Graphics Programming I – Assignment #1 (2D ellipses)

Part 1: Due on 13-April-2010

In the first part of the assignment you will implement an SDL-based framework for running OpenGL code. This framework will consist of four parts:

- Initialization routine and support code: this routine will contain all of the code to initialize SDL. This includes initializing a timer with a callback that will generate events. The callback function will use a global flag to enable or disable generation of events. Initially this flag will be disabled.
- Window creation routine: this routine creates an SDL drawing surface with the attributes required by your application. For this project, the requirements are at least 5 bits of red, green, and blue for the color buffer. This function takes the height and width of the desired surface as parameters.
- Event loop: this is the main loop of the program. This routine will enable generation of timer events (by setting the global flag) and process events as they are received. At the very least, the event loop must respond to the `SDL_QUIT` event and to a keyboard press of "q" by terminating the event loop.

  After processing all user input, if a timer event was received, the event loop calls the drawing routine.
- Drawing routine: the initial version of the drawing routine will be very simple. Each time the drawing routine is called, it will set the clear color to a new value by calling `glClearColor`. The window will then be cleared by calling `glClear` with `GL_COLOR_BUFFER_BIT` set in the parameter. So that the new window color will be displayed, `SDL_GL_SwapBuffers` will be called.

  Setting the red, green, and blue components of the clear color to a value based on the sine of the elapsed time since program start should provide reasonable colors.

The "Using SDL with OpenGL" tutorial at `http://people.freedesktop.org/~idr/OpenGL_tutorials/` will be very helpful for this stage.

Part 2: Due on 20-April-2010

In this part of the assignment, you will implement a simple 2D graphics effect using GLSL shaders. Using the fragment shader, draw a grid of ellipses. The axes of each ellipse will oscillate between 10 pixels and 100 pixels.

It is recommended that this assignment be implemented in three phases.

- Implement set of utility routines to compile a shader, generate a program from a pair of shaders, and create an empty buffer object. Use the `gluLoadTextFile` and `gluUnloadTextFile` routines in the supplied GLU3 library to load a simple vertex and fragment shader. Use these shaders to render a simple triangle strip in the shape of a rectangle.

  The "GLSL Hello World" tutorial at `http://people.freedesktop.org/~idr/OpenGL_tutorials/` will be very helpful for this stage.
- In the fragment shader, divide screen space into a grid of 100 pixel by 100 pixel cells. This is done using the `mod` function on `gl_FragCoord`. The result of `mod(gl_FragCoord, 100.0)` is the pixel location within a cell. This value will be on the range [0, 99]. Convert this to a Cartesian coordinate on the range [-50, 49]. If the absolute value of the X and Y coordinate is less than 25, set `gl_FragColor` to a color of your choosing. If it is greater than or equal to 25, set it to a different color of your choosing. This will result in a grid of 50 pixel by 50 pixel squares being drawn. The squares will have 50 pixels of "blank" space between them in each direction.

  The "Fragment Shader Introduction" tutorial at `http://people.freedesktop.org/~idr/OpenGL_tutorials/` will be very helpful for this stage.
- Use the equation of an ellipse to draw ellipses instead of squares. Here a and b are the lengths of the X and Y axes of the ellipse.

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \tag{1}$$

Since division is somewhat expensive, think about ways to implement this equation efficiently.

- Instead of using constant values for a and b in your shader, specify these values as a single `vec2` uniform. Each time the drawing routine is called, update the value based on the elapsed time since program start. This will be very similar to the changing clear color in the first part of the assignment. It is advisable to use sine to control one axis and cosine to control the other.

  The "Using Uniforms" tutorial at `http://people.freedesktop.org/~idr/OpenGL_tutorials/` will be very helpful for this stage.

Criteria	Excellent	Good	Satisfactory	Unacceptable
Completion	Program correctly implements all required elements in a manner that is readily apparent when the program is executed. User interface is complete and responsive to input. Program documents user interface functionality.	Program implements all required elements, but some elements may not function correctly. User interface is complete and responsive to input.	Program implements most required elements. Some of the implemented elements may not function correctly. User interface is complete and responsive to input.	Many required elements are missing. User interface is incomplete or is not responsive to input.
Correctness	Program executes without errors. Program handles all special cases. Program contains error checking code.	Program executes without errors. Program handles most special cases.	Program executes without errors. Program handles some special cases.	Program does not execute due to errors. Little or no error checking code included.
Efficiency	Program uses solution that is easy to understand and maintain. Programmer has analysed many alternate solutions and has chosen the most efficient. Programmer has included the reasons for the solution chosen.	Program uses an efficient and easy to follow solution (i.e., no confusing tricks). Programmer has considered alternate solution and has chosen the most efficient.	Program uses a logical solution that is easy to follow, but it is not the most efficient. Programmer has considered alternate solutions.	Program uses a difficult and inefficient solution. Programmer has not considered alternate solutions.
Presentation & Organization	Program code is formatted in a consistent manner. Variables, functions, and data structures are named in a logical, consistent manner. Use of white space improves code readability.	Program code is formatted in mostly consistent with occasional inconsistencies. Variables, functions, and data structures are named in a logical, mostly consistent manner. Use of white space neither helps or hurts code reability.	Program code is formatted with multiple styles. Variables, functions, and data structures are named in a logical but inconsistent manner. Use of white space neither helps or hurts code reability.	Program code is formatted in an inconsistent manner. Variables, functions, and data structures are poorly named. Use of white space hurts code reability.
Documentation	Code clearly and effectively documented including descriptions of all global variables and all non-obvious local variables. The specific purpose of each data type is noted. The specific purpose of each function is noted, as are the input requirements and output results.	Code documented including descriptions of most global variables and most non-obvious local variables. The specific purpose of each data type is noted. The specific purpose of each function is noted, as are the input requirements and output results.	Code documented including descriptions of the most important global variables and the most important local variables. The specific purpose of each data type is noted. The specific purpose of each function is noted.	No useful documentation exists.

This rubric is based loosely on the "Rubric for the Assessment of Computer Programming" used by Queens University (http://educ.queensu.ca/ compsci/assessment/Bauman.html).