Lab 4 - Mesh

Due Monday by 6pm **Points** 10 **Submitting** a file upload **File Types** zip
Available after Sep 23 at 6pm

1. Download **meshDemo.zip**.
2. Follow the example in class to complete the vertex and fragment shaders "vert.rgb.glsl" and "frag.rgb.glsl".
3. Follow the example in class to complete the Mesh class (skeleton provided).
   - Mesh.constructor
     - Description:
       - constructs a new mesh
     - Args:
       - gl : the WebGL context
       - program : the shader program which will be used to display this mesh
       - positionArray : the array of data for the position attributes of all vertices
       - indexArray : the index array for use in WebGL's drawElements method
       - position : the transform's initial position, default is the default Vector
       - rotation : the transform's initial rotation, default is the default Quaternion
       - scale : the transform's initial scale, default is Vector(1,1,1)
     - Tasks:
       - instantiate the transform with the desired position, rotation and scale (use "super")
       - store the WebGL context in the field, "this.gl"
       - store the shader program in the field, "this.program"
       - store the shader program's vertex position attribute location in the field "this.positionAttribLocation"
       - store the shader program's world transform uniform location in the field "this.mWorldUniformLocation"
       - create a buffer object for the vertex positions; store it as the field "this.positionBufferObject"
       - populate the position buffer object with the data in positionArray (cast as a Float32Array)
       - create a buffer object for the index array; store it as the field "this.indexBufferObject"
       - populate the position buffer object with the data in indexArray (cast as a Uint16Array)
   - Mesh.activate
     - Description:
       - set up the WebGL context in preparation to draw this mesh
     - No args!
     - Tasks:
       - update the transform
       - tell the WebGL context which shader program to use
       - bind the indexBufferObject to WebGL's element array buffer

        - bind the positionBufferObject to WebGL's array buffer
        - enable a vertex attribute array for the positionAttribLocation
        - set up the vertex attribute pointer for the vertex position attribute
        - set the shader program's world transform matrix (located at "this.mWorldUniformLocation") to this mesh's world matrix
    - Mesh.draw
        - Description:
            - prepare the WebGL context to draw this mesh
            - draw the mesh
            - clean up the WebGL context
        - No args!
        - Tasks:
            - call this.activate
            - draw this mesh with WebGL's drawElements method
            - unbind WebGL's array buffer and element array buffer
4. Independently complete the RGBMesh class (skeleton provided).
    - RGBMesh.constructor
        - Description:
            - constructs a new RGBMesh
        - Args:
            - gl : the WebGL context
            - program : the shader program which will be used to display this mesh
            - positionArray : the array of data for the position attributes of all vertices
            - indexArray : the index array for use in WebGL's drawElements method
            - normalArray : the array of data for the normal attributes of all vertices
            - colorArray : the array of data for the color (rgb) of all vertices
            - position : the transform's initial position, default is the default Vector
            - rotation : the transform's initial rotation, default is the default Quaternion
            - scale : the transform's initial scale, default is Vector(1,1,1)
        - Tasks:
            - construct a mesh with all applicable arguments (most of them)
                - Hint: Mesh is RGBMesh's super class
            - store the shader program's color attribute location in the field "this.colorAttribLocation"
            - store the shader program's normal attribute location in the field "this.normalAttribLocation"
            - create a buffer object for the color attribute data; store it in the field "this.colorBufferObject"
            - populate the color buffer object with the data from colorArray (cast as a Float32Array)
            - create a buffer object for the normal attribute data; store it in the field "this.normalBufferObject"
            - populate the normal buffer object with the date from normalArray (cast as a Float32Array)
    - RGBMesh.activate (overwrites Mesh.activate)
        - Description:
            - prepare the WebGL context to draw this mesh
        - No args!

- Tasks:
  - call Mesh.activate on this RGBMesh
    - Hint: Mesh is the super class of RGBMesh
  - bind the colorBufferObject to WebGL's array buffer
  - enable a vertex attribute array for the shader's vertex color attribute (i.e. the attribute located at "this.colorAttribLocation")
  - set up the vertex attribute pointer for the vertex color attribute
  - bind the normalBufferObject to WebGL's array buffer
  - enable a vertex attribute array for the shader's vertex normal attribute (located at "this.normalAttribLocation")
  - set up the vertex attribute pointer for the vertex color attribute

5. At this point, you should be able to run the provided index.html and see two cubes orbiting and spinning.
6. Create a file "shapes.js". In it create the Cube class with the following static methods:
   - positionArray
     - Description:
       - creates and returns a position attribute array for a cube
     - No Args!
   - indexArray
     - Description:
       - creates and returns an index array for a cube, corresponding to the output from positionArray
       - don't forget, counterclockwise front faces!
     - No Args!
   - normalArray
     - Description:
       - creates and returns a normal attribute array for a cube corresponding to the positionArray
     - No Args!
   - solidColorArray
     - Description:
       - creates and returns a color attribute array for a cube where all vertices are the same color
     - Args:
       - color as a triple [r, g, b], all floats from 0 to 1, denoting the red, green and blue of the desired color
   - Cube.create
     - Description:
       - creates and returns a cube
       - for now, the cube will be an RGBMesh
       - in future labs we will add extra arguments allowing for the creation of a textured mesh (UVMesh) instead!
     - Args:
       - gl : you know what this is by now
       - program : this too
       - fill : an array with 3 floats between 0 and 1, of the form [r, g, b] specifying the color of the cube

        - e.g. [1.0, 0.0, 0.0] for "red"
    - position : position Vector, default is default Vector
    - rotation : rotation Quaternion, default is default Quaternion
    - scale : scale Vector, default is Vector(1,1,1)
- Tasks:
    - construct the position, index, normal and color arrays for a cube (using the other Cube class static methods)
    - construct and return an RGBMesh using these arrays and the appropriate arguments above

7. Use Cube.create() to shorten demo.js; no attribute arrays should be constructed in the demo when you're done!
8. zip up your completed demo (all files from the original zip, plus "shapes.js") and submit.

You may find the following link useful; it provides explanation of which methods need to be used to initialize an object, versus the ones which need to be used every time the object is rendered: **Drawing Multiple Objects (https://webglfundamentals.org/webgl/lessons/webgl-drawing-multiple-things.html)**

File Upload | Google Doc | Dropbox | Google Drive

Upload a file, or choose a file you've already uploaded.

File: Choose File No file chosen

+ Add Another File

Click here to find a file you've already uploaded

Comments...

Cancel | Submit Assignment