University of Calgary, CPSC453
Assignment 3
OBJ Model Loading

Release Date: Monday, October 26, 2015
Due Date: Monday, November 16, 2015 at 10:59 am
Weight of this assignment: 10%
Total Marks: 100 Marks (+ 20 Bonus)

Overall Description

For this assignment, you are to write a model loader program for OBJ files. The application is able to select an OBJ file, read it in, and render the mesh in a number of ways on the screen. As with prior assignments, a number of affine transformations may be performed on the model. It also provides the user with options for changing the view and drawing. To facilitate this, the program has a number of user input tools through the mouse, keyboard, and menu.

The assignment will be written in C++ using the Qt GUI toolkit. In this assignment you will be working with OpenGL commands for drawing, using matrices for transformations and supporting basic GUI for interaction.

A number of example models have been provided. The user may choose which model to load and render it on the screen. Even though models come in different sizes it should still be centered and fit within the viewable window on loading. This may require determining the model's dimensions, and calculate the default transformation required to reposition the model or the view.

Standard model and view transformations will be available. These are mostly the same from prior assignments, however the view will be transformed based on the "lookat" setup, and the model will rotate based on trackball rotation. To visualize the view transformations, a patterned square will be drawn and fixed in the scene, similar to our gnomon from Assignment 2.

Many of the provided models are characters adapted from the MD2 files of the old Quake engine. These characters have both a main mesh as well as a weapon or prop mesh. To support these characters, the application must be capable of loading in additional OBJ files. Since the props are in the same coordinate system as the main mesh, they will need to have the same transformations applied as the main mesh. This forms a small example of a hierarchical model with the main mesh the parent of the prop mesh. We will refer to these props as *submodels*, and they inherit the transformations of their parent model.

Three main drawing modes will be supported: wireframe, face and textured. The first two we saw in Assignment 1. To support texturing, the application must be able to select a texture (2D image) and apply the texture coordinates to the model. These texture coordinates are found within the OBJ model, with one texture coordinate (u, v) for each vertex.

To ensure nice rendering of the scene, normals need to be computed for the model. The two modes are face normal - where each vertex on the same face receive the same normal, generating a flat shading style; and vertex normal - where each vertex has its own normal, computed as an average of the adjacent face normals, generating Phong shading style. Normals may be optionally displayed on the mesh. An easy approach is to generate a small line with one point located at the vertex or center of the face, and the other point some small length away in the direction of the normal. Toggling between face and vertex mode should visualize different normals.

Lastly, one should be able to select a currently active model. This selected model (and any of its associated submodels) is affected by the model transformations. Additionally, when loading a texture, it is applied to the selected model. For full marks, 3D picking must be used for selection. For partial marks, an additional menu should be created allowing the application to select between loaded models.

You are are allowed to use the existing Qt matrices, the matrices you created in Assignment 2 or helper matrices in the **glm** library. You may also wish to use the `QFileDialog` and `QImage` class for file and image loading. A perpsective projection has been implemented and should be used for the assignment.

The Interface

The user interface will be written in Qt. You will need to implement the following functionality. The letters in ”()” indicate the keyboard short cut; remember - both upper and lower case should work for the keyboard shortcut.

- A `File` menu with the following menu items:
  - `Open Model (CTRL+O)`: Load a new model. This becomes the selected model
  - `Open Submodel (CTRL+B)`: Load a new model (eg. a weapon), setting the selected model as its parent. This new becomes the selected model.
  - `Load Texture (CTRL+T)`: Load a texture for the selected model
  - `Reset Models (R)`: Reset models to their default positions

  - `Reset View (R)`: Reset camera view
  - `Reset All (A)`: Reset models and camera
  - `Clear (C)`: Clears all models
  - `Quit (Q)`: Exit the program. (This one is already implemented - do not break it!)

- A `Draw` menu, with the following menu items:

  - `Wireframe (W)`: Draw the scene in wireframe mode.
  - `Face (F)`: Draw the scene in face mode.
  - `Textured (T)`: Draw the scene in textured mode.
  - A nested `Normals` menu, with the following menu items:
    - `Face (CTRL+F)`: Draw the scene using face normals
    - `Vertex (CTRL+V)`: Draw the scene using vertex normals
    - `Show Normals (N)`: Toggle between drawing normals indicators

    `Face` and `Vertex` should use radio buttons to indicate which state is selected.
  - `Select Model (S)`: Turns on picking mode to select a currently active model. Picking mode is deactivated when a user selects a model, or presses the escape key `(ESC)` (and clears the selection).

  `Wireframe`, `Face` and `Textured` should use radio buttons to indicate which state is selected.

- Mouse Movements:

  - Operations are initiated by pressing the appropriate mouse button and terminated by releasing that button. The `SHIFT` and `CTRL` modifiers cause the view movements to be used instead.
  - **LMB**: Translate the selected model, or if none, all models, along the X and Y axis.
  - **MMB**: Translate the selected model, or if none, all models, along the Z axis. Only use the horizontal change in mouse position.
  - **RMB**: Rotate the selected model, or if none, all models, using trackball rotation.
  - **LMB** with `SHIFT`: Translate the position of the camera along the X and Y axis.
  - **MMB** with `SHIFT`: Translate the position of the camera along the Z axis. Only use the horizontal change in mouse position.
  - **RMB** with `SHIFT`: Rotate the *up* direction of the camera. Only use the horizontal change in mouse position.
  - **LMB** with `CTRL`: Translate the camera's look at location along the X and Y axis.

  - **MMB** with `CTRL`: Translate the camera's look at location along the Z axis. Only use the horizontal change in mouse position.
  - The maximum and minimum scales should be restricted to a reasonable range.

  You will need to make reasonable decisions about how much to scale or rotate for every pixel's worth of mouse motion. For example, if the mouse isn't moving, there should be no scaling or rotation.

An on-screen display, such as a status bar, should indicate: which drawing mode (Wireframe, Face or Textured), which normal style (Face or Vertex) and which model is selected.

As with previous assignments, this is implemented in C++ with Qt. We have provided some code for you that sets up a window and draws an example triangle. You will need to add the required functionality. As with Assignment 1, when building your data buffers to send to the provided shaders (ie. vertices, per-vertex colours, per-vertex normals and per-vertex texture coordinates), use Vertex Buffer Objects and Vertex Array Objects. This is a more efficient display style.

Donated Code

You have been provided with the following files:

- `main.cpp` - The main point of entry into the application.
- `window.cpp, window.h` - The application window, to which you may add widgets.
- `renderer.cpp, renderer.h` - A widget that will display your rendering. This is where the core part of your code will go.
- `objModel.cpp, objModel.h` - Routines for loading OBJ files. You may wish to add code here.
- `per-fragment-phong.vs.glsl, per-fragment-phong.fs.glsl` - Simple phong shader code.
- `textured-phong.vs.glsl, textured-phong.fs.glsl` - Phong shader code that supports textures.
- `models.zip` - A collection of OBJ models.
- `demos.zip` - A simple example of transformations, including code for trackball rotations which you will likely want to adapt for this assignment.

To compile and run the provided program, execute:

```
qmake -project QT+=widgets
qmake
make
./a3
```

The provided code creates a simple starter UI. As a test, it draws a triangle in the center of the window. You need to modify this code to load in your models and textures, draw using the rendering styles and perform the requested transformations. A suggested to-do list, in an order that will help you is as follows:

- Draw a simple patterned square in the scene (eg. checkerboard).
- Load an OBJ file. You may wish to draw them in wireframe mode before face mode.
- Reposition the model based on its dimensions so it fits in the center of the window.
- Incorporate the model and view transformations, including virtual trackball and the camera's look-at mechanism.
- Calculate the face normals, and then the vertex normals, passing them off to the shader. Then optionally draw the normals on the model.
- Add support for one submodel using the provided weapons. Be able to transform it separately from the main model.
- Add support for textures.
- Add support for 3D picking.
- Add support for multiple models and submodels.

You may wish to create helper methods, helper classes or use data structures such as queues or stacks, depending on your design. C++ standard template library offers stacks, queues and other collection mechanisms which may be used. Describe your design decisions in your `README`.

Bonus (20 Marks)

The core of this assignment is a decent amount of work. However, if you have time, and the creative inclination, there are a lot of ways this can be made much more interesting. You are encouraged to experiment with the code and implement these sorts of changes, as long as you have already met the assignment's basic objects. The maximum additional marks from bonuses is 20. It is at the discretion of the TA to determine coolness factors in awarding bonus points.

- Support animation. (up to 5 marks)
  - A separate folder of OBJs has been provided indicating an animated sequence. Within these sequences are mini-animations (2-10 frames each). Automatically loop through all types of animation (see animation list below) and adjust display speed to match the specified fps (10 marks),

```
// animation list
    typedef enum {
        STAND,
        RUN,
        ATTACK,
        PAIN_A,
        PAIN_B,
        PAIN_C,
        JUMP,
        FLIP,
        SALUTE,
        FALLBACK,
        WAVE,
        POINT,
        CROUCH_STAND,
        CROUCH_WALK,
        CROUCH_ATTACK,
        CROUCH_PAIN,
        CROUCH_DEATH,
        DEATH_FALLBACK,
        DEATH_FALLFORWARD,
        DEATH_FALLBACKSLOW,
        BOOM,
        MAX_ANIMATIONS    } animType_t;
```

- Support animation GUI. (up to 3 marks)
- Perform one of our subdivision techniques on the mesh. (up to 10 marks)
- Highlight the selected model. (up to 3 marks)
- Support scene generation: loading multiple models, saving and loading transformation descriptions, etc. (up to 8 marks)
- Support undo/redo for transformations. (up to 5 marks)
- Provide an alternative rendering style. (up to 5 marks, based on coolness factor)

If you make an amazing modification (an actual feature, not an unintended bug...), document it in your `README` and it will be considered and graded at the discretion of the TA for a maximum of 10 marks.

If you make extensive changes, additionally offer a "compatibility mode" by default. You should support at least the user interface required by the assignment. You can activate your extensions either with a special command line argument or a menu item. Document this in your `README` file.

Non-functional Requirements (20 Marks)

Documentation

1. You must provide a **README** file. A sample one has already been provided.
2. Your README file should contain:
   (a) Your name and UCID.
   (b) Short description of algorithms you implemented to complete the program. This includes how you set up the submodel transformations and calculate the normals.
   (c) A brief description of the data-structures you used to implement the assignment. This includes what matrices you chose to store in the renderer.cpp and their purpose.
3. You must provide at least one screenshot of your assignment demonstrating its capabilities. Additional screenshots are necessary to demonstrate any implemented bonuses. Screenshots should be named 'firstname.lastname_a3_#.png' where # is $1 - n$ for n screenshot images.

Source Code

1. All your source code must be written in **C/C++** and properly commented. All graphics rendering must be done using **OpenGL**. All event handling and windowing must be performed via **Qt**. Your source code must compile on the lab machines in MS 239 without any special modifications. Your source code must be clear and well commented.
2. You will lose marks for inefficient and slow implementations.
3. You may reuse source code:
   (a) which has been provided by the instructor for use in the course,
   (b) which has been written by you which implements basic data structures, such as linked lists or arrays,
   (c) which you have received permission from the instructor or one of the TAs of CPSC 453 prior to handing-in your assignment,

4. Any instances of code reuse by you for this assignment must be explicitly mentioned within the README file. Failure to do so will result in a zero in the assignment. Please read the University of Calgary regulations regarding plagiarism `http://www.ucalgary.ca/honesty/plagiarism`.

Functional Requirements (80 Marks Total)

Modelling & Rendering (40 Marks)

1. Simple patterned square is drawn. It is affected by view, but not model transformations. (3)
2. UI for loading models. (2)
3. Draw model and be able to toggle between wireframe (2), face (2) and textured (10). This includes 2D texture loading for the selected model.
4. Compute and optionally display indicators for face normals (5) and vertex normals (7).
5. Render using face normals - flat shading (2) and vertex normals - smooth shading (2).
6. Support weapon submodel loading. (5)

Transforming (25 Marks)

1. On model loading, model fits within window and is centered. (5)
2. Model transformations: translation (3) and trackball rotation (5).
3. Submodel transformations: hierarchical - moves independently or with parent model. (3)
4. View transformations: translate camera position (3), translate camera lookat location (3), rotate through camera's up-direction (3)

UI Interaction (15 Marks)

The user should be able to:

1. Access pull down menus for controls as specified. (3)
2. Display on-screen feedback for current mode and selected object as specified. (2)
3. Able to select models (5) - full marks for 3D picking; maximum 2 for menu listing.
4. Mouse controls for model and view as specified. (5)

Lost Marks

Possible areas for losing marks:

- Model is non-uniformly stretched on loading.
- Model is cropped on loading. Model is clipped or cropped on initial rotation.
- On load, model does not rotate about its center, nor the center of the window.
- Specific model values are hardcoded into the program.
- Application redraw slows down over time.
- Draw updating is not live with mouse movements.
- Transformations are buggy, jerky, uneven, too fast or too slow.
- No screenshot. No README.

Demo

You are required to give an approximately 5 minute live demo to your TA. Failure to show up at the presentation will result in a zero in the assignment. You will need to schedule your demo with your TA - they will have details about how your tutorial section demos will run.