps are:

1)ActivateTexture: In order to tell OpenGL that we want to use a texture unit.

2)BindTexture: Bind our texture to the **currently** active texture unit (in this case it's GL_TEXTURE0).

3)Uniform1i: Set out "textureSampler" uniform variable inside the fragment shader to use whatever is bound to texture unit "0" (a.k.a. GL_TEXTURE0).

```
// Task 6: texture
//*/
// Bind our texture in Texture Unit 0
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, texture);
// Set our "textureSampler" sampler to use Texture Unit 0
glUniform1i(textureSampler, 0);
//*/
```

Question 4 ***Enable GL_BLEND and change the alpha parameter (RGBA) in the fragment shader; what do you observe?***

```
// Task 6.1
// Enable blending for transparency
// change alpha in fragment shader
//*/
```

```
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
//*/
```

***Question 5** Enable GL_CULL_FACE; what do you observe?*

```
// Task 6.2
// Cull triangles which normal is not towards the camera
//*/
glEnable(GL_CULL_FACE);
//*/
```

**Homework Tasks:**

**Task1:** Answer (briefly) the Questions 1-5

**Task2:** Make the camera go up and down with the Q and E keys respectively

**Task3:** Make the camera tilt sideways left and right (like peeking around a corner) with R and T keys.

**Task4:**

a)Load the texture displacement.bmp located in the lab03 folder.

b)Overlay the displacement texture on top of the water texture (tip: use mix(displacementColor,WaterColor, 0.5) )

c)Make the displacement texture also move in time but in a slightly different speed compared to the water texture.

d)Is the result more realistic than just using the water texture or not? Explain.

## Reading

Anton's OpenGL 4 Tutorials - Anton Gerdelan pp. 110--148