THE UNIVERSITY OF WESTERN AUSTRALIA

SCHOOL OF COMPUTER SCIENCE & SOFTWARE ENGINEERING

CITS3003 Graphics & Animation - 2013: Project Part I

DUE: 3pm Tuesday October 8 (Week 10) – Group Code (12%) +

Individual Code (4%) + Individual Report (4%)

This is the first part of the project for CITS3003. This part contributes 20% of your unit mark.

Groups: You should complete this project in groups in two, but you may work alone if you chose. Each individual is also required to separately write their own report and to separately design and implement their own small extension of the specified functionality. Groups of three may be allowed with special permission and will be marked against an appropriately higher standard. Changing groups for Part 2 of the project will also require special permission. Each individual must submit a file `groups.txt` via cssubmit that has one line for each group member, each with the member's student number followed by their name, by **3pm Tuesday 17th September**.

Submission: By **3pm Tuesday 8th October** a member of each group should submit the group's code via http://secure.csse.uwa.edu.au/run/cssubmit, and all group members must submit an individual report. Individuals may optionally submit their own version of the code with their report if they feel this better represents their individual extension than the group's final version.

Originality: You may discuss with other students the general principles required to understand this project, but the work you submit must be the sole effort of your group, and the individual components must be the sole effort of yourself. Your code should be based on the supplied starting point, unless you have special permission to use another environment. You are also allowed to base parts of your code on the lab sample solutions, as well as the examples from lectures and the recommended text. If you base your code on code from any other source you ***must*** clearly reference the origin in a comment that also says exactly which parts of your code are based on code from that origin – it will be considered academic dishonesty if you omit any such references.

The main project task

You are required to complete an implementation of a simple *scene editor* that allows a collection of objects to be arranged in a scene and various properties of them to be changed, such as colour, shininess and texture. Sample executables have been provided for you to try – your program should provide the same functionality as these.

Files provided

The Linux directory `/cslinux/examples/CITS3003/project-files` contains (or will very soon, before Monday) the files that you will need for this project, including:

- `sample-linux.tgz` and `sample-linux-full.tgz` these contain an executable for the sample solutions that should be used as a reference for the functionality you are required to build – for Linux in the labs. The full version includes the `models-textures` directory, which you'll need when not in the CSSE labs, and even when in the labs you may want to use a copy on a flash drive to speed things up.
- `sample-windows-full.zip` - the same for Windows NT/XP/Vista/etc. Might also work under Wine if you have a different Linux from that in the labs.

- `project1-start.zip` - a starting point for your project with code for reading the supplied models and textures into data structures, drawing models with textures and simple lighting, and much of what you'll need for your menus and mouse interactivity. Should compile under both Linux and Windows (via MinGW/MSYS).
- `models-textures` - a directory containing 55 3D models and 30 textures used by the scene editor. The models created from 3D scans using techniques recently developed by researchers at UWA. Note: these models are **NOT** to be publicly distributed.

[Note: this folder contains 240MB, and you won't need a copy if you work on the lab machines.]

- `models-textures.zip` - zip file (85MB) if you want to work on your own machine, or if you want to use a copy on a flash drive to load faster.

Overview of Functionality to be Implemented

Your scene editor needs to support the same functionality as the sample solution, including the following. Note that much of the functionality is already implemented in the starting point – the parts that you need to implement have been chosen to be relatively quick to implement while still covering all the relevant concepts.

A. The starting point draws the ground, an initial mesh object and a sphere showing the position of a single point light source. But it has the camera facing always facing the same way – you can moving it forwards and backwards using the scroll wheel, via the `viewDist` variable, but there's no way to move it otherwise.

   The variables `rotUpAndOverDeg` and `rotSidewaysDeg` are already set appropriately when moving the mouse with the left button down, or the middle button down (or shift + left button). Modify the `display` function so that the camera rotates around the centre point according to these variables, in the same way as the sample solution.

B. Similarly, changing the location and scale of objects is already implemented, via the `loc` and `scale` members of the `SceneObject` structure, which is stored in `sceneObjs` for each object in the scene. But the angles array in this structure doesn't affect what's drawn, even though it is appropriate set to contain the rotations around the X, Y and Z axes, in degrees. Modify the `drawMesh` function so that it appropriately rotates the object when drawing it in the same way as the sample solution.

   The starting point code also moves objects in different directions compared to the sample solution- you should also fix this (hint: there's more than one way to do this).

C. Implement a menu item for adjusting the amounts of ambient, diffuse and specular light, as well as the amount of shine, modifying the `materialMenu` and `makeMenu` functions so that they change the appropriate members of the SceneObject structure. The code to use these to affect the shading via the calculation for the light is already implemented.

   Follow the corresponding implementations of other similar menu items. Use the `setTool` function which has four arguments which are pointers to four float's that should be modified when moving the mouse in the x and y directions while pressing the left button or the middle button (or shift + left button). After each pointer argument is a float that scales the affect of the mouse movement – recall that the shine value needs to be able to increase to about 100 or so.

D. In the starting point, triangles are "clipped" (not displayed) if they are even slightly close to the camera. Fix the `reshape` function so that the camera can give more "close up" views of objects. Hint: there's more than one thing you'll need to change.

E. Also modify the reshape function, so that it behaves differently when the width is less than the height: basically whatever is visible when the window is square should continue to be visible as the width is decreased, similar to what happens if the window height is decreased starting with a square.

F. Modify the vertex shader so that the light reduces with distance – exactly how it reduces is up to you, but aim for the reduction to be noticeable without quickly becoming very dark, similar to the sample solution.

G. Move the main lighting calculations into the fragment shader, so that the directions are calculated for individual fragments rather than just the vertices. This should have a very noticeable effect on the way the light interacts with the ground, since the ground only has vertices at the corners.

H. Generally specular highlights tend towards white rather than the colour of the object. Modify the shader code so that the specular component always shines towards white, regardless of the texture and the colour assigned to the object.

I. Add a second light to both the C++ code and the shaders. The second light should be directional, i.e., the direction that the light hits each surface is constant and independent of the position of the surface. In particular, use the direction from the second light's location to origin as the direction of the light. Like the first light, use `sceneObj[2]` to store the coordinates and colours of the second light, and make it otherwise similar to the first light, including placing a sphere at the coordinates of the light.

J. Each group member must individually design and implement at least some small, but non-trivial additional functionality. E.g., (1) allowing objects in the scene to be deleted, or (2) duplicated. (3) Adding a wave motion to the ground or (4) to the scene objects. (5) Allowing spot lights. (6) varying colours over time (All of these are acceptable, but towards the simpler end.)

K. Other additional functionality by the group may earn a small number of bonus marks, but not more than 2 marks out of 20. Generally focusing on other tasks is a better use of your time. E.g.,(1) loading and saving scenes, (2) clicking on objects to edit them, (3) setting paths for objects to move along over time, (4) motion under physics, (5) particle effects, (6) attaching one object to another so that they move and rotate together. (These are tougher than above, but also suitable as your individual additional functionality.)

Individual Report

As well as your source code, each group member must **separately** submit a file containing a brief report (about 2 pages) that quickly describes:

- How well your program works, including a list of any functionality that doesn't work.
- The steps you followed in building your program, focusing on the problems that you needed to solve in this process.
- An outline of your personal contribution to your groups project. (Unless you worked alone.)
- An explanation of the design and implementation of your individual additional functionality and any other novel features of your project that you'd like to draw attention to.
- A reflection on your experience with this project. (Largely for my benefit.)

Assessment

Your project part 1 contributes 20% of your unit mark, and will be assessed on:

- correctness - how well your program implements the functionality. (9% group + 3% individual)
- coding - structure, clarity and appropriate use of Open GL. (3% group + 1% individual)
- individual report (4% - but may also affect how other aspects are assessed)
- bonus marks – for additional functionality of your own design (up to 2%)