Getting Started

Here are some ideas for assignment 1.

1. Question 1: this one's not too hard. You have to modify `make_scene` and `draw_scene`, plus you have to make sure the viewport matches the scene dimensions, which is handled by the `Camera` in `main()`.

   Basically, question 1 tests whether you could install glfw and glm.

2. Question 2: for this question, you need some understanding of the glfw event model. Notice the loop in the `main` program: draw the scene, then wait for a user event. When it occurs, redraw the scene, and wait again.

   Keyboard events are reported by `processInput`. This is called at the top of the `main` event loop.

   All other events are reported through call-back functions:

   - `framebuffer_size_callback` : this handles window-resize events. The `Camera` must be updated. You don't have to modify this.

   - `mouse_motion_callback` : this gets called lots of times, each time the mouse moves a little bit. You should call `camera::mouse_to_world` to locate the mouse in world coordinates, and save the mouse position in a global variable.

   - `mouse_button_callback` : this gets called when a mouse button is pressed or released. If the left button was pressed, you should do as the comments tell you: check if the mouse was on some vertex, and if so, set a global variable that says which vertex is now active. If the user did not click on a vertex, then *no* vertex is now active.

   DO NOT DO ANY DRAWING IN THE CALLBACK FUNCTIONS. All the drawing should be done in `draw_scene`.

3. When the program runs, it expects a command-line argument, which is the name of a graph file (I'm giving you several graph files). The format of the files is as follows:
   - first line : # vertices
   - second line : # edges
   - all other lines : two vertices, which define an edge.

   So your `main` program's first action should be to read the file.

   If you're working in Visual Studio, you'll have to locate the `Debug` folder that contains the program's `.exe` file, and open a shell there. Then type `a1q2 graph-simple.txt` or something like that, to run the program.

4. You will have to create `Mesh` objects to represent the shapes in the graph: the vertices and the edges.

   When the program starts, the vertices should be arranged in a circle of radius 1 and center (0 0), equally spaced. The math for this is just polar-to-cartesian conversion:
   - x_i = cos(angle)
   - y_i = sin(angle)

   where angle ranges from 0 to 2π, in equal increments.