CG Programming III – Assignment #3 (SSAO)
Due on 06/12/2013 at the final

In this assignment you will be required to *partially* implement screen-space ambient occlusion. There are two major portions of the SSAO pipeline, and you will implement one of them: the geometry-aware filter.

Augment your current demo by adding many more torii to the scene. Two good scenarios will show the effect of the filter:

- Stack of tires: Add several torii stacked on top of each other like a stack of tires. The torii should not intersect, and there should not be a gap between them.
- Donut X: Add some torri that intersect each other in an X formation.

Perform some preliminary refactoring of your demo.

- Generate a *new* FBO with color and depth. This FBO should exactly match the dimensions of the window. Both the color and depth attachments to this FBO should be textures (*not* renderbuffers).
- Modify the main scene rendering pass to render to this FBO instead of the window.
- Configure the main scene FBO as the `GL_READ_FRAMEBUFFER` and the window was the `GL_DRAW_FRAMEBUFFER`. Use `glBlitFramebuffer` to copy the the scene to the window. The demo should produce the same output as before the new FBO was added.
- Create a new shader program. This program will have a simple "pass through" vertex shader that copies the input vertex to `gl_Position` and applies a scale-and-bais operation to the vertex before writing it to a `vec2` output. The fragment shader will output the color read from a texture using the vertex shader `vec2` output as the texture coordinate. I will refer to this as the "filter shader".
- Configure the window as both the `GL_READ_FRAMEBUFFER` and `GL_DRAW_FRAMEBUFFER`, and bind the texture attached to the FBO to texture unit 0. Using the shader program from the previous step, draw two triangles to cover the square from $(-1, -1, 0)$ to $(1, 1, 0)$. This will replace the previously added `glBlitFramebuffer` call. The demo should still produce the same output.

Once you get to this point, make a backup copy of your project.

Before implementing the bilateral filter, you will implement a simple Gaussian filter.

- Generate a 1D table of Gaussian filter weights. The filter diameter should be 5. Let $\sigma = 5/6$ and $G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{x^2}{2\sigma^2}}$ for $x \in [-2, 2]$. See Wikipedia[1] for more details. Add this table as a constant array of floats to your filter fragment shader. Call this table `w`.
- Modify the main function in the filter fragment shader to use `textureOffset` to read a row of five pixels around the input texture coordinate. Each pixel value should be multiplied with the corresponding entry in the `w` table (e.g., the pixel at offset -2 goes with entry 0, offset -1 with entry 1, etc.). Sum the results and output the resulting color. It should look a littler blurrier.
- Encapsulate one iteration of the code from the previous step in a macro called `S`. `S` should take two integer parameters: the x offset and the y offset. Replace the previous code with five invocations of the new macro. `y_offset` will always be 0 in this step. Use `x_offset + 2` as the table index.
- Modify `S` to also use y offset to determine the filter weight. Multiply the value from `w` for the x offset with a value from `w` for the y offset. The result should be similar to before, but darker.
- Instead of invoking `S` five times, invoke it 25 times: use a 5×5 grid of pixels. This result should be much blurrier than before. You've now implemented a simple Gaussian blur.

[1] http://en.wikipedia.org/wiki/Gaussian_blur

Now you will begin implementing the bilateral filter. This special filter blurs data while respecting geometric discontinuities.

- Modify the filter fragment shader to have a second texture. Bind the depth buffer from the rendering FBO to this texture[2].
- Recall the definition of the bilater filter:

$$A_p = \frac{1}{k(p)} \sum_{p' \in \Omega} g_d(p' - p) g_r(z_p - z_{p'}) A_{p'}$$

  where

$$k(p) = \sum_{p' \in \Omega} g_d(p' - p) g_r(z_p - z_{p'})$$

  If g_r always returns 1 and $g_d(p' - p)$ is the Gaussian weight function, the bilateral filter is exactly the Gaussian filter that you just implemented.

- Add a function to the filter fragment shader, `float Gr(float a, float b)`. The function should return a value that is inversely proportional to the difference of the input values. Since $|z_p - z_{p'}| \in [0, 1]$, something like `exp(1 - abs(a - b)))` or `pow(1 - abs(a - b), 2.)` should work nicely (try plotting these on the range $[0, 1]$ in Excel or gnuplot).
- Modify `S` to use `Gr` as the g_r from the bilateral filter equation. This means you will also need to add a new variable, `k` that acculuates the `Gr` values. This will be used as $k(p)$ in the bilateral filter equation.

[2] Use `sampler2D`, not `sampler2DShadow`. Also, do not set `GL_TEXTURE_COMPARE_MODE` to `GL_COMPARE_REF_TO_TEXTURE`.

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).