CS 361 Scott Barlowe

Whacky, Tacky Turnstile

Description

In this assignment you will be building a very bizarre turnstile wall out of three-dimensional objects in a file called **wall.js**. The scene consists of two walls, two gray doors, and a dark pole around which the turnstile can rotate. The walls should be separated by and centered around the turnstile. You may not use **BoxGeometry** or any other auxiliary libraries. In a **wall.html** file, make the size of the window 1024 x 768. There are several sites listed the **Resources** folder in Blackboard that will resize your browser for testing. SHIFT-CTL-M also works for setting the window size. If you have not found a method for efficiently debugging, I encourage you to use SHIFT-CTL-J.

Walls

Each wall should be separated by the width of a turnstile door, have 5 cubes along the x axis, and have 4 cubes along the y axis. Each block should be 40 units long, wide, and tall. Although you may choose the color scheme, the cube color should alternate in a checkerboard fashion on the front, back, side, and top exactly as shown in the example. Each wall must be built by defining each vertex on an initial block with Vector3/Face3 and then defining different colors for each face on the initial block. The vertices of the initial block should not be hardcoded but instead be defined in terms of the center of the turnstile. For each wall, use loops that create clones of the initial block and then offset each block according to its place in the wall. The checkerboard pattern can be achieved by appropriate rotations as the clones are created.

Turnstile

Each turnstile door should be gray, 10 units thick, 80 units wide, and 160 units tall. The doors should be placed at 90 degrees from each other and intersect at each other's center. One door should be parallel to the wall and the other door should be perpendicular to the wall. The turnstile pole should be centered horizontally in the middle of the ground, centered in the space between each wall, and at the intersection of the doors. You should be able to see a small portion of the pole at both the top and bottom of the doors.

Ground

The ground will be represented by creating an instance of **PlaneGeometry** with a green material. The ground should be 600 x 600.

Interactions

Pressing 'q': The turnstile doors rotate a small angle around the pole and the walls remain stationary. (Can be done repeatedly.)

Pressing 'p': The walls rotate a small angle around the turnstile and the turnstile doors remain stationary. (Can be done repeatedly.)

Pressing 'r': All objects (except for the camera) are returned to their initial state.

Pressing '0': The camera is positioned at its initial position in the upper right corner closest to the viewer.*

Pressing '1': Moves the camera to the upper right corner farthest from the viewer.*

Pressing '2': The camera is positioned at the vertical and horizontal center of the scene and directed at the front of the wall.*

***Note:** To switch camera positions, create a new instance of an orthographic camera identical to the initial camera, set the new instance to the initial camera variable, and then set the position of the new camera instance. All camera positions should be configured such that both walls and turnstile are completely within the window and such that the camera is looking at the center of the scene. A *small* portion of the ground's corners can be outside of the window if necessary.

Implementation

Your project should consist of the files **wall.html** and **wall.js**. **wall.js** should consist of the following functions and any others that you may find useful.

init

Initializes the renderer, initializes an orthographic camera, initializes the scene, and adds a "keydown" event listener. Initialize the camera with

```
var viewLength = 500;
var aspRat = canvasWidth/canvasHeight;
var camera = new THREE.OrthographicCamera(-aspRat*viewLength/2,
                aspRat*viewLength/2,  viewLength/2, -viewLength/2, -1000, 1000);
```

draw

Along with any other necessary code, calls the following functions and any others that you may find useful.

initGeom

Initializes the arrays for the vertices, faces, and color of the initial cube.

buildGround

Builds the ground for the walls and turnstile.

buildFristCube

Uses the structures initialized in **initGeom** to create the first cube of the wall.

buildWall (side)
Builds the wall (either right or left depending on the value of **side**) by using clones of the first cube.

buildTurnStile
Builds and configures the turnstile doors and pole.

renderScene
Renders the scene.

DELIVERABLES

Compiling and Executing: Your program should execute with Google Chrome or Mozilla Firefox. Place both of your files in the same directory level. To grade your project, I will simply open **wall.html** with Chrome or Firefox and set the resolution to 1024 x 768.

Format: Your code must adhere to accepted program conventions regarding line length, documentation, spacing, etc. A part of your grade will be how effectively you eliminated hardcoded values.

Teams: You may work in teams of one or two students. If you work in teams of two, make sure that the team members are listed in **wall.js**'s comment section. People in different groups may talk about the concepts in the assignment but may not share code.

Submission: Compress your files together and turn in the single file to the assignment link in Blackboard by midnight on **April 5th**. You may turn in your assignment up to five days late. Five points will be deducted for everyday late, including holidays and weekends, after which the program will not be accepted.

EXTRA CREDIT

Notice that the walls seem a little distorted. To get a more realistic view and for 5 points added to your added to your project score, add interactions activated by pressing 3, 4, and 5 that accomplish the same as pressing 0, 1, and 2 but use a perspective camera. We will cover perspective cameras later so you may have to do some independent research to learn more. To receive extra credit, your project can not be late.

GRADING

Programs that do not compile will receive a score of 0. You will be graded according to the following:

Program execution **75%**
(Minimum program functionality as described above and the elimination of hardcoded values)

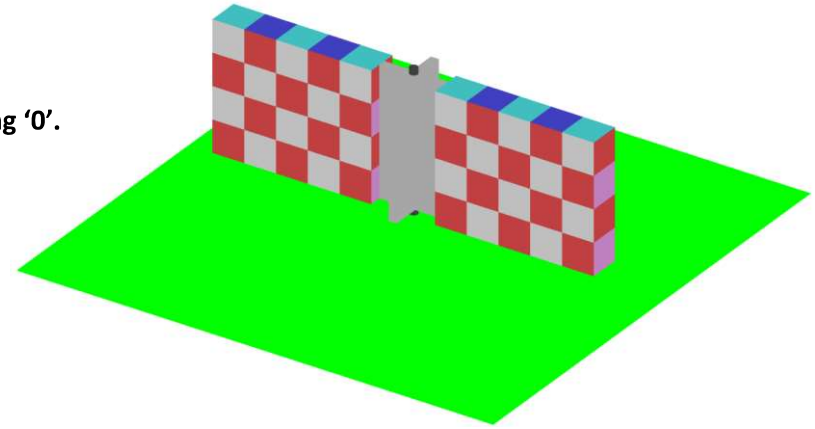
Quality **10%**

(Quality of the rendered image including aesthetic appearance)

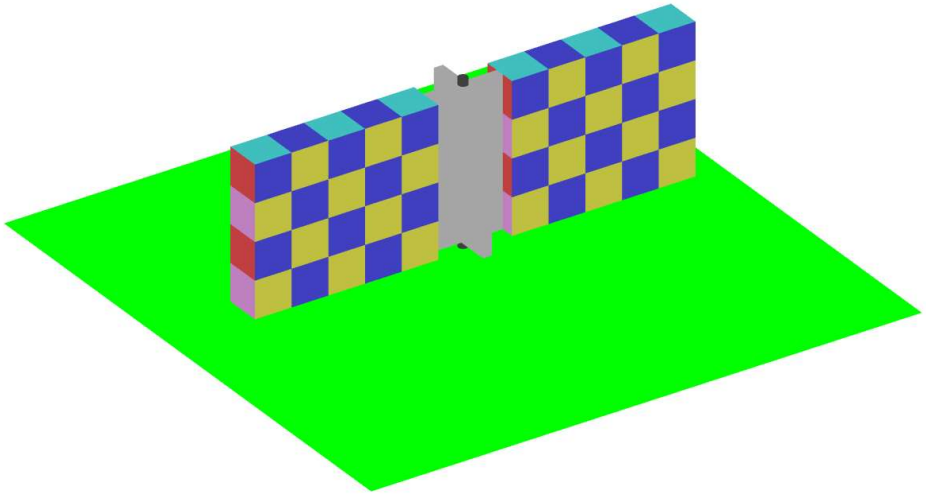
Coding Style/Documentation **15%**
(Line length, spacing, comments, function length, elimination of hardcoded values, etc.)

Examples**

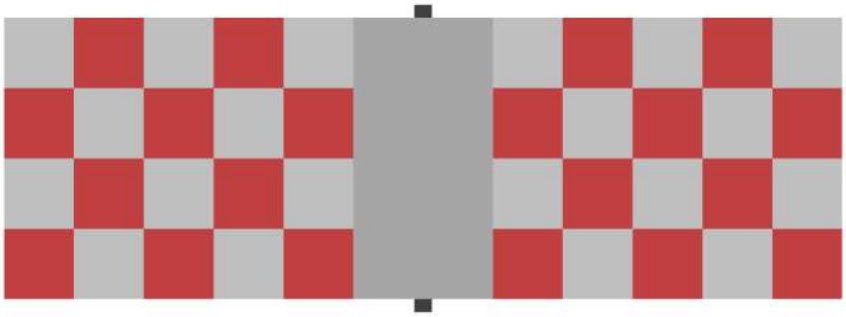
Initial view and after pressing '0'.



View after pressing '1'.



View after pressing '2'.



****Note:** For the other interactions, see the video posted in Blackboard along with this assignment. There are more interactions in the video that you are not required to implement.