CSCI E-236 Assignment 2
Making Waves

Due Date: Monday, April 12th, noon

1. Assignment and Grading

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

- 20 points: Water mesh
- 30 points: Indoor Scene
- 20 points: Outdoor Scene
- 30 points: Water physics
- 10 points: Extra credit

What we're providing: reflect.tga, refract.tga, heightmap.tga

What to hand in: Your zipped project files, including three workspace files mesh.dsw, outdoors.dsw, and indoors.dsw, and any texture maps you use. Don't include any release or debug directories. Your project should compile and run without requiring the TA to make changes.

Assignments submitted after the due date will not be graded.

2. Introduction

Water can be a compelling part of any scene. But it's not trivial to render; refraction, reflection, and surface movements all contribute to the appearance of water.

You'll be rendering two scenes: a swimming pool and an outdoor scene. The water will be displayed with reflection mapping, refraction mapping, and physical water simulation running entirely on the GPU. You will also use bump mapping and a skybox to bring realism to the scene.

There is no skeleton framework, except for code you find online and on the course web page. As always: You may use pieces of other people’s code, but you must acknowledge it clearly.

3. A water mesh

First, you'll want to create a basic OpenGL application to view a water mesh. The window should be 600x600 (resizable if you like).

3.1. A basic mesh
Our water will be modeled as a planar triangulated grid. For performance reasons we want you to use the Nvidia Vertex Array Range extension to define and transfer the water mesh. However, to make it easier to get started you may want to use vertex arrays, or triangle stripping, or just triangles – whatever you like.

Since rendering quality can depend on tesselation, the user should be able to change it on the fly. **When the user presses + (or -), that should double (or halve) the degree of tesselation.** For example, if the surface is tesselated into 64x64 (triangulated) quads, and the user presses “+”, the surface should be tesselated into 128x128 (triangulated) quads, and so on. To allow the user to see this tesselation directly, **allow the user to toggle to wireframe mode by hitting 'w'.** Remember, tesselating the surface differently should not change its overall size in OpenGL world space. When your program starts, use a relatively coarse tesselation of 16 x 16.

### 3.2. Camera and controls

The camera should initially point at the middle of the mesh, with a 45 degree field of view and a 1:1 aspect ratio. Controls should work as follows:
- When the **left mouse button** is held down, mouse movement in any direction should rotate the mesh. You might pick mouse-left/right as azimuth, and mouse-up/down as elevation – whatever you like, as long as it allows examining the mesh from any direction.
- When the **right mouse button** is held down, mouse movement (up and down) should zoom the mesh (in or out).

Remember, we just want a good view of the mesh here, so that we can get to the Cg part.

### 3.3. Performance Counter and Measurements

Measuring performance of an interactive application is very important. We would like you to implement a simple performance counter that measures elapsed time and displays “fps” (frames per second) in the window. Display the number of triangles of the water mesh as well.

Once the timer is working, we would like you to measure your performance for the following mesh tesselations:

16 x 16, 32 x 32, 64 x 64, ... , 1024 x 1024 (or higher, if your graphics card is still going strong)… Use the Nvidia Vertex Array Range extension to render your water geometry. Record your measurements in a .txt file and hand it in as part of your homework.

### 3.4. Copy/paste time

We suggest that you put your simple water mesh generator and viewer aside – you will need it when implementing water physics (see section 6). Now you should create two additional projects: one for your indoor scene, and one for your outdoor scene. Since both will likely share code written above, you can make two copies of the homework so far, and/or allow your two projects to share source files.
To make your TA's life easier, you'll want name the three dsw files accordingly ("mesh.dsw", "indoors.dsw" and "outdoors.dsw") before you zip them up for submission.

4. A swimming pool

Now we can get to the good stuff. To model an indoor water environment, you must include the use of refraction, reflection, and vertex oscillation. We've provided images for you to use for reflection and refraction (reflect.tga and refract.tga).

You may recognize those images from the paper, "Refraction Mapping for Liquids in Containers," by Alex Vlachos, in Game Programming Gems. And one or two of you may even recall that they use a simple kind of raytracing to do their refractions.

Needless to say, we don't expect you to implement raytracing in your shader. Even the Cg tutorial admits that refraction may involve some fudging. Use the images provided, and do the best job of refraction/reflection you can.

Finally, the outside of the pool can just be flat gray. If you want to Phong shade it, and throw in some lights, you can, but it's not required.

5. An outdoor scene

With more attention to the environment, we can make our water look even better.

We've supplied a 64x64 height map as a grey scale image. Each pixel value (between 0 and 255) corresponds to the height (y) of that vertex above the plane (x, z). This height map can be tesselated into triangles to form a simple terrain. You need to calculate per-vertex normals as well.

Since the height map's elevation ranges from 0-255 we can put our water level at 128. Place a water mesh (from section 3) at that height. Implement the same water mesh and camera controls as in part 3 of this homework. The scene will look good from above, and predictably wrong from below.

Next you will use Terragen to create textures for a skybox. If you Google for “terragen” and “skybox”, you'll find some resources; here is one such link:

http://fps.brainerd.net/terragen.htm

The terrain itself should be bump mapped. Create a greyscale heightfield image; be as creative as you like. Then convert it to a normal map with NVIDIA's normal map generator:

http://developer.nvidia.com/object/map_generator.html

The light direction for the bump map should correspond with your placement of the sun in your Terragen skybox (approximately: eyeball it if necessary).
Finally, apply refraction and reflection to your water using code from your indoor scene. Use the skybox textures for your reflection mapping. Make up a refraction texture using Terragen. Remember, refraction involves a bit of fudging – we're not writing a raytracer here. Be creative – as long as it looks good, we are happy.

6. Water Physics

OK, you asked for a challenge, so here it goes… We would like you to animate your water mesh using physics calculations performed entirely on the GPU. Yep, you heard right…

We suggest you first use your simple water mesh viewer from section 3 (without reflections, refractions, or shading) to get started. Once your water is animated using a simple mesh, move on and try to implement the same physics for your pool and outdoor scenes.

First, familiarize yourself with Mark Harris' render to texture class. You can find it at:

http://www.markmark.net/misc/rendertexture.html

Now look at the cloth simulation example of Simon Greene (on the course web page). It uses a mass-spring model to animate the cloth.

You have two choices:

a) Implement water physics using a simple mass-spring model. Make sure you choose your constants appropriately so that the water looks indeed like water. Start off the water movement by moving the corner vertices of your mesh up and down with a sin function. All other vertices should react based on the mass-spring physics.

b) Implement water physics using the wave equation shown in class. There is – as far as we know – no example code for this (definitely not for the GPU). Heck, we don’t even know if this works… But we do know that some of you like a challenge.

Remember: all (or most) of your physics computations must run on the GPU. Once your water mesh is moving, compute per-vertex normals and add it to the pool and outdoors scenes. Compute reflections and refractions as well.

7. Optional Extra Credit

Implement caustics in your indoor scene. You can do this however you like (though if you do a really bad job, you won't get many points). One way of doing this is illustrated in Gamasutra (requires registration):

http://www.gamasutra.com/features/20030903/crespo_01.shtml