

VISUAL SMOKE SIMULATION WITH ADAPTIVE OCTREE REFINEMENT

Lin Shi Yizhou Yu
Department of Computer Science
University of Illinois at Urbana-Champaign
{linshi,yyz}@uiuc.edu

ABSTRACT

Three dimensional fluid simulation becomes expensive on high resolution grids which can easily consume a large amount of physical memory. This paper presents an octree-based algorithm for visual simulation of smoke on ordinary workstations. This method adaptively subdivides the whole simulation volume into multiple subregions using an octree. Each leaf node in the octree also holds a uniform subgrid which is the basic unit for simulation. Because of the octree partition, the physical memory of the workstation only needs to be sufficiently large to hold a small number of subgrids with the majority of the subgrids stored on hard disks. A previous smoke simulation algorithm based on a semi-Lagrangian scheme has been adapted to this hybrid octree-based data structure. A pair of PullUp and Push-Down procedures are designed to solve the Poisson equation for pressure at each octree node. A novel node subdivision and merging scheme is also developed to dynamically adjust the octree during each iteration of the simulation so that regions containing more details are more likely to be subdivided to achieve better accuracy. The result is an algorithm that can solve smoke simulation on large grids using a limited amount of memory.

KEY WORDS

Euler Equations, Poisson Equation, Semi-Lagrangian Tracing, Subdivision, Merging

1 Introduction

Recently, there has been significant progress in graphical fluid simulation [9, 16, 8] which provides visually compelling approximate solutions to the Euler and Navier-Stokes equations. Visual quality instead of numerical precision is the primary goal of these simulations. They give impressive performance on either 2D grids or coarse 3D grids. However, since the memory requirement of a 3D grid increases dramatically at high resolutions, the single uniform grid adopted by these methods prevents them from using high-resolution grids on an ordinary workstation. The uniform grid is wasteful especially when the fluid volume only occupies a small portion of the grid, but has the potential to move from place to place.

This paper presents a method for adaptive mesh refinement based on octrees [15, 14]. This method subdivides the whole space into multiple subregions using an octree. Each leaf node in the octree in turn holds a uniform

subgrid which is the basic unit for simulation. Because of the octree partition, the physical memory of the workstation only needs to be sufficiently large to hold a small number of subgrids. As the simulation advances, additional subgrids can be loaded from the hard disk into the physical memory to replace existing subgrids. Because of the adaptive nature of the octree, regions with higher velocity or density variations can receive finer subdivision than others so that the amount of computation is distributed wisely among regions. The octree also evolves with time with nodes recursively split and merged from iteration to iteration to redistribute the amount of computation.

1.1 Related Work

This work has been influenced by previous works from both computer graphics and computational physics. In graphics, there has been steady progress on fluid simulation [10, 13, 5, 9, 16, 8]. Foster and Metaxas [9] recently used relatively coarse grids to produce nice smoke motion in three-dimensions. Their simulations are only stable if the time step is sufficiently small. To alleviate this problem and make the simulations more efficient, Stam introduced a model which is unconditionally stable and could be run at any speed [16]. This was achieved using a combination of a semi-Lagrangian advection scheme and implicit solvers. However, the simulations still suffered from too much numerical dissipation. Fedkiw *et. al.* [8] introduced vorticity confinement and a higher-order interpolation technique into visual simulation of smoke. As a result, the simulations can keep finer details on relatively coarse grids. Inviscid Euler equations instead of Navier-Stokes equations were used in their simulations.

In computational physics, there has been plenty of work using either multilevel grids or adaptive mesh refinement to improve the computational efficiency. Brandt [3] summarized multilevel computations in fluid dynamics. Berger, Olinger and Colella [2, 1] proposed a new adaptive meshing approach for a class of partial differential equations. This approach embeds nonaxis-aligned rectangular subgrids into a top-level coarse axis-aligned grid. These subgrids may evolve over time so new subgrids may appear and disappear. Quadtree or octree-based adaptive refinement have also been proposed in [12, 7, 4]. In these methods, cells are generated from an octree-based subdivision with smaller cells around critical regions. Compared with fluid simulation in graphics, these methods are con-

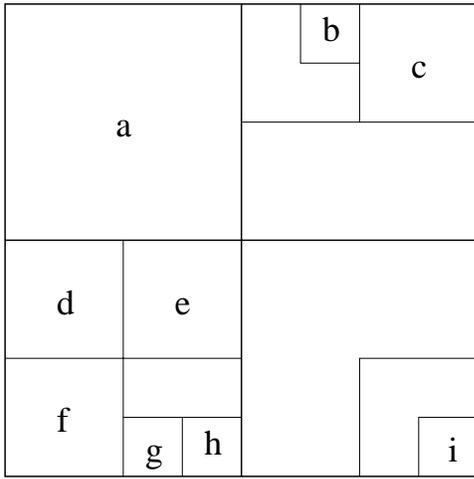


Figure 1. An example of 2D space partitioning using an incomplete quadtree. Note that a parent node may have between 1 and 4 children nodes. For example, leaf nodes b and i are the only children nodes of their parent nodes.

cerned about maintaining the precision of the simulation while improving the efficiency.

2 The Data Structure

A hybrid data structure incorporating both the advantages of octrees and uniform grids is designed in this section. The entire fluid volume is first adaptively subdivided using an octree so that more nodes are allocated to local regions whose associated physical quantities, such as velocity and density, have more details and variations. These kind of regions usually need a larger amount of computation to solve the details to a certain precision. However, the octree space partition is kept at a coarse scale and an octree node is further divided using an internal uniform grid.

One important feature of the octree is that it is *incomplete* in the sense that an internal node does not always have eight children. When the octree is constructed and a node needs subdivision, we use some criteria to check whether each of its octant needs a higher resolution grid, and only generate a child node when the corresponding octant satisfies the criteria. Thus, an internal node may have between one and eight children nodes. A uniform grid is set up at each leaf node as well as at each of those internal nodes with less than eight children nodes. This is because at least one of the octants of those internal nodes is not covered by any of its children's grids. The uniform grids at the octree nodes can have different sizes. For example, a distinct size can be used for nodes at a certain depth of the octree. However, for simplicity, we use the same grid size for all the nodes. Thus, the true resolutions of grids at two adjacent depths of the tree always differ by a factor of two. Fig. 1 shows an example of 2D space partitioning using the same scheme as in octree construction. The resulting incomplete

quadtree is shown in Fig. 2. Because the octree is incomplete, a relatively large number of internal nodes need to have their own uniform grids. Nevertheless, the amount of consumed memory is actually less than the amount that would have been consumed if we had built a complete octree since the branching factor is eight.

The constructed octree is by no means static. During a physical simulation, the nodes may be dynamically subdivided and merged according to some criteria. This is due to the fact that physical quantities may migrate from place to place, and details may be spawned and eliminated from time to time. At each step of a simulation, we perform a check on all nodes assigned with an internal grid. If any of them needs subdivision, all the generated children nodes are also checked for subdivision recursively. This entails criteria that can guarantee to terminate this process after a finite level of recursion. Such a criterion will be introduced in the next section. If an existing node has children and all of its children are leaf nodes, its children are checked for the possibility of being merged. If the criteria for merging are satisfied, the data saved at its children nodes are processed and merged into its own grid, and its children nodes are subsequently eliminated.

We can see that this hybrid data structure is actually a general scheme. When the octree only has a root node, this node must have a uniform grid. This situation is equivalent to a single uniform grid set up for the whole volume. When the uniform grids inside the octree nodes only have one voxel, we have a pure octree structure. However, the overhead involved with dynamic node subdivision and merging discourages a pure octree with every leaf node only covering one tiny voxel. Also, uniform grids are usually more convenient for physical simulation. Having reasonably sized uniform grids at the bottom level of the octree allows the application of simulation algorithms designed for uniform grids without much modification. Meanwhile, this can usually bring efficiency since uniform grids can often lead to sparse matrix systems that can be solved efficiently.

3 Smoke Simulation

The data structure proposed in the previous section is general, and is not restricted to any specific simulation algorithm. It is likely to be effective for a broad class of volumetric simulation algorithms. However, in this paper, we are specifically concerned with visual smoke simulation, and would like to apply this data structure to the specific algorithm introduced in [8].

3.1 Smoke Equations and a Basic Algorithm

Since we consider smoke as a collection of particles immersed in a gas-like fluid such as the atmosphere, we need to simulate the motion of the gas first. The smoke particles largely follows the velocity of the gas. We consider gases as inviscid and incompressible fluids whose velocity is governed by the Euler equations

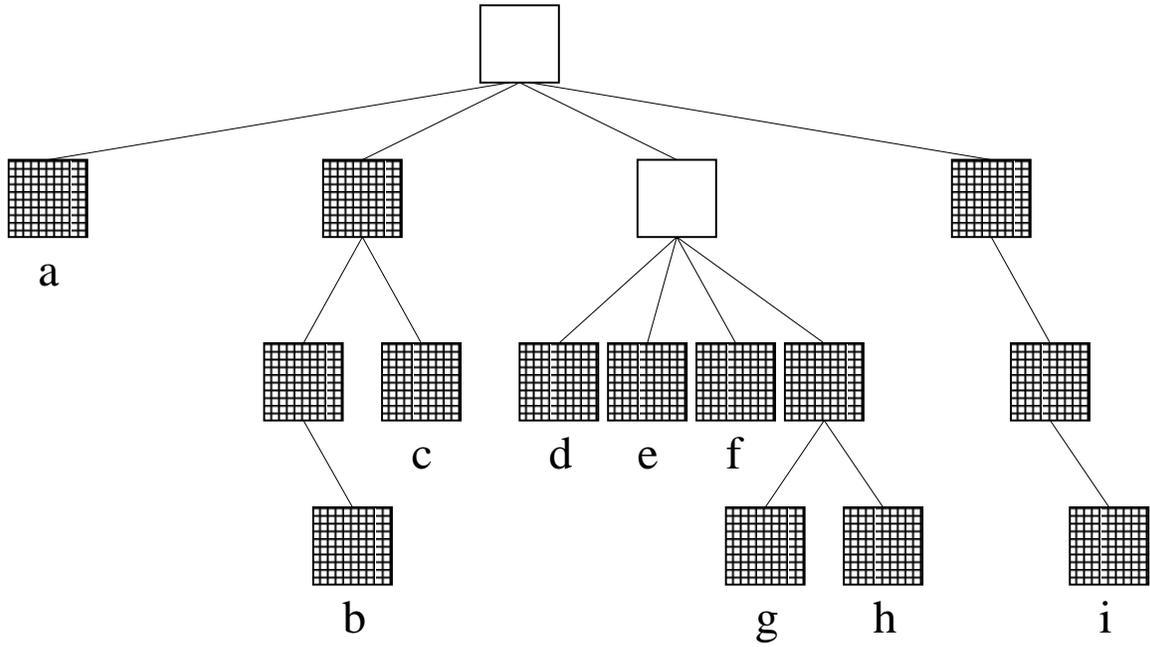


Figure 2. The quadtree corresponding to the 2D space partitioning shown in Fig. 1. Every filled node is assigned an internal uniform grid. Each leaf node is labeled with a letter, and is corresponding to the leaf node with the same letter in Fig. 1.

$$\nabla \cdot \mathbf{u} = 0 \quad (1)$$

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla) \mathbf{u} - \nabla p + \mathbf{f} \quad (2)$$

where Eq. (1) is for the conservation of mass and (2) is for the conservation of momentum. Since we are only concerned about the visual quality of smoke simulation instead of its numerical accuracy, the pressure gradient, ∇p , is only used for the reinforcement of incompressibility. A Poisson equation for the pressure is solved in [8] to obtain a divergenceless velocity field.

The temperature and density of the smoke are assumed to be controlled by the following equations

$$\frac{\partial T}{\partial t} = -\mathbf{u} \cdot \nabla T \quad (3)$$

$$\frac{\partial \rho}{\partial t} = -(\mathbf{u} \cdot \nabla) \rho + S_\rho \quad (4)$$

where the first terms on the right hand side indicate the advection of these scalar fields along the velocity field of the gas, and S_ρ is a source term with negative values for a sink.

To solve the equations (1) and (2), we follow the basic steps outlined in [8]: first add advection and force to the velocity field at the previous time step to obtain an intermediate velocity field $\{\mathbf{u}^*\}$ for the current time step by solving $\frac{\mathbf{u}^* - \mathbf{u}}{\Delta t} = -\mathbf{u} \cdot \nabla \mathbf{u} + \mathbf{f}$; then project the intermediate velocity field by solving the pressure using the Poisson equation $\nabla^2 p = \frac{1}{\Delta t} \nabla \cdot \mathbf{u}^*$ to enforce zero divergence [6]. The fi-

nal solution for the velocity field at the current time step is obtained from an adjustment to $\{\mathbf{u}^*\}$, $\mathbf{u} = \mathbf{u}^* - \Delta t \nabla p$.

The advection part is solved using a semi-Lagrangian scheme [17] instead of finite difference. That is, from the currently considered voxel, trace a path along the reversed velocity direction, and the path ends when the accumulated elapsed time reaches Δt . The velocity at the end of the path is transferred to the currently considered voxel. The path tracing part is carried out by further dividing the time step into subintervals and tracing the velocity field at these subintervals.

There are two typical boundary conditions for the Poisson equation, the Neumann boundary condition and Dirichlet boundary condition. The Neumann boundary condition imposes $\frac{\partial p}{\partial \mathbf{n}} = 0$ at a boundary point with normal \mathbf{n} . The Dirichlet boundary condition specifies the pressure at boundary points directly instead of its normal derivative.

To transport smoke density, the advection term of Eq. (4) uses the gas velocity field to move smoke. It is implemented by using the semi-Lagrangian scheme again. This time, it is the smoke density, instead of the velocity itself, that is being transferred.

3.2 The Octree-Based Algorithm

The basic idea of the octree-based algorithm follows a divide-and-conquer style and solves the fluid equations for each small uniform grid inside the octree nodes instead of for a uniform grid over the entire volume. The solution obtained from the octree-based algorithm has spatially varying resolution. Obviously, we cannot solve each octree node independently since one of the most important fea-

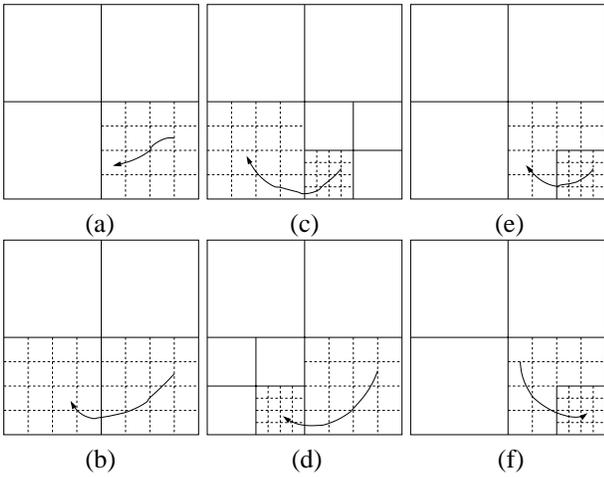


Figure 3. Six possible cases for semi-Lagrangian path tracing in an octree. Solid lines represent octree partitions. Dashed lines represent the uniform grids inside octree nodes. The voxel at the head of the path is called the source, and the voxel at the tail of the path is called the destination since we trace from the destination to the source and the velocity at the source is transferred to the destination. (a) Both the source and destination voxels belong to the grid of the same node. (b) The source and destination voxels belong to two different nodes with the same resolution. (c) The resolution of the destination node is higher than that of the source node. (d) The resolution of the destination node is lower than that of the source node. (e) The destination node is a child of the source node. (f) The destination node is a parent of the source node.

tures of fluids is dynamic feedback. From the above basic algorithm for smoke simulation, we can see that the two most important steps are tracing semi-Lagrangian paths and solving the Poisson equation. Both steps require interactions among different octree nodes. Their adaptations to the octree structure are presented below.

The semi-Lagrangian scheme does not need changes as long as the tracing is kept inside an octree node. However, when its path intersects with one of the six bounding faces of the node, we need to find the appropriate neighboring node where the path can continue. First of all, such a node must have its own internal grid; otherwise, tracing cannot continue. Second, there may be more than one neighboring nodes with an internal grid. They are at different levels of the octree. Among them, we need to pick the node with highest resolution to achieve best accuracy. Since we have an incomplete octree, sometimes we may not be able to find a neighboring node (Fig. 3(e)). Instead, there exists a node containing the current node at a coarser resolution. Then, we have to exit the current high-resolution node and keep tracing at the coarser resolution. Fig. 3 enumerates all possible situations during this path tracing process.

We solve the Poisson equation for pressure hierarchi-

cally since it needs boundary conditions at the bounding faces of each node. Note that directly imposing the Neumann boundary condition or the Dirichlet boundary condition at the bounding faces of an octree node is inappropriate because most likely these bounding faces lie inside the fluid volume and the pressures at these locations are unknowns that need to be solved. To carry out our hierarchical method, we designed a pair of PullUp and PushDown procedures. During these two procedures, a *temporary velocity grid* is set up in those nodes without a uniform grid, such as the two empty nodes shown in Fig. 2. In the PullUp procedure, we move from the leaves to the root of the octree and generate average velocity fields at coarse resolutions on the way. The velocity at a voxel in a temporary grid is the average of the velocities at the eight corresponding voxels in one of its children nodes. Once we finish filling all of the temporary velocity grids, the PushDown procedure is executed to solve the Poisson equation. We start from the temporary velocity grid at the root. Solve the Poisson equation on this grid and obtain a pressure distribution. This pressure distribution is then interpolated at the bounding faces of the children nodes to allow the specification of the Dirichlet boundary conditions there. Then we repeat this process recursively at the children nodes. The Dirichlet boundary conditions for the Poisson equation at a node are always interpolated from the pressure distribution calculated at the coarser level except at the root where we can either apply the original Neumann boundary condition or the Dirichlet boundary condition because the boundaries of the root node is actually the boundaries of the entire simulation volume. Once the pressure term is obtained for all the leaf nodes, its gradient is subtracted from the velocity to finish the projection step as mentioned in Section 3.1.

There is a tradeoff on the resolution of the temporary velocity grids. If it is too high, solving the Poisson equation at top levels of the octree is still expensive and we lose the advantages of having an octree. If it is too low, the interpolated boundary conditions for the children nodes may not be sufficiently accurate. However, in practice, we have not observed artifacts by setting this resolution the same as the resolution of the uniform grids inside leaf nodes. This is probably because the pressure distributions at coarse levels are never used for solving velocity fields directly, but only used for generating the boundary conditions at finer resolutions.

3.2.1 Subdivision and Merging

As mentioned in Section 2, we dynamically subdivide and merge octree nodes during each iteration of the simulation. We would like to subdivide a node where the amount of detail and variation becomes excessive, and merge nodes otherwise. Node subdivision is performed recursively. That means newly generated nodes are checked for subdivision immediately. Such a subdivision scheme requires a criterion that can eventually terminate the recursion.

For smoke, we are most concerned with the variations in smoke density. Following the notation in Eq. (4),

we denote smoke density as a three dimensional function $\rho(x, y, z)$. Define $\nabla_x \rho = \rho(x + 1, y, z) + \rho(x - 1, y, z) - 2\rho(x, y, z)$ which is a finite difference of $\frac{\partial^2 \rho}{\partial x^2}$. $\nabla_y \rho$ and $\nabla_z \rho$ can be defined likewise. Then our criterion for subdivision can be written as

$$C(x, y, z) = \max(|\nabla_x \rho|, |\nabla_y \rho|, |\nabla_z \rho|) > T \quad (5)$$

where T is a specified threshold. Once there exists a voxel (x_0, y_0, z_0) in the grid of an octree node such that $C(x_0, y_0, z_0) > T$, the octree node should be subdivided. The velocity field as well as the density distribution of any newly generated child node is obtained by trilinear interpolation which means three successive linear interpolations along the three major axes. It is straightforward to prove that after trilinear interpolation,

$$\max_{(x,y,z) \in n'} C(x, y, z) \leq 0.5 \max_{(x,y,z) \in n} C(x, y, z) \quad (6)$$

where n and n' are octree nodes and n' is a descendant of n . This property guarantees that the recursive subdivision process terminates eventually as long as $T > 0$. Once a node needs subdivision, only those octants that contain voxels satisfying the criterion in Eq. (5) need to generate children nodes because we build an incomplete octree.

On the other hand, if a node has children and they do not have sufficient details any more, they can be merged and removed. The preconditions for merging include i) none of the children nodes is newly generated in the same iteration; ii) none needs subdivision; and iii) all of them should be leaf nodes. To further check whether the children nodes should be merged, a new density function is computed for the considered node by subsampling the smoke density distributions of its children nodes. If the maximum difference between the subsampled density and the original densities at the children nodes is smaller than a threshold, the children nodes should be merged and removed.

Density-based subdivision and merging serve as the cleanup step at the end of each iteration of the smoke simulation. However, subdivision may happen at a different time when the semi-Lagrangian tracing is performed. The traced path starts from a destination voxel which needs an updated velocity vector, and ends at a source voxel whose velocity is going to be transferred to the destination voxel. The resolution of the source voxel may be higher than that of the destination voxel (Fig. 3(d)&(f)). That indicates the high resolution detail at the source voxel might be lost if the destination voxel were not subdivided. Therefore, we decide to recursively subdivide the node where the destination voxel resides until it reaches the same resolution as the node of the source voxel. Of course, at the end of the iteration, these new nodes are checked for merging to prevent over-subdivision.

3.3 Implementation

Our basic implementation for solving the fluid equations follows the one introduced in [8]: dice the work space into

discretized voxels, and define scalar quantities at the center of the voxels and vectors on the voxel surfaces. The velocity at any point inside a voxel can be obtained by linearly interpolating each component of the velocity vector separately.

We also implemented both open and closed boundary conditions for 2D and 3D grids. The closed boundary condition sets the velocity perpendicular to the boundary to be zero while the open boundary condition assumes the velocity fields inside and outside the grid are continuous at the boundary.

Advection is done using the semi-Lagrangian method, which needs to compute a trace matrix. Pixels are traced back along a path with an elapsed time equal to the simulation time step in the velocity field. And the new value of the quantity (scalar field, vector field) is assigned the value at the position traced to. But once it is out of the boundary, it is assigned a value consistent with the chosen boundary condition.

The projection step involves solving the Poisson equation, which results in a linear equation with a sparse matrix. Conjugate gradient method is the natural way of solving this problem. Most of the time, we use a solver from the FISHPAK library since it is easy to incorporate and extremely efficient [18]. We also built our own solver by calling a sparse linear solver from LAPACK [11] which also does a good job on this problem.

If there are objects present in the fluid field, the objects need to be discretized on the simulation grids as in [8]. Basically, if a voxel is occupied by an object, it needs to be labeled as occupied and semi-Lagrangian paths should be clipped against these occupied voxels. The subdivision of the octree also needs to be adjusted to represent the objects better. Octree nodes should be divided into a finer scale if they are partially occupied by any of the objects. And the generated children nodes are not allowed to merge as long as the objects remain there. Except for this extra step in subdivision, the remainder of our algorithm should be kept the same.

4 Results

	# nodes	Advection	Projection	Total
16^3 grid	455	10.39	4.20	33.23
32^3 grid	75	13.30	4.87	37.05
64^3 grid	31	42.61	14.42	136.92
Uniform	1	10.43	5.49	36.82

Table 1. A comparison of the average running time(seconds) per frame between a uniform $128 \times 128 \times 128$ grid and the same three octrees used in Fig. 5. The memory usage by the uniform grid is approximately the same as the average memory consumption of the octree with grid size $32 \times 32 \times 32$. This comparison also includes the individual timing on the advection part and on the projection step.

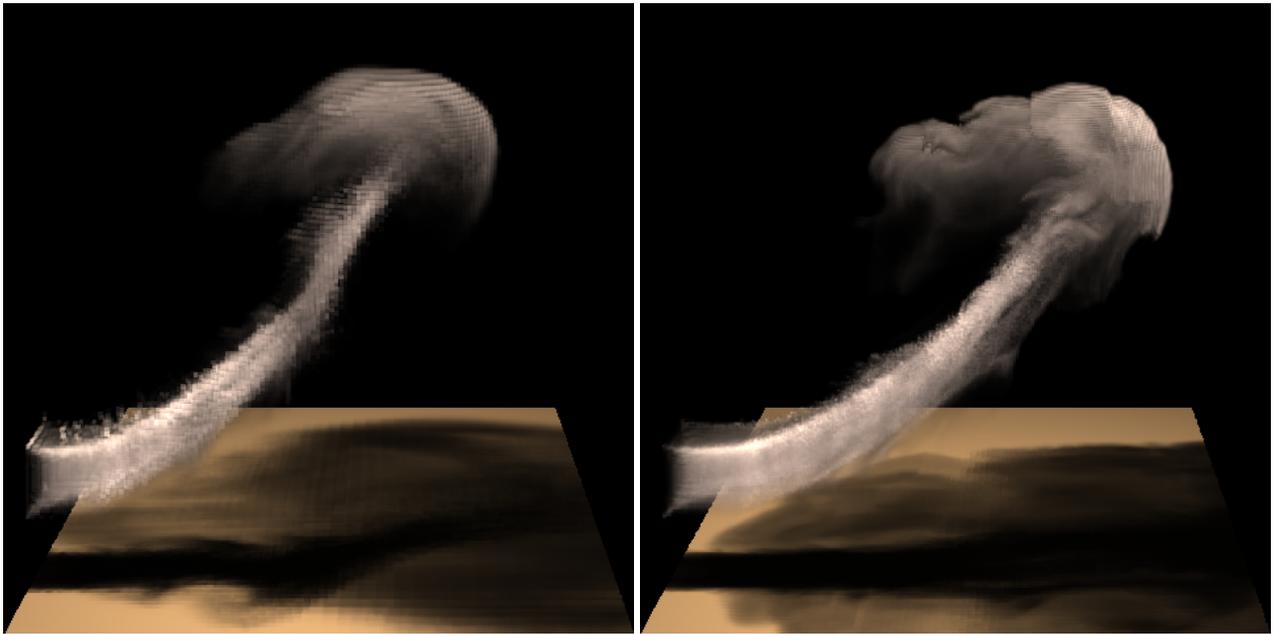


Figure 4. The left image is a frame from a simulation on a $128 \times 128 \times 128$ uniform grid. The right image is a frame from an octree-based simulation with approximately the same amount of memory. The octree does adaptive grid refinement around the smoke, therefore has less artifacts.

We have successfully implemented and tested the octree-based algorithm. We have run most of the simulations on a Pentium 4 1.7GHz processor with 1GB physical memory. The maximum uniform grid we can solve on this machine is around $128 \times 128 \times 128$. Obviously, we would need approximately 8GB physical memory to simulate a uniform $256 \times 256 \times 256$ grid. This amount of memory is already beyond the reach of ordinary workstations although the grid resolution is still not impressive. However, with the help of our adaptive refinement scheme, we were able to generate high-resolution octree nodes around critical regions. Some of the leaf nodes generated during the simulation reach a resolution higher than an equivalent $256 \times 256 \times 256$ grid. This kind of performance is achieved under the restriction that all the nodes are held in the memory. Otherwise, we can easily have unlimited resolution by storing most of the nodes on hard disks and only load into memory a few nodes related to the currently considered one. Although frequent disk access can impair the performance, we still expect that the octree-based algorithm has better data locality than an algorithm solving uniform grids.

We have tested examples of smoke simulation using octrees with different internal grid sizes from $16 \times 16 \times 16$ to $64 \times 64 \times 64$. The height of the octrees is adjusted so that they all have the same maximum resolution which is equivalent to $256 \times 256 \times 256$. Fig. 5 shows the number of nodes at each frame during a typical 100 frame simulation. At the beginning of the simulations, the octree is only subdivided around the smoke source, resulting in a very small number of nodes. This number increases as the smoke propagates, and it eventually stabilizes. The octree with grid

size $16 \times 16 \times 16$ allocated the most number of nodes, but consumes the least amount of total memory. This is expected because with a smaller sized leaf node, this octree does better adaptive node refinement around important regions. Tab. 1 gives a comparison of the running time per frame between a uniform $128 \times 128 \times 128$ grid and the three different octrees used in Fig. 5. Our octree-based algorithm is expected to run slower than the uniform grid since it needs time to manage the nodes. Again, the octree with grid size $16 \times 16 \times 16$ gives best performance, and even runs a little bit faster than the uniform grid despite all the overhead involved with the PullUp and PushDown procedures and for the subdivision and merging of a relatively large number of nodes.

We also produced a comparison between two video sequences simulated by a uniform $128 \times 128 \times 128$ grid and an octree with grid size $32 \times 32 \times 32$. Overall, the octree consumes the same amount of memory. The images are rendered using a simple volume ray tracer which does not smooth the final results in order to show the quality difference generated from a uniform grid and the octree-based adaptive refinement.

5 Conclusions

This paper presented an octree-based algorithm for visual simulation of smoke on ordinary workstations. This method adaptively subdivides the whole simulation volume into multiple subregions using an octree. Each leaf node in the octree also holds a uniform subgrid which is the basic unit for simulation. A previous smoke simulation algo-

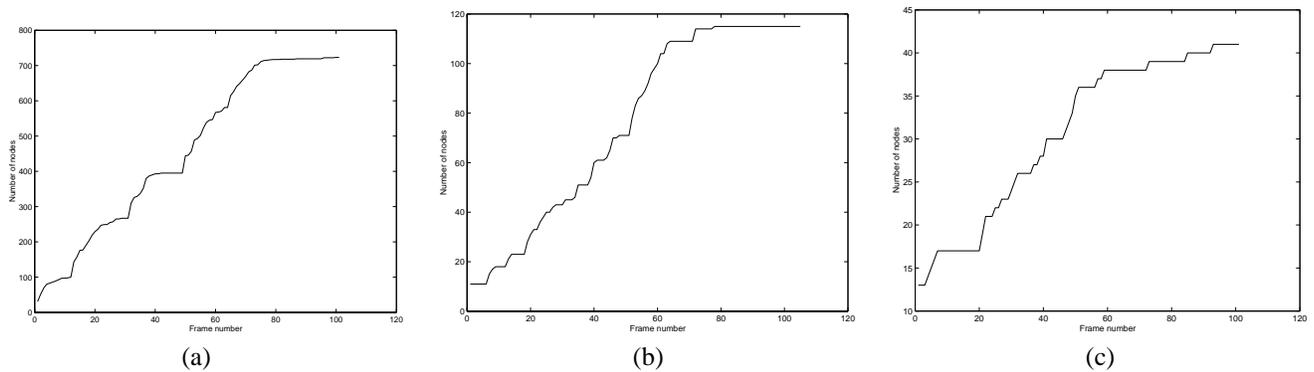


Figure 5. Statistics for a 100 frame simulation. Octrees with the same maximum resolution but different maximum height and grid sizes were tested. The first octree has a grid size $16 \times 16 \times 16$ and a maximum height of 4. The second octree has a grid size $32 \times 32 \times 32$ and a maximum height of 3. The third octree has a grid size $64 \times 64 \times 64$ and a maximum height of 2. (a) The number of allocated nodes in the first octree at each frame; (b) The number of allocated nodes in the second octree at each frame; (c) The number of allocated nodes in the third octree at each frame;

rithm has been adapted to this octree-based data structure. A novel node subdivision and merging scheme is developed to dynamically adjust the octree during each iteration of the simulation. Because of the octree partition, the physical memory of the workstation only needs to be large enough to hold a small number of subgrids with the majority of the subgrids stored on hard disks.

Acknowledgements

This work was supported by National Science Foundation CCR-0132970. We thank the reviewers for their valuable comments.

References

- [1] M.J. Berger and P. Colella. Local adaptive mesh refinement for shock hydrodynamics. *Computational Physics*, 82:64, 1989.
- [2] M.J. Berger and J. Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *Computational Physics*, 53:484–512, 1984.
- [3] A. Brandt. Multilevel adaptive computations in fluid dynamics. *AIAA*, pages 1165–1172, 1980.
- [4] E.F. Charlton and K.G. Powell. An octree solution to conservation-law over arbitrary regions(oscar). In *AIAA 97-0198*, 1997.
- [5] J.X. Chen, N. Lobo, C.E. Hughes, and J.M. Moshell. Real-time fluid simulation in a dynamic virtual environment. *IEEE Computer Graphics and Applications*, pages 52–61, 1997.
- [6] A. Chorin. A numerical method for solving incompressible viscous flow problems. *Computational Physics*, 2:12–26, 1967.
- [7] W.J. Coirier. *An Adaptively-Refined, Cartesian, Cell-Based Scheme for the Euler and Navier-Stokes Equations*. PhD thesis, The University of Michigan, 1994.
- [8] R. Fedkiw, J. Stam, and H.W. Jensen. Visual simulation of smoke. In *SIGGRAPH 01 Conference Proceedings*, pages 15–22, 2001.
- [9] N. Foster and D. Metaxas. Modeling the motion of a hot, turbulent gas. In *SIGGRAPH 97 Conference Proceedings*, pages 181–188, 1997.
- [10] M. Kass and G. Miller. Rapid, stable fluid dynamics for computer graphics. In *Proceedings of SIGGRAPH*, pages 49–57, 1990.
- [11] Lapack (linear algebra package). www.netlib.org.
- [12] J.E. Melton, F.Y. Enomoto, and M.J. Berger. 3d automatic cartesian grid generation for euler flows. In *AIAA 11th Computational Fluid Dynamics Proceedings*, 1993.
- [13] J. O’Brien and J. Hodgins. Dynamic simulation of splashing fluids. In *SIGGRAPH 95 Conference Proceedings*, pages 198–205, 1995.
- [14] H. Samet. *Applications of Spatial Data Structures*. Addison-Wesley, Reading, MA, 1990.
- [15] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, MA, 1990.
- [16] J. Stam. Stable fluids. In *SIGGRAPH 99 Conference Proceedings*, pages 121–128, 1999.
- [17] A. Staniforth and J. Cote. Semi-lagrangian integration schemes for atmospheric models: A review. *Monthly Weather Review*, 119:2206–2223, 1991.
- [18] P.N. Swartztrauber and R.A. Sweet. Efficient fortran subprograms for the solution of separable elliptic partial differential equations. *ACM Trans. Mathematical Software*, 5(3):352–364, 1979.