Advanced Computer Graphics Assignment 4: Anti-aliasing

Due date: check nestor

1 Assignment and grading

The complete assignment will be graded on a 0-100 scale, and the score will be broken down as follows

- 30 points: Questions
- 30 points: Extending the vertex shader
- 40 points: Extending the fragment shader

It is important that you understand what you are doing, so we want you to answer some questions that are interleaved in the assignments. Answers should not be too long, just convey the basic idea. The answers to the questions should be saved to a file called `questions4.txt` (Note: `questions4.txt` is supplied, and has the questions written out). You will want to answer them as you complete the assignment, since many of them require experimentation.

What to hand in: Please hand in C and Cg sources for your code for each of the assignments. Also, after you filled in the answers to the questions in file `questions4.txt`, archive everything and submit as per the instructions on Nestor.

2 Reading material

- The course slides (can be found on nestor)
- Cg User's Manual and Reference Manual
- OpenGL reference pages `http://www.opengl.org/sdk/docs/man/`, for documentation on GL extensions (such as GL_EXT_framebuffer_object) see the extension registry `http://www.opengl.org/registry/`.
- Optional: the book Point-Based Graphics, by Markus Gross

3 Getting started

It is advised to start from the code of the previous assignment, as our approximation of anti-aliasing is best combined with Deferred Shading, due to the potentially larger number of fragments generated per splat.

To demonstrate anti-aliasing, a checkerboard pattern on a plane is best. Such a data set has been provided for this assignment (`checkerboard.pts`). `pts` files are in a very simple binary format that can be directly used for rendering. It contains the number of points (`int numpoints`), followed by the actual surfel data. Code for reading these files was provided for the previous assignment.

4 Anti-aliasing

The goal of the assignment will be to extend the previous point based renderer with an approximation to the EWA filter that does anti-aliasing in screen space. In this method, which was explained in the lecture, a second kernel of a certain fixed radius in screen space is computed, and the maximum of both the splat and screen-space kernel value is used as blending weight. In true EWA, the convolution of both kernels would be used, but this approximation gives results that are almost as good with a fraction of the amount of computations (and thus rendering time).

An example of the results that can be achieved is shown in Figure 1.

4.1 Extending the vertex shader

Adapt the vertex shader so that the point size that is computed is correct for the anti-aliased kernel. This means that the value that is produced should be big enough to envelop the splat as well as the diameter of the screen-space filter. Pass the diameter (or radius) of the screen space filter as a uniform, so that it can be changed at run-time.

Question 1: In which case, magnification or minification, will the screen-space filter make the splat bigger?

Question 2: Will it ever make the splat smaller?

Also, add code to compute the projected splat center in screen coordinates, and to pass these to the fragment shader. In the previous lab session we did not care about the screen position of a fragment within a splat, but now we need this to compute the screen-space kernel.

Question 3: A vertex shader always outputs the homogeneous vertex coordinates in clip-space in the `POSITION` binding semantic. Which two steps are needed to get to the window coordinates from there?

4.2 Extending the fragment shader

In the splat fragment shader, change the kernel computation so that an anti-aliased kernel is approximated by using the minimum of the (normalized) squared distances for the planar splat filter and the screen-space splat filter. Make sure that the condition for `discard` remains correct.

Question 4: Why is taking the minimum of the two (normalized) squared distances before computing the kernel weight equivalent to taking the maximum of the resulting kernel function values?

Question 5: For very large sizes of the screen-space filter, around 22, black areas start to appear around splats that are most perpendicular to the viewing plane. The same happens if we try to anti-alias, for example, the lion. Why?

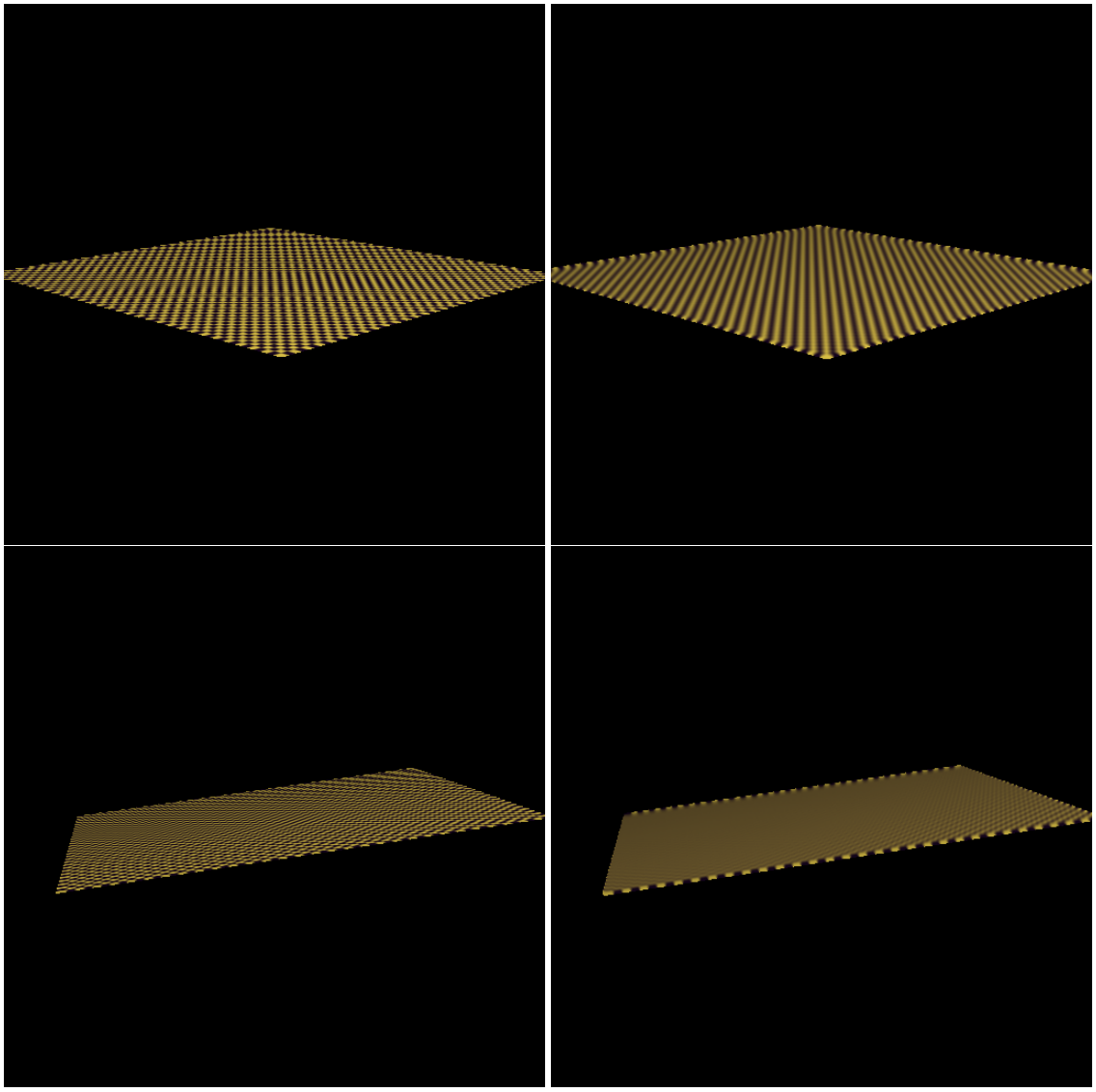

Figure 1: *The checkerboard viewed from two angles; left: without anti-aliasing, right: anti-aliased with a screen-space kernel (radius 3).*