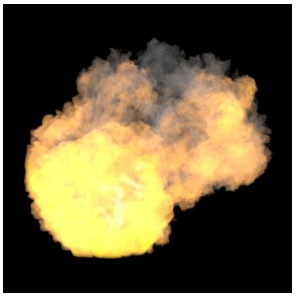
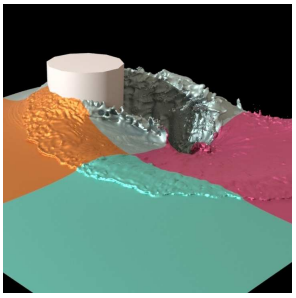
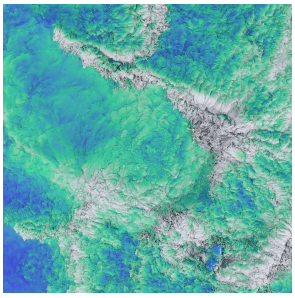
Inside Houdini's Distributed Solver System

Jeff Lait*
Side Effects Software Inc.


(a) Top view of explosion, split 2x2


(b) 9.8 million particle sand with PBD, split 6 ways


(c) 400^3 PLS fluid, split 2x2


(d) 1 billion particle FLIP, split 2x2

Keywords: distributed simulation, level set, fluids, FLIP

Concepts: •**Computing methodologies** → **Distributed algorithms;** •**Networks** → *Peer-to-peer protocols;* •**Applied computing** → *Physics;*

1 Introduction

Distributing a simulation over multiple computers enables simulations that exceed the capacity of any individual machine. Distributed systems, however, tend to be hard to deploy and maintain. Our approach can work with commodity hardware and is agnostic to the specific type of simulation. Users have successfully deployed on a variety of hardware over many years and accelerated a wide variety of types of simulations.

2 Spatial Distribution

The faster turn around times from distribution are coupled with decreased efficiency. The main drive for distributed simulations was to be able to complete simulations that were otherwise impossible due to total resource requirements. To break a problem up, we divide space into slices, one per machine. Each machine only has knowledge of the simulation data in their slice and a thin boundary around it. Because communication bandwidth is proportional to the area of the slice boundary, it is important to minimize the slice area to volume ratio. We divide volume grids uniformly and divide particle simulations by artist placed cutting planes.

Houdini's simulations rely on operator splitting to give artists control over their look and behaviour. We have observed that almost all user-created operations, such as novel forces, diffusions, and metamorphoses, require only local knowledge of the simulation.

*e-mail: jlait@sidefx.com

SIGGRAPH '16, July 24-28, 2016, Anaheim, CA,
ISBN: 978-1-4503-4282-7/16/07
DOI: http://dx.doi.org/10.1145/2897839.2927421

For each operator, we can consider the distance its effect will be felt within a single substep. Provided this is less than the boundary width, no special distribution code is required for the operator. Inter-particle forces in an SPH-simulation, for example, are bounded by the kernel size, so can be computed in a naive fashion. Advection of a Eulerian grid, likewise, can be performed if the maximum boundary velocity does not cause motion farther than the boundary width. A five voxel boundary, for example, can support a CFL of five for the advection step.

When combining operators, the maximum influence is bounded by the sum of the individual distances. In practice we have not had to strictly follow this bound. Some operations work on different parameters, so are independent. And most operations have a rapidly decaying influence, so while convolving the influence function will grow the support, truncating the tail usually introduces minimal error. This can be likened to the truncated support used for Blinn metaballs.

3 Tracking

To ensure we could handle simulations too big for any one computer, we avoided having any machine be in charge of the entire simulation. Instead, each machine runs its own full, independent, session of Houdini, focused on its own slice of space, and only is aware of the distributed nature of its simulation when certain operators are run.

We do need a way for these machines to find each other, however. This is complicated because we need to run within third party renderfarms, where we have no control on machine assignment. Each machine is only provided a single address and port: that of a tracker. The tracker's job is to accept sync requests from each slice machine. When all the slices have reported in, it can send back to all of them their respective addresses and communication ports. The slice machines are then able to open direct peer-to-peer communication with each other. We thus allow full efficiency on switched networks by avoiding a centralized data exchange. When machines complete their exchange they report back to the tracker. Once all machines have checked in, a final message is sent from the tracker to release the machines from their synchronization point. While this final stall is not strictly required for all synchronization processes, reasoning about distributed systems is sufficiently complex. Hence, we elected for reliability and reproducibility over pure efficiency.

3.1 Networking

Without control of the installed network stack, we had a choice of TCP, UDP, or a higher level protocol built on top of those systems.

Unlike realtime applications, our synchronization stages cannot accept missing packets or re-ordered packets, as we wish to produce deterministic results. Since building on top of UDP would require re-inventing TCP's reliability guarantees, we concluded the base layer should be TCP.

We abstracted this layer by building a class to handle a synchronization event. Each event builds its own class, causing each synchronization to reconnect to the tracker to discover ports and synchronize. Since distribution is applied when simulations are already large in size, this overhead is not significant. Also, since each synchronization is reported by name to the tracker, the tracker is able to produce a human readable state of the simulation. This makes it easy to detect split-brain problems, such as two machines deciding to take different substeps, without the use of a C++ or network debugger.

4 Common Synchronization Points

Please refer to the supplementary materials for more detailed analysis of our synchronization methods and pressure projection.

5 Results

Our framework has allowed us to distribute smoke, fire, water, and sand simulations using PLS, SPH, FLIP, PBD, and multigrid solvers. Spatial division and boundary exchange can form a kernel that allows easy distribution of many effects without coupling the distribution system with the solving system.

By distributing we gain not only speed, but the ability to break hard limitations of individual machines. We are able to run GPU-based OpenCL simulations on four separate video cards that would have exhausted the memory of a single card. Similarly, with CPU-based simulations we can fuse the memory of multiple machines to avoid swapping, allowing the simulation to complete in a practical amount of time. We are also able to simulate over four billion points in a FLIP simulation without encountering any 32-bit index limits as no individual machine had to see all of the particles.

Table 1: *Performance; per-frame times*

Test	Single	Distribute	Speed Up
PBD Sand, 9.8MPart, 6 slices	1072s	195s	5.5x
FLIP, 184MPart, 4 slices	180.5s	56.1s	3.2x
PLS, 400^3, 4 slices	469s	142s	3.3x
Explosion, 269MVoxel, 4 slices	168s	95s	1.8x

Position based sand uses particle exchange every substep along the boundary. Because we use Jacobi iterations, each iteration's influence is bounded by the particle radius. Further, if the mass is uniform across the slice boundary, while the influence boundary grows with every iteration, its effect swiftly falls off. This common downside to explicit smoothing becomes for us a virtue. The PBD simulation also uses automatically varying slice positions to balance the work load, quite important as the pile of sand swiftly has a very uneven distribution.

The FLIP simulation is a tank simulation cut into four equal pieces. Since global pressure projection is properly distributed, the key is to have a boundary sufficient to cover the advection stage.

The particle level set simulation is a half-full tank. It uses the global distributed pressure projection and thus, like FLIP, has its boundary determined by the advection stage.

The explosion is a multigrid IOP simulation split equally into four along the X and Z axes (with Y up). The grid is resizing, however, requiring considerable data to be transferred between machines as ownership of voxels changes hands. The multigrid solve is not distributed; instead, each slice naively solves its own segment with closed boundary conditions on its neighbour. This provides a frame-delayed boundary condition that produces visually plausible results but differs noticeably from the undistributed simulation.

References

BAILEY, D., BIDDLE, H., AVRAMOUSSIS, N., AND WARNER, M. 2015. Distributing liquids using openvdb. In *ACM SIGGRAPH 2015 Talks*, ACM, New York, NY, USA, SIGGRAPH '15, 44:1–44:1.

FLORES, L., AND HORSLEY, D. 2009. Underground cave sequence for land of the lost. In *SIGGRAPH 2009: Talks*, ACM, New York, NY, USA, SIGGRAPH '09, 6:1–6:1.

IRVING, G., GUENDELMAN, E., LOSASSO, F., AND FEDKIW, R. 2006. Efficient simulation of large bodies of water by coupling two and three dimensional techniques. *ACM Trans. Graph. 25* (July), 805–811.

LAIT, J. 2011. Correcting low frequency impulses in distributed simulations. In *ACM SIGGRAPH 2011 Talks*, ACM, New York, NY, USA, SIGGRAPH '11, 53:1–53:2.