CG Programming III – Assignment #1 (shadow textures)

In this assignment you will be required to implement shadow textures. To show shadow textures in action, render a scene with *at least* two objects and a single light source. Select some set of objects to be shadow casters and some (disjoint) set to be shadow receivers.

This assignment will be graded in two parts. Each part has a separate due date.

Part 1 due on 18-April-2012

- Add code to the initialization path to create a framebuffer object with a single color attachment. This color attachment should be a texture (as opposed to being a renderbuffer). The framebuffer should be one quarter the screen size (half width and half height).
- At the end of your drawing routine, just before `SDL_GL_SwapBuffers`, use `glBlitFramebuffer` to copy the framebuffer to the lower left corner of the screen. At this point, that will just be a black rectangle.
- Refactor the drawing code to a separate routine.
- Modify the main drawing routine to call the refactored drawing routine twice. The first time should draw to the FBO, and the second time should draw to the screen. The FBO copied to the screen will no longer be a black rectangle, and it will also be different than the image drawn to the screen due to the lack of a depth buffer.
- Before drawing to the FBO, set the clear color to white. Set a different shader. This shader will always output black from the fragment shader.
- Before drawing to the FBO, set a different view-projection matrix. This matrix should view the scene from the point-of-view of the light instead of the camera.

At this point, the body of your main drawing routine will look something like:

```
// Render the shadow texture
glBindFramebuffer(GL_FRAMEBUFFER, shadowFBO);
glViewport(0, 0, shadowFBO_width, shadowFBO_height);
glClearColor(1., 1., 1., 1.);
glUseProgram(shadow_render_program);
draw_scene(light_view_matrix, shadow_casters);

// Render the scene
glBindFramebuffer(GL_FRAMEBUFFER, 0);
glViewport(0, 0, window_width, window_height);
glClearColor(clearR, clearG, clearB, clearA);
glUseProgram(render_program);
draw_scene(camera_view_matrix, shadow_casters);
draw_scene(camera_view_matrix, shadow_receivers);

// Copy the shadow texture image on top of the lower left
// corner of the scene.
glBindFramebuffer(GL_READ_FRAMEBUFFER, shadowFBO);
glBindFramebuffer(GL_DRAW_FRAMEBUFFER, 0);
glBlitFramebuffer(0, 0, shadowFBO_width, shadowFBO_height,
                  0, 0, shadowFBO_width, shadowFBO_height,
                  GL_COLOR_BUFFER_BIT,
                  GL_NEAREST);

SDL_GL_SwapBuffers();
```

The program should render the scene with the shadow texture superimposed over the lower-left corner of the window.

Part 2 due on 25-April-2012

The second part uses the texture generated in the first part as the shadow texture.

- Create a dummy, 1x1 texture that contains a single white texel.
- In the drawing pass that renders the actual (shadowed) scene, use the dummy texture when drawing the shadow casters. Use the texture from `shadowFBO` when drawing the shadow receivers.
- Modify the fragment shader to apply a texture using projective texturing. Since the texture is white in the non-shadow regions and black in the shadow regions, simply modulate the shadow texture color with the final computed color. Be sure to apply the near-plane test mentioned in the lecture notes to prevent anti-shadows!

It is strongly recommended, though not required, that something be drawn at the position of the light. Using a small sphere or a single point primivite should work.

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).