Normal Mapping

Guided by Professor Michael Manzke

Implemented by: Yuzhou Shao

Student id: 19322035

Youtube Demo: https://youtu.be/oaNRAWpWQK0

My Submission Includes:

A Youtube Demo

A Report

A zip File including

main.cpp

test_vs-NormalMapping.glsl(vertex shader)

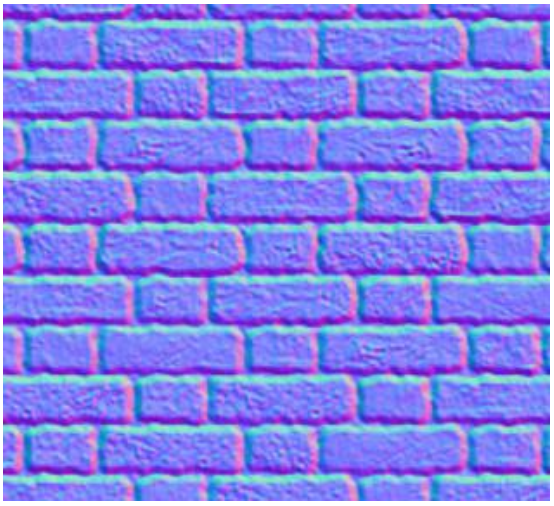
test_fs-NormalMapping.glsl(fragment shader)

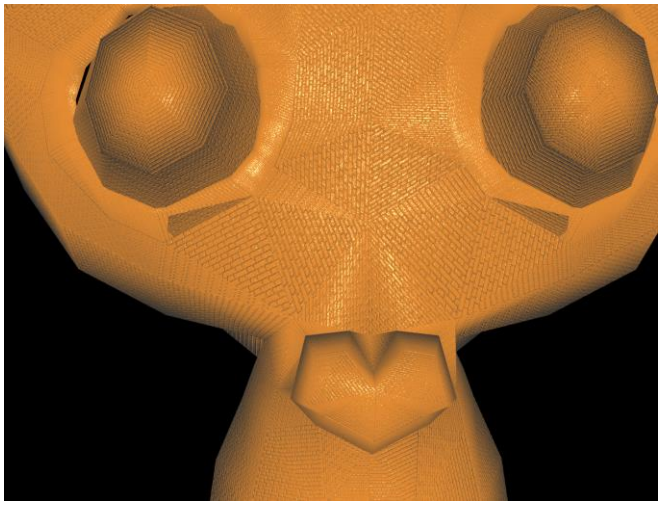
To implement this project, I have followed and studied Anton's OpenGL 4 Tutorials https://antongerdelan.net/opengl/index.html by Dr. Anton Gerdelan, Trinity College Dublin, Ireland.

I have used his code and also modified in my own approach in some parts.

I have used GLSL in my shaders.

I used below left image for Normal Mapping to make a unique object with bump textures as below right.



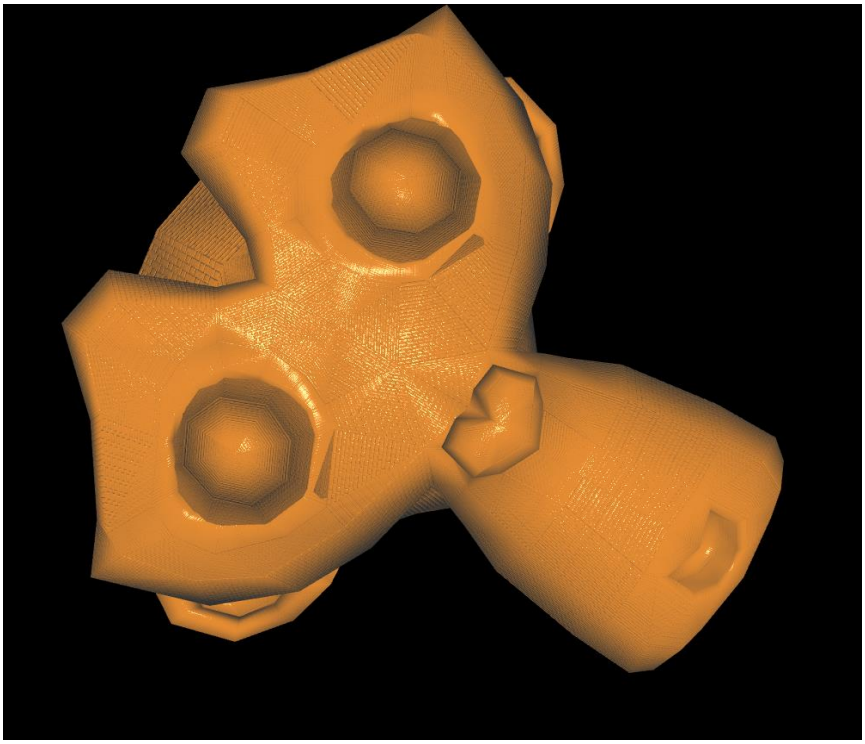


Secondary Objectives:

I have used glfw loops for keyboard control to implement an **rotating object**.

```
253         if (glfwGetKey(g_window, GLFW_KEY_UP)) {//Originally was PAGE_UP, Shao changed
254             cam_pos[1] += cam_speed * elapsed_seconds;
255             cam_moved = true;
256         }
257         if (glfwGetKey(g_window, GLFW_KEY_DOWN)) {//Originally was PAGE_DOWN, Shao changed
258             cam_pos[1] -= cam_speed * elapsed_seconds;
259             cam_moved = true;
260         }
261         if (glfwGetKey(g_window, GLFW_KEY_W)) {
262             cam_pos[2] -= cam_speed * elapsed_seconds;
263             cam_moved = true;
264         }
265         if (glfwGetKey(g_window, GLFW_KEY_S)) {
266             cam_pos[2] += cam_speed * elapsed_seconds;
267             cam_moved = true;
268         }
269         if (glfwGetKey(g_window, GLFW_KEY_LEFT)) {
270             cam_yaw += cam_yaw_speed * elapsed_seconds;
271             cam_moved = true;
272         }
273         if (glfwGetKey(g_window, GLFW_KEY_RIGHT)) {
274             cam_yaw -= cam_yaw_speed * elapsed_seconds;
275             cam_moved = true;
276         }
```

```
277         // update view matrix
278         if (cam_moved) {
279             mat4 T = translate(identity_mat4(), vec3(-cam_pos[0], -cam_pos[1],
280                 -cam_pos[2])); // cam translation
281             //mat4 R = rotate_y_deg(identity_mat4(), -cam_yaw);     //Original
282             mat4 R = rotate_z_deg(identity_mat4(), -cam_yaw);       //Shao edited
283             //mat4 R = glm::rotate(identity_mat4(), -90.0f * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
284             mat4 view_mat = R * T;
285             glUniformMatrix4fv(view_mat_location, 1, GL_FALSE, view_mat.m);
286         }
287
```



Above is my output

To make the object more **photo-realistic**, in my fragment shader,

I have implement diffuse light, specular light and phong light following by Dr. Anton's tutorial.

```
18
19      // sample the normal map and covert from 0:1 range to -1:1 range
20      vec3 normal_tan = texture (normal_map, st).rgb;
21      normal_tan = normalize (normal_tan * 2.0 - 1.0);
22
23      // diffuse light equation done in tangent space
24      vec3 direction_to_light_tan = normalize (-light_dir_tan);
25      float dot_prod = dot (direction_to_light_tan, normal_tan);
26      dot_prod = max (dot_prod, 0.0);
27      vec3 Id = vec3 (0.7, 0.7, 0.7) * vec3 (1.0, 0.5, 0.0) * dot_prod;
28
29      // specular light equation done in tangent space
30      vec3 reflection_tan = reflect (normalize (light_dir_tan), normal_tan);
31      float dot_prod_specular = dot (reflection_tan, normalize (view_dir_tan));
32      dot_prod_specular = max (dot_prod_specular, 0.0);
33      float specular_factor = pow (dot_prod_specular, 100.0);
34      vec3 Is = vec3 (1.0, 1.0, 1.0) * vec3 (0.5, 0.5, 0.5) * specular_factor;
35
36      // phong light output
37      frag_colour.rgb = Is + Id + Ia;
38      frag_colour.a = 1.0;
```

Above is my output

External libraries and 3rd party source code

```
#include "gl_utils.h"
#include "maths_funcs.h"
#include <GL/glew.h>    // include GLEW and new version of GL on Windows
#include <GLFW/glfw3.h> // GLFW helper library
#include <assert.h>
#include <string.h>             // assimp forgot to include this
#include <assimp/cimport.h>     // C importer
#include <assimp/postprocess.h> // various extra operations
#include <assimp/scene.h>       // collects data
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
```

The 3rd party source including gl_utils class, maths_funcs class, stb_image.h and suzanne.obj.

Reference:

Surface Mapping Notes by Professor Michael Manzke, Trinity College Dublin, Ireland

Antons_opengl_tutorials_book
https://github.com/capnramses/antons_opengl_tutorials_book/tree/master/20_normal_mapping

Tutorial 13 : Normal Mapping

Tutorial 13 : Normal Mapping (opengl-tutorial.org)