Project 1: No Lights, Definitely a Camera, and Assemblies

CS 351-1, Winter 2025

Expectations

Projects in CS 351-1 are the core of the course, and you should expect to spend substantial time on each project. Projects are graded based on three component: demo, writeup, and code submission. See the Project 1 rubric for grading details. While the project rubric is strict, there are many (infinite) ways to achieve the requirements in the rubric. Creativity and effort will be rewarded with more generous grading (and will make the projects hopefully more fun), so feel free to experiment! You are always welcome to ask if you are not sure if your project design will match the rubric criterion.

Academic Integrity

As with assignments, you may freely use any source (so long as you cite any code or algorithms you directly use) and discuss problems with classmates, but you *may not* explicitly share code or exact solutions with classmates. **Note**: I will be explicitly checking for plagiarism in projects much more carefully than assignments (using MOSS and related tools), so please be careful to cite your sources. I take academic integrity violations quite seriously, and I would really rather not have to make your (and my) life miserable by having to escalate a plagiarism or cheating report.

Overview

Thus far in class, we have covered the basics of computer graphics and WebGL, including concepts like the VBO, geometric transformations, rotations, and basic 3D models. For this project, we will be applying this knowledge to build a simple 3D scene. This project will reinforce our understanding of the rendering process thus far, and act as a basis for future projects.

By completing this project, you will have built the following:

- **A rudimentary 3D scene**, a collection of 3D geometry with positions and vertex colors.
- **A simple camera**, which is locked to a single axis (for the purposes of this project).
- **A 3D animation**, with the ability to interact live with your 3D models.
- **A writeup**, describing how to interact with your project, and showcasing your project.

As always, you may feel free to implement more than this to build an even cooler scene, but the rubric will describe the minimum requirement you should follow. There are a bunch of requirements in this rubric that might be confusing on first read, so we highly recommend asking (on Ed or in office hours) if you're not sure on a requirement or would like clarification.

Submission

You will be expected to submit **one zip folder** containing at least the following files:

- project1.pdf: a writeup description of your project1
- project1.html: the main HTML file to run your project (you can rename this file)
- project1.js: the main JS file to run your project (you can also rename this file)

Additionally, you may submit any number of additional HTML and JavaScript files. All additional files submitted must be (briefly) described in your `project1.pdf` writeup. If your code relies on the provided `lib` utility folder, please include that folder in your submission.

Generally, you'll want to make sure that opening project1.html in a web browser will allow your project to run without issue. In Chrome, and other browsers, you can do this by double-clicking your main HTML file or by typing in: file:///C://path/to/project1.html (change C:// if your drive is different).

You may instead use a local server to open non-JS files (such as `.obj` files). If you do so, instead you should be able to run your project by running:

```
python -m http.server
```

In the folder containing your code, and then navigating to `localhost:8000` in your browser.

Making sure that your project is easy to run will help the grading team to grade your project quickly and fairly. If your project requires a different setup than either of these two approaches, please let us know in the writeup summary.

Materials

You are initially provided with two materials (additional starting code materials will be provided following Lecture 7):

- **project1.pdf**: this project description file
- **project1_rubric.pdf**: an exact project grading rubric
- **project1starter.html**: starter HTML code for this project
- **project1starter.js**: starter JS code for this project

You may, however, use any code provided as part of in-class demos, including your own previous assignments, so long as you cite where the code originated.

Some Tips! For some students, this may be the first time you have been asked to create a project of this scale for class. With this scale comes challenges that may be novel to you, most notably lots of bugs. If you haven't already, take this project as an opportunity to get familiar with:

- Git and Github (or something similar): Version control is going to be your best friend in all of your software projects, but *especially* for graphics (where it's easy to "break" things unintentionally with so many moving parts).
- Web Browser Inspectors: Using an inspector is extremely helpful for building and debugging anything that targets the web, and lets you see the console, how things are laid out in the DOM, etc. Just push F12 in most browsers.
- Your Editor: One of the best skills you can have when programming is familiarity with your editor, so maybe take some time to learn a shortcut or two! There are some fancy setups for running and updating JavaScript code that we don't cover in class which you might find useful to explore.

Project Guide

The following section is going to be a rough guide for how to complete the project. For specific grading details and descriptions, please refer to the rubric.

Part 1: Getting some models loaded

Your first priority is to get a basic 3D rendering system setup. This involves a lot of setup code, like creating a basic shader, initializing it, setting up your VBO, passing your data to GPU, etc. You will find that the starter code provided does a lot of this setup for you up-front, but it's worth thinking about (and reading through) some of the baseline code we've provided to make it easier to work with later.

When setting up a new feature in a graphics project, focus on getting the simplest "piece" working first – you should start by adding a triangle (or simple mesh) to the screen, with vertex-specific colors. The starter code we have provided should render a teapot to the screen and a thin green line -- this line is actually a grid (a very thin green grid), that you will be able to see better once you implement a camera.

When adding new models to the project, you should refrain from resending mesh data to the GPU when possible (that means you should be only calling gl.bufferData(...) once on startup, or if a mesh explicitly changes). Instead, you should focus on dynamically passing in your projection, view, and object matrices (via uniforms) into your shader.

Once you have figured out how to add something simple, you should start adding your custom models. You can create basic models in Blender, find them online, or hard-code vertex data directly. Each model should have at least 6 visible vertices, and at least one model should have per-vertex colors (so at least one model should have vertices of multiple distinct colors). You need at least three unique models for this project. We have provided helper functions for loading an `.obj` file that you may use to get the mesh data into your program.

When constructing your scene, keep in mind that models should be loaded exactly once (when the program starts) and setup in the shader attributes. You will not need to "swap" models out if you did everything correctly, and instead just adjust uniforms and draw each model independently.

Here are some design decisions that you'll encounter when doing this:

- What should my VBO look like? What attributes do I care about? What are my strides and offsets?
- What uniforms should I send to my vertex shader? When should I pass the uniforms?
- What work do I do on the GPU versus the CPU?
- How can I keep my code clean, allowing me to add more meshes in my scene in the future?

There are no right answers to these questions, but there are definitely wrong ones. Implement what makes sense to you, and remember our goal at this point is to get something on the screen.

This will likely be the longest part of the project, and understanding the concepts here are a core part of this class. Make sure that you really understand this – this project and other projects will build upon this foundation.

Beware of this Common Bug!

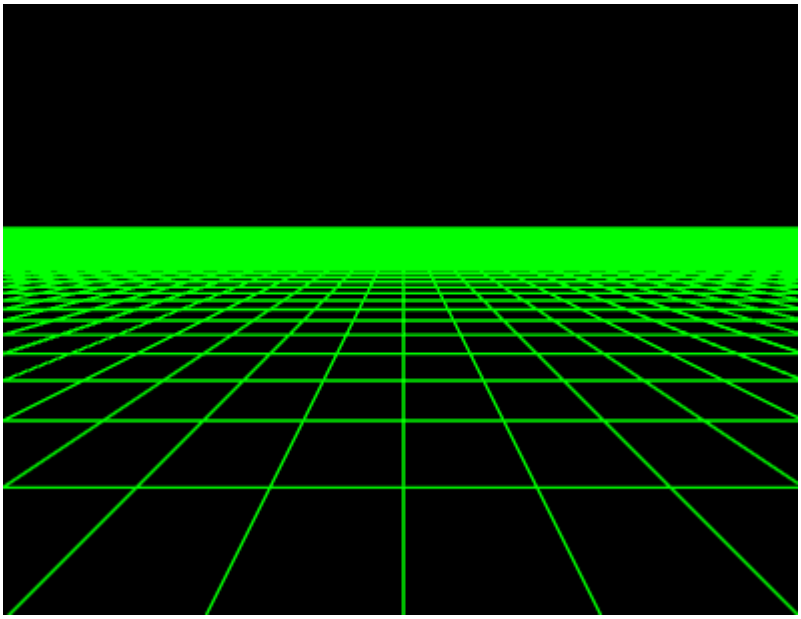
Whenever there is a mismatch in the data you bind to the VBO and how the VBO was configured, it often results in either nothing being drawn at all (assuming you did all the needed setup & shaders are compiling correctly), or an abomination like this:

Make sure to double check that the data that you are sending to the VBO is in the same format as you described using strides and offsets. It may be useful to draw out your VBO on paper!

Part 2: Adding a Camera

Depending on your implementation thus far, you may find this to be pretty simple, or require a mild refactor. Your camera should be able to be translated along at least a single axis. You do not need to add rotation to the camera, but you should be able to move it around. You should also implement the ability to switch between orthographic and perspective cameras in some user-friendly manner.

When initially implementing a camera, a way to test that your camera is working is to see if the default grid we provided appears on the screen (and moves "sensibly" as your camera moves). You are expected to keep a grid of some sort in your project submission, since it helps so much with seeing "correct" camera movement. See the image below for an example of what this grid should look like after you've implemented a camera:



Please make sure to document how to control your camera in the webpage. Some example controls could be using the arrow keys to move the camera, and/or pressing space to switch between orthographic and perspective cameras.

It will be useful to review how projection and view matrices work for this part. Recall specifically that the provided standard library has functions for orthographic and perspective projections. The default camera (before switching) can be either orthographic or perspective, whatever fits your vision of the scene best.

Part 3: Animating Your Scene

Now that you are able to render objects and move a camera round, hopefully this step isn't too bad – take this as time to flex your creative muscle, and showcase all of the work that you've done so far!

You will need to include 3 kinds of animation:

1. At least one model must be moving independently of user control (such as a windmill blade spinning or dog jumping up and down)
2. At least one model must be able to be interacted with by the user while the program is running (this could be through a slider moving a model around the screen, or pressing 'space' on the keyboard being used to launch a rocket)
3. At least one visual "part" of an assembly must be animated independent the rest of the assembly (such as an arm waving). Note that this requirement can be fulfilled through either independent movement or user-controlled movement (so points (1) or (2) in this list).

Note that this last requirement (3) means that one of your objects from part 1 must be constructed as an assembly rather than a single rigid model. You can make a model into an assembly in a variety of ways, but one of the easiest is just to break apart your mesh into two distinct models entirely, and animate each separately.

For organizing your animation code, recall our design discussion in class on scene graphs, and check out webGLfundamentals for concrete code design ideas.

Part 4: Writeup

For the last part of the project, you should create a write up. It should include:

- A descriptive title for your project (such as "Grass Waving in the Wind" or "Rainbow of Spinning Shapes")
- Your name and netid
- A short paragraph describing your project
- At least two images showing off your project
    - We recommend making at least one of these images show the result of your project after user interaction. This can help us out with grading if we can't figure out the user interaction for some reason.
- A 1 or 2 sentence summary of all the files submitted for the project
- Instructions for how to interact with your project
- If you are unsure on a rubric item, justification for how a feature in your project satisfies the requirements
    - This is totally optional, but if you're unclear on a rubric item but believe you met the requirements, adding this justification can help us see "what you were thinking"