Warmup

Assigned: 1/27/2016
Due date: 1/31/2016 11:59pm

Introduction

This is a warm-up project to get you familiar with writing fragment shaders in ShaderToy. You will draw a sphere on a striped plane. It should look something like this:

![C-grade implementation](image)

Refresher: OpenGL and Fragment Shaders

In the OpenGL graphics pipeline, shaders are programs that can be assigned to geometric primitives in order to compute face colors, transform vertices, or even generate additional geometry on the fly. Shaders are very powerful because they leverage GPU parallelism and reduce the amount of data that the user needs to copy over to the GPU.

The table below lists common use cases of shaders.

<table>
<thead>
<tr>
<th>Shader Type</th>
<th>Function</th>
<th>Example Usage</th>
<th>Naive Approach</th>
<th>With Shader</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertex</td>
<td>performs per-vertex transformations</td>
<td>create a cloud of points and animate it like a pulsating jellyfish.</td>
<td>re-compute the point positions using the CPU and re-copy the new positions to the GPU for every draw call</td>
<td>Send an initial point cloud to the GPU only once, and use a vertex shader to move the points around on the GPU.</td>
</tr>
<tr>
<td>Geometry</td>
<td>Generates new geometry</td>
<td>Draw a jellyfish</td>
<td>Create pixel buffer of point locations, send over to GPU</td>
<td>Send an empty array to the GPU, use geometry shader to generate the entire jellyfish on the GPU</td>
</tr>
</tbody>
</table>
Shaders are written in the GLSL language, which superficially resembles C.

**Getting Started**

Go to [https://shadertoy.com](https://shadertoy.com) and register for an account. ShaderToy is a fragment shader editor written completely in the web browser.

In ShaderToy, all you get to work with is a fragment shader, attached to a single GL_QUAD drawn across the entire window. Despite the simplicity of this setup, you can do a lot of procedural graphics with “just” a fragment shader. ShaderToy maintains a collection of community-contributed shaders [1], including procedurally generated landscapes, path tracers, and even an implementation of “Flappy Bird”.

All shaders you create should be saved as “private”; public shaders will be considered a violation of collaboration policy.

**Requirements**

**Basic**
- Draw a sphere on a striped plane (does not have to look exactly like the reference image).
- The code you write will be similar to CS123’s “Intersect” and “Ray” assignment.
  - Trace a ray into the scene
  - Find the closest object you intersected
  - Calculate the surface normal of the intersected object
  - Compute the diffuse lighting on the object and output that value the final color of the fragment.

**Bells and Whistles:**
- 2 Add more spheres, make the ground more complex
- 4 Anti-alias the spheres so that their edges are no longer jaggy
- 2 Make the sphere shiny (and make anti-aliasing still work!)
- 2 Camera motion. More points awarded if you do something fancy
- 4 Add a shadow to the sphere(s)
- 5 Make the shadow nicely blurred (i.e., consider the illumination done by an area light)
- 3 Make the sphere a beach-ball (i.e., striped colored wedges)
- 7 Make the ground beneath the beach-ball “reflect” the color of the ball

**Grading**
For a grade of C+: A functioning ShaderToy renderer that reproduces the ball-on-striped-plane example. If camera motion is implemented, does not crash or fail in any way.

For a B: The reference image is kind of aliased and boring to look at. At least 8 points of bells and whistles added. They don’t have to be the ones in the list of suggestions; those are just to get you started. If your interest is animation, perhaps you can make the ball move or deform in some interesting way (simply moving left and right with constant-speed motion won’t get you much credit).

For an A: Do something that impresses us. We will be demoing them in class on Monday after the due date.

Support Code & Implementation Hints

- You can take a look at https://www.shadertoy.com/view/IX3D8, in which we’ve defined the sphere parameters and written the main() function for you. The main function basically sets up a 3D camera. Your task is to implement the render(in vec3 ro, in vec rd) function and write any auxiliary functions you might need (raymarch intersection, getting surface normals for lighting, getting shading of objects, to name a few).
  - We encourage you to look at other shaders on shadertoy.com for inspiration, but do not blindly copy other people’s code. During class demos (or during grading), we may ask you to explain individual lines of code in detail. Those who have plagiarized may have trouble with this.
  - When you look at a shadertoy example and get inspired, be sure you understand the assumptions being made by that code (e.g., “this works only if all lights in the scene are point lights”), and don’t mistakenly use ideas based on assumptions counterfactual to those in your scene.
- In CS123, you probably wrote intersect by looping through the scene graph to find the shortest scalar multiple of the eye ray that analytically intersects an object. You can do that for this assignment, but one alternative is to implement ray-marching, which makes use of signed distance functions. The basic idea is to march a point along the eye ray until it comes within epsilon of an object in the scene, then return vec2(total distance marched, ID of the object).
- http://iquilezles.org/www/index.htm contains a bunch of excellent articles with tricks like implementing signed distance functions and anti-aliasing. If you decide to borrow techniques from here, be sure to cite them and make sure you understand them fully (we may quiz you).
- Our C-grade implementation is about 100 lines of code and took a few hours to write. You can probably do this in 50.
- Here is a shader that provides a hint on how to use blurring to anti-alias 2D circles. This technique can be extended to work with 3D spheres.
  - https://www.shadertoy.com/view/ltfGD8
- The best way to debug GLSL shaders and inspect variable values is to simply draw them to the screen.
- Save your work often ("Save" button in ShaderToy). Sometimes GPUs can cause machines to crash.

**Handing In**

Your handin folder should contain a README file containing a URL to your private shader and list a summary of the things you implemented (including extra credit). Please also hand-in a copy of your shader source file (i.e. ball.txt) so we can make sure you don’t make modifications to your code after the hand-in deadline.

In a terminal, run `cs224_handin ball`

**Past Examples**

<table>
<thead>
<tr>
<th>Antialiasing, stripes, soft shadows, specular reflections on plane and sphere</th>
<th>Ball stripes, Camera Motion, Soft shadowing, Specular highlight, Reflections, animations, texture mapping, fog, bump mapping, AO</th>
</tr>
</thead>
<tbody>
<tr>
<td>extra spheres - zig-zag floor - sine-wave curtain - animations - specular highlighting on spheres - reflections on spheres - sphere shadows - adaptive supersampling (antialiasing)</td>
<td>Glass refraction/reflection, reflective ground, animation, user interaction, image-based lighting</td>
</tr>
</tbody>
</table>