Eurographics/ACM SIGGRAPH Symposium on Computer Animation (2005)
K. Anjyo, P. Faloutsos (Editors)

Modeling and Animating Gases with Simulation Features

Joshua Schpok[†] William Dwyer[‡] and David S. Ebert[§]

Purdue University

Abstract

In modeling natural phenomena, artists often compromise the benefits of direct control for the visual realism of physics-based simulation. For gases, Eulerian simulations traditionally provide realistic results, but a poor representation for artistically shaping the media. In our system, users work with a more intuitive set of continuously extracted features whose manipulation feeds back into the original simulation. This novel approach overcomes common control issues by providing modeling tools to manipulate high-level behavior in Eulerian simulations. We employ techniques in feature extraction, real-time gas simulation, and volume rendering to build an interactive system to sculpt three-dimensional flows.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Three-Dimensional Graphics and Realism]: Animation of Natural Phenomena;

1. Introduction

Since humans identify separate, discrete components in natural processes, waves in water, plumes in hot gas, flames in fire, and similar structures can be used as intuitive modeling and animation primitives. However, if not skillfully constructed, the collective result may appear unrealistic because these structures are the products of complex local and global physical processes. On the other hand, physically-based simulations achieve realistic results by enforcing feasibility. Rather than directly modeling the features, one controls the conditions producing them, which requires a more scientific perspective of the phenomena (and the simulation algorithm). Given the chaotic nature of gases, the results may be difficult to control and inconsistent over different resolutions and time-steps.

Therefore, there is a trade-off between direct modeling techniques, which may not capture nor convey the nuances of amorphous phenomena, and physical simulations, which can be difficult to control. In visual effects, the goal is a compelling result regardless of physical accuracy. Here, we present a system for combining physics-based simulation and derived high-level flow features to achieve visual realism without forfeiting all control to a dynamics-based solution. Our priority is visual realism and interactivity through an intuitive mouse-driven interface. To achieve this goal, we present the Eulerian simulation as a set of high-level features using interactive feature extraction techniques. In this system, vortices, uniform flows, divergences, and densities can be adjusted within the volume through ellipsoidal flow feature widgets. When the user adjusts the parameters through the widget, the changes are sampled back into the simulation grid, providing direct interaction with larger-scale flow behavior derived from the physics-based simulation. This lets users work exclusively with high-level features, whose behavior is driven by the simulation.

The key contribution of this approach is the interactive, direct manipulation of naturally occurring behavior detected in Eularian simulations. The strong correlation between the features and the simulation enables the animator to make focused adjustments. Our system continuously extracts features, allows the user to adjust the flow animation using these features, and applies the resulting feature changes back into the simulation. The algorithm performs fast enough to run with an interactive Navier-Stokes simulation and our results are rendered in real-time with hardware-accelerated advected texturing. Our results convey realistic,

[†] e-mail: joshua@schpok.net
[‡] e-mail:wdwyer@stanford.edu
[§] e-mail:ebertd@purdue.edu

high-resolution simulations created with natural, adjustable, high-level primitives.

2. Previous Work

As gas and water simulation in computer graphics run faster and appear more realistic, more attention has been focused on controlling fluid flows. Typically, control is leveraged by modifying the dynamical properties of the media [FM97, REN*04], or adding control mechanisms to combine and tweak existing simulations [Wit99]. These methods, however, rely on users being familiar with the dynamics of fluid flow. Keyframing densities provides a more conventional paradigm through optimizing the driving forces and calculating the inter-frame derivatives [TMPS03]. New solutions employ the adjoint method [MTPS04], or avoid optimization using coarser closed-form approximations [FL04].

Rather than specifying density field targets, we look at more directly modifying the simulation. We use ellipsoidal primitives to approximate the visible density and individual *flow primitives* [WH91, WFG94] defining flow behavior in their vicinity. A key contribution of our approach is that these features are not just synthetically created by the user, but driven and extracted from an Eulerian simulation. Previous work has explored modifying the density approximation [PCS04, REN*04], but here we present densities and velocities as separate extracted primitives.

Feature detection and extraction is an established field in computational simulation [PVH*03]. Here we develop coarse approximations motivated by the broad previous work in vortex extraction [JMT03], flow topology [TLHD03], and procedural modeling [EMP*02]. With mathematical characterizations of certain flow phenomena, we can implement algorithms to detect and synthesize high-level features useful in flow modeling.

3. Approach

The goal of our system is to efficiently manipulate high-level flow structures in an interactive Navier-Stokes simulation. Our approach finds these features in the simulation and permits the user to work with the individual regions in which they occur. An overview of the processes composing the system is shown in Figure 1. For every frame, features are extracted from a simulation and presented as adjustable widgets. Users can interactively "adjust" features while the simulation runs by applying elementary transformations to these feature subvolumes.

At every simulation step, we detect relevant density and flow features and overlay their spatial boundaries on the simulation. Figure 2 is a screen capture of our system in such a session. The features are represented with ellipsoidal regions, which can be reorientated within the simulation domain. For

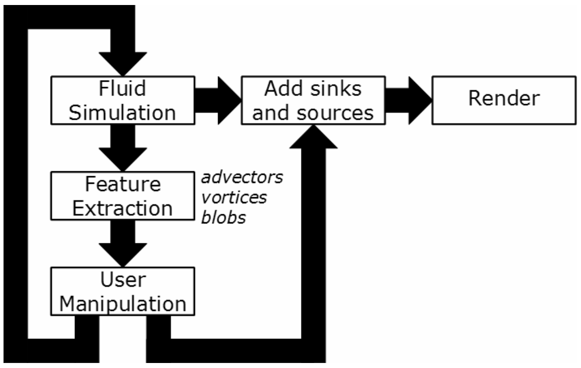
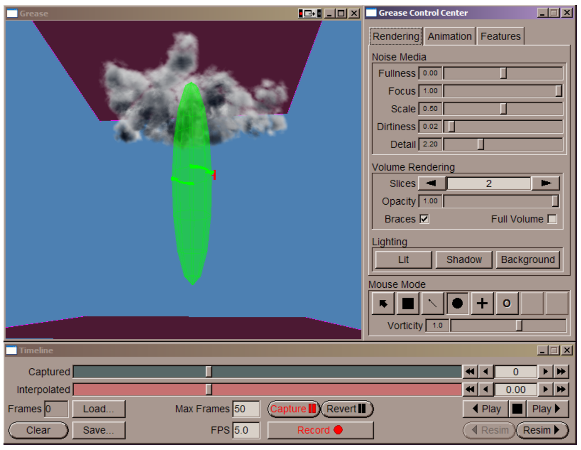


Figure 1: *Outline of task processes per frame in our system. Note divergent features are not a part of the simulation loop as they do no occur in our mass-preserving simulation.*



Fullness 0.00
Focus 1.00
Scale 0.50
Dirtiness 0.02
Detail 2.20
Slices 2
Opacity 1.00
Braces ☑ Full Volume ☐
Vorticity 1.0
Captured 0
Interpolated 0.00
Frames 0
Max Frames 50
FPS 5.0

Figure 2: *In this gas design session, a long vertical vortex is represented with the green ellipsoid. This simulation generated the cyclone in Figure 7.*

flow features, velocities can also be rescaled to vary the magnitude or reverse the flow. Since the system extracts these features at every frame, we do not store a history of features (except for sinks and sources) and rely on their being satisfactorily derived through the extraction process.

These manipulations reflect back onto the simulation grid as a spatial transformation on the ellipsoidal subvolume. Users can then reposition, scale, and rotate the feature's spatial area. This process is outlined in Figure 3. The feature's subvolume is subtracted from the simulation grid, appropriately transformed, and written back into the volume, similar to cutting and pasting regions on a raster grid. Density features only affect the density field and velocity features only affect the velocity field. To fill the space vacated by the feature, densities are set to zero, and velocities are approximated by averaging the border voxels' velocities.

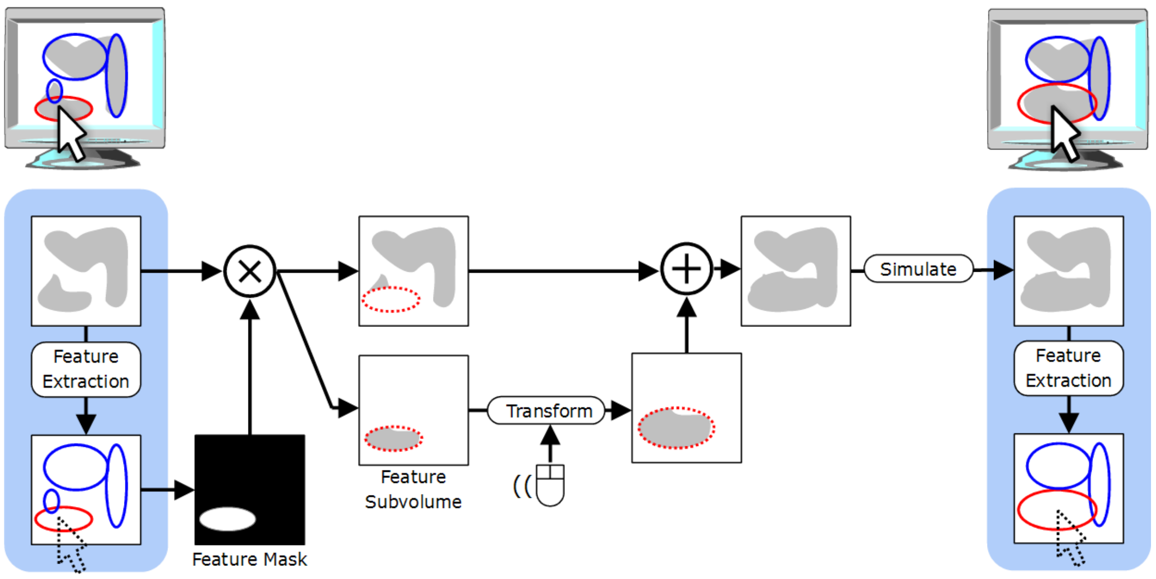


Figure 3: *One frame of the interactive manipulation process*

This editing process can leave coarse compositing artifacts. Committing small manipulations over many frames can minimize these errors, but for large changes, the feature's boundary values may be coarsely inconsistent with the surrounding simulation. To smooth the transition around the feature subvolume, we "feather" its boundaries when writing it into the simulation grid. While this process preserves visual continuity, we must ensure mass conservation by using a divergence-free projection before the subsequent simulation step.

Our physical simulation is based on a stable fluids solution [Sta99]. Our simulation resolution in this paper's examples are 32x32x32 to ensure interactive performance, though our techniques easily scale to higher resolutions. We permit the user to interact with the simulation through the widgets in the following ways:

- **Building Initial Conditions:** Features are a high-level primitive to describe the initial forces driving a simulation. In addition to constructing the environment by painting velocity and density, users can write in new procedurally-based representations of our features.
- **Real-time Manipulation:** While the simulation runs, features can be guided by dragging them with a mouse. Since these changes occur in real-time, the user gets continuous feedback through the simulation.
- **Revising Simulations:** A user can revisit the frames of a simulation to fine-tune the flow. By selecting a certain frame, densities and velocities can be adjusted in a way similar to building initial conditions. When a change is made somewhere along the timeline, the change can be applied as an adjustment to the frame at that point and, if desired, overwrite the subsequent frames with a simulation solution from that frame onwards.

In setting up and revising recorded simulation frames, the simulation process is paused to show the single frame of interest. When the density is not advected, it may be unclear how the new flow is pushing the density. Our implementation of texture advection [Ney03] conveys the direction and rate of flow, which is instrumental in previewing the adjusted flow.

4. Simulation Features

We present an ellipsoidal decomposition of the density field and three primitive velocity features to shape flow: uniform advections, vortices, and divergences. Each of these features can be created, removed, and modified in the interactive environment, but are not stored in a history of frames. Instead, an extraction algorithm is used to segment and quantify features from the simulation at every frame. To create a new feature defined on the grid, we also require a procedural model to generate the voxel values. Both these methods should be computationally simple enough to execute in real-time, and therefore pursue coarser approximations.

There may be inaccuracies in our extraction arising as incorrectly classified regions, features that go unclassified, and inconsistencies between the procedural formulation and the resulting extraction. To address mischaracterizations, a threshold term, α, is used to calibrate the extraction sensitivity. Occasionally, newly created features may not match their extracted representation. For example, they may merge with a

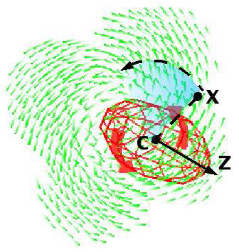


Figure 4: *A vortex parameterization (red) of a flow field (green) using the feature extraction algorithm developed in Section 4.1.1, with center* **c** *and axis* **z**.

neighboring feature with similar properties. Differences in perceived and detected features is a limitation of the automatic extraction approach, but may be resolvable with more accurate methods.

4.1. Vortices

Vortices are easily recognized by human observers, but lack a complete, formal physical definition. We use the definition of a vortex a "multitude of material particles rotating around a common center" [Lug83]. This is typically quantified as a velocity Jacobian with complex eigenvalues. This characterization motivates our vorticity-based approach to vortex extraction and synthesis.

4.1.1. Extraction

Our vortex extraction method finds regions with high vorticity and fits ellipsoids around them (see Figure 4). The vorticity magnitude at a given point is calculated as the magnitude of the curl of the neighborhood's velocity field [FSJ01]. We cluster neighboring voxels with high vorticity and consider each cluster a separate vortex. To perform this operation quickly, we initially seed a vortex at each voxel with a vorticity magnitude above some threshold, α, then merge neighboring vortices satisfying the following criteria:

$$\begin{aligned} \mathbf{z}_1 \bullet \mathbf{z}_2 &\geq \frac{\alpha}{r_{0_1} + r_{0_2}} && (1) \\ |\mathbf{z}_1 \times (\mathbf{c}_1 - \mathbf{c}_2)| &\leq \alpha && (2) \\ |\mathbf{z}_2 \times (\mathbf{c}_1 - \mathbf{c}_2)| &\leq \alpha \end{aligned}$$

α is the user adjustable calibration term, which we normally set to a value near 2.0 in our applications. Here $\mathbf{c}_i$ is the origin of vortex i, r_{0_i} its vortex radius in voxels, and $\mathbf{z}_i$ is a vector extending along the vortex's rotational axis. These criteria restrict merging vortices to those whose rotational axes are similar within a tolerance (Equation 1) and are close along this axis (Equation 2).

Note that as vortices grow, the merging criteria become more restrictive. Though vortex cores are modeled as straight segments in our ellipsoidal model, this axis, in reality, is not necessarily straight [JMT02]. For instance, we see closed vortex core rings in our examples that induce plumes. Here, we must subdivide the curved axis into a series of straight segments. This restriction prevents merging more vortices deviating from the prevailing axis.

If a pair of vortices (labeled 1 and 2) satisfy the merge criteria, they are replaced with a new vortex (labeled 0). Its vorticity v_0 and rotational axis $\mathbf{z_0}$ are a radius-weighted interpolation of the original two.

$$\begin{aligned} t &= \frac{r_{0_2}}{r_{0_1} + r_{0_2}} \\ \mathbf{z}_0 &= \frac{\mathbf{z}_1 + t(\mathbf{z}_2 - \mathbf{z}_1)}{|\mathbf{z}_1 + t(\mathbf{z}_2 - \mathbf{z}_1)|} && (3) \\ v_0 &= v_1 + t(v_2 - v_1) && (4) \end{aligned}$$

The resulting radius r_{0_0} and axis length $|\mathbf{z}_0|$ is scaled to the spatial extents of the the source vortices' union.

Classifying regions purely with a vorticity measure has been shown to incorrectly classify non-vortex regions [JH95], which has motivated more complex tensor decomposition and stream-line tracing schemes [SP99, SB94] and a wealth of vortex extraction research. While more accurate methods of vortex extraction could be used, we use this more efficient vortex parameterization, which we can compute at every iteration. Therefore, our technique will incorrectly classify some vortices, but these occur infrequently and are small enough to be dismissed by the user.

4.1.2. Synthesis

To write a new vortex feature into the velocity field, we update each point $\mathbf{x}$ within the vortex's radius r_0. A velocity is directed along the cross product of the direction to the point and the vortex's rotation axis $\mathbf{z}$. The velocity is scaled to the user defined magnitude v, linearly rolling off with distance. We use the following function to calculate this velocity, where $\mathbf{r} = \mathbf{x} - \mathbf{c}$.

$$\mathbf{f}(\mathbf{x}) = \begin{cases} \left(1 - \frac{\|\mathbf{r}\|}{r_0}\right) \frac{\mathbf{r}\times\mathbf{z}}{\|\mathbf{r}\times\mathbf{z}\|} v & \|\mathbf{r}\| \leq r_0 \\ 0 & \|\mathbf{r}\| > r_0 \end{cases} \qquad (5)$$

To add a vortex to the simulation, we add $\mathbf{f}(\mathbf{x})$ to all voxels $\mathbf{x}$ in the volume. We use this vortex model to create cyclonic funnels in Section 6.1, scaling rolling motion in convective plumes, and to add additional swirling behavior in vapors.

4.2. Uniform Advection

Regions of uniform advection are useful for adjusting prevailing winds, steady jets, and streams. Typically, directly painting densities into a simulation produces a plume which may not advect density as uniformly as desired. With uniform advection features, users have more control of the velocity uniformity within its bounds. These primitives provide

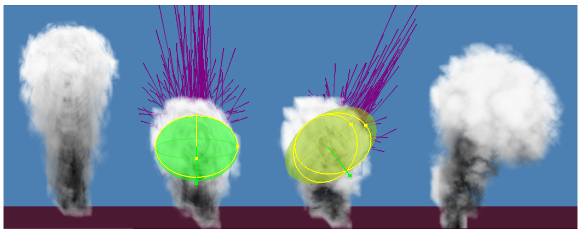


Figure 5: *Altering a jet with an extracted uniform advection feature. The first image shows the result of a buoyant plume, whose primary velocities from an intermediate frame are extracted and visualized in the second image. These velocities are redirected in the third image, and the fourth image shows the altered simulation result.*

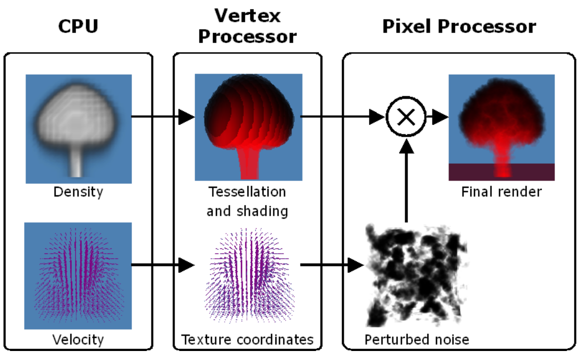


Figure 6: *Our system distributes processes across the available computational units.*

a more direct way to steer density because we are directly adjusting the direction and magnitude of an area.

4.2.1. Extraction

The method to extract regions of uniform advection is similar vortices in Section 4.1.1. We initially seed uniform advection features at all voxels with a velocity over some adjustable threshold. We use Equation 1 as a merge criteria between neighboring features, but now with $\mathbf{z}$ as the velocity direction. Equation 2 is ignored to permit expansion in all directions. The new combined feature is sized and interpolated by Equations 3 and 4 in the same way, with v now representing velocity magnitude.

4.2.2. Synthesis

To create an area of uniform advection, the area is blended with the advection vector $\mathbf{v}$ using distance-based roll-off:

$$\mathbf{f}(\mathbf{x}) = \begin{cases} \left(1 - \frac{\|\mathbf{r}\|}{r_0}\right)\mathbf{v} & \|\mathbf{r}\| \leq r_0 \\ 0 & \|\mathbf{r}\| > r_0 \end{cases} \tag{6}$$

In Figure 5, an intermediate step of a rising plume is redirected and scaled down to alter the final solution. Since the extracted feature encapsulates a neighborhood of consistent velocities, the adjustments holistically changes the rising motion, enabling a higher level of manipulation.

4.3. Sinks and Sources

Sinks and sources are useful for explosions, whirlpools, and tornadoes. However, their divergent behavior does not occur in ideal (non-compressible) fluids. In fact, this behavior is canceled in our simulation by the divergence-free transform. While there are elegant ways to implement divergent behavior [FOA03], we subtract divergent forces before the Navier-Stokes simulation, but add them back in after the simulation step. This permits these forces to persist through rendering and manipulation, but they are intentionally removed from the simulation process. As a result, divergent forces do not evolve or advect densities, but their force is still conveyed through advected textures. The force of a source or sink is defined by the following equation:

$$\mathbf{f}(\mathbf{x}) = \begin{cases} \left(1 - \frac{\|\mathbf{r}\|}{r_0}\right)\frac{\mathbf{r}}{\|\mathbf{r}\|}v & \|\mathbf{r}\| \leq r_0 \\ 0 & \|\mathbf{r}\| > r_0 \end{cases} \tag{7}$$

where positive values of v define sources and negative values are sinks. This feature is confined to a user defined area and, with our approach, does not evolve through the simulation. Consequently, the restricted domain does not require feature extraction since its boundaries are already defined. We simply use the ellipsoidal region of the current frame for the following frame.

In the tornado example in Figure 7, we use a sink to draw texture coordinates in toward the vortex initially forming the funnel. This motion conveys a spout pulled down along the central axis that is simultaneously swirled by a vortex feature.

5. Rendering

To view the gas simulation, we implement a graphics-accelerated volume renderer. A summary of the system is outlined in Figure 6. Since the simulation is most efficiently implemented on the CPU, we defer most rendering computation to programmable graphics hardware.

Our first step is shadowing. The only shading that we compute on the CPU is approximate shadowing since direct light integration through a feed-back buffer has shown to bottleneck hardware volume renderers [KPH*03]. Because texture coordinates are computed later in hardware, this lighting will shadow each voxel of the simulation density but disregards the influence of texturing later in the pipeline. In our applications, this shadowing is sufficiently detailed for our diffusive gases.

To render a single frame, we store the latest simulation volume as a grid of vertices. Every voxel has a corresponding vertex positioned within the grid, and density, velocity, and shadow quantities are assigned as vertex attributes. These vertices form tessellated planes aligned to the axis closest to the eye ray. To add additional detail, we use advected texturing [Ney03], which calculates the local displacement of texture coordinates due to advection. This technique conveys fine particulate details smoothly moving through our simulation field. To improve image continuity, users can draw additional intermediate slices at the expense of rendering performance.

6. Applications

Our system provides a new framework to build and modify scenes of gaseous phenomena. With this interface, we have created clouds, dust plumes, and fire by controlling the density and velocity fields.

To begin modeling, the user may import existing simulation sequences, or paint or shape implicit ellipsoids to create new densities and velocities. All controls are adjustable in real-time and can be saved to apply to other scenes. The companion movies demonstrate the interface, shown in Figure 2. Once a scene is defined, it may be exported to offline renderers for higher fidelity results.

Our applications use simulation grids of $32 \times 32 \times 32$, but are rendered over 128 slices to improve image continuity. Frame rates fluctuate around 5-15 fps on a 3.0 Ghz processor equipped with an NVIDIA GeForce 6800 Ultra graphics accelerator. Performance is bottlenecked by the simulation or the fragment program's volumetric noise texture lookups, varying with flow velocities and the viewport's visible area. In our examples, a simulation iteration takes approximately 90 ms, vortex extraction 20 ms, while rendering runs concurrently at 50 ms. In conditions with lower velocities and the camera within the gas, these benchmarks are nearly reversed, with rendering dominating the processing time.

6.1. Cyclones

We can generate a variety of cyclone like effects using vortex and density features. Beginning with a flat cloud base, we can generate a vertical vortex structure and keyframe the feature stretching down from the clouds, as seen in Figure 2. The simulation will draw densities down around the axis forming the desired funnel (Figure 7). This effect can be enhanced with a small downward advective velocity feature, or by manually keyframing the density primitives to that effect.

In this example, we have created and adjusted the Eulerian simulation exclusively through high-level primitives. The cloud structure and motion are directly modeled, but animated with a realistic physically-based Eulerian simulation. Using both tools, artists can modify the natural evolution of the physical simulation *while it happens* without recalculating initial conditions and restarting the simulation. This approach was also used to create the billowing head in Figure 8.

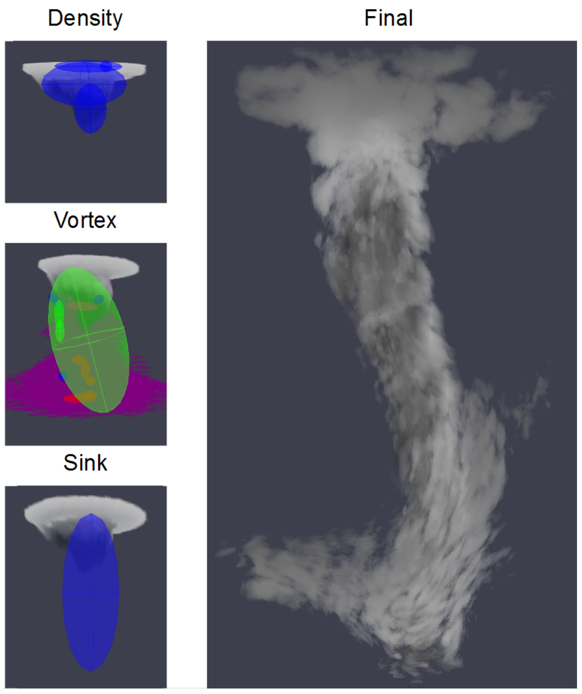


Figure 7: *Tornadoes can be created by driving a fluid simulation with vortex, sinks, and density features to draw visible media into a swirling funnel.*

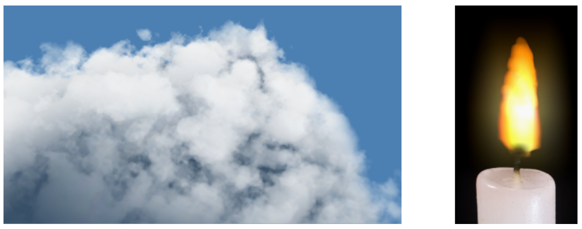


Figure 8: *A cloud formation and candle flame animated using a mixture of feature placement and simulation.*

6.2. Fire

With feature primitives, we are able to emulate various combustive phenomena. Detonations exhibiting compressible behavior can be modeled using a combination of density and velocity sources. For environments requiring more uniform and less detailed flames, such as the candle in Figure 8, it may be advantageous to produce these through a physical simulation and modify the results.

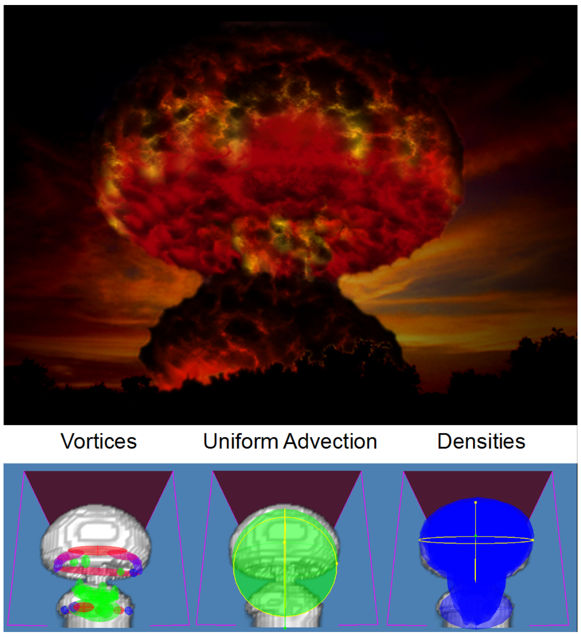


Figure 9: *Mushroom cloud created by simulated bouyancy in Equation 8, and its extracted features.*

To model the convective motion of hot buoyant gas, thermal energy is modeled by the density field H. Hot air rises because it is comparatively less dense than cold air. We model buoyant velocities $\mathbf{b}$ due to the thermal gradient ∇H as a function of gravity $\mathbf{g}$:

$$\mathbf{b} = \mathbf{g}|\nabla H| \qquad (8)$$

These velocities are added to the velocity field before each simulation step, making densities rise, curl, and dissipate like flames. Large gradients representing high temperatures form distinctive mushroom clouds as in Figure 9. Here, a high-density source was placed near the bottom of the simulation domain, and we use the thermal field H as the visualized density field. By tweaking the gravity vector $\mathbf{g}$ and intensifying vortices extracted from the simulation, the turbulent structure of the density field was further enhanced.

6.3. Clouds

Clouds are products of moist thermal gradients where water condenses into a visual form. Users can approximate this behavior with feature primitives, but may also take advantage of the physical simulation to give rise to this phenomena naturally. Using a buoyancy/condensation model, users can seed the volume, which naturally rises leaving a visible condensate trail, but later revise the breadth and height of its ascent, or alter conditions to give rise to more dramatic phenomena.

Hot, humid air is modeled as the thermal field H as in Section 6.2 and visible condensation as another field, Q. For simplification, we assume ambient water vapor is uniformly present and only model its transition into condensation. The change in the condensation volume Q is the advection by the flow field plus the new condensing water defined as

$$\text{mix}(h) = \gamma e^{\left(\frac{17.67h}{h+243.5}\right)}. \qquad (9)$$

This modified mixing ratio is a parameter of the local heat h fitted to collected meteorological data [RY88]. A user-defined parameter γ adjusts the rate of condensation, typically a function of altitude and gravity.

While physical modeling provides a more realistic paradigm to create clouds, users still retain high-level control with our feature manipulation. These processes were used to create the developing storm in Figure 10. Buoyant heat was seeded at the bottom of the simulation, and as the condensation forms, these clouds were reshaped with the density features. Similarly, the advective forces can be redirected with uniform advectors to produce different cloud formations.

7. Conclusion

We have developed a new gas editing technique using feature extraction to create high-level artistic controls that drive a realistic physical simulation. Users can create and modify density and flow elements of a Navier-Stokes simulation through a set of feature widgets. While previous techniques enable these primitives solely as a persistent source of density or energy, feature extraction permits us to apply these primitives in an initial frame and “refind” them as the simulation evolves.

Our system not only provides a new modeling approach to establish initial conditions, but also to revise recorded sequences. In addition to letting user-defined features evolve naturally, feature extraction also finds features naturally occurring in the field. This permits the user to select and change the behavior of features as the simulation progresses, enabling a more “hands-on” approach to flow modeling.

The resulting simulations are rendered interactively by a hardware-accelerated implementation of advected textures. This deterministic technique conveys detailed motion along the velocity field, and is scalable to higher resolution offline rendering while faithfully reflecting even the most unusual designs (Figure 11). Combined with our modeling approach, our system produces realistic animated gases through physically-based simulation, controlled with a set of intuitive high-level primitives.

8. Future Work

We plan to extend our system to combine basic flow primitives into higher-level primitives that construct common

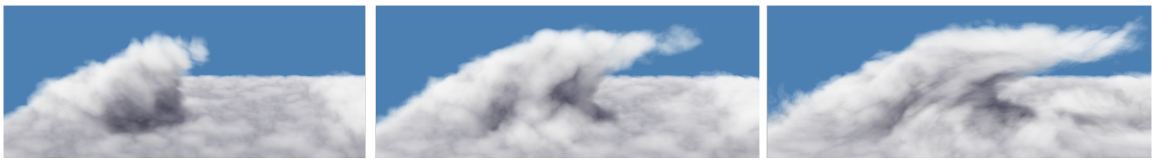


Figure 10: *A growing thunderhead constructed from seeded heat.*

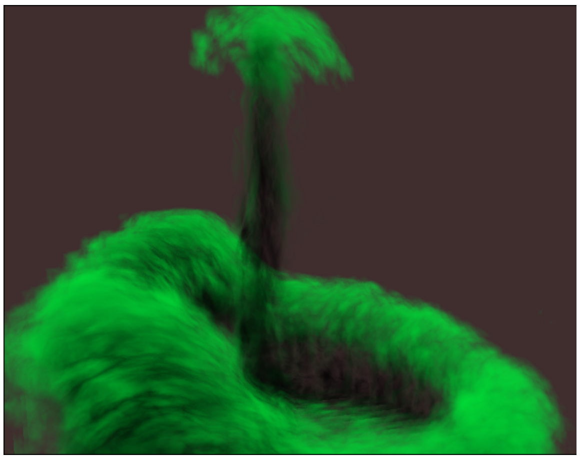


Figure 11: *Pouring a swirling, splashing gas.*

forms of flow phenomena (plumes, whirlpools). We will investigate other structural approximations for features and explore a lofting modeling approach with spline-based feature skeletons. These more complex flows motivate improving the performance or the simulation and rendering, both current bottlenecks, to remain interactive at higher resolutions.

9. Acknowledgements

This material is based upon work supported by the Purdue Research Foundation TRASK program and the US National Science Foundation under grants: NSF ACI-0081581, NSF ACI-0121288, NSF ACI-0328984, NSF IIS-0098443, NSF ACI-9978032, and NSF ACI-0222675. We would like to thank the anonymous reviewers for their valuable comments to improve this paper, Alias, Inc. for their donation of software and NVidia for their donation of hardware to support our research.

References

[EMP*02] EBERT D. S., MUSGRAVE F. K., PEACHEY D., PERLIN K., WORLEY S.: *Texturing and Modeling: A Procedural Approach*. Morgan Kaufmann Publishers Inc., 2002. 2

[FL04] FATTAL R., LISCHINSKI D.: Target-driven smoke animation. *ACM Trans. Graph. 23*, 3 (2004), 441–448. 2

[FM97] FOSTER N., METAXAS D.: Controlling fluid animation. In *CGI '97: Proceedings of the 1997 Conference on Computer Graphics International* (Washington, DC, USA, 1997), IEEE Computer Society, p. 178. 2

[FOA03] FELDMAN B. E., O'BRIEN J. F., ARIKAN O.: Animating suspended particle explosions. *ACM Transactions on Graphics 22*, 3 (2003), 708–715. 5

[FSJ01] FEDKIW R., STAM J., JENSEN H. W.: Visual simulation of smoke. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (2001), ACM Press, pp. 15–22. 4

[JH95] JEONG J., HUSSAIN F.: On the identification of a vortex. *Journal of Fluid Mechanics* (285 1995), 69–94. 4

[JMT02] JIANG M., MACHIRAJU R., THOMPSON D.: A novel approach to vortex core region detection. In *Proceedings of the symposium on Data Visualisation 2002* (2002), Eurographics Association, pp. 217–ff. 4

[JMT03] JIANG M., MACHIRAJU R., THOMPSON D.: Detection and visualization of vortices. In *Visualization Handbook*. Academic Press, 2003. unpublished. 2

[KPH*03] KNISS J., PREMOZE S., HANSEN C., SHIRLEY P., MCPHERSON A.: A model for volume lighting and modeling. *IEEE Transactions on Visualization and Computer Graphics 9*, 2 (2003), 150–162. 5

[Lug83] LUGT H. J.: *Vortex flow in nature and technology*. Wiley, 1983. 4

[MTPS04] MCNAMARA A., TREUILLE A., POPOVIĆ; Z., STAM J.: Fluid control using the adjoint method. *ACM Trans. Graph. 23*, 3 (2004), 449–456. 2

[Ney03] NEYRET F.: Advected textures. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer animation* (2003), Eurographics Association, pp. 147–153. 3, 6

[PCS04] PIGHIN F., COHEN J. M., SHAH M.: Modeling and editing flows using advected radial basis functions. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation* (2004), ACM Press, pp. 223–232. 2

[PVH*03] POST F. H., VROLIJK B., HAUSER H., LARAMEE R. S., DOLEISCH H.: The state of the art in flow visualisation: Feature extraction and tracking. *Computer Graphics Forum 22*, 4 (2003), 775–792. 2

[REN*04] RASMUSSEN N., ENRIGHT D., NGUYEN D., MARINO S., SUMNER N., GEIGER W., HOON S., FEDKIW R.: Directable photorealistic liquids. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation* (2004), ACM Press, pp. 193–202. 2

[RY88] ROGERS R. R., YAU M. K.: *A Short Course in Cloud Physics*, 3. ed., vol. 113 of *International Series in Natural Philosophy*. Pergamon Press, 1988. 7

[SB94] SINGER B. A., BANKS D. C.: *A Predictor-corrector Scheme for Vortex Identification*. Tech. rep., 1994. 4

[SP99] SADARJOEN I., POST F.: Geometric Methods for Vortex Detection. pp. 53–62. 4

[Sta99] STAM J.: Stable fluids. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (1999), ACM Press/Addison-Wesley Publishing Co., pp. 121–128. 3

[TLHD03] TONG Y., LOMBEYDA S., HIRANI A. N., DESBRUN M.: Discrete multiscale vector field decomposition. *ACM Trans. Graph. 22*, 3 (2003), 445–452. 2

[TMPS03] TREUILLE A., MCNAMARA A., POPOVIĆ Z., STAM J.: Keyframe control of smoke simulations. *ACM Trans. Graph. 22*, 3 (2003), 716–723. 2

[WFG94] WILLIAM F. GATES M.: *Interactive Flow Field Modeling for the Design and Control of Fluid Motion in Computer Animation*. Master's thesis, UBC, 1994. 2

[WH91] WEJCHERT J., HAUMANN D.: Animation aerodynamics. In *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques* (1991), ACM Press, pp. 19–22. 2

[Wit99] WITTING P.: Computational fluid dynamics in a traditional animation environment. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1999), ACM Press/Addison-Wesley Publishing Co., pp. 129–136. 2