

# Assignment 3: 3D Scene Editor

Handout date: 10/28/2019  
Submission deadline: 11/18/2018, 23:59 EST  
Demo date: 11/22/2019

This homework accounts for 17.5% of your final grade.

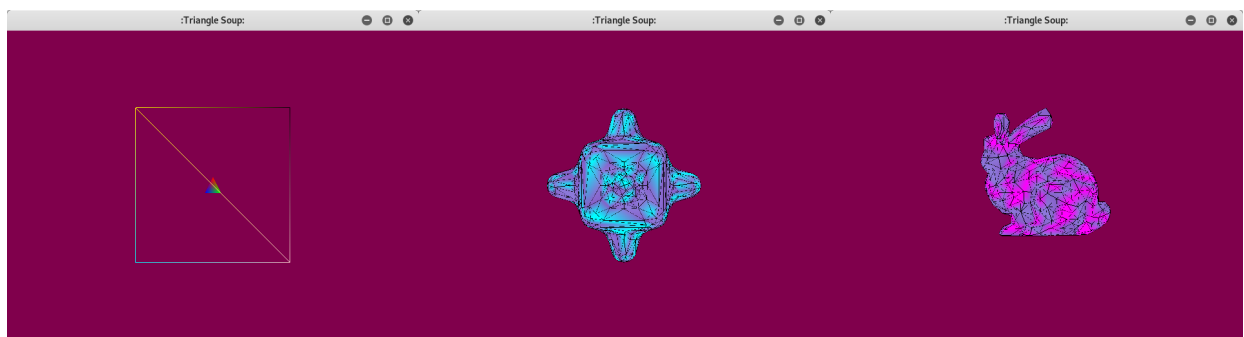
## 1 Mandatory Tasks

For each task below, add at least one image in the readme demonstrating the results. The code that you used for all tasks should be provided.

### 1.1 Scene Editor

Implement an interactive application that allows to add, edit, and delete 3D meshes. The scene should always contain at least one light source. New objects can be added to the scene in three ways:

- The key '1' will add a unit cube in the origin
- The key '2' will import a new copy of the mesh *bumpy\_cube.off*, scale it to fit into a unit cube and center it on the origin
- The key '3' will import a new copy the mesh 'bunny.off', scale it to fit into a unit cube and center it on the origin



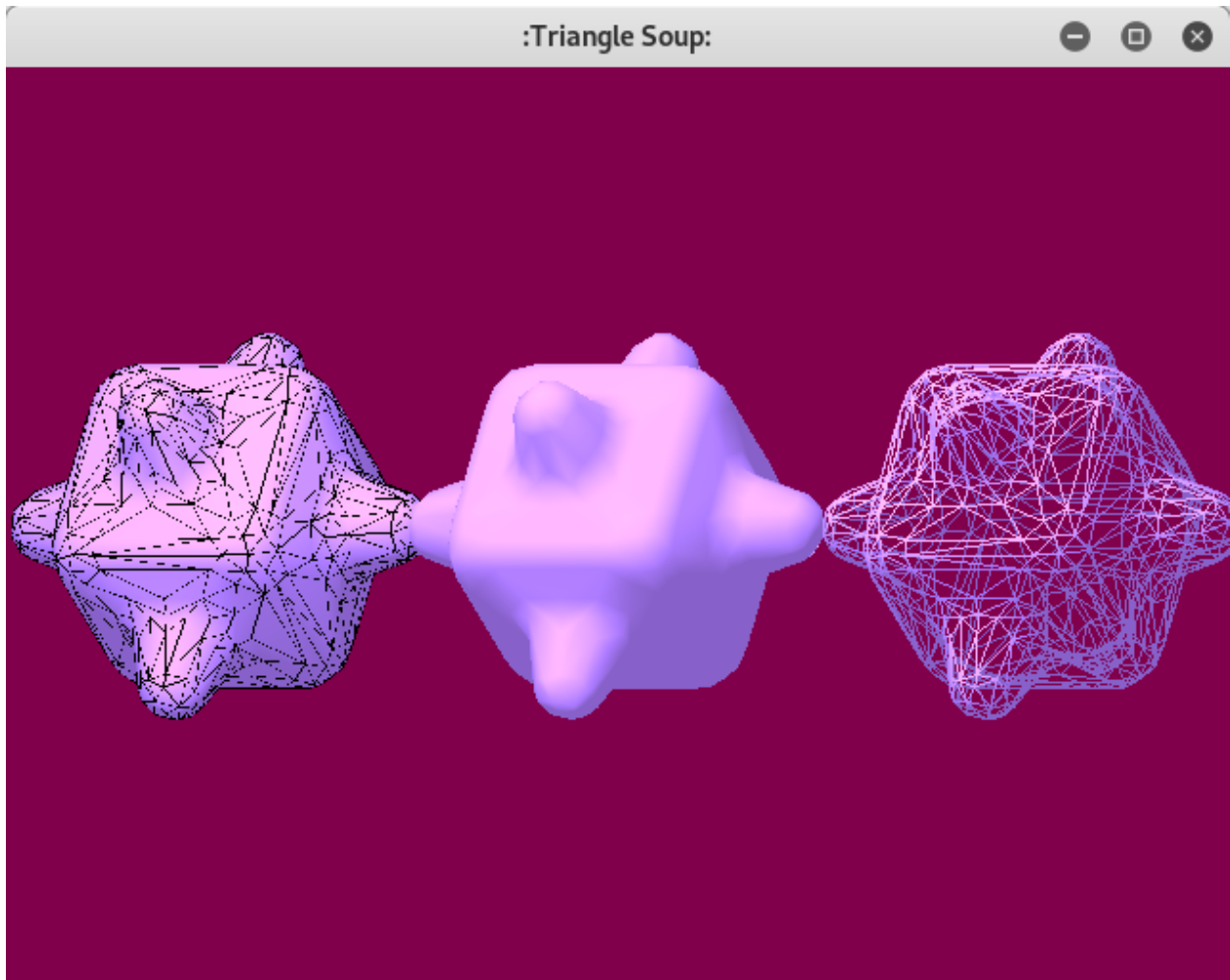
Note that you can have multiple copies of the same object in the scene, and each copy can have its own position, scale, and rotation. For this exercise, all transformations **MUST** be done in the shader. The VBO containing the vertex positions of each object should be uploaded only once to the GPU.

## 1.2 Object Control

Clicking on a object will select the object, changing its color. When an object is selected, it should be possible to translate it, rotate it around its barycenter, and rescale it without changing its barycenter. All these actions should be associated to keyboard keys (and the choice of keys should be detailed in the readme).

**Keys are: h, j, up, down, left, right for rotation, k and l for scaling, and g c v b t y for translation, t to move back and y to move forward to change the shading you use 8, 9, and 0 for wireframe, flat, and phong shading respectively** Each object also has a rendering setting associated with it, which can be one of the following three options:

1. Wireframe: only the edges of the triangles are drawn
2. Flat Shading: each triangle is rendered using a unique color (i.e. the normal of all the fragments that compose a triangle is simply the normal of the plane that contains it). On top of the flat shaded triangle, you should draw the wireframe.
3. Phong Shading: the normals are specified on the vertices of the mesh and interpolated in the interior. The lighting equation should be evaluated for each fragment.



To compute the per-vertex normals you should first compute the per-face normals, and then average them on the neighboring vertices. In other words, the normal of the vertex of a mesh should be the average of the normals of the faces touching it. Remember to normalize the normals after averaging.

**Made an adjacency list like thing by going through the list of indices, then add each “face number” to the a list of faces corresponding to each vertex. I believe it runs in  $O(\text{indices})$  time**

When an object is selected, it must be possible to switch between the different rendering modes by pressing three keys on the keyboard.

### 1.3 Camera Control

Add the possibility to translate the position of the camera (similarly to the previous assignment), but in this exercise the camera should always *point to the origin*. It should be possible to move it around, but the camera should always face the origin.

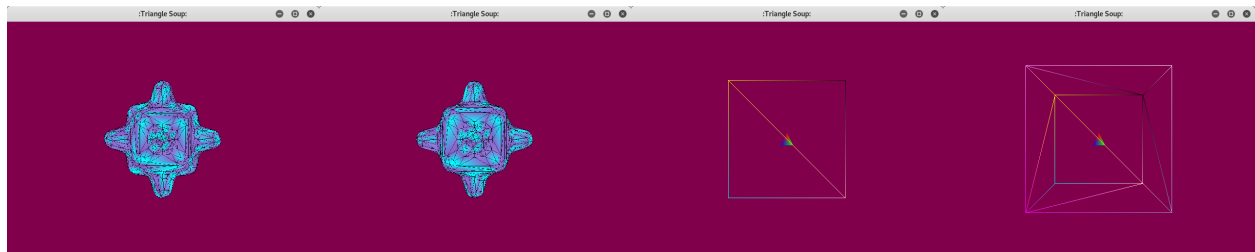
**For regular camera control, you move around using WASD and the plus and minus and equal keys are for zooming in and out**

**For trackball, I actually rotated the scene as I didn't want to deal with matrix stacks, uploading a uniform rotation vector that stored the degrees the scene is rotated for each of its axis, similarly I uploaded a scale float to the VAO. Because I used rotation matrices, it essentially gives you the trackball functionality. You turn on/off trackball using tab, and you rotate around using WASDQE**

Implement both a *orthographic camera* (similar to the one that you used for Assignment 2, but in 3D) and a *perspective camera*. The cameras should take into account the size of the window, properly adapting the aspect ratio to not distort the image whenever the window is resized. All functionalities should work after resizing the window, including object selection and editing of the scene.

**In the VAO I added:**

```
"vec3 projectedPos = (position * rotMat * scale + translation) * w_rotMat* w_scale;"
"float perspectiveFactor = projectedPos.z * 0.5 + 1.0;"
"if(ifPerspective == 0){gl_Position = view * vec4(projectedPos, 1.0);}"
"if(ifPerspective == 1){gl_Position = view * vec4(projectedPos*perspectiveFactor, 1.0);}"
```



### Optional Tasks

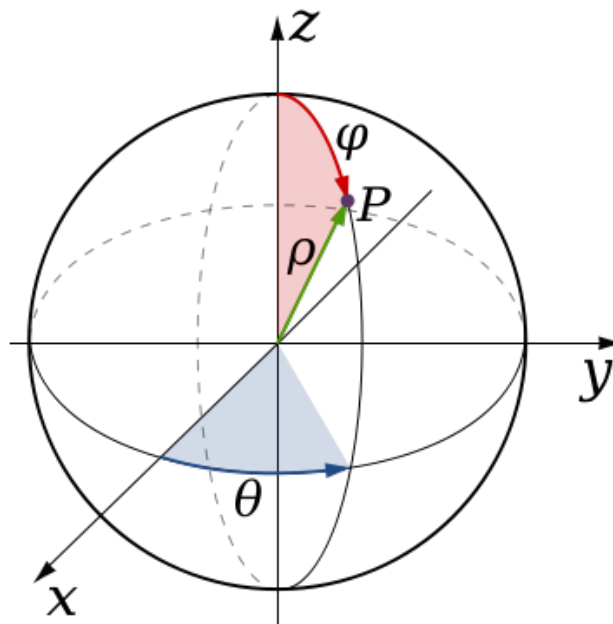
These tasks are optional. The first task is worth 2% of the final grade, the second 3%, and the third task 2%. The optional points are added to the points of the other exercises, but the total sum of points that you gain with exercises cannot be more than 80%.

## 1.4 Export in SVG format

### Hit Z

Add the possibility to export the scene currently drawn on the screen in SVG format [https://en.wikipedia.org/wiki/Scalable\\_Vector\\_Graphics](https://en.wikipedia.org/wiki/Scalable_Vector_Graphics). The exported SVG should be compatible with <https://inkscape.org/>. The triangles should be rendered only using flat shading and they should be deformed according to the current view (orthographic and perspective should both be supported). Hidden triangles should not be rendered (to check for hidden triangles, you can cast three rays to its vertices, and if any vertex is not visible you will simply skip the triangle).

## 1.5 Trackball



### look at camera control for more details

Use a trackball to control the camera. This can be achieved restricting the movement of the camera on a sphere centered on the origin. The easiest way to do it is to parametrize the sphere using spherical coordinates, and to assign keyboard keys to move the camera on the sphere. The camera should always look at the origin.