Programming Assignment #5

Due Nov 20 by 11:59pm **Points** 4 **Submitting** a text entry box or a file upload
Available until Nov 25 at 11:59pm

This assignment was locked Nov 25 at 11:59pm.

Due: Saturday, November 20th (at midnight).

Late Policy: Assignment submission will remain open for 5 additional calendar days after the formal deadline. Late submissions will be penalized by a maximum of 1 point on our 4-point scale (i.e. if you submit 5 days late and your assignment was otherwise deserving a "3", you might get a grade as low as a "2" due to this penalty). Some leniency might be exercised on this general rule based on how late the assignment was, and whether turning in an assignment was a repeating occurrence or a one-time event.

Synopsis: You will write a program that visualizes a 3D scene, as viewed from a movable camera, and use 2D Canvas drawing operations to construct a 2D projected view of it on your screen.

Learning Objectives: To familiarize yourselves with the 3D viewing transform pipeline, especially the transforms that relate the camera coordinate system with the world coordinates, and the projection transforms.

Evaluation: Based on our 4-point grading scheme, as discussed in our introductory lecture. You get a check (“3”) if you turn in a viable, and complete submission (even if it just draws a rectangle like the example in the tutorial). “Above and beyond” grades (i.e. a "4") will be awarded for people who have crafted something particularly cool. As a general rule, no more than 1/3 of all assignments turned in (the very best ones, that is) will be considered for a "4" grade.

Collaboration policy: This is an assignment to be done individually. Code not written by you needs to include proper attribution (see **this post** **(https://graphics.cs.wisc.edu/WP/cs559-fall2020/2020/09/10/collaboration-policy/)** here). It is always ok to use code provided in our in-class examples as a starting point, but you need to add your own effort to raise those examples (or other sources) to what is asked by the programming assignment (i.e. finding some code on some online forum that does all the job for you that is needed to satisfy the assignment is not the intent, if you haven't added any of your own effort to it). If you use somebody else's code (other than our GitHub examples), make sure to clarify in your submission notes what **you** did, and what you repurposed from the external source.

Hand-in: Electronic turn-in on Canvas. Make sure that you turn in all files needed for your program to run. It is acceptable to turn in a single HTML file with your program, but even preferable to separate your code into an .html file and a separate .js file containing the JavaScript code, similar to the examples in our **GitHub repository** **(https://github.com/sifakis/CS559F21_Demos)** (see, e.g. Demos 0-4 from Week 7). If you submit anything else than a single HTML file, please put everything in a

single ZIP archive. Feel free to use the copy of the glMatrix library included in our examples in the GitHub repository (or use them as a starting point) if it's convenient. ***It is not acceptable to submit a link to JSbin for this assignment!***

Description

In Week 7 we saw several examples of how to visualize a 3D scene using Canvas drawing instructions (most of which are found in our **GitHub repository (https://github.com/sifakis/CS559F21_Demos)** under **Week7/**). The general idea was to construct and apply the transforms of a ***3D Viewing Pipeline*** to 3D dimensional points, in such a way as to compute their corresponding ***canvas/viewport*** coordinates, that would allow us to "draw" a three-dimensional line/shape/polygon, by essentially drawing the same type of geometric shape but using the projected points onto 2D canvas (i.e. "viewport") coordinates.

A particularly relevant demo is found under directory **Week7/Demo4** our **GitHub repository (https://github.com/sifakis/CS559F21_Demos)** (also in this **[JSBin] (http://jsbin.com/ficoxeh)** link) where a moving camera is hovering over our 3D world, and creating a projection that corresponds to its own vantage point, which is visualized in a separate window. The convenient feature of this example is that it clearly demonstrates all transforms involved in the 3D viewing pipeline (model, lookAt, projection, and viewport transforms). *What you are asked to do is to create a visualization similar to the "camera" view of this example (the window on the right) where we visualize what the camera "sees" from its vantage point.*

One possible way to achieve this objective would be to take one of the projects you crafted for programming assignments #3 or #4, and "lift" it to 3D by having all the drawing (and motion, if you had any) happen on a *plane* in the 3D world, and setting up a camera so it can move around it and view it from different vantage points. Of course, it would be even more exciting (and advisable) if you modified your code to be natively 3-dimensional: for example, if you had some parametric curves, you could adjust their control points so that they are not all co-planar (so that the curve "escapes" the plane, sort of speak). Or, if you had a hierarchical model included, you could script some of its transformations to be in true 3D, by translating or rotating outside of just the same plane. The following requirements **must** be satisfied in order to get a *satisfactory* "3" grade:

- **Requirement #1**. You must have a moveable camera. The motion can either be automatic, or triggered by manipulation of a slider. It is ok for the motion of the camera to be relatively simple (i.e. translation along a line, or motion along a curve).
- **Requirement #2.** Your 3D world must include a hierarchical model, **or** a parametric curve (or both!).
- **Requirement #3.** Your example should make it apparent that you are observing a 3D scene, as opposed to a 2D object. For example, if you drew everything in the 2D plane (just replacing every coordinate (x,y) with a triple (x,y,0) to "make it 3D"), and just translated the entire drawing up/down (to emulate "moving the camera"), this is really just a 2D drawing exercise, not something that feels like 3D. Having the camera "spin around" an object, having a 3D (non-planar) parametric curve, or having a 3D object are good possibilities for showing off the 3D nature of your program.

- **Requirement #4.** You need to use a *projection* transform. It can be either orthographic or perspective. The aforementioned viewing demo has examples of both, implemented via glMatrix.

As always, you are encouraged to try and exceed these requirements, and if you do well, you can compete for a “4” *above-and-beyond* grade. Here are some ideas:

- Create a complex hierarchical object that uses 3-D motion and transforms!
- Have objects “fly along” complex 3D curves, in ways that control their position and orientation in intricate ways. Think, for example, of an airplane that does a banked turn along a spiral-shaped path …
- Think about the possibility of having solid-drawn (filled) polygons “hide” each other, if they happen to be in front of one another. For example, think of a cube made up of six square faces; from any vantage point, we should only be able to see three of the six faces (the other three are hidden behind the visible ones). Here’s a thought to consider: you could associate a “normal/perpendicular” vector with each face of such a polyhedron, and ask yourself how this vector is transformed in Camera Coordinates. If the normal vector points towards the camera, maybe this is a “visible” polygon. If it points away, maybe it should be “hidden” (i.e. not drawn).