Micah Bojrab
FX Artist

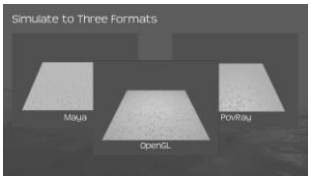
mbojrab@purdue.edu
MicahBojrab.com
260.402.2476

Demo Breakdown:

Navier-Stokes (NS) Hydrostatic Pipe Model Simulation - 0:03
- Fluid simulation with rendering in Mental Ray.
- *Tools Used:* C++; MEL; Maya

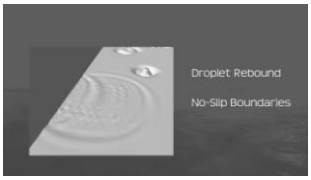
This shot shows a hydrostatic pipe model simulation in C++. The simulation is based on published research by Beneš and Chiba, and builds on of the concept of vertical columns of water continuously attempting equilibrium with neighboring columns. The simulation is a quad-based system, which is completely scalable per user preference. The grid shown in this simulation is 600 by 600 tiles and updates a frame every 1.2 seconds. All assets created for this reel are completely from scratch. No pre-made tools, models, scripts or converters were used.

The shot is further complicated by the size of the data. In all, 400 individual OBJ files were created for rendering. A MEL script was created to efficiently import batches of fifteen files, key the visibility of the OBJ sequence per frame, and save the Maya file with a unique file name. This process automatically continues until a break in the OBJ series is found. The code gives the ability to scrub through large files in real-time and view the animation as if it were a single animated object.



Navier-Stokes (NS) Hydrostatic Pipe Model Simulation - 0:11
- Three simulation renderings from different render engines

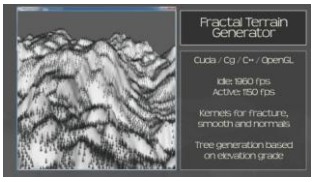
The simulation saves data to three file formats. It renders off-line in OpenGL, where the images are saved to TGA. The simulation also sorts and saves to two geometry file formats: OBJ and POV. OBJ files are used within Maya as stated in the first section. PovRay, an open source command-line style ray tracer, accepts the POV files. Additional small tool scripting was created to organize file indices, and render the PovRay project over a network.



Navier-Stokes (NS) Hydrostatic Pipe Model Simulation - 0:25
- No-Slip boundary for static walls.

The simulation has a randomized rain function, which generates the illusion of rain droplets in a large body of water. The rain has a variable downpour speed that can be controlled through users settings. Additionally, no-slip boundary conditions are used to rebound water against static walls.

Boundary conditions are efficiently achieved by breaking borders and corners into separate entities, where each is controlled by individual equations. This saves time by not having to apply conditionals across the entire grid.

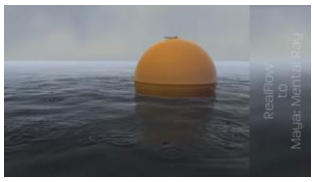


Fractal Terrain with Randomize Forrest Generator - 0:31
- *Tools Used:* CUDA, Cg, C++ and OpenGL

The Fractal Terrain Generator is a GPU-based application in OpenGL, which runs CUDA kernels to modify the ground terrain in real-time. Fractal terrain draws a seeded, randomized vector and tests the dot product between the individual vertices and the vector. The program programs extends this concept in various ways.

Vertex buffer objects (VBO) were used for all renderable objects. A file converter was also created that reads OBJ data and outputs interleaved VBO data in a personalized (.vbo) file. At run-time, the program parses this file much faster than OBJ data and load times are reduced by over 60%. The VBO storing the terrain is registered with CUDA, so data can be both updated and rendered every frame without having to leave the device. The interoperation between CUDA and OpenGL allows the application to output, on average, 1150 fps while having active kernels. Additionally, each kernel calculates smooth normals for the gird, and the application renders to points, wireframe and smooth shaded geometry.

Tree generation is the only function offloaded to the CPU. When the tree callback is registered, the grid data is copied to host memory through buffer mapping; tree transforms are randomized based on grid cells; and elevation grade is calculated and stored per cell. The forrest is limited by the grade in elevation, where trees are not able to grow on steep slopes. All tree data is pre-computed and stored, so at render-time, a simple boolean conditional is used to find renderable trees. The elevation grade can be user controlled, and allows the forrest to interactively grow at various slopes.

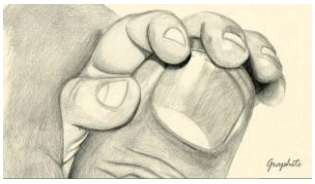


RealFlow Scenes - 1:10

- *Tools Used:* RealFlow and Maya

These simulations were created in RealFlow for my thesis on the important lighting phenomena for rendering water. The study found which lighting components contribute most to the perceptual realism of water. The results were then used to find perceptually-driven reductions in rendering time with limited, or no compromise to visual quality. The results of the study were submitted to SIGGRAPH Asia 2010.

The first scene is a particle system with object interaction. To achieve a more complex simulation, the emitter's velocity was keyed, which stirred the water at even intervals. The particle cloud was meshed within RealFlow and exported to Maya for rendering. The second and third scenes are RealWave objects, which interact with geometry and particles. All assets and textures are created from reference, or in the case of the steel buoy, are created by hand.



Hand Drawing - 1:29

- *Tools Used:* Graphite

This is a simple graphite pencil drawing.