# WebGL for P5.js: Expanding Usability for Beginners!


## Aidan Nelson

an2652@nyu.edu
Cell: +1 (845) 671.0137



## Project Abstract

This project would build upon P5.js' webGL functionality with a focus on helping the beginning coder understand how the computer represents and manipulates objects in 3D space. By expanding the camera controls, documentation, and beginning to implement the missing 3D primitives, I hope to make using the webGL mode in p5.js as easy as possible.


---

## Project Description

Despite that we live and breath in three dimensions, understanding how a computer manipulates and displays objects in 3D space is difficult! Someone learning to code in 3D must, in addition to learning coding fundamentals, understand how the graphics engine's camera parameters (field of view, position / rotation, and clipping planes) affects what we see on screen, how the 3D coordinate system maps to real-world space, and why lighting is required to perceive depth. P5.js' implementation of webGL already does a great deal to reduce this barrier-to-entry by providing a number of default cameras, centering all new primitives at the (0,0,0) point on the canvas, and providing a normalMaterial() which allows the user to build and test ideas without implementing lighting.

This project would continue to build out p5's implementation of webGL with a focus on easing the transition from coding in 2D to 3D. Specifically, I would like to work in the following areas:

- **Interaction**: understanding how a 3D scene is represented on a flat screen is vital for someone to feel comfortable learning to work in the webGL context. Expanding the toolset available to interact with and view 3D space would help facilitate this understanding:
    - **Camera Controls** This would build upon orbitControls() and implement additional functionality (pan/rotate/zoom). Movement contributes a great deal to depth perception, especially when other depth cues (parallax and lighting) are absent. A more advanced camera control would allow a user to see the 'depth' of a 3D sketch without having to first implement any animation or lighting. This has been mentioned in issues here and here, in the orbitControls src here ("implement full

orbit controls including pan, zoom, quaternion rotation, etc.") and in last year's GSOC planning document here. References for this would include:

- Three.js TrackballControls / source here / these controls provide a touchpad-friendly way of entering the three modes of control:
  - Rotation: Left mouse click & drag
  - Movement: Right mouse click & drag
  - Zoom: Two-finger touch & drag
- P5.easycam
- I have already implemented a primitive set of keyboard controls for my own 3D p5 sketches
- Unity scene view controls also seems like a good reference point.

I would discuss priorities for this expansion with my mentor, but initial thoughts are to work in this order:

- **Persistence**: current orbitControls() doesn't retain rotation when mouse is released. I think adding this would allow for a more complete camera API.
- **Movement**
- **Zoom**
- **Projection Mode Switch [optional]** This would allow user to switch between orthographic and perspective projections with a keypress. This is perhaps not necessary for p5's implementation, but would be a good way to visually demonstrate the difference.
- **cameraLookAt() [optional]** This article and this article seem be a good starting point for cameraLookAt(), which would take a 3 dimensional vector as an argument, and move the view to center that point.

- **debugView()** would add a gridded ground plane and 'skybox' of the type seen in Unity. The relationship between ground and sky serves as a visual cue to help the user orient their sketch in 3D space and a grid on the ground would give immediate and clear visual feedback for how the different camera parameters (orthographic or perspective projection and field of view) affect their view.

- **Documentation / Examples**: expanding the reference documentation for the current webGL methods would focus of explaining 3D concepts (projection types, clipping planes, etc.), defining or avoiding unfamiliar vocabulary (frustrum, gluLookAt, modelview matrix, etc.), and generally trying to match the tone and thoroughness of p5's 2D reference documentation.

- **Additional Primitives:** This project would seek to implement some of the missing 3D primitives as mentioned here, beginning with arc(), point() and image(). Ideally, this work

(and my continued learning of webGL) would proceed smoothly and allow me to continue with implementing bezierVertex(), curveVertex(), quadraticVertex() and text() primitives.

---

## Development Process

**Until May 14th:** Without a doubt, finishing this project requires a great deal of learning on my part. Fortunately, much of this learning can happen in tandem with developing for the library, but I would aim to begin on the following before the start of GSOC:

- **ECMA 5 syntax / contributing to open source**: I have been learning javascript with ECMA 6 syntax and using the p5.js library, so would need a greater familiarity both with writing in ECMA 5 syntax and with conventions of vanilla javascript before beginning work. Some helpful resources linked from the 2017 GSOC Processing write-ups include Eloquent Javascript and How to Write an Open Source LIbrary which I think would help give me a better sense of the overall structure and organization of the p5 library. Through this work, I would also come up-to-speed with p5.js' build and test process, and conventions for contributing (how to format PRs, etc.) as most of my use of git/hub until now has been on personal repositories.

- **WebGL / p5 implementation:** I have used webGL through p5 and, to a lesser degree, through three.js, but not written in vanilla webGL. Reading through the p5.js' wiki and issues has given me a sense of how webGL is implemented in P5 and -- more importantly -- why certain decisions were made, but I would need to begin learning vanilla webGL in order to be able to implement additional primitives in the summer. I believe the best way to do this would be to work through the examples at webglfundamentals.org and continue reading through the open (and closed!) webgl issues on p5's github. Another reference that has been recommended to me is WebGL Programming Guide.

**May 14 - June 14th:** My main goals for the first month are to be able to comfortably contribute to the library (formatting of pull requests, unit testing, ecma5 syntax, and in-line documentation), understand the design decisions driving p5's webGL implementation, and begin implementing camera controls. Kate Hollenbach and Stalgia Grigg's writeups have been extremely helpful as I begin this work, but reading through open and closed issues on p5's webGL will help give me a thorough technical and design understanding of the existing feature set. Some specific goals for contribution and learning are as follows:

- **CameraControls**: I would begin work on the additional interactive features proposed above.

- **Documentation:** Prior to working on documentation, I would like to better understand how most p5.js users use the current website and documentation. Do they look to the examples first? Or the 'learn' section? Do they go straight to the documentation? Or do they look to external learning resources (Coding Train, etc.)? I would discuss this with my mentors and approach any future work with this in mind. I personally use the reference docs and would like to expand upon those: adding detail and explanation to the current webGL methods. I would like to see webGL methods (such as 3D primitives and camera modes) have the level of straightforward description present in 2D methods (such as 2D primitives). Additionally, this would help me to get up to speed with the current code base and in-line documentation style.

- **[Learning] WebGL:** Concurrent with other work, I would continue to learn vanilla webGL basics. By this point I expect to have worked through many of the vanilla webGL examples on webglfundamentals, and will continue to work on understanding the complete render pipeline as described in Stalgia's stepthrough.

- **[Learning] Unit Testing:** With the goal of implementing unit tests for any added camera API features and any add'tl 3D primitives later in the summer, I would spend time learning about unit testing broadly, and p5's grunt-mocha based process specifically. Some resources I will look to here, here and here.

- **Develop Timeline:** With the guidance of my mentor(s), and working with any other GSOC'ers working on this webGL implementation, I would develop a more thorough timeline for ongoing work and priorities for the rest of the summer. Ideally, this document would serve as a reference for current and future contributors, and as such would lay out a framework for work extending beyond the summer.

**June 14th - July 14th:** I hope to go into my second month of GSOC more confident with vanilla webGL, the current p5 webGL render pipeline, contributing to p5 (formatting PRs, adding unit testing, and a course of action for the rest of the summer. Much of the work mentioned above can continue into this month, but I would like to focus on:

- **CameraControls / Unit Tests:** Any remaining improvements to cameraControls should happen ASAP so that I can begin writing unit tests for them.

- **WebGL Primitives:** By this point, I will begin working on implementing missing 3D primitives, beginning with arc() and moving on to point() and image().

- **Documentation:** Continued work to update existing documentation and add documentation to any new features.

**July 14th to August 14th:** The last month of my summer would be spent implementing any changes in my code as suggested by mentors and continuing with work begun already, specifically:

- **CameraControls / Unit Tests:** Any remaining updates to unit tests should happen now.

- **WebGL Primitives / Unit Tests:** Write unit tests for arc, image, and point. Once done, continue with implementing bezierVertex, curveVertex, quadraticVertex and text primitives and their corresponding unit tests, completing as many as is possible before end of GSOC.

- **Documentation:** Continued work to update existing in-line reference documentation and add documentation to any new features. Complete a write-up about my GSOC updates!

---

## More about you

Hello!

I am an artist and teacher currently studying at NYU's Interactive Telecommunications Program. Over the past several years, I have been fortunate to work in a variety of creative and technical roles in theater -- as an actor, teacher, director and in technical production -- and still find (some) time to work on personal projects (building electrostatic headphones, cnc routers and starting a company selling overalls!)

I am extremely excited to be proposing this project for Processing for a couple of reasons. First, working in interdisciplinary environments has taught me time and again that no one learns, thinks or expresses themselves in quite the same way. Only by creating an environment in

which everyone is excited to contribute can a project become greater than the sum of its contributors.  Processing seems to have done this in spades and I would love to be a part of it.

Second, I love programming and would like to learn more (specifically about graphics programming and about contributing to open source projects)!  I have been especially inspired by projects like this to learn the fundamentals of webGL and would extremely excited to continue that work through GSOC.  While I am beginner coder, I am eager to help and excited to continue learning.

Below are some of my projects built using P5.js:
Point Cloud Particle System / code
Citibike NYC Bikeshare Visualizer  / code
Herding Behaviors / code

Website: http://www.aidanjnelson.com/
CV: http://www.aidanjnelson.com/files/resume.pdf
Github: https://github.com/aidannelson