About Phoenix FD

• PLUGIN FOR 3DS MAX AND MAYA.
• SIMULATING AND RENDERING BOTH LIQUIDS AND FIRE/SMOKE.
• USED IN MOVIES, GAMES AND COMMERCIALS.
Phoenix FD core

- SIMULATION & RENDERING.
- SIMULATION CORE - GRID-BASED SOLVER:
  - EXTERNAL FORCES.
  - DIVERGENCE CLEANING (CONSERVATION).
  - ADVECTION.
- HYBRID SOLVER.
- FLIP SOLVER.
Phoenix FD conservation close-up

In any solver type the conservation takes a significant time.

Conservation methods are iterative algorithms doing many iterations on the entire grid.

Phoenix implements 2 double-buffered methods and 2 that work in-place.
Phoenix FD buffered conservations

GPU friendly and easy to multithread.

On the CPU – same multithreading method as for all other supporting algorithms:
Splitting the grid vertically into equal sections.
Phoenix FD standard multithreading

For example: 4 threads.

Data amount for each section:

Thread 3
Thread 2
Thread 1
Thread 0
Phoenix FD standard MT

Issues:

• BAD LOAD BALANCING.
• BAD USE OF L3 CACHE.
Phoenix FD standard MT

Resolving both CPU and cache issues: each thread $n$ works on those horizontal slices where $z \mod N = n$
Phoenix FD in-place conservations

Read, calculate and write immediately the result in the same buffer:

• MUCH STRONGER CONSERVATION.
• EACH CELL DEPENDS ON THE PREVIOUS ONE.
• VERY BAD FOR THE GPU.
• SINGLE-THREADED ON THE CPU.
Phoenix FD in-place conservations

For each cell, calculate its divergence and correct the velocities proportionally so that the divergence is zeroed.
Phoenix FD in-place conservations

The cell’s own velocities cancel out of the divergence calculation.
Phoenix FD in-place conservations

In the 3D grid, a checker pattern emerges, where the black and white cells don’t interact:
Let's split the grid in horizontal slices as in the previous method and try to run each slice in a separate thread:
Phoenix FD in-place conservations

Influence pattern for Thread 0 and Thread 1, showing that Thread 1 must be delayed by 2 rows:

- Thread 3
- Thread 2
- Thread 1
- Thread 0

Diagram showing the influence pattern with Thread 1 delayed by 2 rows.
Phoenix FD in-place conservations

The ‘Harvester’ method:

Thread 0
Thread 1
Thread 2
Thread 3 (waiting)
Phoenix FD rendering

Real-time GPU-based volumetric rendering

- THE USUAL RENDERING WITH THE CPU
- THE ADVANTAGES AND LIMITATIONS OF THE GPU
- THE IMPLEMENTATION ON THE GPU
Phoenix FD usual CPU rendering

What kind of data we have:

- DENSITY FUNCTION REPRESENTED AS A 3D GRID
- OBSERVER’S POSITION AND ORIENTATION (CAMERA)
- LIGHT SOURCES
Phoenix FD usual CPU rendering

Calculating what we see in a certain direction:

- **THE RESULT IS A WEIGHTED SUM**
- **CALCULATION OF THE WEIGHT**
- **CALCULATION OF THE LOCAL BRIGHTNESS**
Phoenix FD **usual CPU rendering**

The contribution of a short segment:

\[ \text{Lo} - \text{the light intensity without obstacles} \]
\[ \text{Li} - \text{the light attenuation caused by the absorption} \]

\[ \text{Si} - \text{the cross section of the i-th segment} \]
\[ \text{dr} - \text{the integration step} \]
\[ \text{wi} - \text{the weight of the i-th term, in fact the probability to reach p without absorption} \]
Phoenix FD GPU advantages and limitations

Short history of the GPU:

• BASIC FUNCTION – CONVERT TRIANGLES FROM 3D to 2D AND TEXTURE THEM
• THE PROGRAMMABLE FIXED PIPELINE WITH 8 STAGES
• THE PIXEL SHADERS AND THEIR EVOLUTION INTO GENERAL-PURPOSE CALCULATION UNITS
Phoenix FD GPU advantages and limitations

Why the GPU is faster?

- MULTIPLE CORES WITH THE SAME CODE RUNNING ON THEM
- SIMPLER FLOW CONTROL – NO BRANCHES, NO COMPLICATED ADDRESSING MODES
- SOME INTENSIVELY USED MATH IS DIRECTLY SUPPORTED (INTERPOLATION, 3D VECTORS)
Phoenix FD GPU advantages and limitations

The difference from the CPU

- THE EXECUTED CYCLES ARE LIMITED
- LIMITED SYNCHRONIZATION ABILITIES
- NO RANDOM ACCESS TO THE MEMORY
Phoenix FD implementation on the GPU

General overview

• IN MOST CASES THE GPU IS USED IN MULTIPLE PASSES
• A DUMMY PRIMITIVE (USUALLY A BOX) THAT WILL BE ‘RENDERED’ IS PREPARED
• THE LARGE DATA ARRAYS ARE PASSED AS TEXTURES, THE SINGLE CONSTANTS ARE PASSED IN REGISTERS
• THE NEEDED PIXEL SHADER CODE IS ACTIVATED AND THE RENDERING OF THE DUMMY PRIMITIVE IS STARTED
• AFTER THE ‘RENDERING’ THE RESULT IS IN THE BACK-BUFFER
Phoenix FD implementation on the GPU

The light propagation

- For performance reasons the lights are represented as parallel.
- We find the biggest component of the direction vector and propagate the light layer by layer in this direction.
- Each layer is calculated by a single pass on the GPU and the result is returned into the back buffer. We keep it into a texture and pass it back to the GPU for the next layer calculation.
Phoenix FD implementation on the GPU

The light propagation

Light direction

\[ L_i = L_p \times (1-S_i) \]
Phoenix FD implementation on the GPU

The light propagation

- Once having the light map, we can use it multiple times with different camera positions (we can rotate the view).
- We pass the light map to the GPU, along with the density map, the camera, and the light direction.
- The camera ray integration is exactly like the CPU implementation.
THANK YOU!