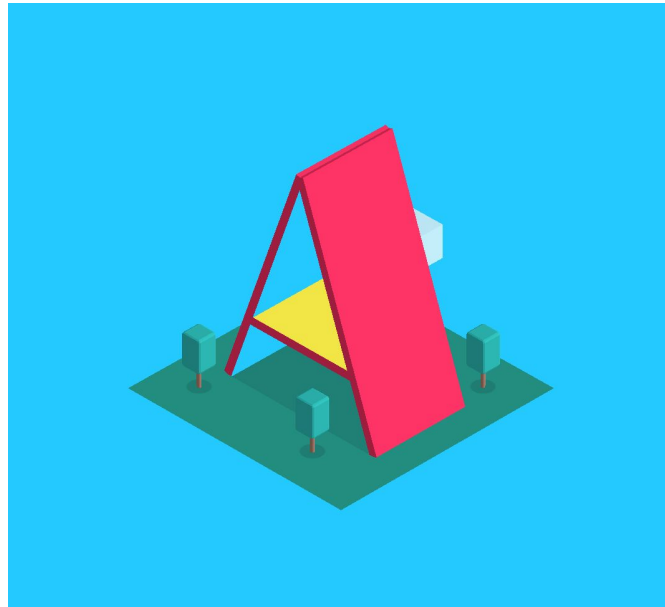


Charles Heller(ID: 010639997)
Soohwan Kim (ID: 010655915)

Assignment 5 Project Report

<https://socketblaster.com/AFrameDemo>

<https://github.com/chheller/AFrameDemo>



Objective

This project aimed to utilize A-Frame in order to provide a dynamic visualization embedded in a website built using NodeJS. The client would be able to save parameters associated with various primitive shapes to MongoDB. Upon client request, the Node server would retrieve data from the MongoDB and pass the information to the client, which would then form the parameters of a primitive shape to be rendered by A-Frame.

Motivations

The ability to readily learn and adapt to new technologies such as A-frame and MEAN stack is an exceedingly valuable skill, and there was great effort made to attempt to use new but relevant technologies in this project to continue to hone such skills. The use of

unfamiliar but relevant technologies such as MongoDB and Node was decided upon to expose ourselves to some of the myriad of different options available in web development and paradigms in coding and development practices.

High Level Architecture

The NodeJS server forms the backbone of the project and utilizes Javascript for the majority of its functionality, with Express being used to create the framework for the backend of the Node server. The client accessible view(webpage) uses JADE to generate its HTML and Javascript for its dynamic functionality that allows for user requests to be handled without reloading the webpage as well as being able to handle multiple functions on one page cleanly and effectively.

The client can choose from three primitives to input parameters for, with the parameter fields adjusting to each different polygon's required parameters. Then the client can send the data fetched from the fields to the server in a JSON format using AJAX. The server saves those various parameters to be saved to MongoDB using Mongoose as an intermediary. Mongoose takes the data and depending on what primitive was selected by the user, saves the data to the database using the correct schema. For example, if the user selected a cube and filled out the desired parameters in the fields and then clicked to submit, the data in the fields would be encoded and passed to the NodeJS server. The NodeJS server would then utilize Mongoose to save to MongoDB.

When the client requests to show all of a certain type of primitive, NodeJS uses Mongoose in order to retrieve the data for the associated collection and passes the data back to the client. The client then feeds the data into A-Frame, a 3d visualization plugin that utilizes HTML5 in order to display any and all polygons that were retrieved from the database. The client may also request to delete any polygons that may exist in a certain collection.

The various user input fields are error checked, with reasonable limits on the numerical inputs. Any input exceeding these limits are automatically corrected to the nearest limit. Invalid inputs such as letters in a number field are rejected. The color field only takes in valid HTML hexadecimal color codes, and if an invalid code is inputted, corrects to FFFFFFFF.

Database Architecture

The database was implemented using MongoDB, a high-performance, NoSQL, schema-less and open-source database solution. Mongo is a key-document based database system, where each entry is a JSON document accessed by its unique ID from within a collection. Collections function as tables would in SQL. The Mongoose middleware for Express allows NodeJS to communicate with mongo efficiently. Mongoose is one of many middlewares available to Express for connecting to MongoDB, including a MongoDB driver itself.

```
var cubeSchema = new Schema( {  
  position : String,  
  rotation : String,  
  color : String,  
  depth : Number,  
  width : Number,  
  height : Number  
}, {collection: 'cubecollection'});  
  
var cube = mongoose.model('cube', cubeSchema);
```

Above is an excerpt from index.js that defines the Mongoose schema for a cube. While MongoDB itself is schema-less, Mongoose implements schemas itself. While this is against MongoDB convention it does provide useful consistency across developers as well as laying out an explicit model ensures consistency.

Server Architecture

The server was built using the Express framework on top of NodeJS. Express creates a backbone and does much of the heavy lifting when starting a website, making it a very powerful and convenient tool. Additionally, Express supports middleware such as Mongoose, MongoDB drivers, and Jade- an HTML pre-processor. Much the work for this assignment was done within index.js, aframedemo.jade, and aframe.js. These files

contain all the code necessary for the page to function, ignoring the setup code done by Express. Server side, index.js is the only file that contains relevant code. There is the code for rendering the page itself as well as responding the POST requests.

```
router.post('/AFrameForm', function(req, res, next) {
  var db = req.db;
  var data = req.body.data;
  var shape = req.body.shape_filter;

  var pos = req.body.posx + " " + req.body.posy + " " + req.body.posz;
  var rot = req.body.rotx + " " + req.body.rotz;

  if(shape == 0) {
    var out = new cube({position : pos, rotation : rot, color : req.body.color, depth : req.body.depth, width : req.body.width, height : req.body.height});
    out.save(function(err, out) {
      if (err) return console.error(err);
      else {
        console.log(out);
        res.send("Cube successfully saved!")
      }
    });
  }
  else if (shape == 1) {
    var out = new sphere({position : pos, rotation : rot, color : req.body.color, radius : req.body.radius});
    out.save(function(err, out) {
      if (err) return console.error(err);
      else {
        console.log(out);
        res.send("Sphere successfully saved!")
      }
    });
  }
  if(shape == 2) {
    var out = new cylinder({position : pos, rotation : rot, color : req.body.color, radius : req.body.radius, height : req.body.height});
    out.save(function(err, out) {
      if (err) return console.error(err);
      else {
        console.log(out);
        res.send("cylinder successfully saved!")
      }
    });
  }
});
```

Above is the entire code excerpt that controls the posting of form data from the client to MongoDB. First, the request data is parsed into useful variables for every primitive shape and then put through a series of conditional statements to determine which model to create from the given data. Once a model is created, it is saved to MongoDB and a response is sent to the client to notify them that their submission has been

successfully saved. Because NodeJS is a single-threaded, non-blocking application, functions are written according to an asynchronous pattern. That is functions and callbacks dictate the flow of data from client to server and vice versa. This is an important functionality of node, as it allows it run very quickly while waiting on slower I/O processes to complete. Because database transactions are relatively slow I/O operations, any transactions must be written asynchronously in order to function properly.

Front End Architecture

The front of the website is written in Jade (now known as Pug). Jade is an HTML pre-processor that takes the tedium out of HTML as well as adding in useful functionality such as loops, conditionals, functions and mixins. Jade is a powerful tool and allows for writing HTML extremely quickly and efficiently.

```
form.shapeform(action="/AFrameForm" method="post" id="formsubmit")
  div.col-md-1
  div.col-md-10(class="text-left")
    div.row
      label Cube
      input.shape(type="radio" name="shape_filter" value=0 checked)
      label X-Position:
      input(type="number" name="posx" style="width:50px" value=0 id="posx")
      label X-Rotation:
      input(type="number" name="rotx" style="width:50px" value=0 id="rotx")

      label.cube Height:
      input.cube(type="number" name = "cubeheight" style="width:50px" value=1
id="cubeheight")

      label.sphere Radius:
      input.sphere(type = "number" name = "radius" style="width:50px" value=1
id="radius")
      label Color:
```

```
      input.color(type = "string" name = "color" style="width:60px" value="000000"
id="color")
```

Above is an excerpt from `iframe.jade`. This excerpt is the first row of three that define the input form. Jade is a whitespace sensitive language, like Python, and such blocks of code are determined by their indentation level relative to their top level neighbor.

Additionally, `iframe.js` contains the clientside javascript code.

```
function formSubmit(event) {

    $.post('/AFrameForm', $(this).serializeArray(), function(res) {

        $("#alertarea").append(res);

    });

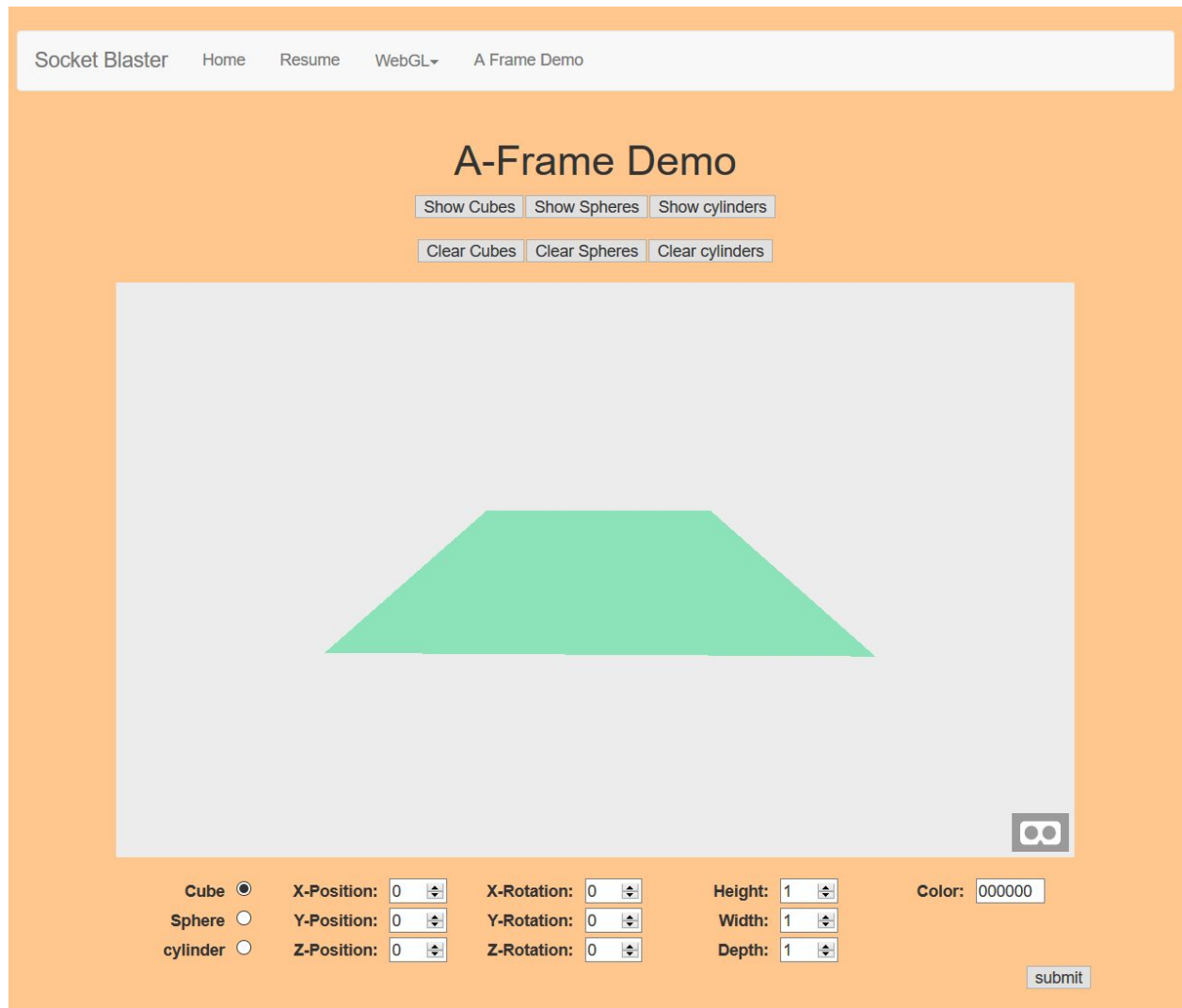
    event.preventDefault();

}
```

Above is an excerpt from `iframe.js` which defines the event handler for the form submission. This function utilizes both AJAX and JQuery to simplify things tremendously. By using AJAX we are able to prevent HTML from redirecting the page and post to the server without changing the page. This is important for giving the page it's reactive feel and aesthetic, which are currently huge topics in the web development industry. Technologies such as AngularJS and ReactJS create a foundation for creating single page, highly interactive and reactive websites.

Additionally, we leverage A-Frame's powerful library to implement WebGL from an extremely high level, implementing only HTML/Jade code to produce an interactive platform that runs on any web platform, including mobile, and offers a Virtual Reality option. A-Frame is a revolutionary technology produced by Mozilla that abstracts much of the difficult and gritty work of creating 3D and VR products through an internet browser.

Results



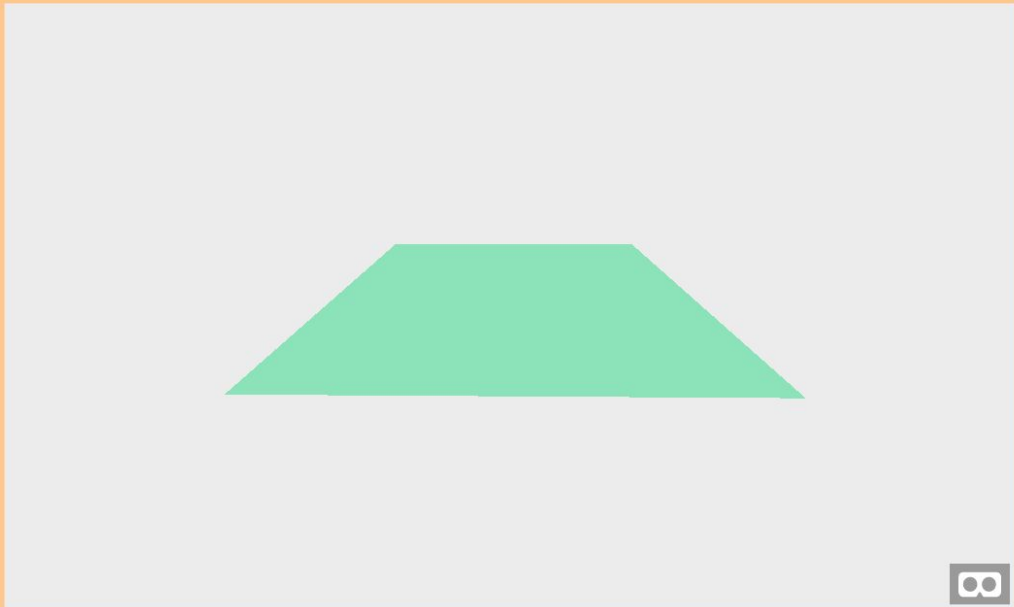
The initial State of the website.

Socket Blaster Home Resume WebGL▼ A Frame Demo

A-Frame Demo

Show Cubes Show Spheres Show cylinders

Clear Cubes Clear Spheres Clear cylinders



Cube ☒

Sphere ☐

cylinder ☐

X-Position: 0

Y-Position: 0

Z-Position: 0

X-Rotation: 25

Y-Rotation: 25

Z-Rotation: 25

Height: 1

Width: 2

Depth: 1

Color: A00000

submit

Submitting parameters for a cube.

Socket Blaster Home Resume WebGL▼ A Frame Demo

A-Frame Demo

Show Cubes Show Spheres Show cylinders

Clear Cubes Clear Spheres Clear cylinders

Cube ☐

Sphere ☒

cylinder ☐

X-Position: 0

Y-Position: 0

Z-Position: 0

X-Rotation: 25

Y-Rotation: 25

Z-Rotation: 25

Height: 1

Width: 2

Depth: 1

Color: A00000

submit

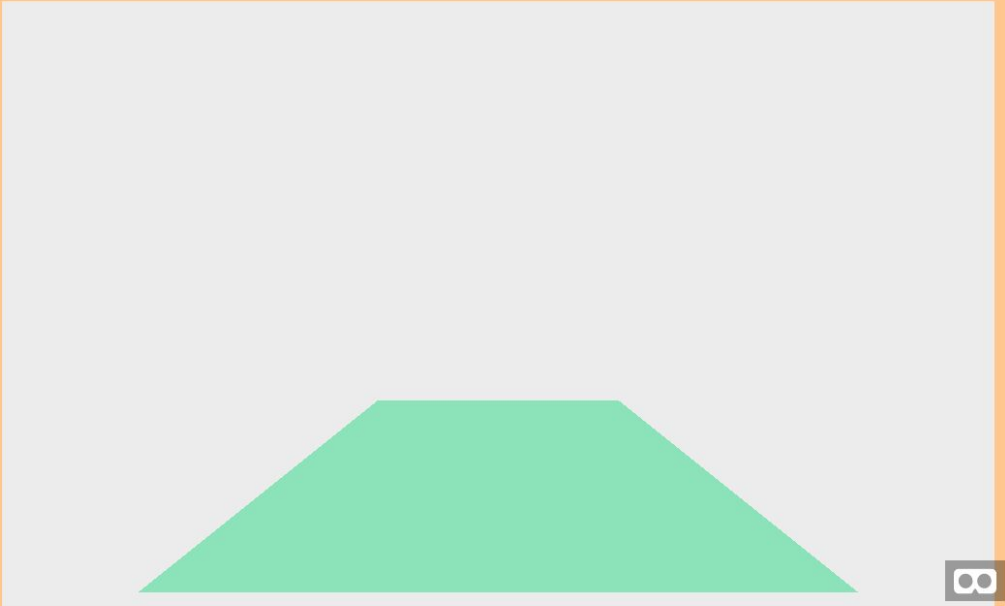
Submitting parameters for a sphere.

Socket Blaster Home Resume WebGL▼ A Frame Demo

A-Frame Demo

Show Cubes Show Spheres Show cylinders

Clear Cubes Clear Spheres Clear cylinders



Cube ☐

Sphere ☐

cylinder ☒

X-Position: 1

Y-Position: 1

Z-Position: 1

X-Rotation: 25

Y-Rotation: 25

Z-Rotation: 25

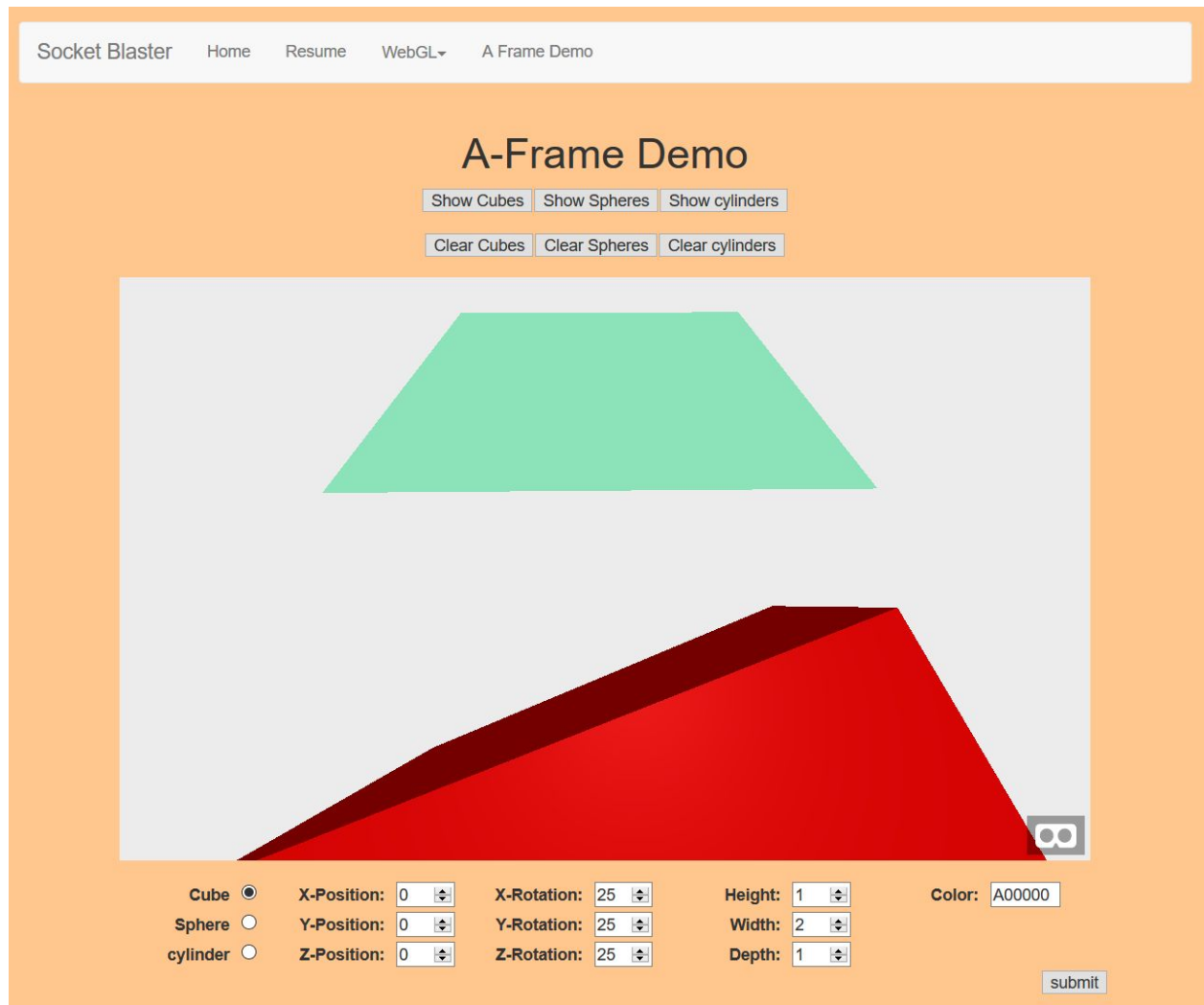
Radius: 1

Height: 1

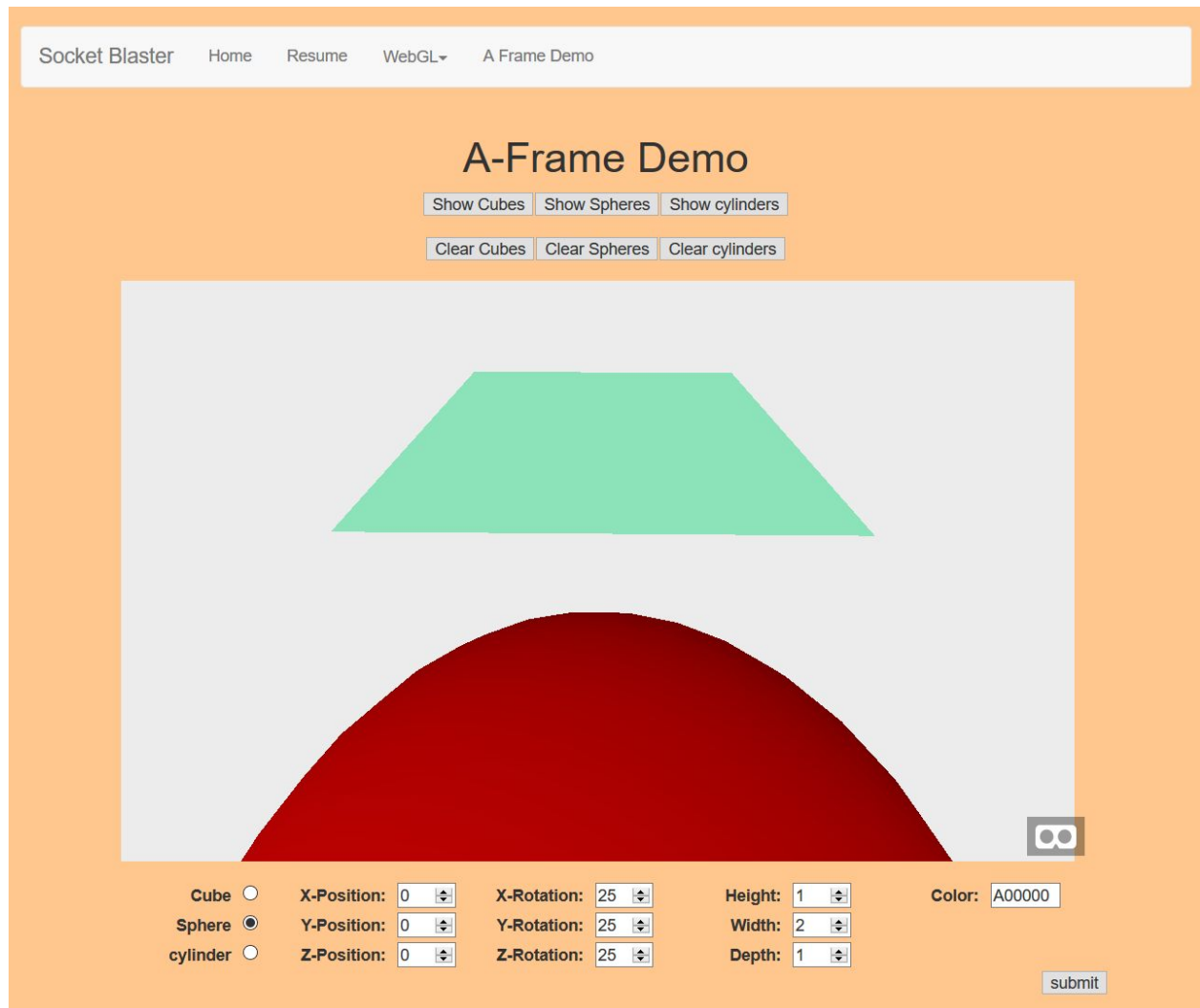
Color: B00000

submit

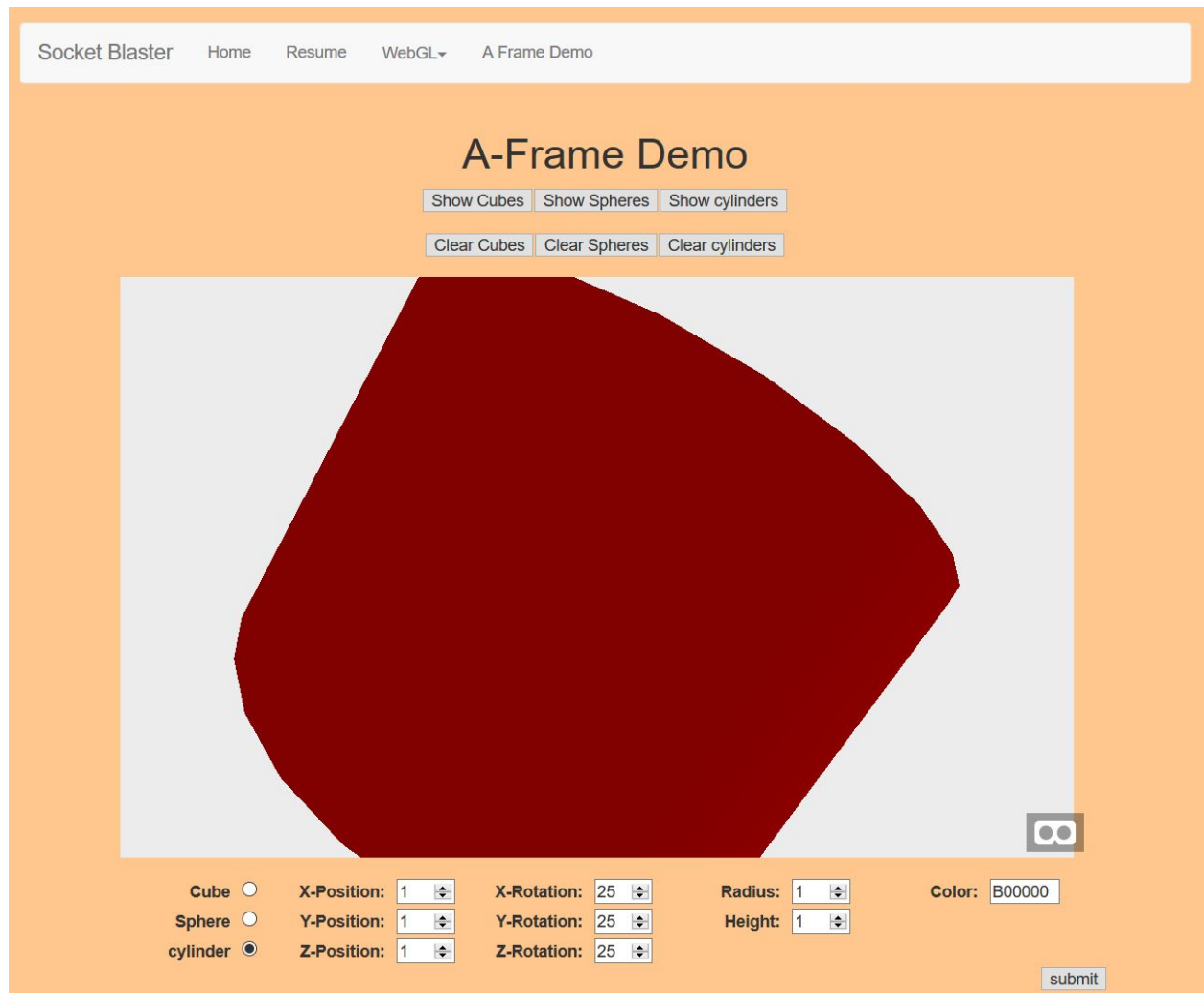
Submitting parameters for a cylinder



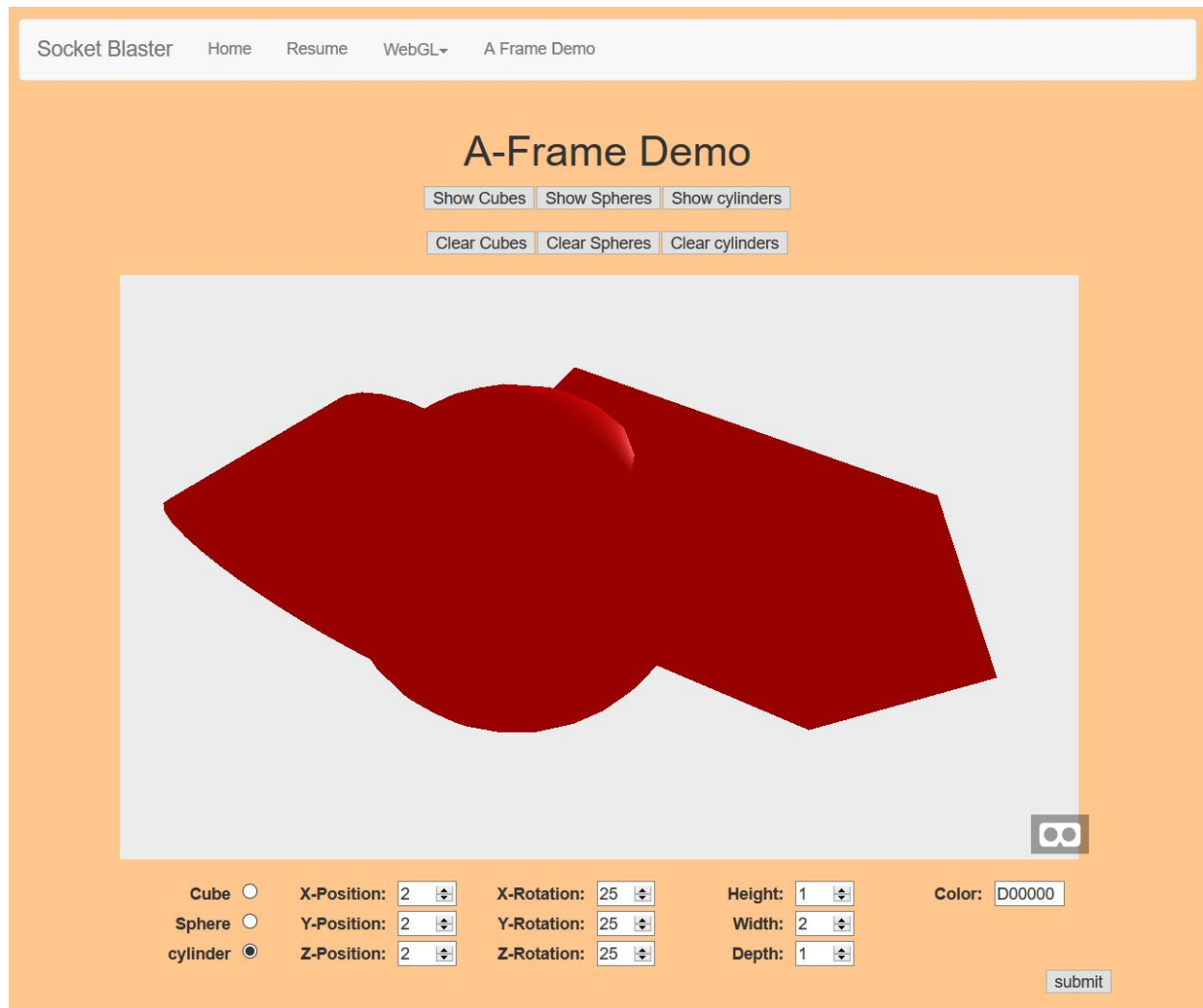
Displaying the submitted cube.



Displaying the submitted sphere.



Displaying the submitted cylinder.



Displaying all of the submitted polygons at once.