[CPSC-4050/6050] ASSIGNMENT #03: SHADING AND TEXTURING

THIS IS A ONE WEEK ASSIGNMENT WITH SUBMISSION DATE AS 25th NOV, 2024.
In total, there are 100 marks to score in this assignment. Your scores will be scaled to carry equal net weight as any other assignment labelled as 'One Week' assignment. There are bonus questions to score maximum extra credits up to 20 marks. These extra credits will be weighted similarly to the main scores and carried forward to the total score for final grading.

SUBMISSION DEADLINE: 11:59 p.m., 25th of November (MONDAY)

- A delay of upto 24 hours will result in a proportional reduction of scores by 10%.
- A delay of upto 48 hours will result in a proportional reduction of scores by 25%.
- A delay of upto 72 hours will result in a proportional reduction of scores by 50%.

Any further delay will be considered as non-submission. If there is an anticipated/actual delay for any medical reason, unavoidable issues, or exigencies, kindly contact the instructor at the earliest.

The objective of this assignment is to learn basics and practice use of shaders, textures, data arrays and buffers for rendering. This assignment will involve use of modern OpenGL. You will be provided with a starter code package with bare minimal framework to drive shaders and access textures. The first aim is to learn how to organize triangle mesh geometry along with other appearance attributes into data buffers and load them up for shading. Next you will learn how to program a vertex shader for performing geometric transformations with traditional projection-view-model matrix formulations. Lastly, you will develop couple of fragment shaders to produce diffuse lighting, specular reflectivity and texturing effects on surfaces.

IMPORTANT:
A starter code package is provided for this assignment. You will mainly modify the stub.cpp file in the package, modify/add shader program files, recompile and run the program as explained in detail, in the workflow section later. You code package should compile and run WITHOUT ANY MODIFICATION on CLEMSON's virtual linux platform under GUI, i.e. XFCE, session (https://computing.clemson.edu/help/virtual.html) . You may use main.cpp for either of the previous assignment/lab_exercises for reference, especially for keyboard interfacing as implemented in Lab 02.

OS required: Linux, Compiler: C/C++,
Build procedure: using make file.

Scenario: You may work with a surface of revolution explained in Part A. Alternately, you may construct a simple vertical cylindrical if you are willing to let go of the scores for Part A and solve the rest of the assignment. Given code framework for modern OpenGL has its camera at a fixed location in the world coordinate system and its view and projection matrices are set and passed on as uniform variables to shader programs. Your surface of revolution should be scaled and positioned such that its center of mass coincides with the origin for the model coordinate system, and it lies within the cube spanning the volume ($-1 \leq xx, yy, zz \leq 1$). Furthermore, the given framework supports a simple rotational animation for your surface of revolution by computing the

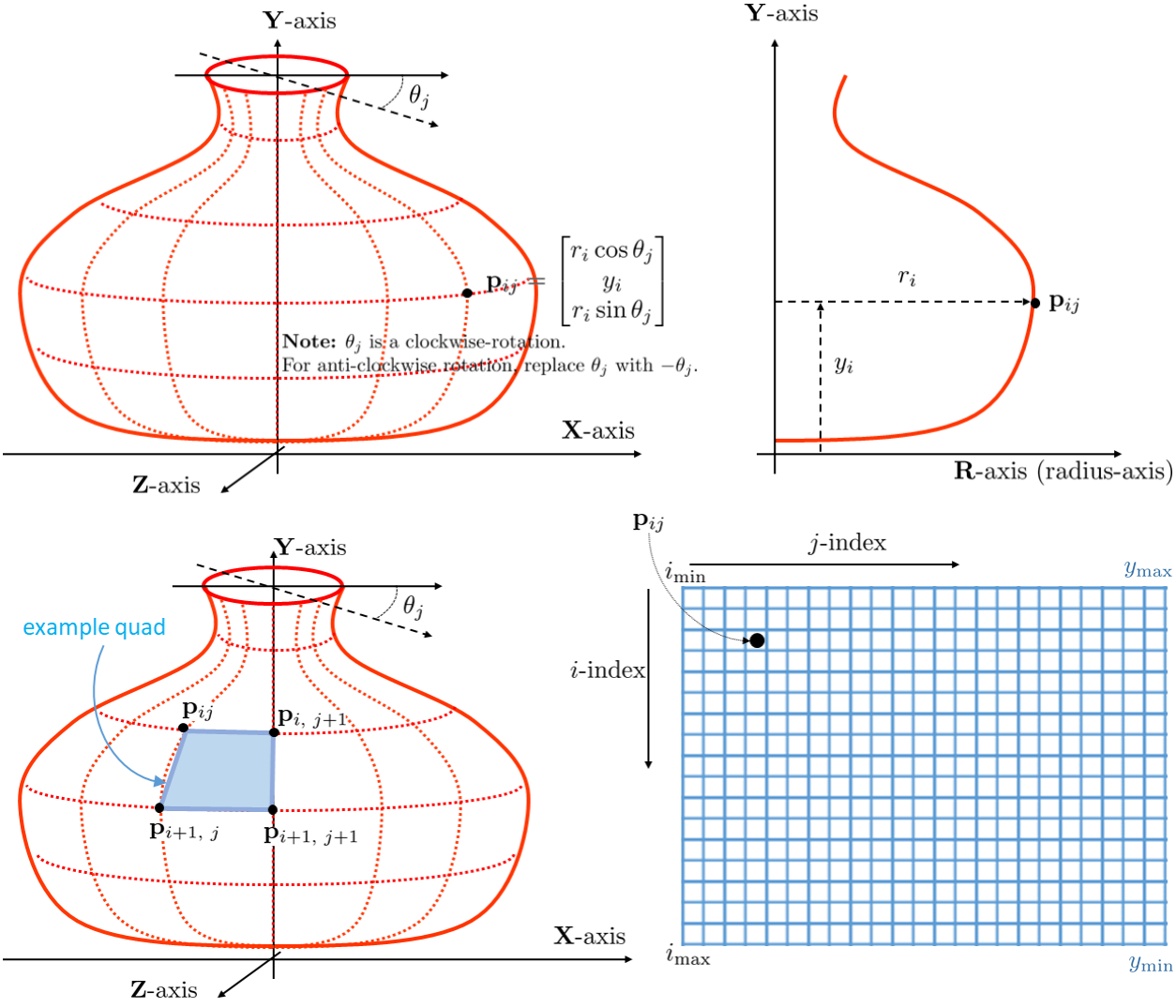
[CPSC-4050/6050] ASSIGNMENT #03: SHADING AND TEXTURING

model matrix in the display loop as passing it as a uniform variable to the shaders. You must implement correct geometric transformations in the vertex shader using these provided matrices. You are provided with bare minimal implementation of sample vertex and fragment shaders along with the code that can read any .jpg/.png file, load it as a texture and passes it on to shaders for sample lookups. The framework uses GLFW library for windowing with modern OpenGL. **For most part, example code relating to the tasks for this assignment is present in stub.cpp itself. Provided starter package is self-contained for basic understanding.**

Objectives:

PART-A (25%): Constructing a Surface of Revolution

For this task you will be re-implementing the stub function loadSurfaceOfRevolution() in stub.cpp. You will be constructing a surface of resolution around the Y-Axis. The following diagram illustrates a surface of revolution and the key aspects for its construction.



[CPSC-4050/6050] ASSIGNMENT #03: SHADING AND TEXTURING

- Inputs: number of steps along the y and θ dimensions, i.e. number of 'i' and 'j' indices respectively in the figure above. Note: you will have to write code to receive these inputs from the user at the start of the program, from within the stub function **loadSurfaceOfRevolution()**.
- Write code to construct a base 2D curve in the Y-R space as shown in top-right image in the figure above. You don't need to display this curve but compute it to form the surface of revolution. Your choices for the base curve are given below. Choose any one and make sure that it fits in the region $(-1 \leq yy, rr \leq 1)$.
  - For 4050: (a) parabolic arc, (b) circular arc, (c) Cubic Bezier curve
  - For 6050: (a) Cubic Bezier curve, (b) Quintic Bezier curve, (c) B-Spline
  - For this base curve, the R-dimension represents the radial-axis and the second dimension represents the **y-axis** for the world coordinate system.
  - With the input parameters (i.e. the number of steps along the **y and θ** dimensions) invoke your curve drawing algorithm to produce as many pairs of (y_i, r_i) as the number of steps along the y axis (plus one).
  - Next, write code to generate the array of point p_{ij} as depicted in the illustrations above. With these points, generate a list of quads, where each quad connects 4 adjacent points as depicted above. This list of quad represents your surface of revolution. Break each quad into two triangles as modern openGL does not accept quads as primitives.
- Organize the triangles generated for above steps into a **vertex data buffer object (VBO)** which is bound to a **vertex array object (VAO)** and pass the buffer data (VBO) on for shading with modern OpenGL.
- Write a basic vertex shader code that performs geometric transformation from the model coordinate space to clip co-ordinate space using projection, view and model transformation matrices that are expressed in the 4x4 format for homogeneous coordinates.

PART-B (25%) : Blinn-Phong Shading

- Compute per-vertex surface normals for ***your surface of revolution***, store and pass them across as a VBO for shading diffuse lighting and specular reflection effects.
- Write a fragment shader program that performs Blinn-Phong shading using ONE fixed **point-light** source, ONE **directional light** source, ONE **ambient light** source and a fixed Blinn-Phong **model (exponent) parameter** for the specular highlight. Compute the diffused, the specular and the ambient light-matter interactions. Use different material colors for the diffused and specular parts but the same color for the diffused and the ambient parts. The point light source should be initially fixed at the same point as the camera location/center. Set the exponent, light colors, and other such parameters arbitrarily within the GLSL shader program or pass them as uniform variables from the C++ program for this part.

[CPSC-4050/6050] ASSIGNMENT #03: SHADING AND TEXTURING

PART-C (25%): Using Textures

- Compute per-vertex texture coordinates for *your surface of revolution*, store and pass them across as a VBO for mapping a texture-image as the diffuse albedo. Texture coordinates are assigned such that any given texture-image in the u-v space ($0 \leq u, v \leq 1$) wraps seamlessly around the surface (wraps along the u-dimension).
- Extend previous (Blinn-Phong) fragment shader to sample a pre-loaded texture image and use those sampled values as the diffuse albedo color in shading.

PART-D (25%): Keyboard Interactivity

- Provide simple keyboard interactivity for dynamically:
  - changing light position
  - changing specular exponent for the Blinn-Phong Model
  - enabling/disabling, i.e. toggling,
    - diffuse shading effects
    - specular lighting effects
    - use of texture as diffuse albedo

Not that for the keyboard interactivities, you need to pass relevant parameters using Uniforms.

All the tasks are the same for both 4050 as well as 6050 levels. Tasks and weightage are already indicated above. BONUS scores will be clamped at 20%. Following are the Bonus Tasks:

CPSC4050	CPSC6050
BONUS: E (20%) Extend the shader in Part C to use a second texture map as a surface normal map for changing both diffused as well as specular shading. Implement interactive toggling (enabling/disabling) the use of this texture map.	**BONUS: E. (10%)** Extend the shader in Part C to use a second texture map as a surface normal map for changing both diffused as well as specular shading. Implement interactive toggling (enabling/disabling) the use of this texture map.
BONUS: F. (10%) Extend the shader in Part C to use a second (grayscale) texture map as a roughness map for changing the exponent for the specular reflection. Implement interactive toggling (enabling/disabling) the use of this texture map.	**BONUS: F. (10%)** Extend the shader in Part C to use a second (grayscale) texture map as a roughness map for changing the exponent for the specular reflection. Implement interactive toggling (enabling/disabling) the use of this texture map.
BONUS: G. (10%) Extend the shader in Part C to add one more directional light to the	**BONUS: G. (10%)** Extend the shader in Part C to add one more directional light to the

[CPSC-4050/6050] ASSIGNMENT #03: SHADING AND TEXTURING

scene. Add interactivity to change its direction	scene. Add interactivity to change its direction

WorkFlow:

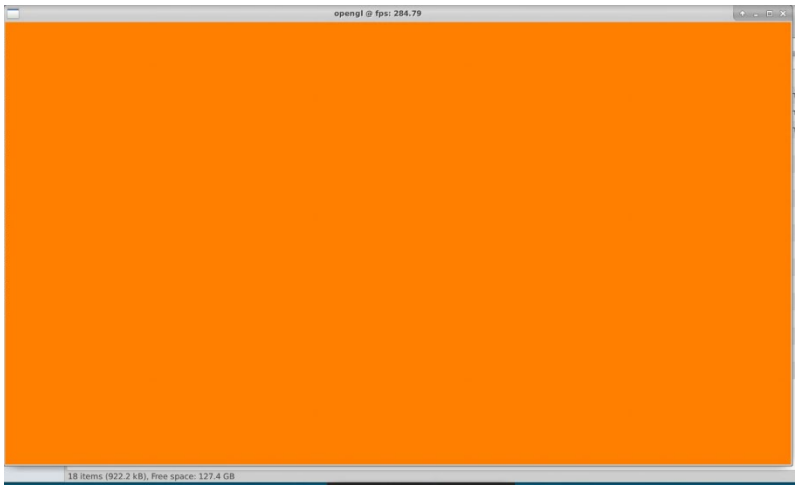
- Download the compressed package from https://clemson.instructure.com/files/24639456/ and save it in your local folder.
- Open a terminal/command prompt and go to the local folder where your package is saved.
- Untar it with the command `tar –xzv cg03_2024.tar.gz` This will create two subfolders named **main** and **common**.
- Change your directory to this subfolder named **main**.
- Apart from other items, this subfolder named **main** contains two source files **main.cpp** and **stub.cpp** and a **Makefile** file. You will be mostly making changes to stub.cpp, vs.glsl and frag.glsl
- To build given package simply type '**make**' or the following at the command prompt:
  >> `make –f Makefile`
  This will produce an executable file named `cg03`
- There are no elaborate make options in this package. So, if you want to make a clean build, you will have to manually delete the .o files and the executable file `cg03`
- To run the default program type `./cg03` **with atleast three command-line input arguments** at the prompt and it will open an OpenGL window. Consider the input command format:

  ```
  ./cg03 arg1 arg2 arg3 arg4
  ```

  `arg1` is the filename for the vertex shader file
  `arg2` is the filename for the fragment shader file
  `arg3` is the filename for the texture file in .jpg or .png format
  `arg4` is an optional argument giving the filename for the second optional texture file in .jpg or .png format. This optional file can be used in bonus tasks.

  For example, giving a command
  >> `./cg05 vs.glsl frag.glsl atlas.jpg` with default build will produce a screen like below:

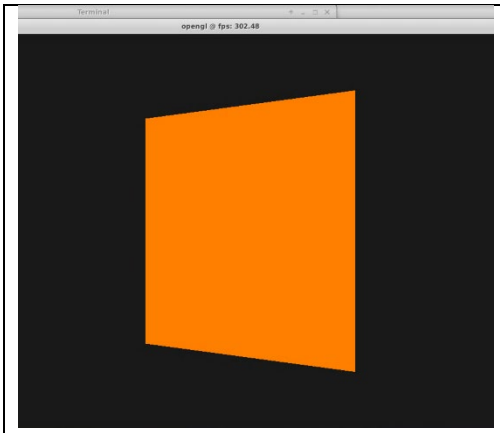
[CPSC-4050/6050] ASSIGNMENT #03: SHADING AND TEXTURING



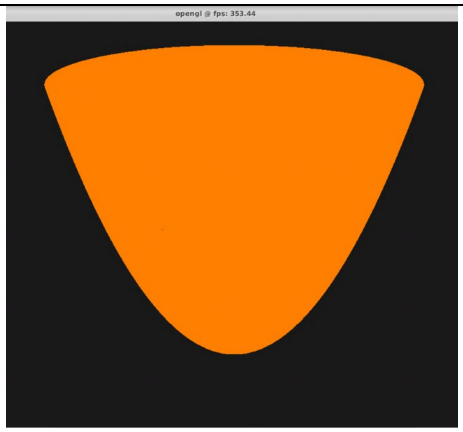
- For PART A:
  You will need to change vs.glsl and the following functions in stub.cpp
  loadSurfaceOfRevolution(), and
  drawSurfaceOfRevolution()

Note that for your surface of revolution, you will have to break each quad into two triangles as quads are not directly supported in modern OpenGL. Also, your triangle mesh will have to be organized into the separate-triangle format which for further simplification is stored as a single dimensional array of float numbers. The loadSurfaceOfRevolution() has a sample code block to example these organizational nitty-gritties for loading a simple square-quad made up of two triangles. The sample code also demonstrates the handling of the VAO and a VBO. There are further comments in these code blocks to explain how things need to be changed.

If you only change the vs.glsl as required without implementing your own geometry, you will see a rotating plate that disappears intermittently:



Without loading surface of revolution

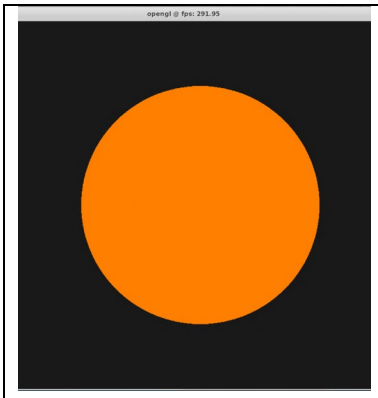
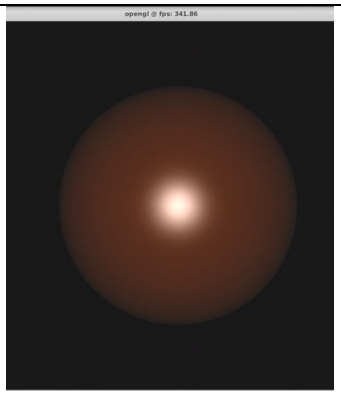
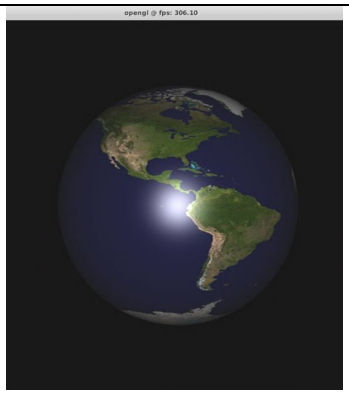


With smooth parabolic Surface

[CPSC-4050/6050] ASSIGNMENT #03: SHADING AND TEXTURING

For digitizing your surface of revolution, you may fix 'm' and 'n' arbitrarily so long as the discretization is smooth enough. An example with a parabola is shown above.

- For PART B and C:
  You will need to change frag.glsl and loadSurfaceOfRevolution() in stub.cpp to create+load a appropriate VBOs for per-vertex attributes (normal/texture coordinates). An example for a spherical globe is shown below:

		
Without modifications to frag.glsl	frag.glsl with BlinnPhong	frag.glsl with BlinnPhong + Texture Mapping

A video for the rotating globe from texture mapping with the atlas from above example is uploaded at: https://clemson.instructure.com/courses/95982/modules/items/1436975

- For PART D:
  You will need to change keyboardFunction() in stub.cpp to capture key-press events. After interactively seeking relevant data from the user pass it on as a uniform variable to the shaders. [HINT: load them when loadUniforms() in stub.cpp is called. This function is invoked before every display rendering.
- For bonus questions you may have to study and modify main.cpp and/or other source code files. If you add newer .cpp files make sure to update the Makefile appropriately.
- You may use math.h for math operations. Although there may not be any need to use additional math packages such as vmath, GLM, e.t.c., if you choose to do so, it will be your responsibility that the source code for such a library is included in your code package and get compiled and linked to your code with the single use of **make** command as before.
- Prepare a PDF report explaining the structure of your code package and how to build and run it along with keyboard interfacing details. Especially, **include a version of Appendix A** from below that clearly associates a key input with corresponding desired operation.

[CPSC-4050/6050] ASSIGNMENT #03: SHADING AND TEXTURING

- Include few screen-grabs for the output of your implementation for each of the task. You need not document each possible operation's output. Some representative cases would do.
- Prepare and upload a single zip file as your submission package that should include:
  - The above report
  - All your source code files and the Makefile with a readme.txt if the need be.

Reference material:

- Chapters 10, 11, 17 and Section 12.1 from the course book Fundamentals of Computer Graphics by Marschner and Shirley, 4th edition. You can access this book from Oreally's online resources.
- Chapter 17 discusses details on VAOs and VBOs beyond those covered through the example code in stub.cpp
- Chapters 2 (for Shader fundamentals including GLSL) and Chapter 3 (Drawing with Modern OpenGL) from the (Redbook) OpenGL Programming Guide Ninth Edition by Kessenich et. al. You can access this book from Oreally's online resources. Consult your TA for details.
- GLSL Essentials: https://clemson.instructure.com/courses/150761/modules/items/2720805
- GFLW keyboard interfacing: https://www.glfw.org/docs/3.3/input_guide.html#input_key