COMP 557 - Fall 2023 - Assignment 2
ArcBall and Shadow Maps

Getting Started

Download the provided code from McCourses and dump it into a new C++ project. Setup the project like the first assignment. Use CMake to generate the solution and build it. Don't forget to specify the resources folder.

You will note that much of the assignment structure matches that of the previous assignment. There is a new inverse matrix within the MatrixStack class that it useful for transforming normals, and some new model classes (QuadTextured, WireCube, etc.), a Camera class that handles the the view and projection, and a Scene class that manages the objects to render. The modeling, viewing, and projection matrices are used in a few different programs: one to draw the light depth, one to draws the light depth's texture for debugging, and one for drawing from the main camera with per pixel lighting.

The assignment will run without errors as provided, but what you see will not be so interesting until you complete the first objectives.

Objectives

The purpose of this assignment is to work with viewing and projection matrices from both a camera at the light (for shadow maps) and from a camera for the drawing the scene. You will also write code to implement ArcBall interaction.

1. ***Viewing Transformation (2 marks)***

   Set up the camera viewing transformation in Camera::updateView() given the member variables (position, look at, up). See that `glm` has a function for creating the lookat matrix, though you are encouraged to understand how to make the matrix yourself.

2. ***Projection Transformation (2 marks)***

   Set up the camera projection transformation in Camera::setPerspective() given the parameters (field of view in the y direction, the aspect ratio, the near and far planes). Note that you can again use glm, or otherwise that you should follow exactly the projection combined with windowing transformation from the lectures, such that the map to the canonical viewing volume (CVV) has near mapping to -1 and the far to 1.

3. ***Implement the TrackBall controls (2 marks)***

   Implement the TrackBall (ArcBall) as discussed in class. Mouse screen positions are projected onto an imaginary ball in the canvas. The fit parameter describes how big the ball is relative to the smallest screen dimension (width or height). With a square window and fit of 2, the ball will just touch the edges of the screen. Values less than 2 will give a ball larger than the window while smaller values will give a ball contained entirely inside.

   Implement the ~~setTrackballVector~~ computeVecFromMousePos helper function to set 3D vectors from 2D points generated from mouse events. Note that if the mouse point is not "on" the screen space ball, then you must compute the point on the silhouette of the ball.

   Using normal length 3D vectors for the current and previous mouse positions, compute a cross product to find the axis and the angle of rotation. The axis will be the direction of the cross product, while the angle is the angle between the two vectors (see glm functions for doing the math). You must scale the angle you computed by the trackballGain value. A matrix can easily have its rotation set by the quaternion generated by glm::axisAngle. Accumulate the rotations in the R matrix member variable, as this is the matrix used within the main render method.

   You can find more information in your class notes and in ARCBALL: A User Interface for Specifying Three-Dimensional Orientation Using a Mouse, a paper inGraphics Interface 1992 by Ken Shoemake.

4. ***Draw Light Frame (1 mark)***

Use the provided axis drawing code to draw the light coordinate frame. Note a call to disable lighting is at the beginning of the light camera debug code as this geometry wants to be drawn in solid colours rather than with lighting computations.

5. ***Draw Light Frustum (1 marks)***

   The point light camera view projection is set up with some reasonable default values for the fovy, aspect, near, and far. For instance, 55 degrees is reasonable for the test scene, and the near and far should contain the shadow casting objects in the scene.

   The objective here is to visualize the frustum by applying the inverse view and inverse projection matrices for the light view on the modeling matrix stack, and then draw a wire cube that is 2 units across.

   See code for drawing a wire cube is provided. You simply need to set up the correct transformations before the call to draw the wire cube.

6. ***Draw Light Depth Debug (1 marks)***

   There is a GLSL program to draw the depth texture on a quadrilateral. The Camera::draw method includes a call to draw the debugDepthMapQuad, and in this objective you must set up a transformation so that the quad draws on the near plane of the light frustum. The size of the quad should match the size of light view frustum on the near plane.

7. ***Shadow mapped lighting (1 marks)***

   Your GLSL program needs to compute the point light camera CCV coordinates for each fragment. You will need a windowing transform to convert the -1 to 1 x and y coordinates to be 0 to 1 for the texture, but you will also need a windowing transform to convert the -1 to 1 depth to be 0 to 1 for the shadow map comparison. Compare the light depth of the current fragment with the light depth of the closest surface to the light. Be sure to use `sigma` as a small offset in your comparison to avoid self shadowing artifacts!

Optional

Up to two bonus marks will be given if you add extra polish to your assignment:

1. Use percentage closer filtering (PCF) to get softer anti-aliased shadows (EASY).
2. Modify the code for multiple lights (certainly requires some reorganization of the code).

Finished?

Great! Be sure your name and student number appears is in the window title, in your readme, and in the top comments section of each of your source files.

Prepare a zip file with your source code folder and readme file if you made one (include resources if you made changes). Only use a *zip* archive! Submit it via MyCourses. **DOUBLE CHECK** your submitted files by downloading them. You cannot receive any marks for assignments with missing or corrupt files!! **Treat deadlines as HARD**. If you submit multiple times, only your last submission will be graded

Note that you are encouraged to discuss assignments with your classmates, but not to the point of sharing code or answers. All work must be your own. Please talk to the TAs or the prof if the academic integrity policies are not clear.