Height Filed Simulation and Rendering

Jun Ye

Data Systems Group, University of Central Florida

April 18, 2013
Outline

- Three primary goals
- Techniques in papers
- My implementation
- My extension
- Demonstration
Primary goals

Simulate a heightfield and render it in real time with user’s interactions. Take the water rendering as an example:

1. Animate a pool of water
2. Render the water
3. Support user’s interactions (e.g. click for ripples and waves) and caustics effects

A WebGL demo: http://madebyevan.com/webgl-water


Fluid Dynamic (Water) Survey

- **Height filed-based**
  - Early works
  - Simulate ocean surface and view from a long distance away
  - PDE, 2D or 3D Navier-Stokes Equations

- **Particle systems-based**
  - More recently
  - Special effects: water fall, drop let, spray and foam, breaking waves, thermal activity, melting, droplets and their streams on a glass plate, etc.
  - Particle systems, metaballs, texture mapping

- **Combined**
  - More complex event, e.g. Interaction with static and dynamic buoyant obstacles
  - PDE+particle systems
A water surface: a tightly stretched elastic membrane

- Horizontal forces are cancelled
- Particles move in only the $z$-direction
- A 2-dimension water surface height map

The vertical position w.r.t. time and space can be described by the PDE:

$$\frac{\partial^2 z}{\partial t^2} = c^2 \left( \frac{\partial^2 z}{\partial x^2} + \frac{\partial^2 z}{\partial y^2} \right),$$

where $c$ is the speed at which waves travel across the surface.
The general solution for a square $L \times L$ section of water is:

$$z(x, y, t) = \frac{2}{L} \sum_{m=1}^{\infty} \sum_{n=1}^{\infty} A_{mn} \sin\left(\frac{m\pi x}{L}\right) \sin\left(\frac{n\pi y}{L}\right) \cos(c\omega t),$$

where,

$$\omega = \frac{\pi}{L} \sqrt{(mx)^2 + (ny)^2},$$

$$A_{mm} = \frac{2}{L} \int_0^L \int_0^L f(x, y) \sin\left(\frac{m\pi x}{L}\right) \sin\left(\frac{n\pi y}{L}\right) dx dy$$
Using an evenly spaced grid of $z$-value (a height file)
- Becomes a discrete problem and FFT can be applied.
- Only consider the most significant components, still heavy.

Use central differences to approximate the PDE:

$$\frac{z_{i,j}^{n+1} - 2z_{i,j}^n + z_{i,j}^{n-1}}{\Delta t^2} = c^2 \left( \frac{z_{i+1,j}^n + z_{i-1,j}^n + z_{i,j+1}^n + z_{i,j-1}^n - 4z_{i,j}^n}{h^2} \right)$$

$$z_{i,j}^{n+1} = \frac{c^2 \Delta t^2}{h^2} \left( z_{i+1,j}^n + z_{i-1,j}^n + z_{i,j+1}^n + z_{i,j-1}^n \right) + \left( 2 - \frac{4c^2 \Delta t^2}{h^2} \right) z_{i,j}^n - z_{i,j}^{n-1},$$

where $c$ is the speed, $h$ is the length of single grid.
\[ z_{i,j}^{n+1} = \frac{c^2 \Delta t^2}{h^2} (z_{i+1,j}^n + z_{i-1,j}^n + z_{i,j+1}^n + z_{i,j-1}^n) + (2 - \frac{4c^2 \Delta t^2}{h^2}) z_{i,j}^n - z_{i,j}^{n-1}, \]

The computation of \( z_{i,j} \) in the current time needs only

- the values of its four direct neighbors in the last frame, and
- the value of \( z_{i,j} \) in the last two frames.

- Much simpler and suitable for OpenCL.
- \( \frac{c^2 \Delta t^2}{h^2} \leq \frac{1}{2} \), otherwise will not converge

Initial value of the height field is zero (a flat plane). Disturbance can be added to create ripples.
Water Surface Rendering

A ray-tracing-based rendering algorithm is used:

- Water volume is composed of two height fields: water surface and ground
- Single level recursivity assumption: for each viewing ray, find the intersection with the water surface, then the intersection of the refracted ray with the ground (step length=2 in path tracing)
- Four possible scenarios when rendering
Ray-water surface intersection algorithm (Binary Search Tree)

Precompute the water surface bounding box (safe box);

\[ [t_{\text{near}}, t_{\text{far}}] = \text{RayBBIntersection}(\text{ray}, \text{safeBB}); \]

\begin{algorithm}
\begin{algorithmic}
\While {t_{\text{far}} - t_{\text{near}} > \delta}
\State \[ t = \frac{t_{\text{near}} + t_{\text{far}}}{2}; \]
\If {\[ y_t < \text{heightfield}(x_t, z_t) \text{ \&\& } y_{t_{\text{near}}} > \text{heightfield}(x_{t_{\text{near}}}, z_{t_{\text{near}}}) \]}
\State \[ t_{\text{far}} = t; \]
\Else
\If {\[ y_t > \text{heightfield}(x_t, z_t) \text{ \&\& } y_{t_{\text{far}}} < \text{heightfield}(x_{t_{\text{far}}}, z_{t_{\text{far}}}) \]}
\State \[ t_{\text{near}} = t; \]
\EndIf
\EndIf
\EndWhile
\State \text{return } t;
\end{algorithmic}
\end{algorithm}
My Plan and Implementation

– Using the Nokia Model, fill up with water
– Simulate the height field
– Rendering the water volume with path-tracing
– Human interaction (Mouse drag ripple, etc)
– Texture mapping
– Caustics
Implementation: Water surface simulation

- 128×128 height field map
- Update the height filed map every frame in javascript and pass it to the kernel
- Bilinear interpolation to compute the normal at water surface

\[
f_1 = \frac{x_2 - x}{x_2 - x_1} \bar{n}_1 + \frac{x - x_1}{x_2 - x_1} \bar{n}_2
\]

\[
f_2 = \frac{x_2 - x}{x_2 - x_1} \bar{n}_3 + \frac{x - x_1}{x_2 - x_1} \bar{n}_4
\]

\[
\bar{n} = \frac{y_2 - y}{y_2 - y_1} f_1 + \frac{y - y_1}{y_2 - y_1} f_2
\]
Implementation: Water surface rendering

Water surface simulation results:

- without normal interpolation
- with normal interpolation

Jun Ye (UCF)
Three attempts to produce caustics:

– Path-tracing with static water surface
  • photo-realistic but only work for static water, snap shot
– Path-tracing with dynamic water surface (50 samples per pixel per frame)
  • lots of noise, slow
– Photon-mapping and texture-mapping
  • less noise, fast
Implementation: Caustics

A simplified photon-mapping based method:

- Two-pass rendering
- Backward: light ray; forward: eye ray
- Light ray produces photon map and only store photons on the floor and side walls
- Photon map is converted to texture map, and used for rendering
- 320×320 photon map and texture map
Implementation: Results of caustics rendering

Comparison of three methods

Path-tracing, static water

Path-tracing, dynamic water 50 samples

Photon-mapping, dynamic water
Hardware configuration

- $800 \times 664$ canvas resolution
- nVidia GTX 660 (960 OpenCL units)
- FPS = 17
Project calendar

April 3
April 4
April 5
April 6
April 8
April 9
April 10
April 12
April 15
April 16
April 17
April 18
Thank you.