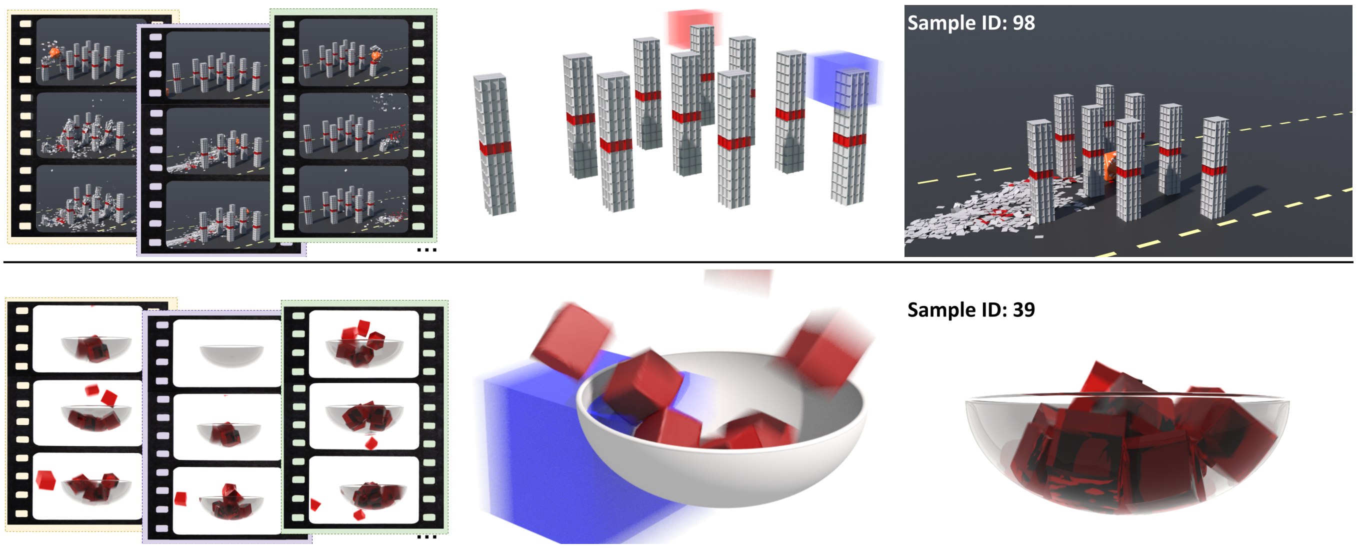
Unified Many-Worlds Browsing of Arbitrary Physics-based Animations

PURVI GOEL and DOUG L. JAMES, Stanford University, USA



(a) (b) (c)

Fig. 1. **Unified Many Worlds Browsing** enables curiosity-driven exploration of (a) simulation ensembles of arbitrary physics-based phenomena created by sampling animation parameters, such as (Top) the rigid-body interactions of a "bowling die" launched at card "pins," or (Bottom) the behavior of Jell-O® thrown in a bowl. (b) A user can navigate ensembles at interactive rates by defining and dragging exploratory spatiotemporal queries (blue and red moving boxes) to find interesting and distinct scenarios such as (c, Top) only knocking down particular card pins, or (c, Bottom) keeping all Jell-O® in the bowl.

Manually tuning physics-based animation parameters to explore a simulation outcome space or achieve desired motion outcomes can be notoriously tedious. This problem has motivated many sophisticated and specialized optimization-based methods for fine-grained (keyframe) control, each of which are typically limited to specific animation phenomena, usually complicated, and, unfortunately, not widely used.

In this paper, we propose *Unified Many-Worlds Browsing* (UMWB), a practical method for sample-level control and exploration of physics-based animations. Our approach supports browsing of large simulation ensembles of arbitrary animation phenomena by using a unified volumetric WorldPack representation based on spatiotemporally compressed voxel data associated with geometric occupancy and other low-fidelity animation state. Beyond memory reduction, the WorldPack representation also enables unified query support for interactive browsing: it provides fast evaluation of approximate spatiotemporal queries, such as occupancy tests that find ensemble samples ("worlds") where material is either IN or NOT IN a user-specified spacetime region. WorldPacks also support real-time hardware-accelerated voxel rendering by exploiting the spatially hierarchical and temporal RLE raster data structure. Our UMWB implementation supports interactive browsing (and offline refinement) of ensembles containing thousands of simulation samples, and fast spatiotemporal queries and ranking. We show UMWB results using a wide variety of physics-based animation phenomena—not just Jell-O®.

CCS Concepts: • **Computing methodologies** → **Animation**; **Physical simulation**; **Volumetric models**.

Additional Key Words and Phrases: Motion control, interactive control, browsing, ranking, simulation ensembles, volume data, compression, RLE, WorldPack, ray tracing Jell-O®.

ACM Reference Format:
Purvi Goel and Doug L. James. 2022. Unified Many-Worlds Browsing of Arbitrary Physics-based Animations. *ACM Trans. Graph.* 41, 4, Article 156 (July 2022), 15 pages. https://doi.org/10.1145/3528223.3530082

Authors' address: Purvi Goel, pgoel2@cs.stanford.edu; Doug L. James, djames@cs.stanford.edu, Computer Science, Stanford University, 353 Jane Stanford Way, Stanford, CA, 94305, USA.

0730-0301/2022/7-ART156 $15.00
https://doi.org/10.1145/3528223.3530082

1 INTRODUCTION

Tuning parameters of physically based animations is a notoriously tedious task required to either explore a motion outcome space or achieve desired behaviors, such as "what goes where" scenarios, e.g., tossing pieces of Jell-O® into a bowl. Manual exploration of the

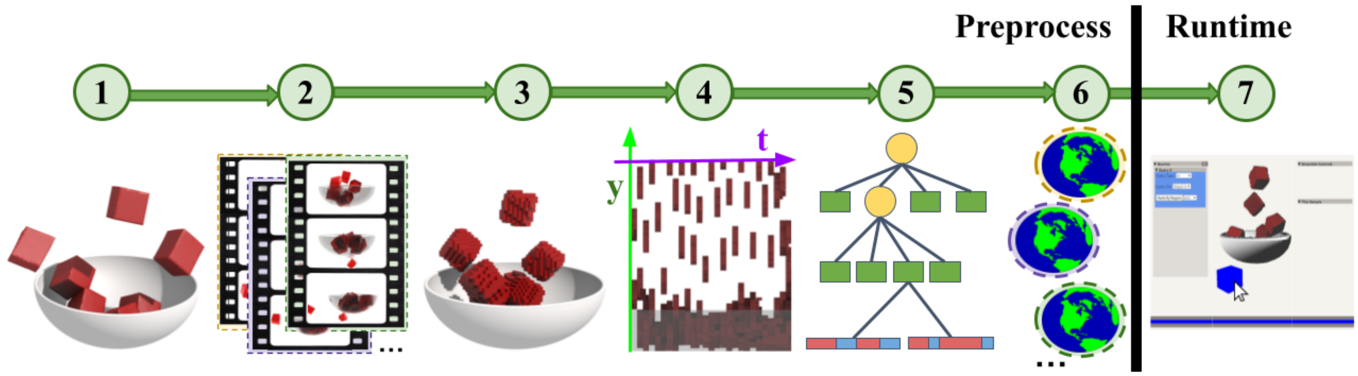


Fig. 2. **Overview:** Given (1) a parameterized physics-based animation, (2) the user samples a simulation ensemble, ideally in parallel. To reduce storage costs and support efficient processing, each simulation sample is summarized by (3) performing volumetric rasterization of quantized simulation state, such as binary occupancy data, then (4) performing temporal run-length encoding (RLE) of each voxel, followed by (5) spatially hierarchical decorrelation. The resulting (6) compressed WorldPack simulation summaries are then used at runtime in (7) our Unified Many-Worlds Browser to perform GPU-accelerated ray tracing of coarse simulation state, and accelerate spatiotemporal user-interaction queries and ranking.

parameter space, such as by sampling many possible cases (a.k.a. "wedging"), can be tedious and difficult, but, regrettably, it is standard practice for animators everywhere.

Simultaneously, decades of advances on physics-based motion control by the computer animation community has enabled sophisticated numerical methods for enabling fine-level motion control, such as keyframing and sketching. Formulated as multi-objective optimization problems, the desired animations are estimated using sampling- and/or derivative-based optimization techniques. These methods can achieve surprising and sometimes delightfully improbable animation results for a wide range of animated physical phenomena, such as particles, rigid bodies, articulated characters, cloth and shells, smoke, liquids, etc. Alternately, coarser sample-level motion control can be achieved using interactive browsing techniques, e.g., for rigid bodies, and avoid the need for specifying detailed keyframe constraints and sophisticated optimization methods.

Unfortunately, the current state of the art in motion control is that completely different methods are required to support different animated phenomena (rigid bodies, cloth, fluids, etc.) and accommodate the inherent complexities of simulation control (kinematics, dynamics, discretization, solvers, optimization, control specification, user interface, etc.). Furthermore, even if a system did have integrated motion controllers for every phenomenon, there exists no simple framework for unifying these approaches into one system that is easy and fun to use by practitioners. Moreover, motion controllers generally require an artist to have a clear goal in mind to target their specialized solvers, optimizers, and keyframes towards, rather than enabling curiosity-driven exploration of a simulation's range of possibilities. So, we ask, is there a simple approach for data exploration and motion control that can unify arbitrary physics-based phenomena, or do we need a different method for every single flavor [Heckbert 1987]?

In this paper, we propose *Unified Many-Worlds Browsing* (UMWB) for sample-level control and exploration of arbitrary physics-based animations—a reimagining of Twigg and James [2007] for rigid-body animations. Please see Figure 1 for a preview of our results, and Figure 2 for an overview of our technique. Our approach begins by sampling (and optionally refining) animations in parallel from a distribution of possible input parameters (initial conditions, geometric and design variations, material properties, control forces, stochasticity, etc.) to create an animation ensemble. Simulation summarization is performed to support unified browsing of arbitrary animations. We volumetrically rasterize and quantize 3D animation state, such as geometry and fields of interest, to create binary occupancy and low-fidelity attribute value grids. These fields are further compressed using temporal run-length encoding (RLE), followed by spatially hierarchical decorrelation, and then stored in a compressed format, a WorldPack, for unified browsing. For further scalability, the original high-fidelity simulations can be discarded—most are never used—then recomputed later, if needed, using their sample parameters, thereby resulting in significant storage reductions.

The compressed volumetric WorldPack sample representation provides just enough information to summarize the animation, while also enabling fast unified query support for interactive browsing. Given user-specified spatiotemporal regions, e.g., a mouse-drawn bounding box, we can perform fast spatiotemporal occupancy tests to find ensemble samples ("worlds") where material is either IN or NOT IN a specific spacetime region, and thus enable "what goes where" animation queries. The compressed representation enables fast evaluation of general query and ranking predicates, such as spatiotemporal integrals.

We further leverage WorldPack's spatially hierarchical and temporal RLE raster representation to perform hardware-accelerated ray tracing. Although of a greatly reduced visual fidelity compared to production quality animations, our system can display hundreds of multi-physics simulation summaries at interactive rates with modest in-core memory footprints.

Finally, we demonstrate our prototype UMWB implementation with physics-based animation content generated by a wide variety of phenomena and simulators. Our system can support interactive browsing (and offline refinement) of ensembles containing thousands of simulation samples, efficiently perform spatiotemporal queries at interactive rates, and rank and display selections interactively in a browser for easy use by animators. Our browser demonstration and data are available at https://github.com/stanford-gfx/umwb.

2 RELATED WORK

Tuning Parameters and Design Spaces. A fundamental problem in physics-based animation is the tuning of many simulation and control parameters. Manual exploration of parameter spaces is often done using tedious online interactive tuning, or offline brute-force (parallel) sampling of so-called parameter "wedges" (e.g., in SideFX Houdini [SideFX 2021]) or animation ensembles (c.f. "Design Galleries" [Marks et al. 1997]). Our query-driven exploration scheme is inspired by a larger body of work on user-guided exploration of high-dimensional, highly varied design spaces. Some design spaces can be sampled in real time: the solver or model can instantly produce an outcome from a set of input parameters, in such domains as parametric BRDFs [Talton et al. 2009], color grading of photographs [Koyama et al. 2020], and 2D particle systems [Shimizu et al. 2020]. Fast solvers are conducive to a fast, interactive user exploration experience. In order to support interactive browsing when solvers do not run at interactive rates, such as in this paper, prior work relies on computing an ensemble of simulation samples in an offline step, such as work on Design Galleries [Marks et al. 1997]. This pipeline was applied to fire and smoke volumes at low resolutions [Bruckner and Möller 2010]. Our browsing approach enhances exploration of arbitrary animation ensembles, e.g., using spatiotemporal queries.

Rigid-body Animation Control. Many methods for control and steering have emerged in response to the often unintuitive, finicky, and time-consuming process of tuning rigid-body animation parameters to produce a desired outcome. These techniques introduce algorithms that work in tandem with users, solving, searching, or optimizing for a set of initial conditions that produce a user-specified desired simulation end state or outcome. The dream of fine-level keyframe control of physics-based simulations led to several advances. Witkin and Kass [1988] expressed keyframe control as the solution to a constrained spacetime optimization problem, and explored how to programmatically imbue physics-based animations with anticipation, follow-through, squash-and-stretch, and timing [Lasseter 1987]. Later work explored genetic algorithms [Tang et al. 1995], interactive spacetime control using a prototype interface for sketching motion edits [Cohen 1992], and entire rigid-body motions using multiple-shooting optimization [Popović et al. 2003].

Idealized simulation models are commonly enriched using (stochastic) parameter variations, e.g., perturbed contact normals, to produce more varied yet visually *plausible dynamics* by virtue of realistic statistical variations [Barzel et al. 1996] and the limitations of human perception [O'Sullivan et al. 2003]. Producing richer animation ensembles enables opportunities for *sample-level control* wherein animation control is achieved by selecting desired outcomes from available ensemble samples, for example, to produce a plausible yet desirable pool break in virtual billiards [Barzel et al. 1996], or to impose spatiotemporal object constraints or steer ensemble refinement [Twigg and James 2007].

In pioneering work, Popović et al. [2000] explored interactive gradient-based optimization (with sampling to avoid local minima) and was able to find a number of plausible yet exciting motions in real time for smaller examples. Chenney and Forsyth [2000] expressed desirable animations using mathematical objective functions that were then optimized using statistical sampling techniques based on Markov Chain Monte Carlo (MCMC) to achieve interesting sample-level control, e.g., of bowling outcomes. Sample-level optimization was also used for audio-constrained synthesis of rigid-body motions in [Langlois and James 2014]. While these fine-level (keyframe) approaches produce many promising results, the optimization problems can be very difficult and the solvers can struggle, e.g., fall into unsatisfying local minima, be extremely slow on larger problems (which interferes with interactive user experiences), suffer from physical infeasibility, e.g., due to collisions, and, ironically, be difficult for practitioners to use and tune parameters for.

To avoid fine-level keyframe control and its inherent optimization difficulties, Many-Worlds Browsing (MWB) [Twigg and James 2007] only provides coarser sample-level control on a (precomputed or refineable) simulation ensemble. MWB leverages human intelligence and interactive tools to let the user explore and query the space of all ensemble samples. While user-specified spatial queries were able to find needle-in-a-haystack rigid-body simulation outcomes, the method was limited to a single phenomenological domain—rigid body dynamics. UMWB generalizes to other simulation domains of importance to computer animation (deformation, fracture, smoke and fire, liquids, etc.), and provides a practical unified, interactive browsing framework for arbitrary animations. The original MWB approach only used approximate, compressed center-of-mass for inexact spatial queries, whereas we use approximate volumetric WorldPack representations with good memory performance for modern multi-physics simulations.

Beyond Rigid Bodies. There are several works that have explored simulation control and specialized guiding techniques for domains other than rigid bodies. Examples include fluid animation controllers [Foster and Metaxas 1997] and keyframe-based smoke control [Fattal and Lischinski 2004; Treuille et al. 2003]. Adjoint control methods assist with many control parameters for fluids [McNamara et al. 2004] and particle systems [Wojtan et al. 2006]. Other methods for fluids include guide-based techniques [Sato et al. 2021; Shi and Yu 2005]; neural style transfer [Guo et al. 2021; Kim et al. 2019] for indirect control; strategically placed control forces and interactive local edits [Pan et al. 2013; Schoentgen et al. 2020; Thürey et al. 2006]; methods for stylizing fluids with neural networks [Kim et al. 2020; Yan et al. 2020]. Unfortunately methods for calculating control forces to carefully guide simulations are domain-specific and, for instance, formulations tuned to controlling smoke do not trivially generalize to controlling rigid bodies or elastic objects. For example, specialized methods exist for coarse-to-fine control of thin shells using moment-based constraints [Bergou et al. 2007].

Control techniques for multiphysics simulations that can generalize to several physical models are less common. Some attempts include controllers for reduced-order dynamical systems [Barbič and Popović 2008]. Recently, Ma et al. [2018] trained a controller using deep reinforcement learning for coupled fluids and rigid bodies, but only offers control on the fluid in the domain.

Volumetric Data Structures. Our work on browsing-specific compression formats is closely related to several previous papers on data structures optimized for storing large, sparse volumes. Sparse regular grids are common in computer animation applications, for representing everything from scenes to vector fields. As a result, a great number of papers explore different ways to handle common issues with these grids, such as high memory footprints and expensive data access, including VDB [Museth 2013], DT-grid [Nielsen and Museth 2006], SPGrid [Setaluri et al. 2014], and DB+grid [Museth 2011]. Several methods are inspired by volumetric octrees [Samet 2006] for exploiting spatial sparsity, such as sparse paged grids [Aanjaneya et al. 2017] and wide-branching tile trees [Nielsen et al. 2018]. However, most of these data structures target dynamic data using instantaneous, per-frame representations; in contrast, our work targets browsing of precomputed simulation ensembles where simulation samples are not modified at browsing time. In this context, NanoVDB [Museth 2021], which stores immutable simulation snapshots with static topology, is more similar to our WorldPack representation. One key difference is that NanoVDB and its predecessors store a separate serialized data structure for each timestep, independent of data at other timesteps. However, many simulations contain both temporal and spatial sparsity and redundancy, especially when quantized heavily. Therefore, rather than storing a data structure per-timestep, our proposed WorldPack data structure leverages both in-space *and* in-time compression to further reduce the memory footprint and accelerate queries.

Run-length encoding (RLE), our temporal compression strategy, has been widely applied to volume data in previous work, particularly for representing sparse level sets [Houston et al. 2006, 2004], and signed distance functions [Curless and Levoy 1996]. But these methods use RLE solely for in-space compression; to our knowledge, applying RLE compression in-time on volume data is a largely unexplored direction, in part because of the pervasive nature of per-frame animation processing. Coupled in-time and in-space compression appears in common 2D video compression schemes like GIF89a and H.264 [Sayood 2017], and can effectively reduce memory costs while preserving quality. In contrast, our data structure is designed specifically for fast processing of spatiotemporal user queries on low-fidelity volume animation data.

Finally, our WorldPack queries exploit the fact that vector sets compressed using RLE (or PackBits) can be used to accelerate vector computations (dot products, summation, etc.) [Oyamada et al. 2018].

Animation Compression. Many-Worlds Browsing used rigid-body trajectory compression to support larger in-core ensembles [Twigg and James 2007], and the center-of-mass trajectory for fast, approximate spatial queries. Rigid-motion compression rates are good for ballistic trajectories, but degrade for objects undergoing many complex collisions (c.f. [Jeruzalski et al. 2018]). In computer animation, compression schemes exist for various animated content, such as deforming mesh animations [Lengyel 1999; Sattler et al. 2005], character animation databases [Arikan 2006], fluid animation fields using DCT-based compression [Jones et al. 2016], etc., however they tend to be domain-specific and not easy to integrate with approximate browsing queries on arbitrary animated content. In contrast, we exploit a unified low-fidelity representation to achieve compression, fast rendering and approximate queries of arbitrary animations during ensemble browsing. In other fields, the use of compressed representations of data that allows for answering queries about that data is referred to as "sketching" [Nelson 2011]. Lastly, low-fidelity simulations for browsing are orthogonally related to the use of adaptive precision simulation [Hu et al. 2021; Yeh et al. 2009].

3 SUMMARIZING SIMULATIONS WITH WORLDPACK

3.1 Design Goals

3.1.1 Simulation Summaries. Unfortunately, storing ensembles in core for on-the-fly query evaluation can quickly become intractable given the sheer amount of simulation data often dumped. For example, a single smoke animation sample can contain several gigabytes of simulation data, and we anticipate ensemble sizes on the order of hundreds to thousands of samples. Furthermore, it is impractical to use raw simulation ensemble data to support interactive query performance and the visualization of selected results.

To this end, we require a data structure that can *summarize simulation data* in support of the following UMWB design desiderata:

(1) Arbitrary & Unified: We desire support for arbitrary physics-based animation models (solids, fluids, etc.) by using an approximate but unified data representation.
(2) Expressive: We desire sufficient spacetime resolution to express spatiotemporal animation queries of practical interest.
(3) Low Memory: We desire in-core query processing and thus require a low-memory footprint for the entire summarized animation ensemble.
(4) Fast Queries and Ranking: The data structure should be optimized for interactive evaluation of spatiotemporal queries.
(5) Fast Rendering: Since the original simulation data is unavailable, the summary must itself provide a fast visual substitute.

Design decisions made under these tenets resulted in WorldPack, a data structure for interactive exploration of large simulation ensembles. With WorldPack, users may interact with rasterized ensemble data at interactive rates in order to explore hundreds to thousands of ensemble samples and narrow down the outcome space to only the samples they find interesting.

3.1.2 Simulation Data Quantization. Raw simulation data dumped from solvers are often at the high precisions required for per-timestep solves. We emphasize that the intention of the WorldPack data structure is to store an expressive but lightweight sketch of an animation over time; therefore, all simulation data that we store in WorldPacks will be quantized to lower precision.

Spacetime Occupancy: For what-goes-where queries that form the bulk of our user interaction model, just knowing if a spatiotemporal region contains an object or not is generally sufficient to sift through ensemble samples. We consider this a question of *spacetime occupancy* in which simulated material of interest is identified by the user using an indicator function $I_\Omega : (\boldsymbol{x}, t) \rightarrow \{0, 1\}$. Specifically, let the object material occupy the spacetime region $\Omega \subset \mathbb{R}^3 \times \mathbb{R}_+$,

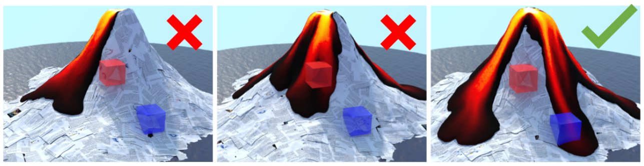


Fig. 3. **Spatiotemporal Queries** are drawn in world space with accompanying temporal bounds and a predicate to be applied on the region. Common predicates for what-goes-where queries test for geometric occupancy: IN queries include simulations that include any occupancy of a given object–in this case, the lava–in the region. NOT IN queries discard simulations with occupancy in the region. The above thumbnails show four simulation samples tested against the same two queries. The samples evaluate from left to right as *false* (lava is not present in the IN region), *false* (lava is present in the NOT IN region), and *true*.

then the continuous indicator function is

$$I_\Omega(\boldsymbol{x}, t) = \begin{cases} 1 & \text{if } (\boldsymbol{x}, t) \in \Omega, \\ 0 & \text{if } (\boldsymbol{x}, t) \notin \Omega. \end{cases} \tag{1}$$

Attribute Values: For attribute values that need to be stored at higher fidelity than an indicator function (like color, normals, or object IDs), we can quantize the range of values to save memory. While we lose accuracy for these values, we gain the flexibility to compare samples, store large ensembles in core, and identify relative differences and trends.

3.1.3 Spatiotemporal Query Model. In our proposed system, a user interacts with simulation ensemble data via *spatiotemporal queries*, analogous to those used in relational databases. Queries fetch specific simulation data fields or **keys** (e.g., temperature, velocity, normals, boolean occupancy, etc.), apply a **reduction** (e.g., `max`, `average`, `sum`, `any`, etc.) and evaluate whether an accompanying **predicate** is *true* or *false* for a given spacetime **region** R. For example, a user may ask for ensemble samples that satisfy various queries (see Figure 3):

- IN_R Query: *"There exists some geometric occupancy"* in a region R. A simulation sample occupying Ω must have $\Omega \cap R \neq \emptyset$ in order to satisfy an IN_R query.
- NOT IN_R Query: *"There is no geometric occupancy"* in a region R. A simulation sample occupying Ω must have $\Omega \cap R = \emptyset$ in order to satisfy a NOT IN_R query.
- Attribute Value Queries: *"Average temperature greater than 40"* in a region R is an example of a more specific query with attribute-value predicates.

Users may chain together queries to explore arbitrary physics-based ensembles in a simple and unified manner. From our definitions, it follows that evaluating if queries are *true* or *false* for a given ensemble sample amounts to processing spatiotemporal operators, e.g., repeatedly and efficiently applying data reduction operators (such as `min`, `max`) to simulation data region R, to check the scalar in a query predicate.

3.2 WorldPack Concepts

3.2.1 Exploiting Spacetime Structure. While simulation spatial dimensions can grow to intractable sizes, we generally do not need to store data for every single spacetime point in the domain. First, we observe that for many artist animations, *quantized* simulation data contains a large amount of *spatiotemporal redundancy*: values in a given spatial neighborhood are often relatively consistent over portions of time—an extreme case of this redundancy is with the rasterized, boolean (0 or 1) occupancy indicator function where changes between frames only occur on the object's boundary. Second, relevant data exhibits *sparsity*, as it is usually constrained to compact regions within the domain. This sparsity may manifest itself as empty space (*"density is 0 in region R"*) interspersed with static geometry (*"occupancy exists in region R"*). Many popular data structures like VDB[Museth 2013] exploit the spatial sparsity of simulation domains for better compression and efficient sampling. However, these data structures are generally stored as snapshots for per-frame processing, e.g., an artist may dump a sparse volume for each frame of an animated sequence. While popular data representations can vastly improve memory footprints for a given timestep, storing per-timestep volume representations over hundreds or thousands of timesteps can quickly get out of hand. It is not uncommon for a cache of per-timestep sparse volumes for a single animated sequence to demand gigabytes of memory.

Because sparse volumes are customarily compressed using strategies that leverage *spatial redundancy* and dumped per-timestep, they fail to exploit *spatiotemporal redundancy*. We observe that simulation data is often both consistent in space and time. To reference our previous example, spatiotemporal redundancy can appear as *"density is 0 within region R for a span of T timesteps."* Utilizing redundancy in both space and time is a largely unexplored space, but in order to store ensembles with sizes on the order of thousands of samples in core, we must look further than compression with just a spatial hierarchy.

3.3 WorldPack Construction

3.3.1 Quantization. As discussed earlier, we start by quantizing simulation data to lower-precision values in a field-type-specific manner. For spacetime occupancy, 1-bit quantization is already performed by the indicator function, $I_\Omega(\boldsymbol{x}, t)$.

For a scalar attribute value, $v \in \mathbb{R}$, we discretize the continuous interval $[v_{min}, v_{max}]$ into b_v bins, where v_{max} (or v_{min}) are the maximum (or minimum) value that the v attribute ever takes for a simulation sample. Let the length $L = v_{max} - v_{min}$ be divided into b_v distinct bins. Every attribute value v thus falls into a value bin with integer index $i_v \in [0, b_v)$. Rather than storing the higher-precision v attribute value itself, we instead store i_v.

For all following steps, we act on quantized simulation data using ordinal values, e.g., 8-bit value-bin indices, rather than the raw values.

3.3.2 Rasterization. In order to support unified data exploration for arbitrary simulations, regardless of solver-specific representations (e.g., particles, grids, meshes, etc.), we rasterize per-sample, per-timestep simulation data onto coarse, uniform voxel grids, with user-specific cell size, h. All relevant attribute values (e.g., occupancy, temperature, density, etc.) are rasterized onto separate uniform grids. A simulation domain that has a spatial height, width, and depth of H, W, and D, respectively, is then converted into a 3D uniform voxel

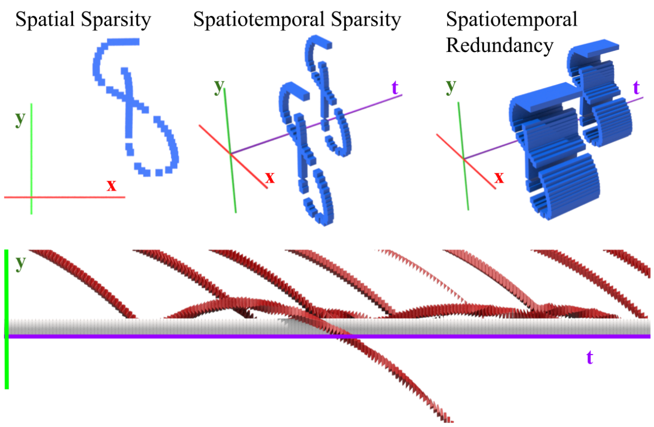


Fig. 4. **Spatiotemporal Redundancy:** (Top left) Spatial sparsity and (Top middle) spatiotemporal sparsity can be exploited to compress 3D and 4D gridded data, respectively. However, for physics-based animations where objects move continuously through time, we also observe (Top right) *spatiotemporal redundancy* wherein local neighborhoods may have the same value while not necessarily being empty. *(Bottom)* An XZ slice from a Jell-O-Bowl simulation shows Jell-O pieces moving through time. While some regions are certainly sparse, many areas are *redundant*: they are not empty, but they are constant within local spacetime neighborhoods.

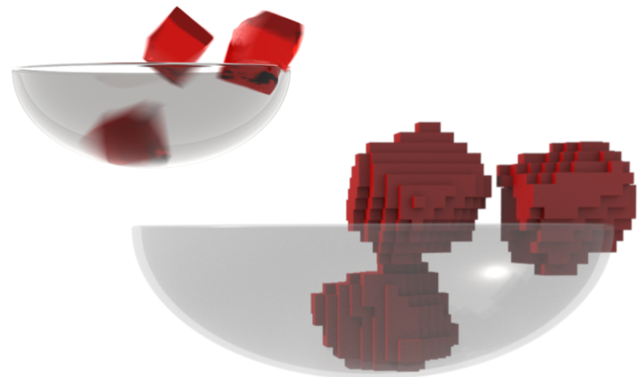


Fig. 5. **Rasterization:** Raw simulation data can be high in spatial resolution, making for expensive storage and processing. We start by rasterizing data from each timestep onto a coarsened uniform grid to create a lightweight summary. Here, we show a sample from our Jell-O ensemble. We project quantized data—such as the $I_\Omega(\boldsymbol{x}, t)$ indicator function (*"there is some Jell-O in this raster block" vs. "there is no Jell-O in this raster block"*—onto a coarse uniform grid. In this example, coarsening is done using a `max` reduction on 0/1 occupancy values.

grid of dimensions $N_x \times N_y \times N_z = \lfloor H/h \rfloor \times \lfloor W/h \rfloor \times \lfloor D/h \rfloor$. We refer to the ijk^{th} cubical cell as a **raster block**, and denote its spacetime region as $\Omega_{ijk} \subset \mathbb{R}^3 \times [0, T_{max}]$. The projection of simulation data to a single raster block value can be done via standard reductions. For example, spacetime occupancy is rasterized by approximating a conservative `max` reduction of the indicator function:

$$I_{ijk} \equiv \max_{(\boldsymbol{x},t)\in\Omega_{ijk}} I_\Omega(\boldsymbol{x}, t). \tag{2}$$

Discussion. Rasters only need to be coarse summaries of simulation data, so for browsing, a large h is preferable and enables efficient inspection of simulation-data trends across coarse spacetime chunks. Additionally, because WorldPacks are built as summaries of already-dumped simulation data, the finer resolutions required for simulation computations or high-quality rendering are unnecessary.

3.3.3 Temporal Compression. Let us consider a single ensemble sample at a time. For every timestep in this sample S, we have rasterized relevant attribute values into summaries composed of raster blocks. Because quantized simulation state is usually temporally redundant, we observe that, for a given attribute value, a raster block can have the same or similar values for significant stretches of time. Therefore, we use run-length encoding (RLE) of the raster block's values *in time* for the length of the simulation. RLE converts repeated data values into a sequence of <*count, value*> pairs, e.g., given the sequence of values: (2, 2, 2, 0, 4, 4, 4, 4, 4, 4, 4, 4), RLE would produce the following <*count, value*> tuples, commonly referred to as *runs*: <3, 2>, <1, 0>, <8, 4>. Notice how RLE has exploited the sequence's temporal redundancy, to reduce the number of values being stored from 12 to 6.

While strictly-repeating values are rare in floating-point simulation data, quantization creates redundancy in sequences that are very close in value. As a result, temporally encoded sequences usually have a smaller memory footprint in practice. Processing RLE-encoded sequences when evaluating temporal integrals is also often faster than storing a series of individual values because there is usually a) less to process, and b) processing does not require decoding the entire sequence (c.f. [Oyamada et al. 2018]). We discuss specifics of RLE-in-time operations in §4.

3.3.4 Spatial Compression. Next, we exploit the spatial consistency of simulation data: spatiotemporal neighborhoods often exhibit similar behavior. So, we build a shallow, wide octree on the RLE-in-time signals. Starting from the root, we only subdivide a node if the RLE sequences contained in the node's raster blocks vary. Otherwise, storing a single RLE sequence for the node's region is sufficient. The maximum depth for a hierarchy $Depth_{max}$ is chosen to be a small constant; because all RLE-in-time sequence data is stored at leaf nodes, a shallow structure allows us to traverse from root to leaf in a bounded number of steps.

Node Representation. Because octrees are regular hierarchies, we only need to store the dimensions and position of the root node. After that, as long as we store node children in a fixed order, the location and size of internal and leaf nodes can be inferred from depth [Samet 2006].

At every node, we store a flag indicating if the node has children or not; a flag value of *true* indicates that a node has 8 children, while a flag value of *false* indicates that a node has 0 children. Nodes store RLE-in-time data differently depending one of three cases, as in Figure 7:

- *Internal nodes* do not store RLE-in-time sequences, and only need to store a reference (either through pointers or memory

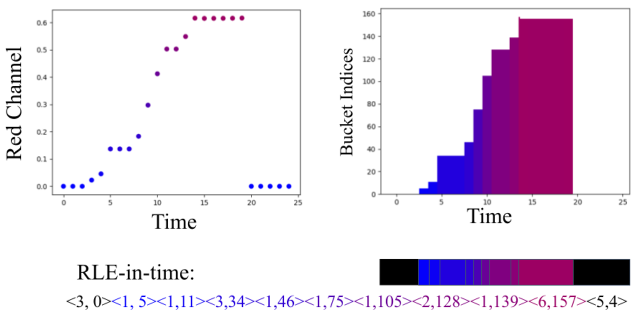


Fig. 6. **RLE-in-time Compression:** WorldPack utilizes RLE-in-time to reduce the behavior in a single raster block over the course of hundreds of timesteps. Following this step, each raster block will have a single RLE encoded signal. (Top Left) Some attribute values are represented in simulators with higher-precision floating point numbers, which may be close in value but do not strictly repeat. (Top Right) Similarly to how we coarsened space, we quantize the attribute value axis so that values relatively near in time and value fall into the same value bin. Finally, we RLE-in-time encode lower-precision bin IDs rather than higher-precision floating point values (Bottom) into *<count, value>* pairs. In the case of geometric occupancy, we use just two bins: occupancy either exists, or does not exist, within a raster block. For attribute values requiring finer-grained storage like angles or color, we use 256 bins.

offsets) to the node's children. (Nodes marked in *light purple* in Figure 7)

- *Leaf node at a smaller hierarchy depth than* $Depth_{max}$*:* In this case, we only store the single RLE-in-time sequence since all descendent raster blocks are identical. (Nodes marked in *green* in Figure 7)
- *Leaf node at* $Depth_{max}$*:* In this case, some raster blocks within the node's bounds have differing RLE-in-time sequences. So, the node stores RLE-in-time sequences for each of its raster blocks. (Nodes marked in *yellow* in Figure 7)

3.3.5 Linear Representation. A final, optional step is to linearize the data structure. Because the data structure is static and read-only, we replace all pointers between parent and children nodes with 32-bit memory offsets. Linearized WorldPacks are stored in breadth-first order in a contiguous, unbroken block of memory.

Linearized WorldPacks can be transferred rapidly from disk to memory, and from CPU to GPU. The latter enables accelerated processing of computationally intensive algorithms like volume rendering, which is relevant for visualizing WorldPacks in our browser (see §4).

4 ACCELERATED BROWSING FUNCTIONALITY WITH WORLDPACKS

4.1 Spatiotemporal AABB Queries

Users may define query regions as AABB spacetime bounding boxes, after which the octree-based spatial hierarchy is used to perform efficient processing of WorldPack-AABB intersections. Processing a query uses *three steps* described in §4.1.1-§4.1.3 (see Figure 8).

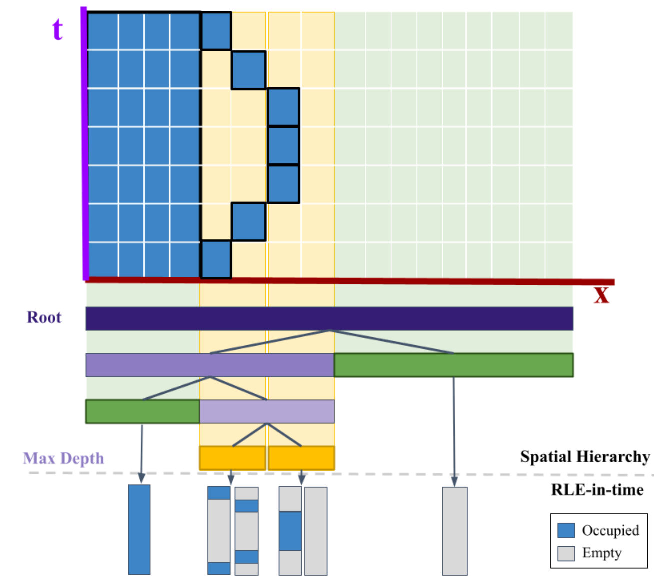


Fig. 7. **WorldPack Structure:** We show a WorldPack constructed for geometric occupancy on a domain with one spatial dimension, x, and a temporal dimension t. More complex domains are too large to be shown here, but even this toy example contains similar features to our examples: large areas of spatiotemporal redundancy interspersed with areas of distinctness. (Top) Both axes are partitioned into spacetime raster blocks; raster blocks occupied at $(\boldsymbol{x}, t)$ are shown in blue. Every raster block has an associated RLE-in-time sequence. (Bottom) We build a shallow, wide octree on the raster blocks. If any raster blocks within an internal octree node $O_{internal}$ have different RLE-in-time sequences, we split O into child nodes: these internal nodes are shown in purple. If O is at the hierarchy's $Depth_{max}$ and has any spatiotemporally distinct raster blocks, O is a leaf, and stores an RLE-in-time sequence for each of its raster blocks: such nodes O are shown in yellow. If O is not at the hierarchy's $Depth_{max}$, and all raster blocks in O have the same RLE-in-time sequence, then O does not branch and only stores a single RLE-in-time sequence for the entire region: such nodes $Depth_{max}$ are shown in green.

4.1.1 $AABB_{xyz}$/Octree Intersection (Step 1/3). First, we need to locate the WorldPacked data falling within the AABB spatial coordinates. Because WorldPack represents spatial data with an octree-inspired hierarchy, we can simply traverse the hierarchy in depth-first order to quickly collect all leaf octants that overlap with the AABB spatial bounds [Samet 2006]. WorldPacked data is laid out in a pointerless, linear fashion, so the traversal is performed using stored memory offsets from parent to children nodes.

As discussed in §3 and illustrated in Figure 7, each leaf octant O_{leaf} can either store a single RLE-in-time sequence if the entire region is spatiotemporally redundant (indicated by the green nodes in Figure 7), or an RLE-in-time sequence for each raster block within the region if O_{leaf} contains any spatiotemporally distinct raster blocks.

In the latter case, we check all individual raster blocks in O_{leaf} for intersection with the AABB spatial bounds, in case the AABB region only partially overlaps with the octant.

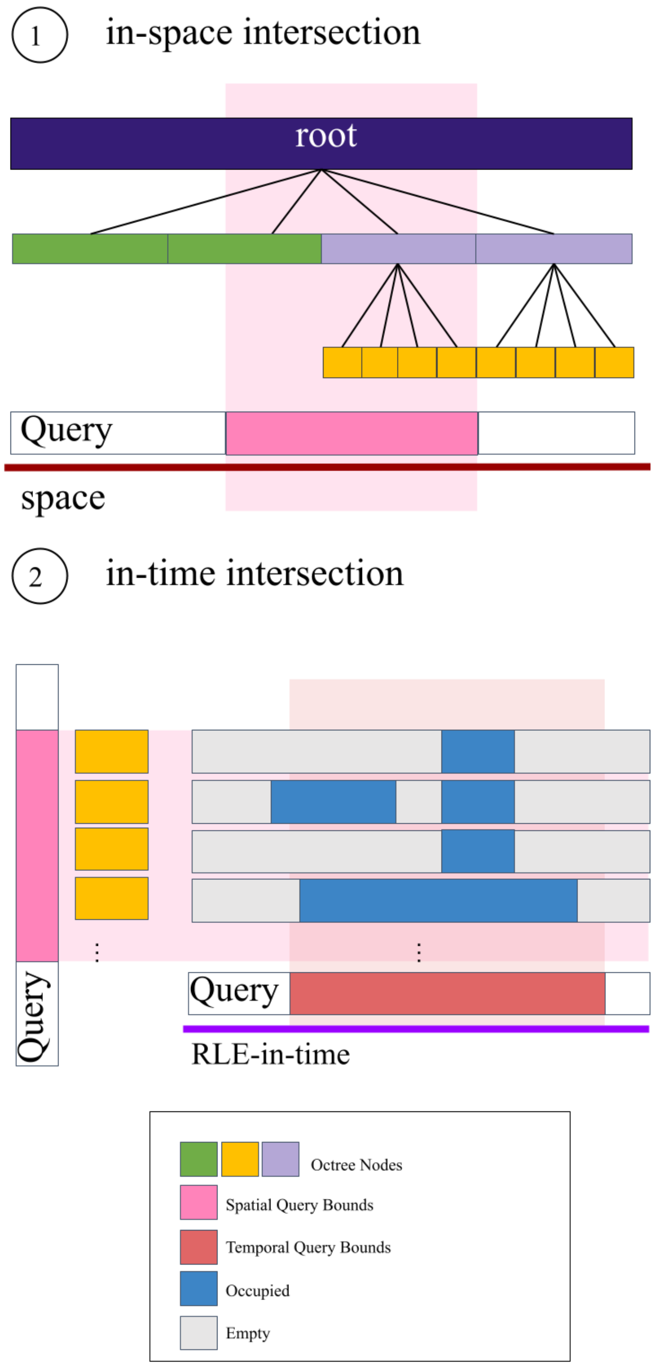


Fig. 8. **Query Evaluation on WorldPack :** Query evaluation is composed of three high-level steps. (Left) First, using a familiar AABB-octree intersection, we locate the leaf nodes in the WorldPack spatial hierarchy that lay within the spatial bounds of the query. (Right) Second, we find which RLE-in-time units stored at these leaf nodes fall within the query's temporal bounds. Finally, we apply the predicate to overlapped sequences.

4.1.2 AABB$_t$/RLE-in-time Intersection (Step 2/3). Once we have the leaf octants and raster blocks that are completely encapsulated by the spatial bounds, we need to find which parts of their full RLE-in-time sequences fall into the AABB region's temporal interval T_{query}. Every RLE-in-time <*count, value*> unit constitutes a single interval with length *count*. An RLE-in-time sequence is thus a series of intervals that, by construction, do not overlap with each other. We can treat the operation as a search for overlapping ranges between T_{query} and the RLE-in-time interval sequence and apply a 1D sweeping method. We might find overlapping ranges faster with, e.g., a non-overlapping interval tree, though, in practice, more advanced structures are unnecessary because the number of <*count, value*> units composing our RLE-in-time sequences is quite small; we report the average number of <*count, value*> units per raster block for WorldPacks storing geometric occupancy in Figure 9.

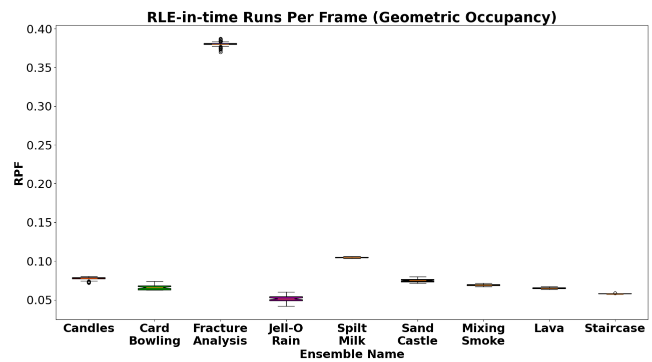


Fig. 9. We calculate RLE-in-time Runs-Per-Frame (RPF) as the number of RLE-in-time <*count, value*> runs per dense non-empty raster block, divided by the number of total frames compressed with RLE-in-time. We show the RPF for each WorldPack storing geometric occupancy data, and plot the average for each ensemble. As the boxplot shows, the RPF is quite small, generally around 0.1: given an RPF of 0.1 results in about 10x more compression than data not compressed with RLE-in-time. Because the number of runs per raster block is small, in-time processing generally occurs quite fast, and does not require advanced data structures.

4.1.3 Evaluate Predicate (Step 3/3). Once we have determined which RLE-in-time runs overlap with T_{query}, we process their *values* based on the query's predicate. These operations are conducted directly on the RLE-in-time compressed representation, without need for decompression: an RLE-in-time sequence implies that a *value* occurs at a raster block over *count* consecutive timesteps. For example, if the predicate asks to sum all *values* in spatiotemporal region R_{query}, perhaps for later ranking of ensemble samples by total density in R_{query}, we aggregate the ($count \times value$) products for all RLE-in-time runs within R_{query}:

$$sum_{block} = \sum_{i=0}^{\ell_{block}} n_i V_i \tag{3}$$

where ℓ_{block} is the length of the RLE-in-time sequence for a given raster block, n_i is the *count* of the i^{th} RLE unit, and V_i is the *value* of the i^{th} RLE unit. If the predicate instead performs an IN query for spatiotemporal region R_{query}, we check for RLE-in-time sequences in R_{query} with *value* 1, the single-bit indicator for occupancy:

$$IN: \max_{0\le i<\ell_{block}} V_i > 0, \tag{4}$$

$$NOT\ IN: \max_{0\le i<\ell_{block}} V_i == 0. \tag{5}$$

Early exits are used to speedup query evaluation. We refer readers to [Oyamada et al. 2018] and [Ding-tao et al. 2008] for a more detailed discussion of accelerated primitive operations on RLE sequences. Like Oyamada et al. [2018], we find that operating on RLE sequences is usually faster than iterating and processing individual values at every timestep; we show an ablation in Figure 10.

The overall speedup of query evaluation using our WorldPack data structure and WorldPack-AABB intersection algorithm is significant; most IN and NOT IN queries across region sizes, positions, and ensembles are processed within 10^{-3} seconds.

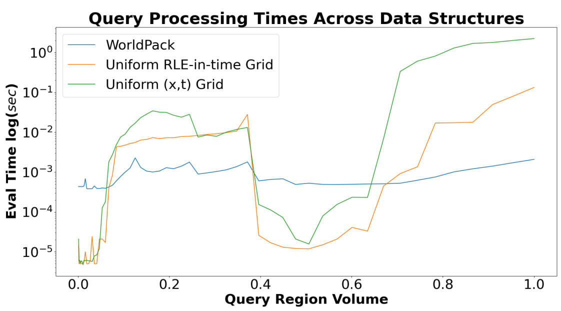


Fig. 10. **Query Processing** We measure the processing time for 64 IN queries on a sample from our Jell-O ensemble. Query spatial regions are all placed at the center of domain, and have increasing region spatial volume, shown on the x-axis. We evaluate these queries using three data structures: (a) WorldPack (blue), (b) a uniform RLE-in-time grid (orange) which is uses RLE-in-time compression but no spatial hierarchy, and (c) raw data (green), which uses a spacetime uniform grid but no spatial or temporal compression. We measure time in seconds and plot with a log-scale. As these query regions increase in size, WorldPacks keep a steady processing time due to their spatial hierarchy. The dip in processing times at a query volume of 0.4 is due to early exit: as the query regions grow, they eventually encapsulate some new geometric occupancy that allows us to terminate evaluation earlier.

4.2 WorldPack Rendering

As part of the browsing interface, users may explore 3D simulation samples via a navigable, rendered interface with the ability to pan, zoom, and rotate the camera around the scene. Users may compose and visualize spatiotemporal queries from within this scene.

To avoid dumping large amounts of per-timestep scene geometry for visualization, we chose to render the WorldPack data structure from a fragment shader using a simple ray tracer. WorldPack's in-time compression allows us to render the scene at any timestep from a *single* dumped file, enabling users to flexibly navigate the scene without having to extricate, dump, and store extraneous per-timestep simulation geometry. As we will discuss, WorldPack's spatial hierarchy, spatial sparsification, and support of low-precision attribute value storage are particularly beneficial for GPU ray casting and shading.

4.2.1 Attribute Value Encoding. For basic material shading, we will need to know for $I_\Omega(\boldsymbol{x}, t)$ (a) whether $I_\Omega(\boldsymbol{x}, t) == 1$ (contains geometric occupancy) and if so, (b) the diffuse color $color_\Omega(\boldsymbol{x}, t)$ and (c) the normal $N_\Omega(\boldsymbol{x}, t)$ of the object at $I_\Omega(\boldsymbol{x}, t)$. We need to export this data per raster tile in a low-memory, low-precision format for WorldPack to leverage spatiotemporal redundancy, while keeping just enough information to enable rendering with these values.

We make several simplifications to reduce the amount of data that needs to be WorldPacked. First, rather than storing a 32-bit RGBA color for $color_\Omega(\boldsymbol{x}, t)$, we instead rasterize an 8-bit color ID, which we use to lookup in, e.g., a hash table, the full RGBA color value. While this assumption gives us an upper bound of 255 separate materials in the scene, our examples (see §5) do not strain this limit. We parameterize $N_\Omega(\boldsymbol{x}, t)$ with two polar coordinates θ and ϕ. Both polar angles are binned into 0–255 buckets of size $2\pi/256$ radians, for a total of a 16 bits per normal.

We thus reduce shading information (occupancy, color, and normals) per-raster-block into a 24-bit unit for a given timestep. We pack these lower-precision units into our WorldPack format.

4.2.2 Ray casting. Our WorldPack rendering algorithm is similar to the volumetric rendering pipeline described by [Laine and Karras 2010], which similarly exploits the regular topology of spatial octrees for efficient ray casting. We therefore refer readers to [Laine and Karras 2010] for an in-depth description of ray casting voxel octrees on the GPU; in summary, we perform ray casts into the spatial hierarchy and incrementally traverse leaf nodes intersected by the ray in depth-first order. Once a ray hits geometry, we lookup shading attributes as inputs to a simple Phong shading model, and the traversal is complete.

The key difference between the GPU-accelerated ray casts described in [Laine and Karras 2010] and ours is the encoding of surface geometry. Laine and Karras [2010] use a timestep-specific octree–simulation data transferred to the GPU is from a single snapshot in time (see Figure 11) so leaf-node occupancy tests can be performed with a single lookup. In contrast, our WorldPacks use spatially hierarchical RLE-in-time: each leaf node contains an RLE sequence rather than a single occupancy value. So, we must also perform a temporal traversal: simply checking the RLE intervals for occupancy at the time-step that is currently being rendered. While the extra step adds some computation, we can avoid an expensive data transfer of a timestep-specific octree to the GPU for every frame. In our examples, the number of units per RLE-in-time sequence is small, and so the in-time traversal adds only a few extra shader operations; rendering RLE-in-time is negligibly more expensive than rendering a single frame. In Figure 12, we report the average number of <*count, value*> units per raster block for WorldPacks storing shading information quantized in the manner discussed in 4.2.1.

4.2.3 Optimizations. To speed up data access of an individual raster block's RLE-in-time data, we augment the WorldPack format with leaf node headers. For each raster block in the leaf node, headers record a 16-bit offset value, or bookmark, from the start of the node's data to the beginning of the raster tile's RLE-in-time sequence in

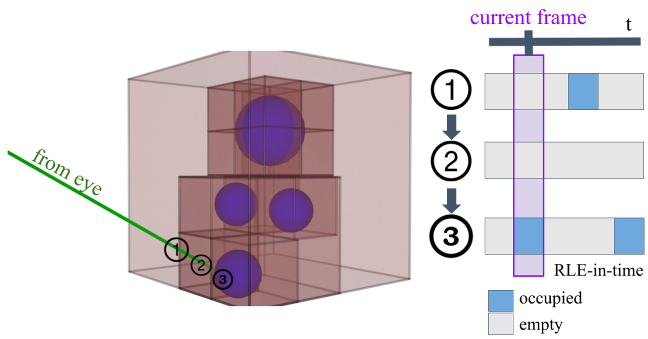


Fig. 11. **Ray Casting WorldPacks:** Rendering WorldPack is reminiscent of rendering a 3D octree. Similar to ray casting a purely spatial hierarchy, a ray is cast from the eye location into the octree and traverses from leaf node to leaf node until it either finds an occupied cell or exits the volume. However, the occupancy test requires decoding the RLE stored at each intersected leaf node to check for occupancy at the current timestep.

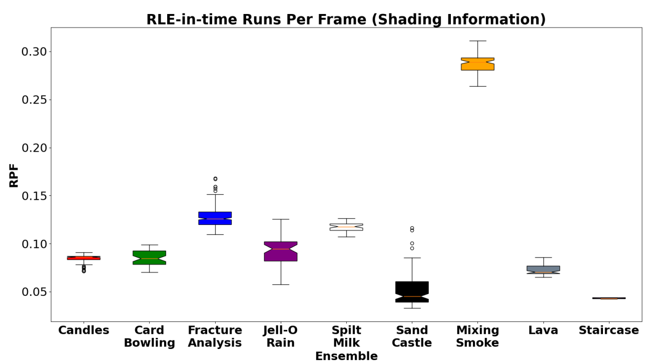


Fig. 12. We plot RPF for each WorldPack storing shading information, and plot the average for each ensemble. While the number of runs per raster block is higher in these WorldPacks than they were for WorldPacks storing geometric occupancy, it is still quite small: so, the in-time traversal adds only a few extra GPU shader operations.

the data stream. Each offset has a set size of 2 bytes; offsets are laid out in ZYX order. Given its location in 3D coordinates, the offset of any raster block is accessed in constant time by converting 3D coordinates to 1D. Rather than processing a stream of RLE-in-time data linearly to find the start of a raster block's block, we instead use the offset to jump directly to the block's position in the stream–also a constant-time operation.

5 RESULTS

We now describe our examples in more detail. Example statistics are shown in Table 1. Please see the accompanying video for all animated results.

Implementation Details. Our approach involves multiple components: (1) physics-based simulation, (2) WorldPack exporter, (3) interactive browser with GPU renderer. To support our claim of arbitrary simulation models, we created a diverse assortment of examples using a wide range of dynamics solvers in SideFX Houdini v18.5 (example-specific details are below). We exported WorldPacks straight from Houdini with a Python-based implementation. Our runtime browser is implemented in Javascript and runs via a local server; because they are quite lightweight, an ensemble's entire set of sample-specific WorldPacks storing geometric occupancy can be stored comfortably in memory. This avoids the need for expensive file IO of WorldPacks when evaluating user queries; WorldPacks can just be loaded once at the start of the browsing session. WorldPacks that store shading information are loaded on an as-needed basis: if the browser needs to display a new simulation sample, we load the sample's shading WorldPack and send it to the GPU packed into a texture2D. We render WorldPacks with shaders built using WebGL. The browser evaluates user queries at interactive rates, updating the set of feasible samples immediately as the user inserts or drags query regions or adjusts predicates. All feasible samples are shown as representative images within a scrollable menu on the right-hand side of the GUI (see video). Users examine an interactive 3D view of individual WorldPacks by clicking on their representative image. Users can avoid manually examining each feasible sample by ranking using other metrics; we show examples in Figures 14 and 15.

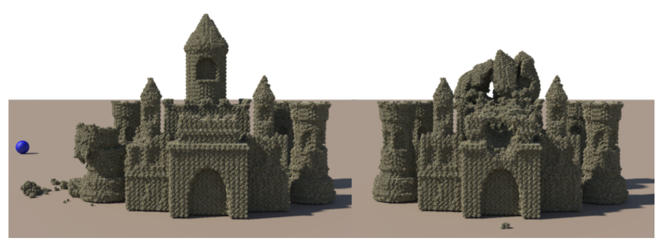


Fig. 13. **Smashing Sand Castles:** A sandbox ensemble is created by launching a ball with randomly chosen starting positions and velocities at a sandcastle fashioned out of wet grain. Using exploratory queries we can effectively say (Left) "give me samples where the *front-left turret* is smashed by the final timestep," or (Right) the *top turret*.

Smashing Sand Castles (Figure 13). Our rasterization-based approach supports a wide range of models, including this sand-grain simulation example (simulated in Houdini using 180,000 grains). By applying appropriate queries, the animator can find scenarios where particular turrets are hit or not hit by a thrown object.

Card Bowling. WorldPacks can be exported for a single simulated subject, such as lava or Jell-O®, or for several. In this example, we sample an ensemble where a 20-sided die with randomly-sampled initial position and velocity is launched at 10 card "pins" arranged like traditional Candlepin bowling. Dynamics are simulated in Houdini using the Bullet rigid-body solver; each bowling pin is comprised of 585 cards. Queries can be applied to one subject, like the die, or to several in combination, with subject-specific predicates on both the dice and the cards to narrow down the ensemble to find interesting bowling strikes (see Figure 1).

Table 1. **Example Statistics:** For each ensemble, we report the number of ensemble simulation samples, the number of timesteps, the number of input parameters that are randomly varied to create the ensemble. We list the T_{max} chosen for WorldPacks storing geometric occupancy and shading information; we choose T_{max} at a finer resolution for the latter. We also report the average size of each ensemble's WorldPacks in MB. All WorldPacks used for evaluating user queries are $128 \times 128 \times 128$ raster blocks in spatial resolution, with a spatial hierarchy depth of 4. All WorldPacks displayed using hardware-accelerated raycasting in our browser's interface are $256 \times 256 \times 256$ raster blocks in resolution, with a spatial hierarchy depth of 6. Finally, we report a rounded average serial construction time for constructing a) each WorldPack individually, and b) each entire ensemble; the majority of the time is incurred by solvers, not WorldPack construction. Note that the Staircase ensemble (final row) utilized several rounds of ensemble refinement, so we report statistics for the final stage of refinement, but the total number of samples collected from the entire refinement process.

Example	Samples	Num Timesteps	Num Params	T_{max} (Occ.;Shade)	Avg WorldPack Size, MB (Occ.;Shade)	Avg Construction Time (min/sample; min/dataset)
Sand Castle	104	70	4	5 ; 3	0.089 ; 1.692	25 ; 1040
Card Bowling	51	155	4	10 ; 5	0.145 ; 2.190	15 ; 750
Fracture	300	24	1	3 ; 2	0.075 ; 5.526	5 ; 1500
Jello Rain	46	349	1	10 ; 5	0.321 ; 13.04	20 ; 920
Smoke	99	164	1	10 ; 10	0.093 ; 3.083	10 ; 990
Candles	109	70	6	5 ; 3	0.259 ; 4.050	5 ; 545
Lava Village	72	154	10	10 ; 10	0.103 ; 3.739	20 ; 1440
Spilt Milk	46	94	3	10 ; 10	0.088 ; 3.871	30 ; 1380
Staircase	484	92	6	5 ; 3	0.093 ; 1.861	3 ; 1452

Browsing Jell-O® Brand Gelatin (Figure 1). We sampled an ensemble of 13 randomly oriented Jell-O cubes falling into a bowl, with each cube simulated as a tetrahedral mesh (5530 tets, 1406 verts.) in Houdini Vellum. Unfortunately some pieces fly out of the bowl. To eliminate simulation samples where Jell-O pieces fell out of the bowl at any time, a user places a large NOT IN query region under the bowl: this constraint allows only samples that do not have geometric occupancy in the selected region.

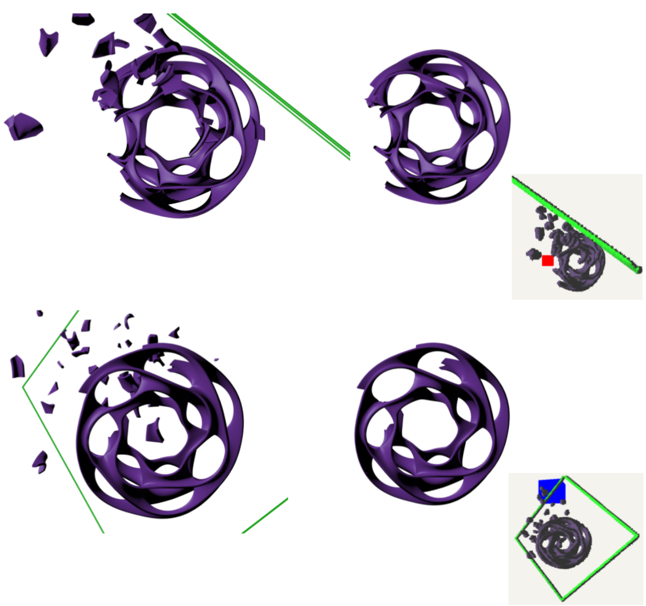


Fig. 14. **Fractured Heptoroid:** We perform fracture analyses on a heptoroid by subjecting it to randomly sampled planar impacts and rasterize WorldPacks in the body-space of the main object. The user can place a NOT IN query acting on the main object over the last few timesteps (Top), to find samples with a specific impact response: "in which samples did *this* part of the heptoroid fracture?" A different part of the outcome space is explored (Bottom) by placing an IN query acting on the tiny detritus and ranking samples to find outcomes with the most debris in the query region.

Material-Space Browsing of Fracture (Figure 14). The WorldPack RLE-in-time representation enables us to evaluate spatiotemporal queries over simulation domains at interactive rates. On our largest ensemble, we perform 300 fracture analyses on a heptoroid by subjecting it to random planar impacts, and rendering collected WorldPacks in the body space of the main fragment. By dragging IN and NOT IN queries around both the main object and the shattered detritus, an analyst can quickly sift through the ensemble, and explore different impact responses and degrees of destruction. Metrics calculated on-the-fly from WorldPacks, like density integrals, help further sift through samples, such as allowing an analyst to rank samples by the density of fragmented material in a query region.

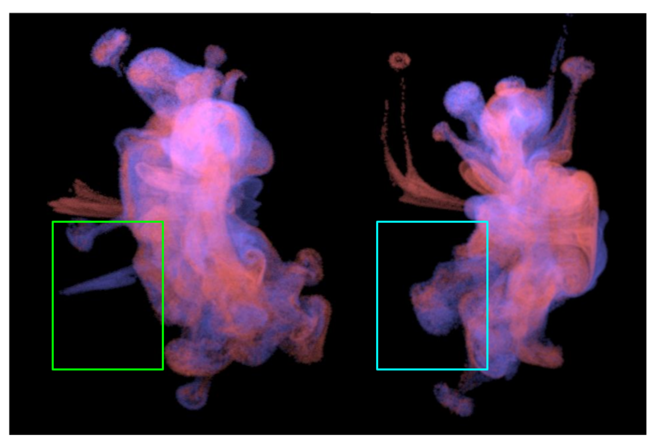


Fig. 15. **Mixing Smoke** a ranking query executed within the lower part of the volume (with spatial bounds shown from an orthographic view in blue/green) orders simulations by the amount of purple smoke within the query region. The integral is calculated on-the-fly from WorldPacked color data. We show the samples with (Left) the lowest mixing–notice the single blue vortex within the query bounds–and (Right) the highest mixing.

Mixing Smoke (Figure 15). We recreated the colliding red teapot and blue bunny smoke objects using the simulator in [Chern et al.

2016] (their Fig. 6 example) in Houdini (128^3 grid, $\hbar = 0.0451$, ±2 jet speeds). We built an ensemble by randomly sampling the rotational orientation of one of the objects and rasterizing smoke occupancy and color; as listed in Table 1, our WorldPacks resolutions match the resolution of the solver grid without need for aggressive spatial coarsening while maintaining reasonable memory footprints, and explored interesting variations and vortical structures. We could also explore the degree of smoke mixing by browsing and ranking based on the amount of purple smoke generated by a simple diffusive mixing process.

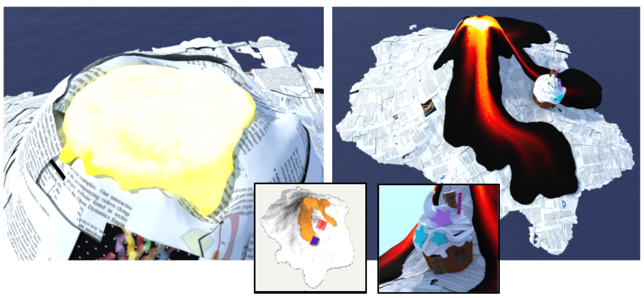


Fig. 16. **Browsing Lava Flows:** (Left) We make an ensemble by varying volcano vent deformations. (Right) Combining IN and NOT IN queries allows the user to find a simulation sample where the lava flow branches in an interesting way, and uses it to create a suspenseful animation: a house in the shadow of the volcano is narrowly missed by an eruption's lava.

Lava Village (Figures 3 & 16). We sampled an ensemble of volcanic eruptions by varying parameters for a volcano's vent shape that affect the amount of flow and angular direction (see Figure 16). As an example design goal, the animator explores where to place a small house so it is safe from a volcano eruption. Unfortunately, given the scale of this viscous FLIP-based fluid animation in Houdini, the sheer amount of simulation data is difficult to load, inspect, and query. As an estimate, dumping per-timestep snapshots of lava geometry in VDB format produces roughly 1.5 GB of data for each simulation sample; more than 100GB for a 72-sample ensemble. We use this number not to compare our functionality with VDB–the authors make design decisions with a different use case in mind–but to highlight the intractability of exploring and querying multiple simulation samples using current, popular data structures. In contrast, our 72 WorldPacks storing geometric occupancy can fit in a total of 7.2 MB (see Table 1): comfortably in a web browser cache. Naïvely evaluating queries to find samples where lava does not flow through candidate house locations is expensive and can not be computed at interactive rates on this much data. Using the UMWB interface, the animator can easily find locations where the house is hit (IN query) or not hit (NOT IN query) by the lava.

Spiral Staircase (Figure 17). Inspired by the articulated rigid-body character falling down the stairs using ensemble refinement in [Twigg and James 2007], we consider an animation where a squishy armadillo falls all the way down a spiral staircase. It is quickly apparent that this unlikely behavior is not present in the initial ensemble: no samples reach the bottom stair. Instead, we create an ensemble

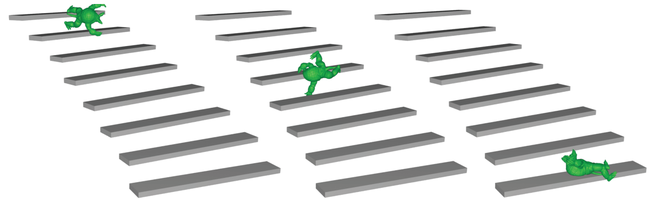


Fig. 17. **Don't Fall Off!** UMWB enables ensemble steering to identify needle-in-the-haystack scenarios. The animator's goal is to generate an animation where a softbody character falls all the way down a spiral staircase without falling off. This unlikely scenario is not present in the first ensemble, so we instead identify a promising sample using the GUI, narrows down the sampling range to generate character motions from there, and repeat the process to steer the ensemble towards the desired result. The figure shows stills from the final outcome.

in which the character is subjected to small, randomly applied impulses that hope to keep it from preemptively tumbling off the edge. The user identifies a promising outcome using the GUI, while easily discarding samples where the character has fallen off the stairs or is too close to the edge. The user then manually narrows the sampling range around the chosen outcome's parameters, and generates a new ensemble of character motions starting from here, and repeats the process. In situ ensemble refinement remains challenging for many multi-physics animations due to high simulation costs, but parallel simulator-in-the-loop refinement is likely in subsequent works. The final result is shown in Figure 17.

Fig. 18. **Candles:** We create an ensemble of candles lit by a flurry of particle-system sparks (Top). IN and NOT IN queries executed on the candle wicks can find samples where candles ignite or didn't ignite. Such functionality allows the animator to explore and inspect different possible combinations of lit and unlit candles (Bottom) without needing to manually tune parameters of the particle system.

Candles (Figure 18). In this ensemble, we randomly perturb the noise, size, and trajectory of sparks to light a set of candles. We model the sparks as a turbulent particle system. At the start of the simulation, candles wicks are coated in fuel. They require a significant number of spark collisions to overcome a temperature threshold for lighting, which is difficult to control due to the noisiness of the spark particle system. We simulate fire spread in Houdini using the Pyro Source Spread solver, and we WorldPack sufficiently hot particles. Applying queries to candle wicks can search for samples where candles–or combinations of candles–did or did not kindle.

Spilt Milk (Figure 19). In this FLIP-based fluid ensemble, a cup of milk is spilled on a table setting by varying the cup's angular direction, its milk fullness, and its table position. Dragging an IN query along the table, an animator can ask: "show me samples where milk has spilled furthest down the table." Including NOT IN queries adds more specificity, such as "show me samples where milk has spilled furthest down the table, *given* that selected place settings stays dry" a means for interactive investigation of "what if?" scenarios.

6 CONCLUSION

We have introduced a practical approach for exploring large datasets of arbitrary physics-based animations, with the following components: a) our WorldPack representation, a unified volumetric for spatiotemporally compressed animation state optimized for data browsing, b) a query-based browsing interface that responds interactively to user input, and c) demonstrations of results on a diverse assortment of animation phenomena.

6.1 Limitations and Future Work

Most of the design decisions made in our development of WorldPack were made under the assumption that we could find and exploit spatiotemporal redundancy in ensemble data. As we have shown, many simulated phenomena in computer animation (multibody dynamics, fluids, fracture, etc.) exhibit suitable redundancy: simply empty space, or repeated values, e.g., relatively constant temperature values in a spatiotemporal location, consistent normals on a piece of barely-moving scene geometry, etc. However, for simulated domains with large amounts of noise-like variations, e.g., snowstorms or sandstorms, WorldPack's RLE-in-time compression will suffer (may even increase sizes), and more sophisticated lossless (or lossy) compression schemes may be required in such cases. Although most of our ensemble animations are short, RLE-in-time compression incurs per-octree-cell decoding overhead in the GPU ray traversal that can increase for longer or "noisier" animations.

We have used fast spacetime-AABB *vs.* WorldPack intersections for interactive queries, but more general spacetime queries are possible by encoding the query region using an WorldPack and performing more expensive WorldPack-WorldPack intersection tests.

Interactive ensemble refinement, which was possible for rigid-body simulation-in-the-loop in [Twigg and James 2007], remains a challenge for many multi-physics animations due to higher simulation costs and the lack of simulator integration in our prototype UMWB system. In addition, our WorldPack representation is immutable, and not designed to support simulation-in-the-loop ensemble refinement.

Direct GPU-accelerated visualization of the WorldPack uniform raster geometry is fast in practice, but some animation tasks may require higher visual fidelity. Alternatives could include adaptive grids, or non-raster and hybrid geometric representations.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for constructive feedback; Kangrui Xue, Sofia Wyetzner, and Vishnu Sarukkai for proof reading; Christopher Twigg for MWB-era discussions on fluid browsing. We thank SideFX for donating Houdini licenses which were helpful in producing examples and renders. We thank Gianluca Iaccarino, Javier Urzay, and other Stanford INSIEME group members for project feedback. This material is based upon work supported by the Department of Energy, National Nuclear Security Administration under Award Number DE-NA0003968.

REFERENCES

Mridul Aanjaneya, Ming Gao, Haixiang Liu, Christopher Batty, and Eftychios Sifakis. 2017. Power Diagrams and Sparse Paged Grids for High Resolution Adaptive Liquids. *ACM Trans. Graph.* 36, 4, Article 140 (July 2017), 12 pages. https://doi.org/10.1145/3072959.3073625

Okan Arikan. 2006. Compression of Motion Capture Databases. In *ACM SIGGRAPH 2006 Papers.* 890–897.

Jernej Barbič and Jovan Popović. 2008. Real-Time Control of Physically Based Simulations Using Gentle Forces. *ACM Trans. Graph.* 27, 5, Article 163 (dec 2008), 10 pages. https://doi.org/10.1145/1409060.1409116

Ronen Barzel, John R Hughes, and Daniel N Wood. 1996. Plausible motion simulation for computer graphics animation. In *Computer Animation and Simulation'96.* Springer, 183–197.

Miklós Bergou, Saurabh Mathur, Max Wardetzky, and Eitan Grinspun. 2007. TRACKS: Toward Directable Thin Shells. *ACM Trans. Graph.* 26, 3 (jul 2007), 50–es. https://doi.org/10.1145/1276377.1276439

Stefan Bruckner and Torsten Möller. 2010. Result-Driven Exploration of Simulation Parameter Spaces for Visual Effects Design. *IEEE Transactions on Visualization and Computer Graphics* 16, 6 (2010), 1468–1476. https://doi.org/10.1109/TVCG.2010.190

Stephen Chenney and D. A. Forsyth. 2000. Sampling Plausible Solutions to Multi-Body Constraint Problems. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '00).* ACM Press/Addison-Wesley Publishing Co., USA, 219–228. https://doi.org/10.1145/344779.344882

Albert Chern, Felix Knöppel, Ulrich Pinkall, Peter Schröder, and Steffen Weißmann. 2016. Schrödinger's smoke. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 1–13.

Michael F. Cohen. 1992. Interactive Spacetime Control for Animation. In *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '92).* Association for Computing Machinery, New York, NY, USA, 293–302. https://doi.org/10.1145/133994.134083

Brian Curless and Marc Levoy. 1996. A Volumetric Method for Building Complex Models from Range Images. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '96).* Association for Computing Machinery, New York, NY, USA, 303–312. https://doi.org/10.1145/237170.237269

Shen Ding-tao, Cui Can, and Wang Jie-chen. 2008. Implementation and Application of Intersection Operation Based on Run-Length Encoding. In *2008 International Conference on Computer Science and Software Engineering*, Vol. 4. 602–606. https://doi.org/10.1109/CSSE.2008.1197

Raanan Fattal and Dani Lischinski. 2004. Target-driven Smoke Animation. In *ACM SIGGRAPH 2004 Papers.* 441–448.

N. Foster and D. Metaxas. 1997. Controlling Fluid Animation. In *Proceedings Computer Graphics International.* 178–188. https://doi.org/10.1109/CGI.1997.601299

Jie Guo, Mengtian Li, Zijing Zong, Yuntao Liu, Jingwu He, Yanwen Guo, and Ling-Qi Yan. 2021. Volumetric Appearance Stylization with Stylizing Kernel Prediction Network. *ACM Trans. Graph.* 40, 4, Article 162 (July 2021), 15 pages. https://doi.org/10.1145/3450626.3459799

Paul S. Heckbert. 1987. Ray Tracing Jell-O® Brand Gelatin. *SIGGRAPH Comput. Graph.* 21, 4 (aug 1987), 73–74. https://doi.org/10.1145/37402.37411

Ben Houston, Michael B. Nielsen, Christopher Batty, Ola Nilsson, and Ken Museth. 2006. Hierarchical RLE Level Set: A Compact and Versatile Deformable Surface Representation. *ACM Trans. Graph.* 25, 1 (Jan. 2006), 151–175. https://doi.org/10.1145/1122501.1122508

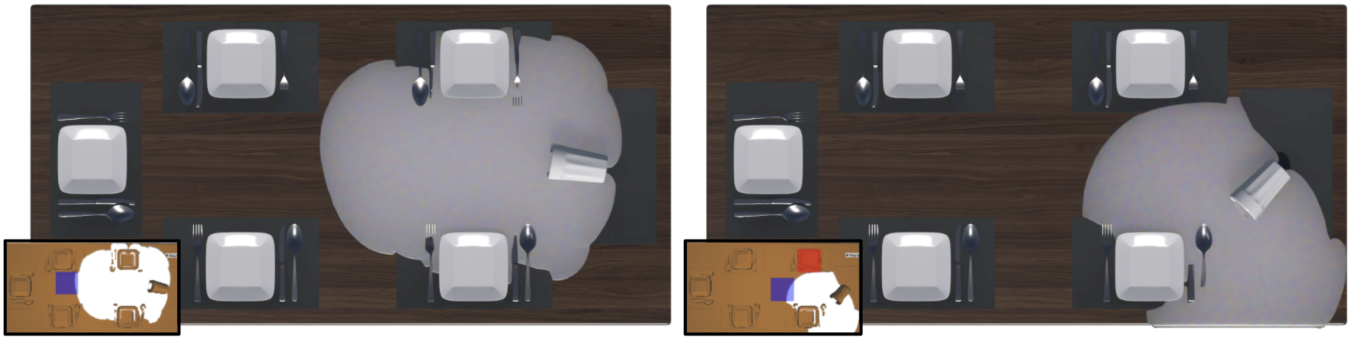


Fig. 19. **There's no use crying over it!** A cup of milk is spilt on a table setting in several ways with varied start conditions. (Left) An IN query on geometric occupancy of the milk can be dragged across the length of the table to find samples where milk spills the farthest, while (Right) a combination of IN and NOT IN queries can ask more nuanced questions, such as finding samples where the milk spills furthest, subject to the constraint that certain areas remain dry.

Ben Houston, Mark Wiebe, and Chris Batty. 2004. RLE Sparse Level Sets. In *ACM SIGGRAPH 2004 Sketches*. 137.

Yuanming Hu, Jiafeng Liu, Xuanda Yang, Mingkuan Xu, Ye Kuang, Weiwei Xu, Qiang Dai, William T. Freeman, and Frédo Durand. 2021. QuanTaichi: A Compiler for Quantized Simulations. *ACM Trans. Graph.* 40, 4, Article 182 (jul 2021), 16 pages. https://doi.org/10.1145/3450626.3459671

Timothy Jeruzalski, John Kanji, Alec Jacobson, and David I.W. Levin. 2018. Collision-Aware and Online Compression of Rigid Body Simulations via Integrated Error Minimization. *Computer Graphics Forum (Proc. SCA)* (2018).

Aaron Demby Jones, Pradeep Sen, and Theodore Kim. 2016. Compressing Fluid Subspaces. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 77–84.

Byungsoo Kim, Vinicius C. Azevedo, Markus Gross, and Barbara Solenthaler. 2019. Transport-Based Neural Style Transfer for Smoke Simulations. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 188.

Byungsoo Kim, Vinicius C. Azevedo, Markus Gross, and Barbara Solenthaler. 2020. Lagrangian Neural Style Transfer for Fluids. *ACM Transactions on Graphics* 39, 4, Article 52 (2020), 10 pages. https://doi.org/10.1145/3386569.3392473

Yuki Koyama, Issei Sato, and Masataka Goto. 2020. Sequential Gallery for Interactive Visual Design Optimization. *ACM Trans. Graph.* 39, 4, Article 88 (July 2020), 12 pages. https://doi.org/10.1145/3386569.3392444

Samuli Laine and Tero Karras. 2010. Efficient Sparse Voxel Octrees. In *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (Washington, D.C.) *(I3D '10)*. Association for Computing Machinery, New York, NY, USA, 55–63. https://doi.org/10.1145/1730804.1730814

Timothy R. Langlois and Doug L. James. 2014. Inverse-Foley Animation: Synchronizing rigid-body motions to sound. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2014)* 33, 4 (Aug. 2014). https://doi.org/10.1145/2601097.2601178

John Lasseter. 1987. Principles of Traditional Animation Applied to 3D Computer Animation. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '87)*. Association for Computing Machinery, New York, NY, USA, 35–44. https://doi.org/10.1145/37401.37407

Jerome Edward Lengyel. 1999. Compression of Time-dependent Geometry. In *Proceedings of the 1999 symposium on Interactive 3D graphics*. 89–95.

Pingchuan Ma, Yunsheng Tian, Zherong Pan, Bo Ren, and Dinesh Manocha. 2018. Fluid Directed Rigid Body Control Using Deep Reinforcement Learning. *ACM Trans. Graph.* 37, 4, Article 96 (July 2018), 11 pages. https://doi.org/10.1145/3197517.3201334

J. Marks, B. Andalman, P. A. Beardsley, W. Freeman, S. Gibson, J. Hodgins, T. Kang, B. Mirtich, H. Pfister, W. Ruml, K. Ryall, J. Seims, and S. Shieber. 1997. Design Galleries: A General Approach to Setting Parameters for Computer Graphics and Animation. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '97)*. ACM Press/Addison-Wesley Publishing Co., USA, 389–400. https://doi.org/10.1145/258734.258887

Antoine McNamara, Adrien Treuille, Zoran Popović, and Jos Stam. 2004. Fluid Control Using the Adjoint Method. *ACM Trans. Graph.* 23, 3 (Aug. 2004), 449–456. https://doi.org/10.1145/1015706.1015744

Ken Museth. 2011. DB+Grid: A Novel Dynamic Blocked Grid for Sparse High-Resolution Volumes and Level Sets. In *ACM SIGGRAPH 2011 Talks* (Vancouver, British Columbia, Canada) *(SIGGRAPH '11)*. Association for Computing Machinery, New York, NY, USA, Article 51, 1 pages. https://doi.org/10.1145/2037826.2037894

Ken Museth. 2013. VDB: High-Resolution Sparse Volumes with Dynamic Topology. *ACM Trans. Graph.* 32, 3, Article 27 (July 2013), 22 pages. https://doi.org/10.1145/2487228.2487235

Ken Museth. 2021. NanoVDB: A GPU-Friendly and Portable VDB Data Structure For Real-Time Rendering And Simulation. In *ACM SIGGRAPH 2021 Talks* (Virtual Event, USA) *(SIGGRAPH '21)*. Association for Computing Machinery, New York, NY, USA, Article 1, 2 pages. https://doi.org/10.1145/3450623.3464653

Jelani Jelani Osei Nelson. 2011. *Sketching and streaming high-dimensional vectors.* Ph.D. Dissertation. Massachusetts Institute of Technology.

Michael B. Nielsen and Ken Museth. 2006. Dynamic Tubular Grid: An Efficient Data Structure and Algorithms for High Resolution Level Sets. *J. Sci. Comput.* 26, 3 (March 2006), 261–299. https://doi.org/10.1007/s10915-005-9062-8

Michael B. Nielsen, Konstantinos Stamatelos, Morten Bojsen-Hansen, Duncan Brinsmead, Yannick Pomerleau, Marcus Nordenstam, and Robert Bridson. 2018. A Collocated Spatially Adaptive Approach to Smoke Simulation in Bifrost. In *ACM SIGGRAPH 2018 Talks* (Vancouver, British Columbia, Canada) *(SIGGRAPH '18)*. Association for Computing Machinery, New York, NY, USA, Article 77, 2 pages. https://doi.org/10.1145/3214745.3214749

Carol O'Sullivan, John Dingliana, Thanh Giang, and Mary K. Kaiser. 2003. Evaluating the Visual Fidelity of Physically Based Animations. *ACM Trans. Graph.* 22, 3 (jul 2003), 527–536. https://doi.org/10.1145/882262.882303

Masafumi Oyamada, Jianquan Liu, Shinji Ito, Kazuyo Narita, Takuya Araki, and Hiroyuki Kitagawa. 2018. Compressed Vector Set: A Fast and Space-Efficient Data Mining Framework. *Journal of Information Processing* 26 (2018), 416–426.

Zherong Pan, Jin Huang, Yiying Tong, Changxi Zheng, and Hujun Bao. 2013. Interactive Localized Liquid Motion Editing. *ACM Trans. Graph.* 32, 6, Article 184 (nov 2013), 10 pages. https://doi.org/10.1145/2508363.2508429

Jovan Popović, Steven M Seitz, and Michael Erdmann. 2003. Motion sketching for control of rigid-body simulations. *ACM Transactions on Graphics (TOG)* 22, 4 (2003), 1034–1054.

Jovan Popović, Steven M. Seitz, Michael Erdmann, Zoran Popović, and Andrew Witkin. 2000. Interactive Manipulation of Rigid Body Simulations. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '00)*. ACM Press/Addison-Wesley Publishing Co., USA, 209–217. https://doi.org/10.1145/344779.344880

Hanan Samet. 2006. *Foundations of multidimensional and metric data structures.* Morgan Kaufmann.

Syuhei Sato, Yoshinori Dobashi, and Theodore Kim. 2021. Stream-Guided Smoke Simulations. *ACM Trans. Graph.* 40, 4, Article 161 (July 2021), 7 pages. https://doi.org/10.1145/3450626.3459846

Mirko Sattler, Ralf Sarlette, and Reinhard Klein. 2005. Simple and Efficient Compression of Animation Sequences. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Los Angeles, California) *(SCA '05)*. Association for Computing Machinery, New York, NY, USA, 209–217. https://doi.org/10.1145/1073368.1073398

Khalid Sayood. 2017. *Introduction to Data Compression.* Morgan Kaufmann.

Arnaud Schoentgen, Pierre Poulin, Emmanuelle Darles, and Philippe Meseure. 2020. *Particle-Based Liquid Control Using Animation Templates.* Eurographics Association, Goslar, DEU. https://doi.org/10.1111/cgf.14103

Rajsekhar Setaluri, Mridul Aanjaneya, Sean Bauer, and Eftychios Sifakis. 2014. SPGrid: A Sparse Paged Grid Structure Applied to Adaptive Smoke Simulation. *ACM Trans. Graph.* 33, 6, Article 205 (Nov. 2014), 12 pages. https://doi.org/10.1145/2661229.

2661269

Lin Shi and Yizhou Yu. 2005. Controllable Smoke Animation with Guiding Objects. *ACM Transactions on Graphics* 24 (01 2005). https://doi.org/10.1145/1037957.1037965

Evan Shimizu, Matthew Fisher, S. Paris, J. McCann, and K. Fatahalian. 2020. Design Adjectives: A Framework for Interactive Model-Guided Exploration of Parameterized Design Spaces. *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology* (2020).

SideFX. 2021. Houdini Engine. http://www.sidefx.com.

Jerry O. Talton, Daniel Gibson, Lingfeng Yang, Pat Hanrahan, and Vladlen Koltun. 2009. Exploratory Modeling with Collaborative Design Spaces. *ACM Trans. Graph.* 28, 5 (Dec. 2009), 1–10. https://doi.org/10.1145/1618452.1618513

Diane Tang, J Thomas Ngo, and Joe Marks. 1995. N-body Spacetime Constraints. *The Journal of Visualization and Computer Animation* 6, 3 (1995), 143–154.

N. Thürey, R. Keiser, M. Pauly, and U. Rüde. 2006. Detail-Preserving Fluid Control. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Vienna, Austria) *(SCA '06)*. Eurographics Association, Goslar, DEU, 7–12.

Adrien Treuille, Antoine McNamara, Zoran Popović, and Jos Stam. 2003. Keyframe Control of Smoke Simulations. *ACM Trans. Graph.* 22, 3 (jul 2003), 716–723. https://doi.org/10.1145/882262.882337

Christopher D. Twigg and Doug L. James. 2007. Many-Worlds Browsing for Control of Multibody Dynamics. *ACM Trans. Graph.* 26, 3 (July 2007), 14–es. https://doi.org/10.1145/1276377.1276395

Andrew Witkin and Michael Kass. 1988. Spacetime Constraints. In *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '88)*. Association for Computing Machinery, New York, NY, USA, 159–168. https://doi.org/10.1145/54852.378507

Chris Wojtan, Peter J Mucha, and Greg Turk. 2006. Keyframe control of complex particle systems using the adjoint method. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*. 15–23.

Guowei Yan, Zhili Chen, Jimei Yang, and Huamin Wang. 2020. Interactive Liquid Splash Modeling by User Sketches. *ACM Trans. Graph.* 39, 6, Article 165 (Nov. 2020), 13 pages. https://doi.org/10.1145/3414685.3417832

Thomas Y Yeh, Glenn Reinman, Sanjay J Patel, and Petros Faloutsos. 2009. Fool me twice: Exploring and exploiting error tolerance in physics-based animation. *ACM Transactions on Graphics (TOG)* 29, 1 (2009), 1–11.