Spatially Adaptive FLIP Fluid Simulations in Bifrost

Michael B. Nielsen*
Autodesk

Robert Bridson*
Autodesk

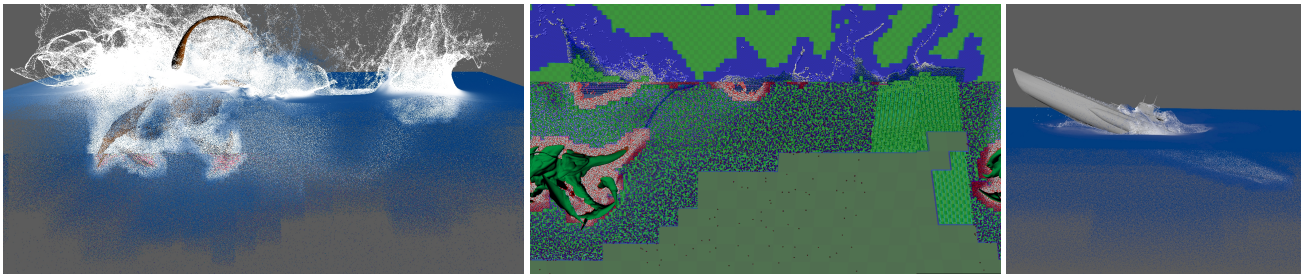


Figure 1: *A "Thresher" jumps into deep adaptive water (left). A slice through the spatially adaptive tile tree with three levels of depth-based adaptivity in voxel size and particle density shown (middle). A submarine emerges from deep adaptive water (right). Thresher courtesy of: model by Eddie Munoz (ed209.artstation.com), rig by Justin Kirk (justinkirkcg.com), animation by Kostadin Martic (springonionstudio.com).*

1 Spatial Adaptivity

Spatial adaptivity has great promise for fluid simulation in visual effects. Given that a $2\times$ increase in spatial resolution incurs an $8\times$ increase in memory for 3D, using high resolution only where needed can offer a huge savings for many interesting scenarios. For example, figure 1 shows a large-scale simulation of a creature emerging from deep water: the depth is critical for the bulk motion, but the velocity field away from the creature and surface is relatively smooth and does not require as high resolution. Adaptive fluid simulation was proposed for graphics over a decade ago [Losasso et al. 2004], and the combination of FLIP and octrees seven years ago [Hong et al. 2009], yet it has been slow to catch on in the industry. Instead, FLIP with uniform resolution sparse voxel tiles has emerged as the standard approach. Sparse voxel tiles efficiently cover arbitrary domains, give excellent cache usage and low data structure overhead, and allow the use of popular voxel algorithms for level set computations etc. with barely any changes. By comparison, typical adaptive octree and/or tetrahedral mesh approaches incur sizeable overheads and may rule out many standard voxel algorithms. SPGrid [Setaluri et al. 2014] is one recent option for efficiently achieving octree-like adaptivity via multiple levels of sparse voxel tiles; in this talk we discuss a more aggressive approach using a wide-branching *tile tree* of voxels. We have fully implemented this in Bifrost in Autodesk Maya, demonstrating that even in uniform resolution we achieve the performance of optimized sparse voxel tile code, and that spatial adaptivity gives significant further improvements in both memory and runtime.

Keywords: fluid simulation, FLIP

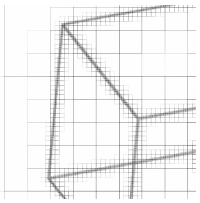
Concepts: •**Computing methodologies** → **Physical simulation;**

*e-mail:michael.nielsen@autodesk.com, robert.bridson@autodesk.com

SIGGRAPH '16, July 24-28, 2016, Anaheim, CA,
ISBN: 978-1-4503-4282-7/16/07
DOI: http://dx.doi.org/10.1145/2897839.2927399

2 Adaptive tile tree

The core Bifrost data structure is the tile tree, a generalized octree where each node is a $5\times5\times5$ tile rather than just 2^3. We allow up to 7 levels in the tree, providing a domain of 390,625 voxels on each side, each level five times finer resolution than the next. For voxel data, a value is stored in each of the 125 cells of each tile, contiguously in a cache-friendly manner: we keep the advantages of sparse voxel tile structures, but store and work with tiles at any level providing spatial adaptivity and level-of-detail (LOD) capability. This is similar to OpenVDB [Museth 2013], which also stores voxel tiles in a tree. OpenVDB supports an address space bounded only by the bit-precision of the coordinates, and employs a short wide-branching tree with configurable tile sizes based on powers of two at each level. While OpenVDB encodes larger voxels at coarser levels, it is optimized for uniform resolution voxels at the finest level. Our tile tree instead provides operations independent of level, and was designed from the start to naturally support LOD spatial adaptivity with a multi-resolution voxel representation at each spatial location. We also differ from OpenVDB in handling multiple channels of data attached to the same tile tree without duplication of tree topology.



The choice of five is noteworthy. Odd-sized tiles allow voxel-centered samples to line up across resolution jumps (instead of being offset by half a cell for even sizes); five is a sweet spot between tiles large enough to amortize data structure overhead and small enough for good cache efficiency and efficient adaptivity around irregular data.

We introduce *haloed tiles* to simplify voxel operations on spatially adaptive data. Sampling a continuous interpolant in a tile or evaluating a finite difference stencil, for example, may involve values in nearby tiles; handling all possible resolution jumps is cumbersome. Instead we provide code to efficiently construct a 7^3 haloed tile around any given tile, either copying the halo values from neighboring tiles at the same level, or appropriately interpolating them from coarser levels as needed. Subsequent operations can then work on the tile as if it were just part of a uniform resolution grid, without explicitly handling spatial adaptivity. We provide halo constructions for linear interpolants as well as C^1 quadratic B-splines, which are smooth even across resolution jumps. We also provide a variety of efficient parallel breadth- and depth-first traversals, where user-supplied code to process a tile is plugged in without needing to explicitly handle

parallelism, even when constructing a new tile tree in parallel.

3 Adaptive Level Sets

The tile tree allows level sets to be represented adaptively even along the surface. One of the keys to a scalable adaptive liquid solver is performing all numerical computations directly with this adaptive representation, as opposed to temporarily reverting to a uniform resolution narrow band representation, for example. We accordingly developed a toolbox of lock-free parallel algorithms including:

- **Adaptive fast sweeping:** borrowing ideas from multigrid, our method computes adaptive distance fields on the tile tree with a fine-to-coarse traversal initializing distance followed by a coarse-to-fine refinement. At each level of the tree we run parallel red-black fast sweeping in haloed tiles. Additional speed-ups come from transferring values between LOD levels with prolongation and restriction operators.
- **Adaptive level set from particles:** our adaptive particle representation is converted to an adaptive level set using a hierarchical union of spheres algorithm combined with spatially varying erosion.

Our toolbox also includes other novel extensions to liquid solver components such as adaptive CSG operations, data extrapolation in the normal direction, filtering and more. Generally we found that the LOD representation, where values are stored at non-leaf voxels as well, is a critical component of efficient adaptive computations.

4 Adaptive Particles

Unlike previous work on adaptive FLIP [Hong et al. 2009; Ando et al. 2013], we view the FLIP particles as just samples of a continuous function as opposed to particles with a certain radius. For particles at the liquid surface this is especially important as offsetting the surface away from the particles by some radius visibly dampens wave propagation. We also neither split nor merge particles, but rather reseed particles based on the continuous interpolant and delete particles based on a sub-grid partitioning to ensure good sampling, maintaining a reasonable particle count per voxel. The particle density adapts to the voxel size and rapidly decays away from refined regions. Particles are stored in the same tile tree, continuously re-blocked into leaf tiles in which they are spatially located, which optimizes all our particle-voxel operations. Also different from prior work, we perform a memory-efficient coarsening and refinement of the tile tree in-place by a single depth-first traversal with a criterion at each voxel to determine if the branch should be retained, coarsened or refined. Similar to Ando et al. [2013] our criterion is based on distance metrics such as distance to the surface or selected solid boundaries (themselves represented by adaptive level sets), but we do not require expensive operations such as a Delaunay tetrahedralization. Our contributions to adaptive FLIP also include adaptive transfer operations between the particles and the grid based on a modified trilinear interpolant. All algorithms are fully lock-free parallel.

5 Adaptive PDEs

Solving the Poisson equation for pressure is critical to incompressible fluids, and for highly viscous liquids solving for the viscous effects implicitly (a parabolic equation) is also essential. We faced two challenges in implementing these for spatially adaptive tile trees: the discretization, and fully parallelizing the linear algebra operations.

Finite difference or finite volume schemes on adaptive grids with T-junctions (like octrees or our tile tree) have generally not been able to achieve matrix symmetry (necessary for the most efficient linear solvers), second order accuracy across resolution jumps, and full accuracy at non-grid-aligned boundaries (cut cells). By contrast, the finite element method handles these almost automatically. However, in uniform regions the usual trilinear elements lead to a 27-point stencil, which is nearly four times more costly than the 7-point finite difference stencil. We achieved the best of both worlds with a modified finite element method, sparsified to a 7-point stencil in every tile by a form of lumping, and automatically assembled together into a global matrix via Galerkin projection. We similarly discretized viscosity to be equivalent to optimal finite difference methods in uniform resolution regions, but gracefully handle spatial adaptivity as well.

Assembling and then solving the resulting linear systems required a variety of sparse matrix operations. However, standard representations of sparse matrices, such as Compressed Sparse Row (CSR), pose significant sequential bottlenecks in operations such as matrix-matrix multiplication. To effectively parallelize our sparse matrix suite we created a segmented sparse matrix structure, where we partition both the rows and columns of a sparse matrix into blocks according to tile trees, and store only the nonzero blocks in a generalized CSR format. Parallelization over both row and columns then becomes trivial, and allows scalable lock-free algorithms for otherwise difficult operations such as transposition and general matrix-matrix multiplication.

6 Results

	Houdini 15	Bifrost	Bifrost Adaptive
# time/frame	59.3s	37.6s	12.4s
# particles (avg)	12.1M	12.1M	3.83M
# voxels (avg)	10.1M	11.8M	3.10M

Table 1: *Benchmark results for the submarine simulation on a Windows 7 machine with two Intel Xeon E5-2687W 3.10 processors, 32 cores and 64GB of RAM. We used one time step per frame, and sub steps for advection with a CFL number of one. All disk caching was disabled.*

Figure 1 and the accompanying video show several examples of Bifrost running adaptive liquid simulations in Autodesk Maya. While currently not every simulation benefits from adaptivity—the flow needs to be adequately deep for it to be worth it—we were able to achieve a factor of five speedup compared to Houdini 15's FLIP solver (Table 1) in our submarine test, for example. The Bifrost adaptive liquid solver is currently in use in production, comes with a C++ API for user extensions, and is freely available for academic use.

References

ANDO, R., THÜREY, N., AND WOJTAN, C. 2013. Highly adaptive liquid simulations on tetrahedral meshes. *ACM Trans. Graph. 32*, 4 (July), 103:1–103:10.

HONG, W., HOUSE, D. H., AND KEYSER, J. 2009. An Adaptive Sampling Approach to Incompressible Particle-Based Fluid. In *Theory and Practice of Computer Graphics*, The Eurographics Association UK, W. Tang and J. Collomosse, Eds.

LOSASSO, F., GIBOU, F., AND FEDKIW, R. 2004. Simulating water and smoke with an octree data structure. *ACM Trans. Graph. (Proc. SIGGRAPH) 23*, 457–462.

MUSETH, K. 2013. VDB: High-resolution sparse volumes with dynamic topology. *ACM Trans. Graph. 32*, 3 (July), 27:1–27:22.

SETALURI, R., AANJANEYA, M., BAUER, S., AND SIFAKIS, E. 2014. SPGrid: A sparse paged grid structure applied to adaptive smoke simulation. *ACM Trans. Graph. 33*, 6 (Nov.), 205:1–205:12.