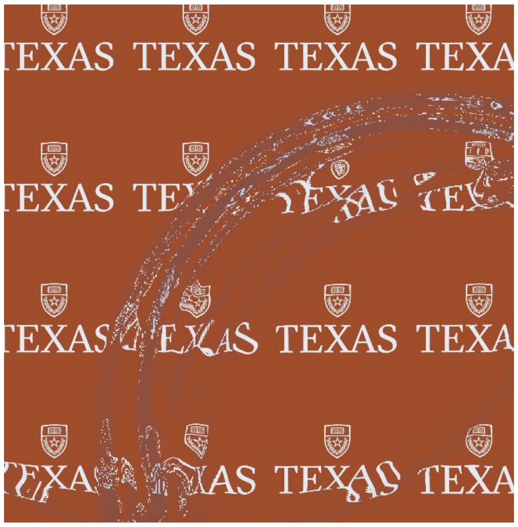
CS 354 : Shallow Water

Mustafa Abban



Introduction

A shallow water simulation developed using webgl. My motivation behind this was to showcase a real time animation that was responsive to the user. For this reason I looked into webgl and felt it was best.

Methodology

WebGL provided a great amount of resources for establishing the shader program. I went about splitting the water texture into three channels. First being the height deviation and the others were the speed in both axii. Then equations are derived from "equations of conservation of mass and conservation of linear momentum (the Navier–Stokes equations)," which hold even when the assumptions of shallow water break down. By not holding across for this hydraulic jump and neglecting Coriolis force, I approached the equation by utilizing water normals to replace the height space derivatives.

Implementation

This is the Non-conservative form of the equation where

- ***(u, v)*** is the speed in the horizontal plane,
- ***h*** is the height deviation of the horizontal pressure surface from its mean height H,
- ***H*** is the mean height of the horizontal pressure surface,
- ***g*** is the acceleration due to gravity ($g = 9.8m/s^2$),
- ***f*** is the Coriolis coefficient. I neglect the effects of the rotation of the earth, so take f = 0,
- ***b*** is the viscous drag coefficient.

$$\frac{\partial u}{\partial t} - fv = -g\frac{\partial h}{\partial x} - bu,$$
$$\frac{\partial v}{\partial t} + fu = -g\frac{\partial h}{\partial y} - bv,$$
$$\frac{\partial h}{\partial t} = -H\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right)$$

As seen the variation of water speed can be approximated by utilizing Euler Method.

$$d\begin{pmatrix} u \\ v \end{pmatrix} = -dt\left(g\begin{pmatrix} n_x \\ n_y \end{pmatrix} + b\begin{pmatrix} u \\ v \end{pmatrix}\right)$$
$$dh = -dt.H\left(\frac{du}{dx} + \frac{dv}{dy}\right)$$

This is implemented in the shader program but there was an issue with sampling and rendering to the same texture. As mentioned in the OpenGL documentation “normally, you should not sample a texture and render to that same texture at the same time. This would give you undefined behavior. It might work on some GPUs and with some driver version but not others.”

So a new shader was built to copy the water texture before sending as input to GPGPU water shader. There is where water normals are used to to simulate height space derivatives. These water normals are then maintained as a normal map texture.

An issue arose when realizing the texture channels were 8 bits. Meaning the channels had enough to store color but not velocity or position. To solve this, I utilized the OES_TEXTURE_FLOAT extension. This allowed for every channel to store a float.

In order to homogenize physical quantities amongst the water, I found it was best to apply a three pixel wide blur that could be applied across the texture.

To compute refraction I utilized the refract(genType I, genType N, float eta) function within OpenGL to compute the UV coordinates on the floor. For determining between reflection or refraction, I utilized Fresnel coefficients where F=water_normal.z and I would use mix(color_sky, color_floor, 0.6+F*0.3) to generate the new color.

Future development / Conclusion

I would love to continue building upon this project. Potentially have it within a 3d space and allow for camera movement. Id also look into allowing the user to change viscosity or drop objects within the water.