Peter Lesslie
Project 2

I have been playing tournament chess since kindergarten, and I have always wanted to make a chess engine so designing a chess board seemed like the perfect project. Designing each of the pieces is sophisticated enough to show off the use of interesting geometries, while the board will eventually be textured. The one aspect that a chess board does not really show well is different types of lighting since they do not matter for the board; I want, however, in the later projects to put a candle next to the chess board to show a positional light source.

This scene is taken from Réti's famous endgame study. It demonstrates an odd phenomenon in a chess endgame where the white king is able to stop the black pawn despite the black pawn being closer to the queening square than the white king. This is achieved by simultaneously threatening to queen the white pawn while also making progress towards stopping the black pawn. The correct moves in this position are: 1. Kg7 h4 2. Kf6 Kb6 3. Ke5 Kxc6 4. Kf4 when the white king can now capture the black pawn and secure the draw. To learn more about this study look at: http://en.wikipedia.org/wiki/Reti_endgame_study.

I generated the board by first creating a square class that takes a position for the lower left corner, a color type, a width, and a thickness. The board class then has the responsibility of creating an 8x8 checkered board with a 1 square border around it. When the render() method of the board class is called it iterates through the std::vector of Squares and calls each Square class' render() method.

Each piece for now is just created with flat sides. I want to in the next project put a sphere on the top of the pawn. I have created a basic sphere class, which is in the directory, but the class only creates a unit sphere around the origin. To use this class I need to implement a matrix to move the sphere to the correct position and adjust the radius. To facilitate the drawing of the pieces the GeneralMV class has quad() and tri() member functions that take vertices and generate the triangles that OpenGL can actually draw. Using these functions, the drawing of the pieces was just a matter of breaking each piece down into rectangles and triangles then passing the vertices into the respective equations. I borrowed this idea from the programs in the Appendix of the E. Angel textbook.

I have four concrete subclasses of the ModelView class: Square, Board, Pawn, and King. The Pawn and King classes are the two more sophisticated ModelView classes since they include multiple rectangles and each has several hundred vertices. Both sides, black and white, each have one king and one pawn, making two instances of both classes. The Square and Board subclasses are simple, but they were need for the scene. The lighting model is simple, and the light source is coming from the viewer —the light position is (0,0,-1) –but this seemed the most appropriate for the scene, and changing it is simply a matter of changing the lightModel variable in the GeneralMV class.

I had the most difficulty getting my matrices correct. I originally did not check the MatrixIF class' methods so I assumed that the arrays being generated would be row-major so I was passing GL_TRUE to glUniformMatrix() to transpose them. I did not realize my error until I was fairly far into the program because my first models were already in the range -1,1 and centered about the origin so the matrices were not doing anything anyways. I also wanted to specify my model's in model space and use a matrix to move the model to world coordinates, but the program structure was not really set up to do this. And, of course, my biggest problem was simply starting on the program too late since I have had to give up all of my weekends in the past few weeks to go to mock trial tournaments.

The major unique feature in my project is that I implemented the ability for users to move the scene using the keyboard. I did this mostly to help myself look at the model from different viewpoints more easily. The other feature, that you suggested in the writeup, is a general modelview class that maintains the pointers to the matrices, kd, and lightModel variables in the GLSL shaders.

For the next project I want to make the squares look better from any angle, this will mean adding more points on each side and possibly moving some of the calculations to the fragment shader. I would like to move all the models to their own model space centered around the origin to facilitate model creation. I want the objects to be specified in .obj files so that I can load different chess sets, and, of course, I need to design the rest of the pieces. For the pawn model I want to put a sphere on the top.