University of Calgary, CPSC453
Assignment 1
Introduction to OpenGL

Release Date: Monday, September 14, 2015
Due Date: Monday, October 5, 2015 at 10:59 am
Weight of this assignment: 10%
Total Marks: 100 Marks (+ 20 Bonus)

General Specification of the Assignment

This assignment will practise working with graphics APIs (OpenGL, Qt, GLSL) to build a graphics application. The assignment will be written in C++ using the Qt GUI toolkit. In this assignment you will be working with simple OpenGL commands for drawing, using matrices for transformations and supporting basic GUI for interaction.

The application is a game, where the user controls falling block formations to create solid lines within the well of game play. When a line is completed it is removed from the game. When blocks reach the top of the well, the game is over.

This assignment is adapted, with permission, from Assignment 1 of CS488 - Introduction to Computer Graphics at the University of Waterloo.

Game Play

The game takes place in a U-shaped well of unit cubes enclosing a grid of width 10 and height 20, in which the block formations can fall. The blocks occupy discrete positions in the grid (ie. they don't fall smoothly, but jump from position to position).

Block formations start within a four unit tall region at the top of the well. Every time a predetermined interval elapses, the current formation falls one unit. The value of 500ms is a good novice interval; 100ms is more challenging. At any time, the current piece can be moved to the left or right, rotated clockwise or counter-clockwise, and dropped the rest of the way down the well. When the piece can fall no further, it stops and any rows in the well that are completely filled are

removed from the game. The game ends when a block cannot clear the starting region.

You should have at least three different speeds at which the block formations fall.

Interface

The user interface will be written in Qt, a cross-platform application and UI framework used in industry by C++ developers. You will need to implement the follwing 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:
  - `New Game (N)`: Start a new game.
  - `Reset (R)`: Reset the view of the game.
  - `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 game in wireframe mode.
  - `Face (F)`: Fill in the faces of the game. Each different piece shape should have its own uniform colour.
  - `Multicoloured (M)`: Similar to `Face` mode, but each cube has six faces of different colours (ie. no two faces should have the same colour). Funky colours and combinations are encouraged.

  The `Draw` menu should use radio buttons to indicate which state is selected.
- A `Game` menu, with the following menu items:
  - `Pause (P)`: Pause the game.
  - `Speed Up (Page Up)`: Increase the speed of the game play.
  - `Slow Down (Page Down)`: Decrease the speed of the game play.
  - `Auto-Increase (A)`: Slowly and automatically increase the speed of the game play.

  A `QTimer` may be helpful with setting up game play timing.
- Mouse Movements:
  - Mouse operations are initiated by pressing the appropriate mouse button and terminated by releasing that button. Only motion in the horizontal direction should be used.

  - The left mouse button should rotate the game around the X-axis.
  - The middle mouse button should rotate the game around the Y-axis.
  - The right mouse button should rotate the game around the Z-axis.
  - When the `shift` key is pressed, use of any mouse button will uniformly scale the game - both the board and the pieces. When the mouse moves to the left, the game should become smaller. When the mouse moves to the right, the game should become larger. 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.

  You are also required to implement a feature known as "persistance" or "gravity." If, while rotating, the mouse is moving at the time that the button is released, the rotation should continue on its own. This decision should be made at the time of the release; after that it should persist independently of mouse movement, until the next button press.

- Keyboard Input:
  - The left arrow key should move the currently falling piece one space to the left.
  - The right arrow key should move the currently falling piece one space to the right.
  - The up arrow key should move the currently falling piece counter-clockwise.
  - The down arrow key should move the currently falling piece clockwise.
  - The space bar should 'drop' the piece, sending it as far down into the well as it will go.

Much of these actions are already implemented within the provided game code (`game.h`, `game.cpp`), however you will need to construct the appropriate GUI and OpenGL for handling and displaying this functionality.

Qt, OpenGL and C++

Qt provides a great deal of GUI management, and has recently expanded to offer more support for OpenGL-style commands. For the assignment, you are to use the OpenGL commands directly where possible. If needed, using Qt to assist with shader loading is permissible. Use of the `QMatrix4x4` Qt class, and other such classes is permitted.

When building your data buffers to send to the provided shader (ie. vertices, per-vertex colours and per-vertex normals), use `Vertex Buffer Objects` and `Vertex Array Objects`. This is a more efficient display style.

For more information, see the Qt documentation: `http://doc.qt.io/qt-5/`.

Walk Through

The provided source code is a good starting ground for building your application. This includes:

- `main.cpp` - The main entry point for the program.
- `renderer.cpp, renderer.h` - The OpenGL widget, where all of the OpenGL-related code should go.
- `window.cpp, window.h` - The application window. Most of the GUI-related code (menus, etc.) should go here.
- `game.cpp, game.h` - A game engine that implements the core of the falling blocks game. Should not need to be modified.
- `per-fragment-phong.vs.glsl` - Vertex shader used to determine the position of every vertex. Should not need to be modified.
- `per-fragment-phong.fs.glsl` - Fragment shader used to determine the colour of each pixel. Should not need to be modified.

To compile and run the start program, execute:

```
qmake -project QT+=widgets
qmake
make
./a1
```

The provided code creates a user interface with an OpenGL window. As a test, it draws triangles where the corners of your game (not including the well) should appear. The camera is setup so that the triangles appear centered and correctly sized. You need to modify this code to render the current state of the game and respond to user interface events. A suggested to-do list, in an order that will help you is as follows:

- Write a function to draw a unit cube using OpenGL.
  You can write a single cube to a Vertex Buffer Object (VBO). You would then create a Matrix4x4 model matrix for each cube and use the matrix functions to translate, rotate and scale this model. You will need to have a buffer of per-vertex unit-length normals to pass to the shader as well. These may be hard-coded for this assignment alone.
- In your render function, draw a U-shaped border for the well, out of the cubes.
- Implement face rendering and wireframe rendering.

- Implement rotation and scaling. You should be able to see the effect on the well.
- Add code to draw the current contents of the game. Each piece type should be drawn a different colour - the colour is up to you.
  Note: The colour is sent off to the shader similar to the vertices.
- Hook up a simple timer (using the `QTimer` class) that calls the game's `tick` method, and re-renders. You should be able to see pieces falling.
- Implement the rest of the controls for game play and the remaining user interface.

Bonus (20 Marks)

There are lots of ways this simple application could be modified to enhance playability and attractiveness. You are encouraged to experiment with the code to 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.

- A scoring mechanism. (5 marks)
- Head-to-head networked play. (10 marks)
- Modified cubes for pieces. The blocks look much better if individual cubes have their edges slightly beveled. (5 marks)
- Animations for certain events. For example, having the board spin around when you lose. (5 marks)
- Additional light and lighting - requires shader modification. (10 marks)

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.

If you make extensive changes to the game, 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.

   (c) A brief description of the data-structures you used to implement the assignment.

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)

Display (25 Marks)

1. Support wireframe mode. (10)

2. Support face colour mode. (10)

3. Support multicoloured face mode. (5)

Game Play (25 Marks)

The user should be able to:

1. Increase the speed of the game play (5)
2. Decrease the speed of the game play (5)
3. Work with the user interface as specified (menus, mouse interactions, etc). (10)
4. Play with the game as described under "Game Play." (5)

Transformations (30 Marks)

1. The game can be rotated. (10)
2. The game can be scaled. (10)
3. Persistence works for rotation. (10)

Lost Marks

Possible areas for losing marks:

- Polygon vertices should be specified in counter-clockwise order (otherwise you may not see the face if you have backface culling on).
- Persistent rotation shouldn't start because of mouse movements for scaling.
- Persistent rotation shouldn't start, get faster, nor slow down on its own.
- Normals should all point outside the cubes.
- Radio button should be initialized correctly.
- Persistence should only happen when the user releases the mouse button while the mouse is moving.
- Not allowing the user to use two buttons at the same time.
- Allowing negative scaling.
- Jerky persistence.
- Rotation and scaling should be continous, which means should start from where it was left last time and not resetting.
- Uncontrollable rotation, scaling etc. e.g. Game rotates constantly, rotates a bit and resets right away, etc.

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.