American University of Sharjah
School of Engineering
Computer Science & Engineering Dpt.
P. O. Box 26666
Sharjah, UAE

Instructor: Dr. Michel Pasquier
Office: EB2-213
Phone: 971-6-515-2883
E-mail: mpasquier@aus.edu
Semester: Fall 2017

CMP 430 COMPUTER GRAPHICS

PROJECT #3

Due date/time: Saturday 9 December 2017 at 6 PM

Instructions: Send to your instructor, before or on the due date, the source code of your programs i.e., *complete HTML files with their Three.js script within.* File names should be numbered after each step i.e., "bouncing1.html", "bouncing2.html", etc. Penalties for late submissions strictly apply (cf. Course Policy).

You are to complete this last assignment *as a team of 2-3 students.* Note that you are always encouraged to discuss with others the class material and related software as well as general ideas of the assignment in order to improve your understanding of the task. You are also welcome to give each other examples that are *distinct* from the assignment in order to demonstrate how to accomplish such task. However, *each team must work independently* and is *not* allowed to *share* their answers or to *copy* other answers or sourced material, etc. Plagiarism and cheating will be penalized accordingly, starting with a 0 grade (cf. Course Policy).

Objective:

In this assignment, you will work as a team to learn and use *Three.js*, a popular, free and open source library that builds on WebGL to allow for Computer Graphics modelling and rendering, to create a simple animation of bouncing objects.

Background:

Before you tackle the assignment proper, you need first to get yourself further acquainted with Three.js, using the provided class tutorial and examples, and the script template included as appendix. Full documentation and countless examples cand be found at https://threejs.org/ (and other Three.js related websites).

Assignment:

You should start by creating your own HTML+Three.js source file, copy-pasting the code provided as appendix. Next, study the code, check Three.js documentation as needed, and proceed to create your own version of the two functions: `makescene()` where you create all necessary objects and add them to your scene, and `update()` where you code the changes occuring at each step of the animation.

Note that you should code these functions incrementally, adding features as you go while following the steps highlighted hereafter. The provided demo creates a wireframe box and places one solid ball at the center, as illustrated in Fig. 1. It also animates the ball, making it move back and forth a small distance along the X axis. In the following, you should generalize the idea to make the ball(s) bounce between the walls of the surrounding box, and move along all X-Y-Z directions…

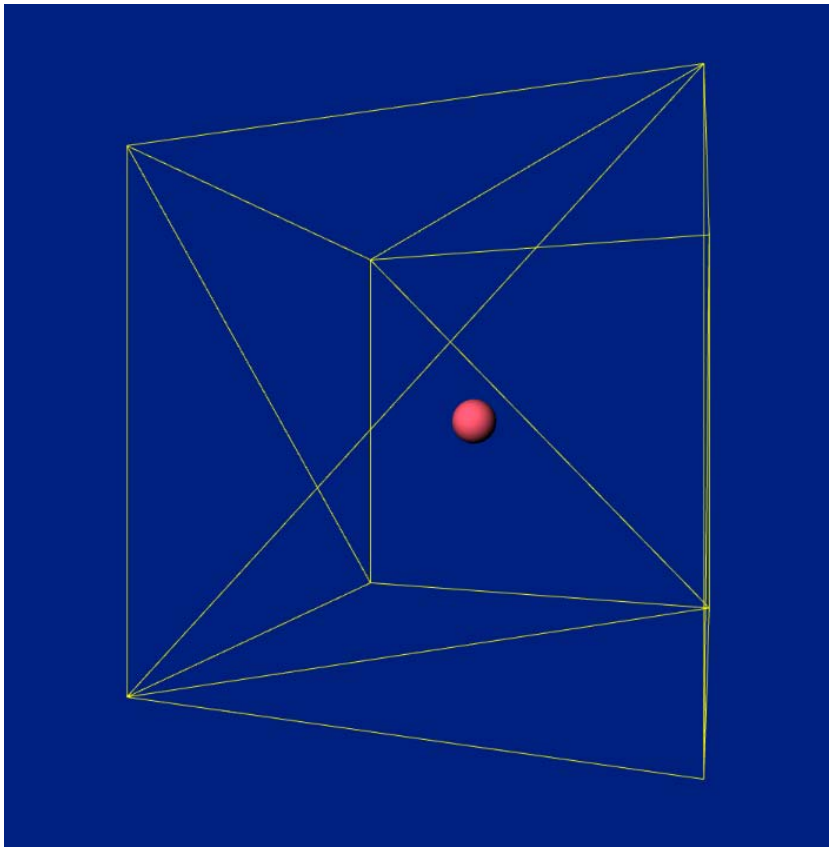

Figure 1: Example of a single ball bouncing endlessly inside a box.

As for coding style, you should follow standard Three.js coding practices, as can be seen in every sample program from class or the Threejs.org website, etc. Use obvious variables names and/or comment as needed. Make sure to define variables and values appropriately. For instance you should define the surrounding box size once at the start e.g., `boxsize = 6`, and subsequently use that variable wherever needed. (Do not hardcode values all over the place!) Basically you should make it easy to change parameters such as size, initial position, direction of motion, step value, number of objects, and so on so forth. (And your code should still work!)

Stage 1 ("C" level) – save file as "bouncing1.html":

Modify the given Three.js code to create a single ball that forever bounces inside a cube. You can set dimensions as you please but the ball radius should be between 1/20 to 1/10 of the cube size. The entire cube should be visible, in a fashion similar to that shown in Fig. 1. Feel free to change light, material colors, etc. as you like if that makes your rendered images and animation look nicer.

You must adapt the given code to move the ball along all three X-Y-Z axes at the same time. You could for instance define 3 deltas e.g., `dx = 1, dy = 1, dz = 1`, that are the motion increments at each animation step, and update the position of the ball accordingly. To make it bounce on the walls that are the cube faces, you only need to reverse the direction of motion when the ball reaches the corresponding face e.g., `dx = -dx`, etc. Your code should work for different deltas e.g., `dx = 2` or `dz = 3`, etc. (you can assume small values) as well as different initial positions of the ball (always inside the cube, of course).

Finally, add a mouse controller so that we can rotate the cube around (and the ball bouncing inside it) as well as zoom in and out of the scene. Again you can simply use/adapt the code from class examples and/or the website.

Stage 2 ("B" level) – save file as "bouncing2.html":

Modify the previous Three.js code to add gravity to the animation. That is, the ball should now accelerate on the way down and decelerate on the way up (where up/down refer to the vertical Y axis). Assume there is no friction, hence the ball will still keep bouncing forever. Basically you need to subtract a small constant (to represent the 9.8 m/s earth gravity) from the Y coordinate. When the ball is going down, this will make it go faster; when going up, it will slow it down.

Now add another parameter to represent friction. You should also have a variable that allows toggling friction on/off. To model friction you need to reduce the displacement by a small percentage, whenever the ball hits one of the walls. This will slowly decrease the amplitude of the motion (along all three axes) so that the ball will eventually stop (at the bottom of the cube).

Next, add a key controller and the necessary code so that pressing the '+' (plus) key will increase gravity while pressing the '–' (minus) key will decrease it. Moreover, pressing the 'f' key should toggle friction on/off.

Finally, replace the cube with 6 planes corresponding to the six cube faces, and modify your code so that pressing 'c' will display the faces with a solid color, while pressing 'w' will display them as wireframe (everything else working the same, of course). This will allow zooming in while the cube faces are displayed as solid walls and seeing the ball bounce against them still (whereas a solid cube is otherwise… solid, so we cannot zoom inside).

Stage 3 ("A" level) – save file as "bouncing3.html":

Place a second ball at the center of the cube. It should be twice the size as the bouncing ball, and of a different color. Modify your code so that the moving ball bounces off the center ball as well as the cube faces. You should implement perfect elastic collision between the two balls (i.e., friction-less bounces).

Now add the code to change the material of the two balls so that pressing the 'd' key will use diffuse shading, pressing the 's' key will use specular shading, and pressing the 'n' key will go back to normal, flat shading. Add lights as you see fit to make it work and have a nice looking animation.

Lastly, modify the texture of the center sphere so that it becomes a perfect mirror. You should have a variable that allows toggling this mirror effect on/off, and pressing the 'm' key will enable or disable it.

Your three programs will be assessed based on, in order, how many of and how well the given specifications are met, the overall niceness of the animations, and the quality of the code itself.

Appendix – Three.js project template

```
<!DOCTYPE html> <!-- see class tutorial / examples for comments, more features, etc. -->
<html>
  <head>
    <title>Three.js Project Template</title>
    <style>
      body { margin: 0; }
    </style>
  </head>
  <body>
  <script src="http://threejs.org/build/three.min.js"></script>
  <script>
    var camera, scene, renderer, mesh, boxsize = 6, dx = 0.05;

    init();
    makescene();
    animate();

    function init() {
      camera = new THREE.PerspectiveCamera( 75, window.innerWidth / window.innerHeight,
                                            0.1, 1000 );
      camera.position.z = 10;

      scene = new THREE.Scene();

      var light = new THREE.PointLight( 0xffffff, 1.2 );
      light.position.set( -2, 2, 6 );
      scene.add( light );

      renderer = new THREE.WebGLRenderer( { antialias: true } );
      renderer.setSize( window.innerWidth, window.innerHeight );
      renderer.setPixelRatio( window.devicePixelRatio );
      renderer.setClearColor( 0x002080 );
      document.body.appendChild( renderer.domElement );
    }

    function makescene() { // replace below with your code ...

      var box = new THREE.Mesh( new THREE.CubeGeometry( boxsize, boxsize, boxsize ),
                                new THREE.MeshBasicMaterial( { wireframe: true,
                                                               color: 0xFFFF00 } ) );
      box.rotation.y = Math.PI / 2.5;
      scene.add( box );

      var ballGeometry = new THREE.SphereGeometry( 0.3, 32, 32 );
      var ballMaterial = new THREE.MeshLambertMaterial( {color: 0xf6546a} );
      mesh = new THREE.Mesh( ballGeometry, ballMaterial );
      box.add( mesh );
    }

    function update() { // replace below with your code ...

      mesh.position.x += dx;
      if ( mesh.position.x > 2 || mesh.position.x < -2 ) // move back and forth
        dx = -dx;
    }

    function animate() {
      requestAnimationFrame( animate );
      update();
      render();
    }

    function render() {
      renderer.render( scene, camera );
    }
    </script>
  </body>
</html>
```