Temporal Set Inversion for Animated Implicits

KAVOSH JAZAR, McGill University, Canada
PAUL G. KRY, McGill University, Canada

Fig. 1. Snapshots of an animated implicit surface at timesteps 68, 150, 265 and 330 showing in red the regions that were evaluated in each timestep to sparsely update a 1024^3 voxelization of the scene only as and where is necessary to maintain bounded error in the surface. For comparison, existing approaches would color the entire surface red in each timestep, thus wasting resources in static or slowly-evolving areas far from the motion. The runtime of our method depends on the moving surface area per frame and achieves here a $\sim 3\times$ speedup compared to full per-frame re-evaluation at a temporal error tolerance of $\delta = 0.01$.

We exploit the temporal coherence of closed-form animated implicit surfaces by locally re-evaluating an octree-like discretization of the implicit field only as and where is necessary to rigorously maintain a global error invariant over time, thereby saving resources in static or slowly-evolving areas far from the motion where per-frame updates are not necessary. We treat implicit surface rendering as a special case of the continuous constraint satisfaction problem of set inversion, which seeks preimages of arbitrary sets under vector-valued functions. From this perspective, we formalize a temporally-coherent set inversion algorithm that localizes changes in the field by range-bounding its time derivatives using interval arithmetic. We implement our algorithm on the GPU using persistent thread scheduling and apply it to the scalar case of implicit surface and swept volume rendering where we achieve significant speedups in complex scenes with localized deformations like those found in games and modelling applications where interactivity is required and bounded-error approximation is acceptable.

CCS Concepts: • **Computing methodologies** → **Rendering**; **Animation**; *Volumetric models*; *Massively parallel algorithms*; Continuous space search; • **Mathematics of computing** → *Interval arithmetic*; *Automatic differentiation*; • **Information systems** → Temporal data; Uncertainty.

Additional Key Words and Phrases: implicit surface, signed distance field, sdf, temporal coherence, sparse voxel octree, root-finding, branch-and-bound, global optimization, nonlinear optimization, constraint satisfaction, error analysis, subpaving, subdivision, isosurface, differentiable programming

ACM Reference Format:
Kavosh Jazar and Paul G. Kry. 2023. Temporal Set Inversion for Animated Implicits. *ACM Trans. Graph.* 42, 4, Article 1 (August 2023), 18 pages. https://doi.org/10.1145/3592448

Authors' addresses: Kavosh Jazar, cryvosh@gmail.com, McGill University, Montreal, Canada; Paul G. Kry, McGill University, Montreal, Canada, kry@cs.mcgill.ca.

0730-0301/2023/8-ART1 $15.00
https://doi.org/10.1145/3592448

1 INTRODUCTION

An animated implicit is defined by a field function $f : \mathbb{R}^n \to \mathbb{R}$ parameterized at time t by a configuration-space point $\boldsymbol{C}(t) \in \mathbb{R}^\tau$. The roots of f form the surface $\mathbb{S}$ of some volume in $\mathbb{R}^n$ and act as the boundary between filled regions where the field is negative and empty regions where the field is positive. More formally,

$$\mathbb{S} = \{\boldsymbol{p} \in \mathbb{R}^n \mid f(\boldsymbol{p}, \boldsymbol{C}(t)) \stackrel{\text{def}}{=} f_t(\boldsymbol{p}) = 0\} = f_t^{-1}(\{0\}). \tag{1}$$

The expressive power of this representation allows it to concisely model unusual topological transformations in time-evolving surfaces. Artists in the demoscene community capitalize on these capabilities to construct compelling 3D scenes through the blending of implicit primitives, iterated fractals, procedural noise, and spatial distortions in generalized smooth CSG trees [Korndörfer et al. 2015]. Arbitrary parameters of these functions are then programmatically perturbed over time to produce realtime animations in a memory footprint on the order of several kilobytes [Quilez 2008]. Users of implicit CAD packages likewise leverage similar modelling techniques to build robust static geometry and rely on the realtime rendering of deforming implicits to facilitate their interactive modelling process [Allen 2019; Schmidt 2006]. Existing rendering solutions however cannot effectively localize these deformations in arbitrary implicits and must thus sacrifice resolution or restrict the range of allowable functions to maintain interactivity while they waste resources redundantly re-evaluating static scene regions every single frame.

We address this deficiency by tracking changes in the field through the rigorous range-bounding [Moore 1966] of its time derivatives to selectively re-evaluate regions only as and where is necessary to maintain a global error invariant. Our simple scheme yields significant speedups in complex temporally-coherent dynamic scenes and enables higher-resolution interaction with arbitrary implicits in computer games and modelling applications where deformations are largely localized and bounded-error approximation is acceptable.

Our approach treats implicit volume rendering as a special case of the continuous constraint satisfaction problem of *set inversion* which seeks the preimage $\mathbb{X} \subseteq \mathbb{R}^n$ of a set $\mathbb{Y} \subseteq \mathbb{R}^m$ under $f : \mathbb{R}^n \to \mathbb{R}^m$. We introduce this framework in Section 3, use it to formalize our

temporally coherent set inversion algorithm in Section 4, apply our algorithm to the special computer-graphics case of animated implicits where $m = 1$ and $\mathbb{Y} = [-\infty, 0]$ in Section 5, and discuss various extensions including swept volume rendering in Section 6. Our code is available at github.com/cryvosh/TemporalSetInversion.

2 RELATED WORK

Here we review various computer-graphics techniques for rendering $m = 1$ implicit surfaces and exploiting their temporal coherence. Section 3.2 further discusses related work on set inversion methods.

2.1 Raytracing

Raytracing reduces the implicit rendering problem to that of a one-dimensional search for the closest root along every ray shot from an eye into the scene. Sphere-tracing [Hart 1996] is a popular root-finder for this purpose due to its simplicity at the cost of converging on only the limited class of Lipschitz-bounded distance-like field-functions with $\|\nabla f\| \leq 1$. Segment-tracing [Galin et al. 2020] accelerates this root-finding process for implicits constructed through the hierarchical blending of compactly-supported skeletal primitives [Wyvill et al. 1999] for which local Lipschitz bounds over line segments and spheres can be analytically derived. These local bounds are often much smaller than their global counterparts, enabling larger, more efficient march steps. Non-linear sphere-tracing [Seyb et al. 2019] enables the rendering of signed distance fields warped by a class of potentially non-invertible explicit forward space deformations like Kelvinlets [De Goes and James 2017], but otherwise inherits the limitations of naive sphere-tracing and is orthogonal to our approach.

Interval ray-tracers [Barth et al. 1994; De Cusatis et al. 1999; Díaz et al. 2008; Fryazinov et al. 2010; Knoll et al. 2009; Mitchell 1990] instead rely on rigorous range-bounding techniques like interval [Moore 1966] and affine [Comba and Stolfi 1990] arithmetic to overcome the convergence constraints of sphere-tracing and root-find arbitrary non-Lipschitz field functions using coherent ray traversal.

Raytracing approaches however do not cache the roots found during ray-traversal and must therefore rediscover the surface from scratch in every frame, even if only the eye is moving. Our approach instead maintains a world-space discretization of the surface that updates only where sufficient surface displacement occurs, allowing the eye to freely move without additional root-finding.

2.2 Meshing

Meshing strategies like marching cubes [Lorensen and Cline 1987] and dual-contouring [Ju et al. 2002] point-sample the field over a dense grid spanning the entire search space to construct a triangulated approximation of the surface which can then be rendered via rasterization or exported into explicit modelling and simulation software. Global discretization approaches are therefore widely used in implicit CAD packages [Allen 2019; Schmidt 2006]. These techniques however miss thin features at low resolutions and scale poorly to higher resolutions due to their cubic runtime, requiring interactive modelling applications to drop their spatial resolution during user interactions to maintain realtime framerates.

Our approach instead builds and efficiently maintains a sparse octree-like discretization of the field that adaptively re-evaluates only the regions that measurably deform in each timestep. Small edits to the surface can therefore be made without a full-scene re-discretization, enabling higher-resolution interactions with animated implicits. Our octree can be used to enable fast temporally-coherent hierarchical mesh extraction [Sharp and Jacobson 2022].

2.3 Branch-and-Bound Subdivision

Branch-and-bound algorithms [Hansen 1992] recursively subdivide space in search of the surface using range-bounding techniques like interval arithmetic [Moore 1966] to disqualify regions that *cannot* contain the surface. This in turn produces a sparse octree-like discretization of the scene that can readily be used for rendering purposes. Interval subdivision approaches have a long history in computer graphics [Duff 1992; Snyder 1992; Tupper 2001] and have recently been applied to neural implicits [Sharp and Jacobson 2022], used for global optimization on the GPU [Sanders 2020], and augmented with function-pruning techniques that locally simplify f over the search domain [Keeter 2020].

We note these approaches solve a special scalar case of the continuous constraint satisfaction problem of set inversion, which seeks the preimage $\mathbb{X} = f^{-1}(\mathbb{Y}) = \{\boldsymbol{p} \in \mathbb{R}^n \mid f(\boldsymbol{p}) \in \mathbb{Y}\}$ given $f : \mathbb{R}^n \to \mathbb{R}^m$ and $\mathbb{Y} \subseteq \mathbb{R}^m$. A branch-and-bound algorithm called SIVIA (Set Inverter via Interval Analysis) [Jaulin and Walter 1993] is employed in general to derive such $\mathbb{X}$ to within a desired accuracy. We introduce SIVIA in Section 3.2 where we further discuss the related work on branch-and-bound rendering of implicit volumes where $m = 1$ and $\mathbb{Y} = [-\infty, 0]$. In Section 4 we introduce our temporally-coherent set inversion algorithm that avoids fully re-discretizing the entire search domain in response to changes in the implicit field due to user interaction.

2.4 Temporal Coherence

One approach to exploiting the temporal coherence of animated implicits involves the tracking of particles over the surface by numerically integrating a derived surface velocity [Stam and Schmidt 2011]. These particles may be rendered directly [Hart et al. 2002; Witkin and Heckbert 1994] or used to guide the deformation of a mesh [Bouthors and Nesme 2007]. Deriving an accurate surface velocity and maintaining a uniform particle distribution is nontrivial. Thus, these techniques in general cannot cope well with unusual topological transformations and serve as a Lagrangian counterpart to our more Eulerian discretization approach.

A second approach likewise relies on constraining the implicit field function to hierarchical Blobtree-like blends of compactly-supported spatially-boundable implicit primitives [Wyvill et al. 1999] where the region-of-influence of a moving body can be derived from the bounding-volumes of its constituent elements [Gourmel et al. 2010; Jevans et al. 1988; Opalach and Cani 1997]. Where such influence is non-zero, the surface must be recalculated. This process can be accelerated using cached local approximations of the non-deforming subtrees [Schmidt 2006; Schmidt et al. 2005]. However, for arbitrary implicits with infinitely-supported blends, such optimizations do not readily apply. Our more general approach

operates at the instruction level and thus imposes little additional structure on the implicit field function.

Beyond implicits, range-bounding techniques like interval arithmetic are used for continuous collision detection and kinetic datastructures in computer animation [Redon et al. 2005; Snyder et al. 1993; Wang et al. 2021; Zhang et al. 2007]. Our approach is similar in spirit to the adaptive rigidification scheme of Mercier-Aubin et al. [2022] which accelerates softbody simulations by locally merging undeforming elements in each timestep to save per-frame resources in low-deformation areas.

3 PRELIMINARIES

In this section we review the necessary background for our approach. This includes an overview of interval arithmetic, an introduction to set inversion, and an in-depth discussion on recent related work in computer graphics. We use the mathematical machinery covered here to formalize our error invariants in Sections 4.1 and 4.5.

3.1 Interval Arithmetic

Interval arithmetic [Moore 1966; Young 1931] is a generalization of real arithmetic to closed intervals that lets us efficiently and automatically derive guaranteed bounds on the range of real functions over interval domains. It is used in nonlinear global optimization to rigorously prove statements about sets of uncountably many reals in a finite number of operations [Hansen 1992; Jaulin et al. 2001].

Definition 3.1. An interval $[x]$ of $\mathbb{R}$ is a closed set defined by its upper and lower bounds $x_-, x_+ \in \overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, \infty\}$ via

$$[x] = [x_-, x_+] = \{x \in \overline{\mathbb{R}} \mid x_- \leq x \leq x_+\}. \tag{2}$$

Definition 3.2. Given *unordered* endpoints $a, b \in \overline{\mathbb{R}}$ we construct a valid guaranteed-nonempty interval between them via

$$[a, b]_u = [\min(a, b), \max(a, b)]. \tag{3}$$

Definition 3.3. The *width* of an interval $[x]$ is given by

$$w([x]) = x_+ - x_-. \tag{4}$$

The width of $\emptyset \in [\mathbb{R}]$ is 0 and the width of $[-\infty, \infty]$ is ∞.

Definition 3.4. We let $[\mathbb{R}]$ represent the set of all intervals of $\mathbb{R}$. Note that a zero-width interval is equivalent to a real number in $\mathbb{R}$.

Interval arithmetic works by overloading the operations of real arithmetic with their interval analogues. The elementary rules of addition, subtraction, and multiplication, for example, are given by

$$\begin{aligned} [x] + [y] &= [x_- + y_-, x_+ + y_+], \\ [x] - [y] &= [x_- - y_+, x_+ - y_-], \\ [x] \cdot [y] &= \begin{matrix}[\min(x_- y_-, x_- y_+, x_+ y_-, x_+ y_+), \\ \max(x_- y_-, x_- y_+, x_+ y_-, x_+ y_+)].\end{matrix} \end{aligned} \tag{5}$$

The range of any continuous monotonic function $m : \mathbb{R} \to \mathbb{R}$ can in general be bounded using the endpoints of the input interval via

$$m([x]) = [\min(m(x_-), m(x_+)), \max(m(x_-), m(x_+))] \tag{6}$$

Continuous non-monotonic function are bounded through the decomposition of the input interval into monotone pieces using branching operations. Squaring for example can be defined via

$$[x]^2 = \begin{cases} [\min(x_-^2, x_+^2), \max(x_-^2, x_+^2)], & \text{if } 0 \notin [x] \\ [0, \max(x_-^2, x_+^2)], & \text{otherwise.} \end{cases} \tag{7}$$

There rules can be composed to range-bound any function f whose constituent operations admit interval analogues, producing an "interval extension" of f which we can bound over an interval $[x]$ via $f([x]) \in [\mathbb{R}]$. Such functions satisfy an inclusion property given by

$$f([x]) \supseteq \{f(x) \mid x \in [x]\}. \tag{8}$$

These interval extensions are said to be convergent if $w(f([x])) \to 0$ as $w([x]) \to 0$. For the purposes of our algorithm, we require only that f, when queried on smaller and smaller intervals $[x]$, converges to the true range of the function over $[x]$ which, for discontinuous f, may be nonzero. This is further discussed in Section 3.2.1.

Definition 3.5. An interval vector, or box, $[\boldsymbol{x}]$ of $\mathbb{R}^n$ is a set defined by the cartesian product of n intervals of $[\mathbb{R}]$ via

$$[\boldsymbol{x}] = \bigtimes_{i=1}^{n} [x_i] = [x_1] \times [x_2] \times \cdots \times [x_n]. \tag{9}$$

Definition 3.6. Given arbitrary unordered points $\boldsymbol{a}, \boldsymbol{b} \in \overline{\mathbb{R}}^n$ we construct a valid guaranteed-nonempty box with corners $\boldsymbol{a}, \boldsymbol{b}$ via

$$[\boldsymbol{a}, \boldsymbol{b}]_u = \bigtimes_{i=1}^{n} [a_i, b_i]_u. \tag{10}$$

Definition 3.7. The *width* of a box $[\boldsymbol{x}]$ is given by the maximal width of its components such that

$$w([\boldsymbol{x}]) = \max_{1 \leq i \leq n} (x_{i+} - x_{i-}). \tag{11}$$

Definition 3.8. We let $[\mathbb{R}]^n$ represent the set of all boxes of $\mathbb{R}^n$.

By passing interval bounds as the arguments to multivariate functions, we can obtain bounds on the image of a function $f : \mathbb{R}^n \to \mathbb{R}^m$ over an axis-aligned box of $\mathbb{R}^n$. This capability is what powers the set inversion algorithm described in Section 3.2.

3.1.1 Overapproximation. Interval arithmetic bounds are guaranteed to enclose the true range of possible results, but are not necessarily tight. Suppose for example we let $[x] = [-1, 1]$ and perform the operation $x - x$. Intuitively, we expect this to yield zero, yet interval arithmetic returns $[-2, 2]$ as it cannot deduce that the two arguments to the subtraction are not independent. This effect is therefore known as the dependency problem, and explains why the squaring operation $[x]^2$ defined in Equation 7 yields tighter bounds than $[x] \cdot [x]$ using the multiplication rule from Equation 5. Interval bounds can likewise be made arbitrarily bad by adding to f factors of $\cos(x) + \cos(x + \pi)$ as interval arithmetic cannot deduce the two terms cancel to zero. We exploit this effect in Section 3.2.1 to explain how existing approaches in computer graphics deal with this issue.

Higher-order range-bounding techniques like affine arithmetic [Comba and Stolfi 1990] or Taylor models [Berz and Hoffstätter 1998] can instead be used to obtain tighter enclosures in exchange for added complexity [Ratz 1996; Vu et al. 2009]. Recent work by Sharp and Jacobson [2022] applies these approaches to neural implicits.

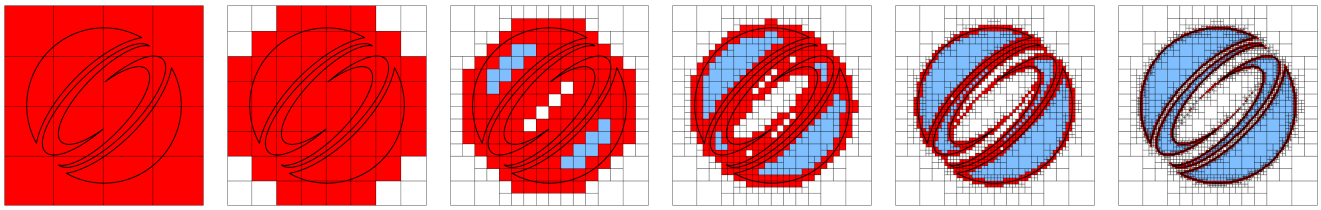


Fig. 2. Subdivision of an implicit SIGGRAPH logo from 4^2 to 128^2 quadtree resolution. We color P_i in red, P_{in} in blue, and P_{out} in white.

3.2 Set Inversion via Interval Analysis

Set inversion is the continuous constraint satisfaction problem of characterizing a *feasible set* $\mathbb{X} = f^{-1}(\mathbb{Y}) = \{\boldsymbol{p} \in \mathbb{R}^n \mid f(\boldsymbol{p}) \in \mathbb{Y}\}$ given $f : \mathbb{R}^n \to \mathbb{R}^m$ and $\mathbb{Y} \subseteq \mathbb{R}^m$. It is solved for range-boundable f to arbitrary accuracy using a branch-and-bound [Hansen 1992] algorithm called SIVIA (Set Inverter via Interval Analysis) [Jaulin and Walter 1993] which encloses $\mathbb{X}$ between sets of boxes called *subpavings* [Jaulin et al. 2001]. These sets naturally form octree-like hierarchies that we can readily use for rendering purposes.

Definition 3.9. A *subpaving* P_s of $\mathbb{R}^n$ is a set of disjoint nonzero-width boxes of $[\mathbb{R}]^n$. Disjoint boxes have non-overlapping *interiors*. Two subpavings are disjoint if their elements are pairwise disjoint.

Definition 3.10. A *paving* of $D \subseteq \mathbb{R}^n$ is a subpaving P_s that exactly covers D such that $\bigcup P_s = D$. Where not ambiguous, $\bigcup P_s$ may be written simply as P_s.

Definition 3.11. A box $[\boldsymbol{b}]$ is *feasible* if and only if $[\boldsymbol{b}] \subseteq \mathbb{X}$, *infeasible* if and only if $[\boldsymbol{b}] \cap \mathbb{X} = \emptyset$, and *ambiguous* otherwise.

Given a bounded search domain $[\boldsymbol{D}] \in [\mathbb{R}]^n$ and spatial error tolerance $\varepsilon \in \mathbb{R}$, SIVIA produces a paving P of $[\boldsymbol{D}]$ from three disjoint subpavings P_{in}, P_{out}, and P_i that approximate $\mathbb{X}$ such that

$$\begin{gathered} P_{in} \cup P_{out} \cup P_i = [\boldsymbol{D}], \\ P_{in} \subseteq \mathbb{X} \subseteq P_{in} \cup P_i \text{ and } \mathbb{X} \cap P_{out} = \emptyset, \\ \forall [\boldsymbol{b}] \in P_i, w([\boldsymbol{b}]) \leq \varepsilon \text{ and } f([\boldsymbol{b}]) \not\subseteq \mathbb{Y} \text{ and } f([\boldsymbol{b}]) \cap \mathbb{Y} \neq \emptyset, \end{gathered} \tag{12}$$

where P_i is comprised of *indeterminate* boxes $[\boldsymbol{b}]$ that cannot be proven feasible or infeasible from only their image under f via

$$\begin{aligned} f([\boldsymbol{b}]) \subseteq \mathbb{Y} &\implies [\boldsymbol{b}] \text{ is feasible,} \\ f([\boldsymbol{b}]) \cap \mathbb{Y} = \emptyset &\implies [\boldsymbol{b}] \text{ is infeasible.} \end{aligned} \tag{13}$$

This paving is built by recursively subdividing $[\boldsymbol{D}]$ until all indeterminate boxes are sufficiently small, thereby naturally producing an octree-like hierarchy via Algorithm 1 where P_{in} approaches the boundary of $\mathbb{X}$ (denoted $\partial\mathbb{X}$) from the inside, P_{out} approaches $\partial\mathbb{X}$ from the outside, and P_i covers $\partial\mathbb{X}$, as visualized in Figure 2. Note that using the tools discussed here, a box $[\boldsymbol{b}]$ *cannot* be proven to be ambiguous, i.e., on the boundary of $\mathbb{X}$. Instead, it can only be proven *not* ambiguous using Equation 13. We note this formalization implicitly underlies the related work on interval subdivision in computer graphics [Duff 1992; Gleicher and Kass 1992; Keeter 2020; Sharp and Jacobson 2022; Snyder 1992; Tupper 2001].

Algorithm 1: SIVIA (Set Inverter Via Interval Analysis)

Input: Range-boundable function $f : \mathbb{R}^n \to \mathbb{R}^m$, set to invert $\mathbb{Y} \subseteq \mathbb{R}^m$, initial search domain $[\boldsymbol{D}]$, spatial error tolerance $\varepsilon \in \mathbb{R}$

Output: Paving P_{in}, P_{out}, P_i of $[\boldsymbol{D}]$

```
P_in ← ∅, P_out ← ∅, P_i ← ∅
Q ← Queue()
Q.enqueue([D]);
while Q ≠ ∅ do
    [b] = Q.dequeue()
    if f([b]) ⊆ Y then
        P_in ← P_in ∪ [b]
    else if f([b]) ∩ Y = ∅ then
        P_out ← P_out ∪ [b]
    else if w([b]) ≤ ε then
        P_i ← P_i ∪ [b]
    else
        Q.enqueue([b_j]) for [b_j] ∈ Subdivide([b])
return {P_in, P_out, P_i}
```

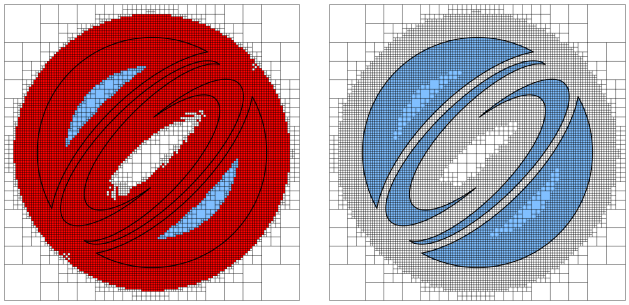


Fig. 3. Here we add a $\cos(x) + \cos(x + \pi)$ term to an implicit SIGGRAPH logo to exaggerate the effects of interval overapproximation and produce a thick indeterminate layer visualized in red (left). Keeter-style point sampling of the indeterminate boxes gives a more accurate result (right).

3.2.1 Point Sampling. Note that $P_{in} \to \mathbb{X}$ as $\varepsilon \to 0$. However, for any $\varepsilon > 0$, the width of the indeterminate subpaving P_i may be arbitrarily large as the converse of Equation 13 does not hold due to interval overapproximation. Indeterminate boxes are therefore not necessarily ambiguous, but may be feasible or infeasible as

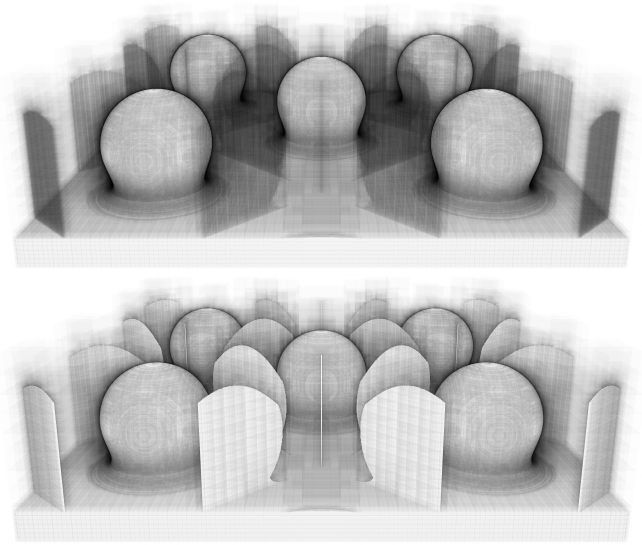


Fig. 4. This field uses the modulo operator to repeat space and instance a blobby sphere. The bottom figure shows the standard SIVIA solution where P_{in} and P_i are visualized as filled boxes resulting in false planes between the instanced subspaces consisting of indeterminate boxes that straddle the discontinuities in the field. Point sampling (top) resolves this issue and produces a more accurate paving. The extensive subdivision in these areas however remains visible in our octree density visualization where per-pixel darkness corresponds to the number of indeterminate boxes marched-through during the raytracing pass.

well [Jaulin and Walter 1993]. To disambiguate such boxes for rendering purposes, Keeter [2020] point-samples f once in the center of each indeterminate leaf box to approximate its feasibility. Figure 3 visualizes this process by adding to f a $\cos(x) + \cos(x + \pi)$ term to exaggerate the effects of interval overapproximation, leading to a thick error layer that point-sampling effectively cleans up. Sharp and Jacobson [2022] augment this approach by point-sampling such boxes 6 times, once in the center of each face. We note however that both approaches fail to capture thin features like planes defined by absolute values of implicit field functions where the probability that a point-sample lands within the implicit volume is effectively zero. Dilating the desired isosurface by subtracting constants from f to expand and therefore recover such features can work for SDFs but not for arbitrary non-Lipschitz-bounded field functions.

Keeter [2020] claims their SIVIA variant described in Section 3.2.2 requires $C0$ continuity in the surface, but we note that the field need not be continuous for rendering purposes and in fact, a large fraction of functions found on platforms like Shadertoy are discontinuous due to their use of the modulo operator to repeat space and thereby instance objects [Korndörfer et al. 2015]. Such fields yield non-convergent inclusion functions where the theoretical guarantee that $P_i \longrightarrow \partial\mathbb{X}$ as $\varepsilon \longrightarrow 0$ no longer applies, but this does not effect the correctness of SIVIA for our needs as Equation 12 still holds. Efficiency on the other hand is reduced in such cases as interval arithmetic cannot deduce the infeasibility of boxes whose images overlap boundary-straddling discontinuities. Consider for

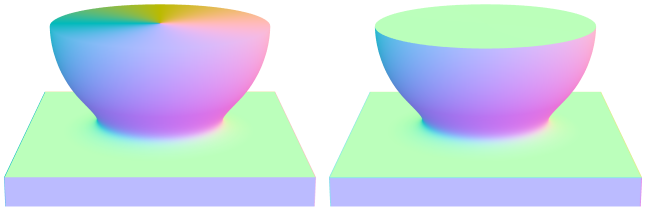


Fig. 5. Left image shows autodiff-derived surface normals which appear incorrect due to a modulo-induced discontinuity in the field that overlaps the surface and cuts it in half. The right image shows the normals one would expect from such a surface, here derived via large-stencil finite-difference.

example the function $f(x) = \text{mod}(x, 1) - 0.5$ over the domain $[x] = [-0.1, 0.1]$. This function contains no roots over $[x]$ yet $0 \in f([x]) = [-0.4, 0.4]$ and thus SIVIA will subdivide down to the minimal box size around the discontinuity at $x = 0$ in search of roots that do not exist. Figure 4 illustrates this effect where the zero-straddling discontinuities in the field are covered by indeterminate boxes in P_i forming solid planes between the instanced subspaces. Keeter-style point sampling however effectively eliminates this issue. If such discontinuities overlap the surface then the normals may not be well-defined as the gradients of the field as illustrated by Figure 5. More plausible normals can in some cases be derived via large-stencil finite differences or otherwise recovered from the local octree topology.

3.2.2 Function Approximation. Keeter [2020] notes that subexpressions of f of the form $\max(g(x), h(x))$ (and min analogously) can be simplified in regions where interval arithmetic proves that $g(x)$ is always greater than or equal to $h(x)$. Within such regions, the max will never select $h(x)$ and it can thus be pruned from the local f by what amounts to an optimizing compiler. This accelerates the evaluation of f in the child subregions so long as there is enough free memory to store the specialized program. If not, the children will simply reuse their parent's programs in subsequent subdivisions.

From a differentiable programming perspective, Keeter's function pruning algorithm simply removes from f every subexpression s for which $\partial f/\partial s = 0$ over the working box $[\boldsymbol{b}]$ where custom interval gradient rules always return 1 for every instruction, except for max (and min analogously) whose gradient is instead defined via

$$\nabla \max([g], [h]) = \begin{cases} (0,1), & \text{if } g_+ < h_- \\ (1,0), & \text{if } h_+ < g_- \\ (1,1), & \text{otherwise.} \end{cases} \tag{14}$$

Here, the partial derivatives 0 and 1 returned by this rule represent zero-width intervals to be used in an interval autodiff system that replaces gradient rules with their interval analogues.

We note this function pruning approach can easily be generalized beyond just min and max instructions by *approximating* subexpressions whose influence over f is below some global threshold. To derive this influence, one needs an approximation s' of s over $[\boldsymbol{b}]$ with an associated absolute error bound Δs that can then propagate through f using first-order error analysis. The forward interval trace of f provides us a crude constant approximation $s' = \text{Mid}(s([\boldsymbol{b}]))$ with $\Delta s = w(s([\boldsymbol{b}]))/2$. We can then derive the error in f due

to a substitution of s with s' using first-order error analysis with standard interval gradient rules as discussed in Section 4.2 via

$$[\Delta f] = \Delta s \cdot \frac{\partial f}{\partial s}([\boldsymbol{b}]). \tag{15}$$

This generalization appears in earlier optimization work by Deussen et al. [2016]; Riehme and Naumann [2015]; Vassiliadis et al. [2016]. Having implemented these techniques on the GPU, we note their main limitation is memory usage. In difficult scenarios, one will often run out of memory before enough approximated programs can be stored to significantly improve runtime performance.

3.2.3 Parallelization. SIVIA recursively subdivides some areas more densely than others leading to an unbalanced, irregular workload not easily mappable onto modern GPUs. Existing implementations subdivide the search-space breadth-first and evaluate each octree level in parallel. Between each level they execute a filtering step to disregard unambiguously filled or empty boxes that require no further investigation. This effectively reduces the sparse set of evaluated boxes into a compact set of indeterminate boxes to subdivide in the next parallel dispatch.

Keeter [2020] implements this filtering step using a bump allocator, where each indeterminate box at the current level atomically increments a single integer to assign itself an index into the compact set. Sanders [2020] and Sharp and Jacobson [2022] instead rely on a parallel prefix-sum scan operation [Merrill and Garland 2016] via CUDA and JAX respectively as implementing this operation efficiently in cross-platform compute shaders is not trivial [Levien 2021]. Both approaches require multiple shader stages and are likewise difficult to directly adapt to a long-running program like ours where data must persist between timesteps. We outline our parallelization strategy in Section 4.6.1.

4 TEMPORAL SIVIA

In this section we extend SIVIA to support fast bounded-error updates to P_{in}, P_{out}, and P_i as the arguments of f vary in time. We assume that time advances in discrete steps and that $f : \mathbb{R}^n \to \mathbb{R}^m$ is a range-boundable function parameterized by τ time-varying parameters encapsulated in a configuration-space point $\boldsymbol{C}(t) \in \mathbb{R}^\tau$ whose position in each new timestep is provided by an external system like user-input and whose position in all previous timesteps is known and accessible. We first focus on the special case of animated implicits defined by scalar $m = 1$ functions where $\mathbb{Y} = [-\infty, 0]$ and $\mathbb{X}(t) = f_t^{-1}(\mathbb{Y}) = \{\boldsymbol{p} \in [\boldsymbol{D}] \mid f(\boldsymbol{p}, \boldsymbol{C}(t)) \in \mathbb{Y}\}$, and later generalize to vector-valued $m > 1$ functions with arbitrary $\mathbb{Y}$ in Section 4.5.

4.1 Error Invariant

Our goal is to exploit the temporal coherence of $\mathbb{X}(t)$ to avoid redundantly repaving the entire search domain $[\boldsymbol{D}]$ from scratch in every timestep. We instead wish to build and efficiently *maintain* a paving $P' = \{P'_{in}, P'_{out}, P'_i\}$ of $\mathbb{X}(t)$ by selectively re-paving regions only as and where is necessary to approximate the ground-truth SIVIA paving P of $\mathbb{X}(t)$ to within a user-specified error tolerance $\delta \in \mathbb{R}$ at all times. This means we are willing to tolerate fluctuations of up to δ in the field before updating P' and therefore accumulate error in P' within a bounded neighborhood of P_i in the form of misclassified boxes. More formally, we guarantee that

$$\begin{gathered} P'_{in} \subseteq f_t^{-1}([-\infty, \delta]), \\ P'_{out} \subseteq f_t^{-1}([-\delta, \infty]), \\ \forall [\boldsymbol{b}] \in P'_i, w([\boldsymbol{b}]) \leq \varepsilon \text{ and } f_t([\boldsymbol{b}]) \cap [-\delta, \delta] \neq \emptyset. \end{gathered} \tag{16}$$

Note that $P' \to P$ as $\delta \to 0$ and that the width of the $f_t^{-1}([-\delta, \delta])$ error layer fluctuates over $[\boldsymbol{D}]$ according to the local Lipschitz bounds of f. Within the ε-width neighborhood of this layer, we make no guarantees about the correctness of P' but outside it, P' must match P exactly. We show the visual consequences of this invariant in Section 5, generalize it in Section 4.5, and discuss an alternative approach in Section 6.1. This invariant was inspired by the formalization of Thick SIVIA by Desrochers and Jaulin [2017].

4.2 Measuring Change

We now wish to bound the change $[\Delta f]$ in the field over a box $[\boldsymbol{B}]$ over an interval of time $[t] = [t_-, t_+]$ to later determine if a paving of $[\boldsymbol{B}]$ built at t_- requires updates to satisfy our error invariant at t_+. This bound must satisfy an inclusion property such that insufficient change measured over $[\boldsymbol{B}]$ implies insufficient change at every point within. More formally, we require that

$$\forall [\boldsymbol{b}] \subseteq [\boldsymbol{B}], f([\boldsymbol{b}], t_+) - f([\boldsymbol{b}], t_-) \subseteq [\Delta f]. \tag{17}$$

Note the finite difference of intervals $f([\boldsymbol{B}], t_+) - f([\boldsymbol{B}], t_-)$ over all of $[\boldsymbol{B}]$ fails to satisfy the desired inclusion property and is unsuitable for our purposes. Consider for example the function $f(x, t) = \cos(x + t)$ over the domain $[x] = [0, 2\pi]$. As time evolves, the roots of this function move yet its range over $[x]$ does not change. The finite difference therefore yields zero and gives no insight into whether sufficient change has occurred in the domain. We instead turn to first-order error analysis and range-bound the time-derivative of f over $[\boldsymbol{B}]$ over $[t]$ to measure $[\Delta f]$ via

$$[\Delta f] = (t_+ - t_-) \cdot \frac{df}{dt}([\boldsymbol{B}], [t]). \tag{18}$$

We note that f need not be continuous in space nor differentiable in time for this bound to be defined. Consider for example the function $f(x, t) = \lfloor x \rfloor + \max(0, t)$ over $[x] = [0, 5]$ over $[t] = [-1, 1]$. Using the gradient rule for max from Equation 14, we find the time derivative of f over this domain is bounded by $[0, 1]$ and thus $[\Delta f] = [0, 2]$ as desired. Discontinuous functions of time likewise produce useful gradients so long as they are locally continuous over the queried time interval. Consider for example the function $f(x, t) = x + \lfloor t \rfloor$ over any $[x]$ over $[t] = [2.4, 2.6]$ where $[\Delta f]$ yields zero as desired if the gradient of floor is defined by

$$\frac{\partial}{\partial t}\left\lfloor [t] \right\rfloor = \begin{cases} [0, 0], & \text{if } \lfloor t_- \rfloor = \lfloor t_+ \rfloor \\ [-\infty, \infty], & \text{otherwise.} \end{cases} \tag{19}$$

More generally, if f is a function of an interactively-adjustable parameter $C(t) \in \mathbb{R}$ whose value over $[t]$ may decrease, then we must differentiate f over the nonempty interval of configuration-space with endpoints $C(t_-)$ and $C(t_+)$ via

$$\begin{gathered} [\Delta f] = (C(t_+) - C(t_-)) \cdot \frac{df}{dC}([\boldsymbol{B}], [c]) \\ \text{where } [c] = [C(t_-), C(t_+)]_u. \end{gathered} \tag{20}$$

We note that $[\Delta f]$ may wildly overestimate the change if $\partial f/\partial t$ is not tightly bounded. Consider for example the function $f(x,t) = x + \sqrt{t}$ over $[x] = [0,1]$ over $[t] = [0,4]$. The time derivative of f over this domain is bounded by $[0.25, \infty]$ and thus $[\Delta f] = [0.5, \infty]$. However, the true range of f itself over $[x]$ is only $[0,3]$ and thus the field at no point in $[x]$ could have changed by more than ± 3 units from t_- to t_+. We can exploit this fact to better approximate $[\Delta f]$ via

$$\begin{aligned}[\Delta f] &= W \cap (C(t_+) - C(t_-)) \cdot \frac{df}{dC}([\boldsymbol{B}], [c]) \\ \text{where } [c] &= [C(t_-), C(t_+)]_u \\ \text{and } W &= w(f([\boldsymbol{B}], [c])) \cdot [-1, 1].\end{aligned} \tag{21}$$

More generally, if f is a function of τ interactively-adjustable parameters encapsulated by $\boldsymbol{C}(t) \in \mathbb{R}^\tau$ then we must differentiate it over the nonempty axis-aligned box of configuration space with corners $\boldsymbol{C}(t_-)$ and $\boldsymbol{C}(t_+)$ to derive our final scalar $[\Delta f]$ via

$$\begin{aligned}[\Delta f] &= W \cap \sum_{i=1}^{\tau} (\boldsymbol{C}_i(t_+) - \boldsymbol{C}_i(t_-)) \cdot \frac{\partial f}{\partial \boldsymbol{C}_i}([\boldsymbol{B}], [\boldsymbol{c}]) \\ \text{where } [\boldsymbol{c}] &= [\boldsymbol{C}_i(t_-), \boldsymbol{C}_i(t_+)]_u \\ \text{and } W &= w(f([\boldsymbol{B}], [\boldsymbol{c}])) \cdot [-1, 1].\end{aligned} \tag{22}$$

Note that $\mathbb{X}(t)$ is a pure function of $\boldsymbol{C}(t)$ and thus we need not know the true range of $\boldsymbol{C}$ *over* $[t]$, i.e., between timesteps, to compare $\mathbb{X}(t_-)$ and $\mathbb{X}(t_+)$. This allows our technique to work in interactive environments where such interim parameter values are unknowable. Consider for example the functions $f_1(x,t) = x + \cos(t)$ and $f_2(x, C(t)) = x + C(t)$ over any domain $[x]$ over $[t] = [0, 2\pi]$ where $C(t_-) = C(t_+) = 0$ are two provided samples of $\cos(t)$. That is, $C(t) = \cos(t)$ is a function whose value we know only at the endpoints of the time interval. In this case, our change query yields $[\Delta f_1] = [-2\pi, 2\pi]$ and $[\Delta f_2] = 0$ as desired. Indeed we can evaluate any subexpression of f which depends solely on time between its values at t_- and t_+ instead of over $[t]$ to obtain tighter bounds.

Computing time-derivatives with automatic differentiation involves gradient evaluation over the forward interval trace of f. This extra computation naturally introduces extra interval overapproximation into the gradient bounds which does not exist in the original bounds on f itself. In practice, these gradient bounds are often extremely conservative leading to reduced efficiency of our algorithm. The more sophisticated range-bounding techniques discussed in Section 3.1.1 can be employed to tighten them.

4.3 Updating Pavings

Our goal now is to use our change query to efficiently update our paving P' initialized to P at time t_- such that it satisfies our error invariant at some new time t_+. Intuitively, we plan to recursively re-pave the domain with SIVIA as usual, but *without* recursing into the existing boxes over which we can prove insufficient change has occurred. In such regions, our invariant remains satisfied and thus no re-evaluation is necessary. We will then repeat this updating procedure in every new timestep to maintain P'.

A first approach may be to simply measure change over just the latest timestep, starting from the root and recursing deeper into boxes where such change is sufficiently large. This approach, however, fails to capture the dynamics of a slowly evolving field

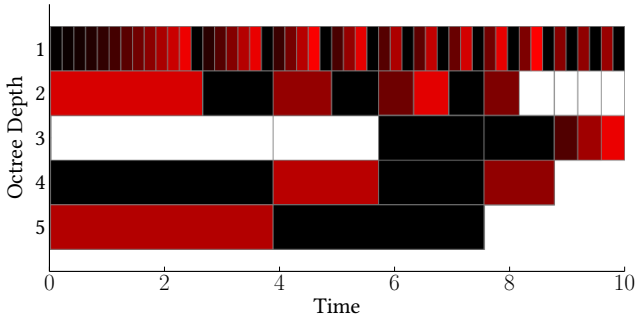


Fig. 6. Cross-section of our octree structure at a point across time showing how the large boxes near the root of our octree (top) evaluate more frequently than the smaller boxes deeper down the hierarchy (bottom) where time-derivatives are more tightly bounded. The width of each box represents the time interval over which the evaluation occured and the redness indicates the amount of accumulated error within the box at that level in the hierarchy at the end of the time interval. Gaps in the evaluations are caused by instant evaluation due to the fast optimization of Section 4.4.

where over no single timestep does the change anywhere exceed δ. In this case, P' will never update and its error will grow unbounded. We therefore seek a method that evaluates f over multiple timesteps.

A second approach may be to store in each box $[\boldsymbol{b}]$ a timestamp $[\boldsymbol{b}].t \in \mathbb{R}$ that represents the time at which the change to the field within $[\boldsymbol{b}]$ last crossed our error threshold and its interior was recursively re-paved. Each re-evaluation of $[\boldsymbol{b}]$ then measures change over the time interval $[[\boldsymbol{b}].t, t_+]$ stretching from its time of last re-paving to the current time t_+. If this change exceeds our threshold and $[\boldsymbol{b}]$ remains indeterminate then we update $[\boldsymbol{b}].t$ to t_+ and recurse into its children.

Under a slowly evolving field, the root box enclosing the search domain will now simply evaluate over longer and longer time intervals until the change eventually exceeds δ and its children are updated in turn. Indeed this approach works due to the inclusion property of interval arithmetic, but is not optimal as each successive query that fails to update the stored timestamp will run over ever-growing time intervals that introduce extra interval overapproximation and thus lead to unnecessary re-paving. We therefore seek a method for accumulating the change that happens over many small timesteps instead of few large ones.

Our third and final approach is to store in each box $[\boldsymbol{b}]$ both a timestamp $[\boldsymbol{b}].t$ indicating now the time this box was last queried and an accumulated error $[\boldsymbol{b}].\delta \in \mathbb{R}$ which tracks the total change to the field in this box since it last crossed our error threshold and had its interior recursively re-paved. This timestamp then updates whenever the box is queried for change, and these changes accrue inside $[\boldsymbol{b}].\delta$ until they eventually cross our error threshold. If $[\boldsymbol{b}]$ remains indeterminate when this occurs then we reset $[\boldsymbol{b}].\delta$ to 0 and recurse into its children.

Under a slowly evolving field, the root box enclosing the search domain will now be queried for change over every latest timestep until the changes accumulate beyond our error tolerance. Note that $[\boldsymbol{b}].t$ is hierarchically organized, i.e., a child cannot have a more recent timestamp than its parent. However, $[\boldsymbol{b}].\delta$ values have no

such structure and thus a child may have more or less error than its parent. A cross-sectional view of this hierarchy at a point over time is shown in Figure 6 and a formal implementation of our approach is presented in Algorithm 2.

4.4 Fast Evaluations

The approach described thus far re-uses the forward-pass of the change-measuring evaluations to deduce the feasibility of each box. The approach works well for slowly-evolving surfaces but quickly degrades in performance beyond even per-frame SIVIA for fast-moving surfaces. Fundamentally, the issue is that we are evaluating f over intervals of time, yet only care about the state of P' *at* the current instant in time. If a box $[\boldsymbol{b}]$ is indeterminate at time $[\boldsymbol{b}].t$ then, by the inclusion property of interval arithmetic, it will necessarily also be indeterminate over any time interval that contains $[\boldsymbol{b}].t$. As a result, our current algorithm wastes resources subdividing down around the indeterminate boxes that contained the surface in the previous timestep yet are empty in the current timestep, effectively doubling our workload. This effect manifests as a ghost-like region that follows fast-moving surfaces in our octree-density visualization shown in Figure 7.

Our solution to this problem is to simply define a threshold $\kappa \in \mathbb{R}$ such that if the measured change to the field over an indeterminate box $[\boldsymbol{b}]$ in a single evaluation exceeds κ then we set a new property $[\boldsymbol{b}]$.fast to true and evaluate the children of $[\boldsymbol{b}]$ not over an interval of time but at the current instant, just to check if there is anything still there. If a child box is non-indeterminate then we stop searching, otherwise we again attempt time-interval evaluation in the grandchildren. This leads quickly-evolving areas to alternate between time-instant and time-interval evaluation as we descend down the octree as shown in Figure 6 where gaps in the time intervals indicate fast evaluations.

Since no change can occur over an instant in time, the indeterminate instant-evaluated children of fast boxes cannot measure change nor decide for themselves whether to recursively re-evaluate their children or not. For correctness, we thus force all such boxes to re-evaluate their children in turn. Suppose for example a box $[\boldsymbol{b}]$ is marked as "fast". Each of $[\boldsymbol{b}]$'s children are then evaluated at an instant in time, and all of *their* children ($[\boldsymbol{b}]$'s grandchildren) are forced to re-evaluate. This optimization may therefore reduce performance in some cases as it is possible that without, the measured change in $[\boldsymbol{b}]$'s children would not warrant further sub-box evaluation. From our testing, such cases seem exceedingly rare.

This simple scheme effectively alleviates the described performance penalties of time-interval evaluation and additionally helps regulate the extra interval overapproximation it introduces. We find from our testing that the exact value of κ does not significantly influence performance so long as it is nonzero and within an order of δ. All of our tests in Section 5 use $\kappa = \delta$.

4.5 Generalization

We have thus far focused on the scalar $m = 1$ field functions used for implicit volume rendering where $\mathbb{Y} = [-\infty, 0]$, but our algorithm easily extends to vector-valued $m > 1$ functions with arbitrary $\mathbb{Y}$ through component-wise generalization of our previous discussion.

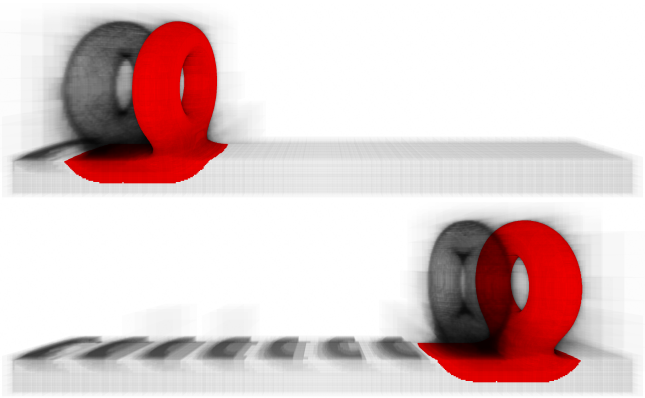


Fig. 7. Two snapshots of an animated implicit torus translating quickly from left to right. The per-pixel darkness indicates the number of indeterminate boxes marched-through during the raytracing pass. Regions colored red were evaluated in the latest timestep to indicate where detectable motion is occuring. Here we see ghosting artifacts where the previous-timestep surface persists in the octree structure. These regions were indeterminate *at* the previous timestep and therefore remain indeterminate when evaluated *over* the latest timestep, thus degrading our performance.

In this case, our temporal error tolerance and fast threshold are given by vectors $\boldsymbol{\delta}, \boldsymbol{\kappa} \in \mathbb{R}^m$ encoding the respective quantities along each axis of the codomain. Our error invariant from Equation 16 becomes

$$\begin{gathered} P'_{in} \subseteq f_t^{-1}(\mathbb{Y}'_{in}), \\ P'_{out} \subseteq f_t^{-1}(\mathbb{Y}'_{out}), \\ \forall [\boldsymbol{b}] \in P'_i, w([\boldsymbol{b}]) \leq \varepsilon \text{ and } f_t([\boldsymbol{b}]) \cap (\mathbb{Y}'_{in} \cap \mathbb{Y}'_{out}) \neq \emptyset \end{gathered} \tag{23}$$

where $\mathbb{Y}'_{in}$ and $\mathbb{Y}'_{out}$ are given by the Minkowski sum of the box $[\boldsymbol{d}] = [-\boldsymbol{\delta}_1, \boldsymbol{\delta}_1] \times \cdots \times [-\boldsymbol{\delta}_m, \boldsymbol{\delta}_m]$ with $\mathbb{Y}$ and $\mathbb{R}^m \setminus \mathbb{Y}$ such that

$$\mathbb{Y}'_{in} = \bigcup_{\boldsymbol{p} \in \mathbb{Y}} \boldsymbol{p} + [\boldsymbol{d}] \quad \text{and} \quad \mathbb{Y}'_{out} = \bigcup_{\boldsymbol{p} \in \mathbb{R}^m \setminus \mathbb{Y}} \boldsymbol{p} + [\boldsymbol{d}]. \tag{24}$$

Our change measure $[\Delta f]$ becomes a vector quantity $[\Delta \boldsymbol{f}]$ which satisfies an inclusion property analogous to Equation 17 given by

$$\forall [\boldsymbol{b}] \subseteq [\boldsymbol{B}], \forall j \in [1 \ldots m], f([\boldsymbol{b}], t_+)_j - f([\boldsymbol{b}], t_-)_j \subseteq [\Delta \boldsymbol{f}]_j \tag{25}$$

which leads to a generalized $[\Delta \boldsymbol{f}]$ analogue of Equation 22 given by

$$\begin{gathered} [\Delta f] = \bigtimes_{j=1}^{m} \left(W_j \cap \sum_{i=1}^{\tau} \Delta \boldsymbol{C}_i \cdot \frac{\partial f_j}{\partial \boldsymbol{C}_i}([\boldsymbol{B}], [\boldsymbol{c}]) \right) \\ \text{where } [\boldsymbol{c}] = [\boldsymbol{C}_i(t_-), \boldsymbol{C}_i(t_+)]_u \\ \text{and } W_j = w(f([\boldsymbol{B}], [\boldsymbol{c}])_j) \cdot [-1, 1] \\ \text{and } \Delta \boldsymbol{C}_i = (\boldsymbol{C}_i(t_+) - \boldsymbol{C}_i(t_-)). \end{gathered} \tag{26}$$

The accumulated error $[\boldsymbol{b}].\delta$ stored in each box likewise becomes an m-dimensional interval vector $[\boldsymbol{b}].\boldsymbol{\delta}$. In practice one could instead store only the maximal component of this quantity to save memory.

4.6 Implementation

In this section we discuss our GPU implementation of Temporal SIVIA in cross-platform compute shaders via WebGPU and WGSL. We implement interval arithmetic and reverse-mode automatic differentiation in our shader using IntervalArithmetic.jl [Sanders et al. 2022] and ChainRules.jl [White et al. 2023] as reference. Like both Keeter [2020] and Sharp and Jacobson [2022], we do not account for floating-point rounding errors. Our implementation evaluates f and its gradients using an interpreter loop similar to Keeter [2020]. This adds tremendous runtime overhead to our evaluations and is not fundamentally required for Temporal SIVIA. We could instead code-generate native GPU interval and autodiff code on the CPU to improve performance, but this speedup would apply equally to our algorithm and per-frame SIVIA and would thus not influence our relative performance results presented in Section 5.

Algorithm 2 implements Temporal SIVIA. It assumes that each box $[\boldsymbol{b}]$ contains additional fields including a set of child boxes $[\boldsymbol{b}]$.children, a bound on the local field $[\boldsymbol{b}].[\boldsymbol{F}]$, a timestamp $[\boldsymbol{b}].t$, an accumulated error $[\boldsymbol{b}].\boldsymbol{\delta}$, and a boolean $[\boldsymbol{b}]$.fast to flag fast-changing regions. We note that $[\boldsymbol{b}].[\boldsymbol{F}]$ is used only to guide our visualization pass and can thus be replaced in practice by a ternary flag that classifies $[\boldsymbol{b}]$ as either filled, empty, or indeterminate. In the first frame of an animation, we initialize $[\boldsymbol{D}]$.children to an empty set and zero out its other fields. The first call to Algorithm 2 then builds an octree-like hierarchy rooted at $[\boldsymbol{D}]$ by recursively populating its children. Subsequent runs of the algorithm on $[\boldsymbol{D}]$ in each timestep then incrementally update the hierarchy.

4.6.1 Parallelization. We parallelize our algorithm using a persistent thread scheduling model [Aila and Laine 2009; Gupta et al. 2012] where long-running threads saturate the GPU and continually fetch work from a global work-distribution queue [Kerbl et al. 2018] until it is eventually emptied and all active threads are finished their tasks. This flexible approach allows us to easily map a recursive workload like SIVIA to a single shader stage with real code that very closely resembles the pseudocode provided in Algorithm 2.

The Line 3 while-loop is parallelized over each warp and the Line 7 for-loop is parallelized over each thread within the warp. Lines 4-6 are executed only by thread 0 in each warp using workgroup shared memory to communicate with the other threads. To terminate the shader, we use a global atomic integer $\mathcal{W}$ to track whether any work is left to be done. We increment $\mathcal{W}$ upon each Dequeue() and decrement it once per warp from thread 0 after all threads break from the Line 7 for-loop. If the queue is ever emptied and the Line 4 dequeue fails then we check $\mathcal{W}$. If it is zero then we terminate all the threads, otherwise we continue looping as there must exist some thread in flight that may produce new work.

For simplicity, our implementation uses only 8 of the 32 available threads per warp. We could therefore easily improve our runtime by increasing the GPU occupancy but this speedup would apply equally to our algorithm and per-frame SIVIA and would thus not influence our relative performance results presented in Section 5.

4.6.2 Memory Management. Our algorithm requires a runtime memory management solution to allocate and deallocate octree nodes as the surfaces moves and occupies new space over time.

Algorithm 2: Temporal SIVIA

Input: Range-boundable function $f : \mathbb{R}^n \to \mathbb{R}^m$, octree root and search domain $[\boldsymbol{D}]$, current time $t \in \mathbb{R}$, function parameterization history $\boldsymbol{C}(t) \in \mathbb{R}^\tau$, spatial error tolerance $\varepsilon \in \mathbb{R}$, temporal error tolerance $\boldsymbol{\delta} \in \mathbb{R}^m$, fast threshold $\kappa \in \mathbb{R}^m$

Output: An octree rooted at $[\boldsymbol{D}]$ encoding P'_{in}, P'_{out}, P'_i.

```
1  Q ← ParallelQueue()
2  Q.enqueue([D])
3  while Q ≠ ∅ do in parallel
4      [B] ← Q.dequeue()
5      new ← [B].children = ∅
6      if new then [B].children ← Subdivide([B])
7      for [b] ∈ [B].children do in parallel
8          t_l ← [b].t
9          [b].t ← t
10         if w([b]) ≤ ε then
11             [b].[F] ← f(Center([b]), C(t))
12             continue
13         if new ∨ [B].fast then
14             [b].[F] ← f([b], C(t))
15             if Classify([b].[F]) = indeterminate then
16                 Q.enqueue([b])
17             else
18                 Destroy([b].children)
19             continue
20         [c] ← [C(t_l), C(t)]_u
21         [b].[F] ← f([b], [c])
22         if Classify([b].[F]) ≠ indeterminate then
23             Destroy([b].children)
24             continue
25         [Δf] ← MeasureChange([b], [c])
26         [b].δ ← [b].δ + [Δf]
27         [b].fast ← any(|[Δf]_j|_+ ≥ κ_j)
28         if any([b].δ_j > δ_j) ∨ [b].children = ∅ then
29             [b].δ ← 0
30             Q.enqueue([b])
31 return [D]
```

We use a parallel queue [Kerbl et al. 2018] to implement a fixed-size memory allocator that distributes indices into a flat vector of boxes representing the tree. The Subdivide operation on Line 6 of Algorithm 2 uniformly subdivides the current box and allocates the memory for its child sub-boxes by deallocating indices from the memory pool or, if it is empty, incrementing a global atomic integer to generate a new index. The dequeue operation then simply returns an index to the pool to free the associated memory.

4.6.3 Visualization. Our algorithm constructs an explicit tree of nested uniform grids stored in a flat 2 GB vector of boxes where each box stores its children as an array of indices into the vector.

We visualize this octree-like structure via raytracing using a per-ray stack to recursively descend the tree, and fast voxel traversal [Amanatides and Woo 1987] to march through the uniform grids. Our construction algorithm uses Keeter-style point sampling in the indeterminate leaf boxes and thus our rays simply stop when they encounter a filled box. For lighting, we evaluate the surface normal at the hit point via finite difference of the implicit field using floating-point arithmetic. Compared to octree construction and maintenance, primary-ray tracing is generally quite fast even in our unoptimized implementation (~5 ms at 1024^3 spatial resolution with 1spp at 1080p). In Section 6 we discuss alternative strategies for normal evaluation and indeterminate box rendering. To visualize the non-uniform structures produced by our contraction scheme in Section 6.3, we interleave ray-AABB intersection tests [Majercik et al. 2018] into our recursive ray traversal.

4.6.4 Workflow. Our implicit scenes begin as GLSL shaders like those found on platforms like Shadertoy. They're then compiled by LLVM [Lattner and Adve 2004] using a Swizzling library to translate the syntax to C++. We then parse the LLVM intermediate representation (IR) into a custom instruction format for interval evaluation and autodiff by our WGSL interpreter loop. We rely on a set of compiler flags and optimizations to minimize the number of branches and memory moves in the IR as our interpreter handles only scalar floating-point instructions. Still, some shaders require manual adjustment to produce parsable output. Note this is not a fundamental limitation and we can, in future, extend our implementation to differentiate through control-flow as well [Innes 2018]. For per-pixel normals via finite-difference, we translate the GLSL shader to WGSL via NAGA [2023] for direct inclusion into our visualization shader. This entire compilation pipeline takes on the order of ~5 s on an i7 3930k CPU.

5 PERFORMANCE

The runtime of our algorithm depends on the *moving* surface area per timestep and we can therefore achieve arbitrarily large speedups compared to per-frame SIVIA by simply increasing the ratio of static to dynamic geometry in our test scenes. These speedups however come at the cost of accuracy. Our temporal error tolerance δ controls this tradeoff according to Equation 16 and thus acts like a second user-adjustable detail slider to accompany the spatial resolution slider ε. We compare our algorithm to per-frame SIVIA on a number of test scenes at various values of δ and ε on an RTX 3080 GPU.

Our visualizations show in red the regions that were evaluated in each timestep to demonstrate how our algorithm localizes per-frame updates near local surface deformations. At higher values of δ, fewer per-frame updates occur, leading to larger speedups at the cost of increased visual artifacts. Table 1 quantitatively compares our algorithm to per-frame SIVIA ($\delta = 0.0$) at various $\delta = 0.01, 0.03, 0.06$ at octree resolutions 1024^3 and 256^3 in terms of average per-frame runtime, total speedups, and average per-frame speedups. Figure 8 shows more explicitly the per-frame speedups on our test scenes at 1024^3 octree resolution. Figure 12 shows close-up views of the artifacts and update patterns on the surface of our scenes to give a qualitative sense for how δ controls our octree quality.

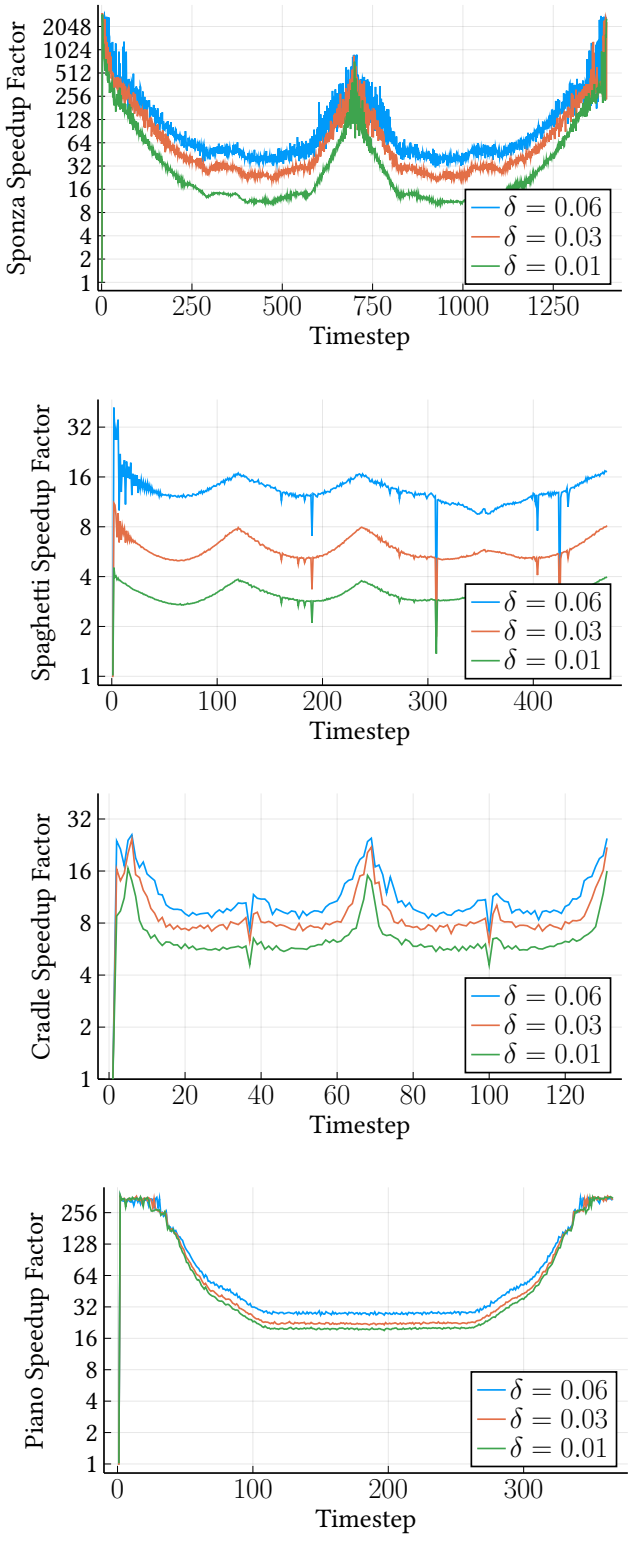


Fig. 8. Semi-log plots of per-timestep speedup factors of our algorithm at various δ compared to per-frame SIVIA ($\delta = 0.0$) at 1024^3 octree resolution.

5.1 Test Scenes

We note that many implicit scenes found on platforms like Shadertoy do not work well with interval subdivision approaches in our experience as they are not typically designed with discontinuities, interval overapproximation, or branching instructions in mind. We describe here our 4 main test scenes.

Sponza. This scene, shown in Figure 9, demonstrates a translating blob smoothly blending into a complex static environment, and most closely resembles the intended use-case for our method in games. It is based on the implicit Sponza scene by Shadertoy user *mmerchante* [Merchante 2018] and contains 306 scalar SSA instructions. Figure 12a shows a close-up view of the varying update patterns and artifacts produced by different settings of δ on a 1024^3 octree.

Spaghetti. This scene, shown in Figure 1, demonstrates the motion of a meatball within a ring of spaghetti. It is based on the Cables2 scene by Shadertoy user yuntaRobo [2020] and contains 296 scalar SSA instructions. Figure 12b highlights the artifacts in this scene at the various δ. Of our 4 tests scenes, this has by far the lowest ratio of static to dynamic geometry and thus yields the smallest speedups.

Cradle. This scene, shown in Figure 10, is based on the Newton's Cradle scene by Shadertoy user *BigWIngs* [Steinrucken 2021] and contains 328 scalar SSA instructions. Figure 12c shows a closeup of the surface at various δ. Note the middle three balls are not static.

Piano. This scene, shown in Figure 11, is based on the Piano scene by Shadertoy user *iq* [Quilez 2013] and contains 324 scalar SSA instructions. Note that after the large initial speedup from $\delta = 0.0$ to $\delta = 0.01$, varying δ does not significantly impact the speedup factor. Interestingly, this is the only scene for which increasing δ sometimes yields a worse average per-frame speedup, as seen in Tables 1 and 3. This may be due to the fast evaluations of Section 4.4.

6 EXTENSIONS

In this section, we discuss various extensions to our algorithm enabled by the local gradient bounds available within each box. We again focus on scalar $m = 1$ field functions for simplicity.

6.1 Lipschitz Correction

We note that our temporal error tolerance δ is a unitless quantity that simply bounds the allowed variation in the field between the $-\delta$ and δ isosurfaces according to Equation 16. The thickness of this error layer therefore varies across the domain according to the local Lipschitz bounds of f. To correct for this, we simply divide nonzero $[\Delta f]$ by the lower bound on the local gradient magnitude to translate error in the field to a bound on the surface displacement in distance units. Table 2 gives the resulting performance metrics.

These local gradient bounds however are often extremely conservative and thus this correction may greatly overapproximate the local surface displacement and lead to unnecessary re-evaluations. Regions with zero-crossing gradient bounds and nonzero pre-normalized change can now no longer tolerate any amount of error in the field and are thus forced to re-evaluate in every timestep, thus resulting in costly updates far from the motion. To illustrate this effect, we add to our test scene fields an infinitesimal global time-varying perturbation to ensure the change is nowhere zero, as would be the case if our scenes were deforming under the influence of a non-finitely supported blending operation. The results are shown in Figure 13 and the corresponding performance metrics of this stress test are given in Table 3. The higher-order range-bounding techniques discussed in Section 3.1.1 may render this approach more practical by better bounding the gradients. We note that many implicit models of interest (e.g., on Shadertoy) are approximate signed distance fields where change in the implicit field value translates almost directly to surface displacement in object-space units. For such scenes, Lipschitz correction is not necessary.

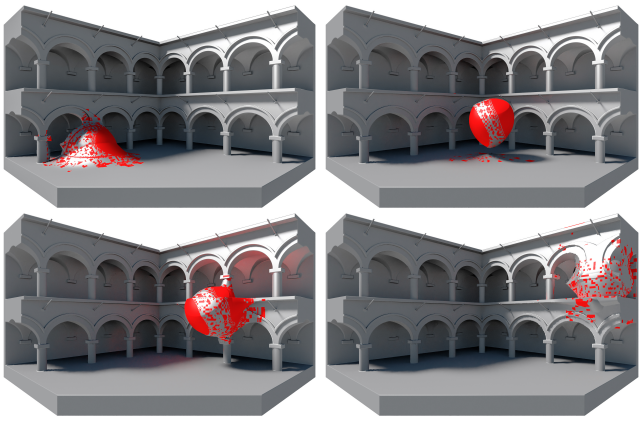


Fig. 9. Snapshots of our "Sponza" scene at frames 236, 365, 437 and 602. Regions colored red were evaluated in the latest timestep. Here $\delta = 0.01$ at 1024^3 octree resolution.

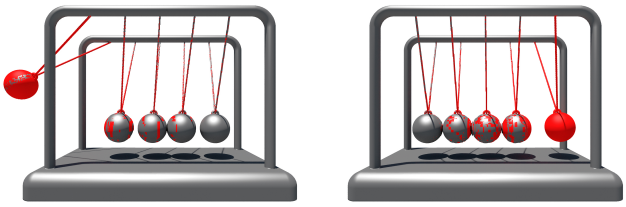


Fig. 10. "Cradle" scene snapshots at frames 11 and 40. Red regions were evaluated in the latest timestep. Here $\delta = 0.01$ at 1024^3 octree resolution.

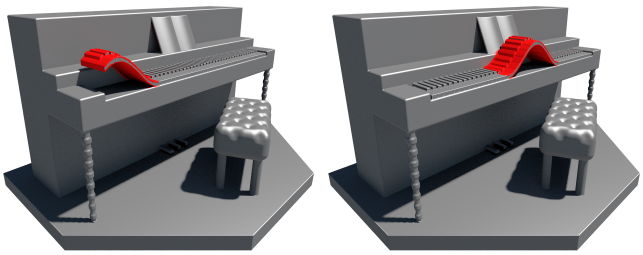
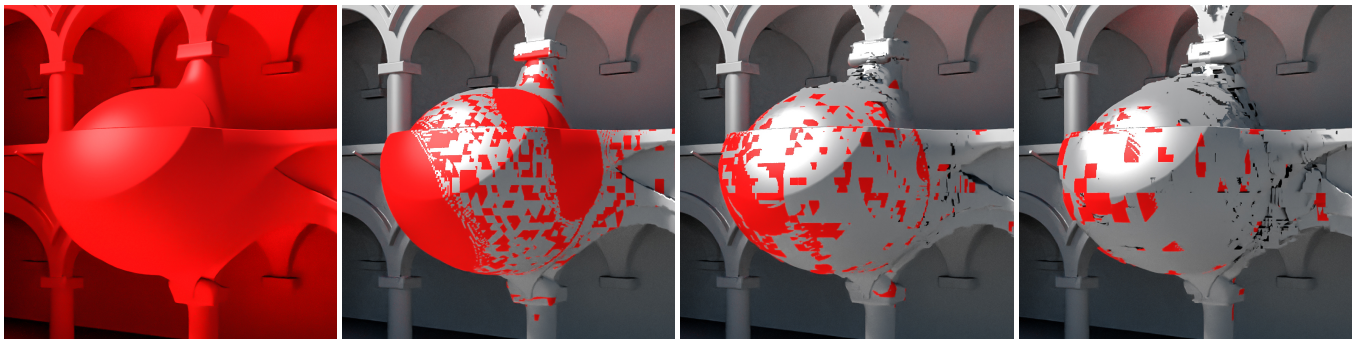
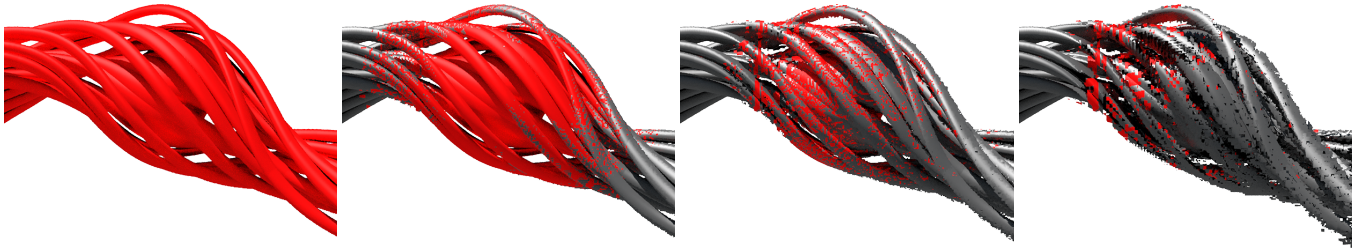


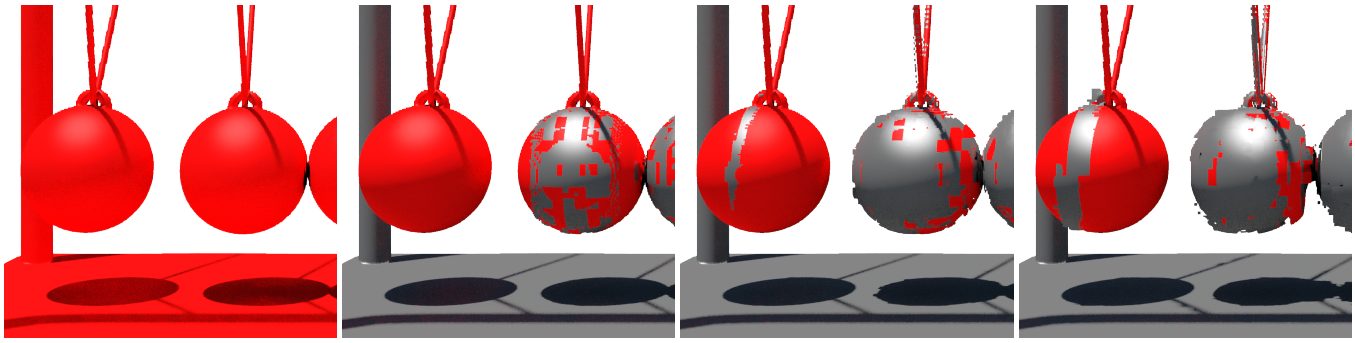
Fig. 11. Snapshots of our "Piano" scene at frames 87 and 190 showing a wave propagating through the keys of a piano. Regions colored red were evaluated in the latest timestep. Here $\delta = 0.01$ at 1024^3 octree resolution.



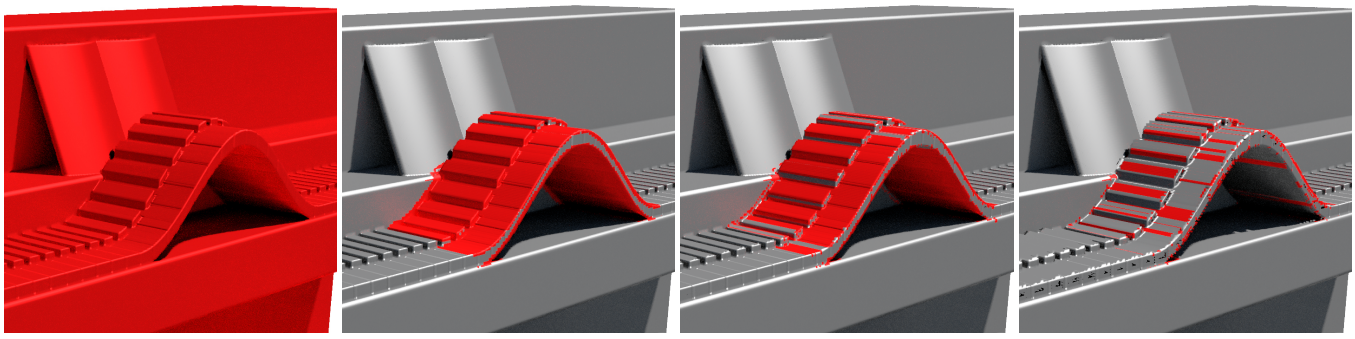
(a) Cropped view of frame 962 of our "Sponza" scene. Here the blob has already reached the right edge of the scene and is moving back towards the left.



(b) Cropped view of frame 330 of our "Spaghetti" scene. Here the meatball is moving towards the left.



(c) Cropped view of frame 35 of our "Cradle" scene. Here the ball on the left is moving towards the right. Note the right ball is not static.



(d) Cropped view of frame 190 of our "Piano" scene. Here the wave is moving towards the right.

Fig. 12. Update patterns and temporal artifacts in our pavings at $\delta = 0.0, 0.01, 0.03$ and 0.06. Regions colored red were evaluated in the current timestep.

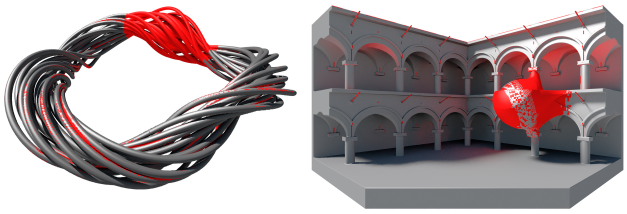


Fig. 13. Lipschitz-normalized scenes with $\delta = 0.01$ now bounding the maximum surface displacement in distance units where the total scene width is 10 units. To each field we've added a small global time-varying perturbation to ensure the change is nowhere zero. At 1024^3, this bounds the error in the paving to within two leaf boxes of the ground truth. Regions whose gradient bounds contain zero cannot tolerate any error and must thus re-evaluate every frame, resulting in red regions far from the motion.

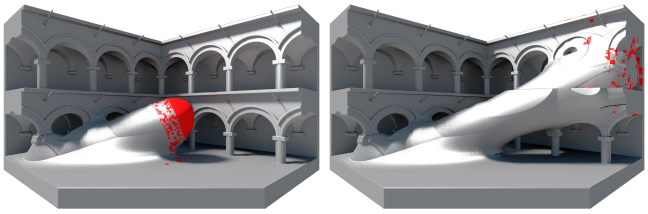


Fig. 14. Swept volume of our Sponza scene at 512^3 with $\delta = 0.01$. Please refer to our supplemental video for swept volumes of our other test scenes.

6.2 Swept Volumes

Our algorithm easily adapts to the problem of efficiently deriving a bounded-error *swept volume* defined formally as the continuous union of an animated implicit volume over time via

$$\mathbb{X} = \bigcup_{t \in [t]} f_t^{-1}([-\infty, 0]). \tag{27}$$

To do this, we simply disable the box deallocations from Section 4.6.2, the fast evaluations from Section 4.4, and the temporal contractions from Section 6.4, and evaluate newly allocated boxes on Line 14 of Algorithm 2 over the latest timestep instead of at the current time t. We early-terminate the line 7 for-loop if the current box $[\boldsymbol{b}]$ is not new and $[\boldsymbol{b}].[F]_+ < 0$ to avoid re-paving already-filled regions.

This approach overcomes the main limitations of "stamping" techniques which approximate $\mathbb{X}$ as a union of the implicit volume at a finite set of discrete timesteps. As noted by Sellán et al. [2021], the stamping frequency required to achieve alias-free output depends on the local scene complexity and surface velocity. Faithfully capturing thin, fast-moving features therefore requires full-scene re-discretization at many small timesteps. Our approach instead evaluates f over continuous time via interval arithmetic, re-discretizes only the moving surface frontier in each step, and automatically varies the local stamping rate over the search domain by re-evaluating regions as infrequently as possible while maintaining a bounded-error approximation of the surface at all times. Figure 14 shows swept volumes of our test scenes.

An alternative approach to deriving swept volumes using interval arithmetic without Temporal SIVIA would be to simply pave the domain in one shot over the entire time interval $[t]$. This approach however is non-interactive and extremely inefficient as the long time interval exaggerates the interval overapproximation beyond practicality. We note the numerical continuation method proposed by Sellán et al. [2021] is otherwise orthogonal to our approach.

Table 1. Performance metrics of our algorithm on our test scenes at various octree resolutions $(\text{Res})^3$ for various values of δ. For each configuration we report the average frametime in ms (Avg. FT), the total speedup factor (Speedup) over the animation, and the average per-frame speedup factor (Avg. Speedup). Timings do not include visualization, only octree updates.

	Res	δ	Avg. FT	Speedup	Avg. Speedup
Sponza	1024	0.0	1304.05	1	1
		0.01	66.17	19.71	71.95
		0.03	31.47	41.44	126.01
		0.06	19.56	66.68	193.99
	256	0.0	85.70	1	1
		0.01	7.03	12.19	16.02
		0.03	4.70	18.25	22.16
		0.06	3.58	23.92	28.33
Spaghet	1024	0.0	686.04	1	1
		0.01	224.82	3.05	3.10
		0.03	120.34	5.70	5.92
		0.06	55.02	12.47	13.38
	256	0.0	37.82	1	1
		0.01	12.06	3.14	3.16
		0.03	10.73	3.52	3.55
		0.06	8.28	4.57	4.63
Cradle	1024	0.0	217.85	1	1
		0.01	35.76	6.09	6.66
		0.03	26.86	8.11	9.19
		0.06	21.67	10.05	11.63
	256	0.0	20.38	1	1
		0.01	5.64	3.61	3.72
		0.03	4.99	4.09	4.20
		0.06	4.66	4.38	4.51
Piano	1024	0.0	540.87	1	1
		0.01	18.86	28.68	87.84
		0.03	17.08	31.67	92.37
		0.06	14.05	38.50	98.42
	256	0.0	39.32	1	1
		0.01	2.92	13.48	14.86
		0.03	2.96	13.29	14.59
		0.06	2.90	13.54	14.94

6.3 Gradient Contraction

To accelerate the convergence of SIVIA, inclusion functions can in general be replaced by interval contractors [Benhamou et al. 1999; Chabert and Jaulin 2009] which utilize constraint propagation to shave off empty parts of an input box $[\boldsymbol{b}]$ to produce a potentially smaller box $[\boldsymbol{b}']$ such that no feasible points are lost in the process. These contraction operations require a full linear-time backwards pass over the forward interval trace of f similar to reverse-mode automatic differentiation. At the end of this process, the input box is

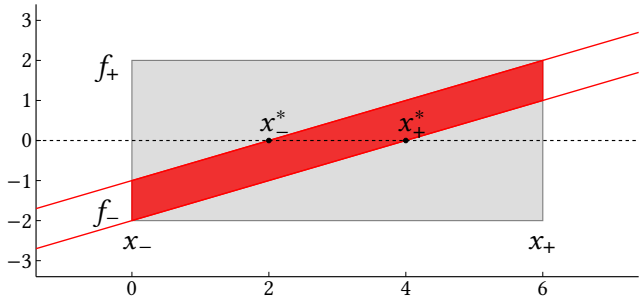


Fig. 15. Gradient contraction visualization. Suppose that $f([x]) = [-2, 2]$ over an interval $[x] = [0, 6]$. This bounds the function within the gray box. Suppose also that $0.5 = |[\partial f/\partial x([x])]|_-$ is a known lower bound on the magnitude of the derivative of f with respect to x over $[x]$. This bounds the function within the red zone between the two red lines. We then intersect this zone with the x-axis to derive a contraction of the input domain $[x^*] = [x^*_-, x^*_+] = [2, 4]$ that bounds the feasible root locations.

not guaranteed to contract at all and we find this outcome is common on the complex implicit functions in our test scenes. Contractions can therefore sometimes increase the overall runtime.

We instead propose a constant-time gradient-based contraction scheme inspired by the interval Newton method [Hansen and Greenberg 1983] that acts where the field is locally continuous and provably monotonic using the freely-available local gradient information required for our main algorithm. Suppose for example that $f([x]) = [-2, 2]$ over $[x] = [0, 6]$ where $\partial f/\partial x([x]) = [0.5, 5]$. We note the function f cannot be defined below the line $0.5x - 2$ or above the line $0.5(x - 6) + 2$ as f cannot change slow enough to reach these areas. More formally, if $\exists\, x' \in [x], f(x') < 0.5x' - 2 \vee f(x') > 0.5(x' - 6) + 2$ then, by the mean value theorem, there must exist some x'' in the open interval (x_-, x_+) where our known gradient bound is violated. This thick linear bound on the function can then be intersected with the x-axis to bound the feasible root-locations over the domain via

$$[x^*_-, x^*_+] = [x] \cap \left(\frac{-f([x])}{M} + [x]_-\right) \cap \left(\frac{-f([x])}{M} + [x]_+\right)$$
$$\text{where } M = \text{sign}\left(\frac{\partial f}{\partial x}([x])\right) \cdot \left|\frac{\partial f}{\partial x}([x])\right|_- \tag{28}$$

Like constraint-based contractors, this scheme is not guaranteed to restrict the domain and in our case often does not due to overapproximated gradient bounds, but comes at almost no additional cost. Higher-order range-bounding techniques that more tightly bound the gradients may yield improved performance. Figure 15 illustrates this process and Figure 16 applies it component-wise to each box in a low-resolution paving of a sphere to produce a smoother result.

6.4 Temporal Contraction

The gradient-based contraction scheme described in Section 6.3 applies just as well in the temporal axis where it reduces to a simple inclusion relationship. If $[\Delta f]$ over a box $[\boldsymbol{b}]$ over a time interval $[t]$ is strictly positive or negative, this implies the field everywhere in $[\boldsymbol{b}]$ is monotonically increasing or decreasing. We can use this fact to obtain a tighter bound on the field over $[\boldsymbol{b}]$ *at* t_+ to potentially

Table 2. Performance metrics of our algorithm on our test scenes at various octree resolutions $(\text{Res})^3$ for various values of δ using Lipschitz correction as described in Section 6.1. For each configuration we report the average frametime in ms (Avg. FT), the total speedup factor (Speedup) over the animation, as well as the average per-frame speedup factor (Avg. Speedup).

	Res	δ	Avg. FT	Speedup	Avg. Speedup
Sponza	1024	0.0	1304.05	1	1
		0.01	115.15	11.32	28.52
		0.03	74.99	17.39	40.40
		0.06	56.82	22.95	48.86
	256	0.0	85.70	1	1
		0.01	10.82	7.92	9.69
		0.03	9.35	9.17	10.90
		0.06	8.51	10.06	11.68
Spaghet	1024	0.0	686.04	1	1
		0.01	285.00	2.41	2.42
		0.03	284.54	2.41	2.43
		0.06	281.68	2.44	2.45
	256	0.0	37.82	1	1
		0.01	12.19	3.10	3.14
		0.03	12.14	3.11	3.14
		0.06	12.17	3.11	3.13
Cradle	1024	0.0	217.85	1	1
		0.01	45.64	4.77	5.07
		0.03	35.69	6.10	6.52
		0.06	31.85	6.84	7.35
	256	0.0	20.38	1	1
		0.01	6.24	3.26	3.32
		0.03	5.76	3.54	3.59
		0.06	5.52	3.69	3.76
Piano	1024	0.0	540.87	1	1
		0.01	18.90	28.62	87.53
		0.03	18.84	28.72	86.56
		0.06	18.68	28.95	88.93
	256	0.0	39.32	1	1
		0.01	2.96	13.26	14.41
		0.03	2.91	13.50	14.74
		0.06	2.91	13.51	14.93

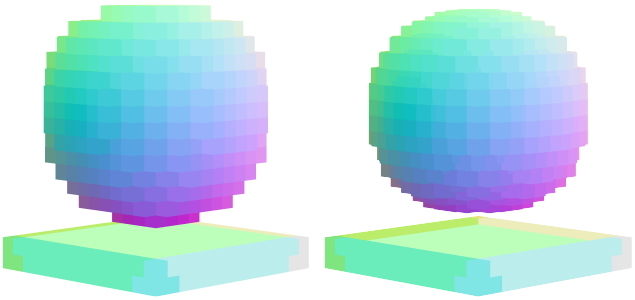


Fig. 16. Gradient contraction of a low-resolution 32^3 paving (left) of a sphere produces a smoother result (right). The effect is less noticeable at higher resolutions and where the gradients are less tightly bounded. Note that boxes can better conform to the surface where the field is nearly axis-aligned.

Fig. 17. Using contractions along the time axis, we can prove infeasible a portion of the previously indeterminate ghost region from Figure 7, visually reducing its density. Note the fast optimization from Section 4.4 is disabled.

disambiguate otherwise indeterminate boxes. Suppose for example that $f([\boldsymbol{b}],[t]) = [-1,12]$ and $[\Delta f] = [2,3]$. This implies the field over $[\boldsymbol{b}]$ at t_+ is bounded by $[1,12]$. For general f more formally,

$$f([\boldsymbol{b}],\boldsymbol{C}(\boldsymbol{t_+})) \subseteq f([\boldsymbol{b}],[\boldsymbol{c}]) \cap f([\boldsymbol{b}],[\boldsymbol{c}]) + [\boldsymbol{\Delta f}]$$
$$\text{where } [\boldsymbol{c}] = [\boldsymbol{C}(t_-),\boldsymbol{C}(t_+)]_u \tag{29}$$
$$\text{and } [\boldsymbol{\Delta f}] \text{ as in Equation 26.}$$

Figure 17 shows the results of this temporal contraction scheme. Like the spatial contractions described in Section 6.3, this scheme has minimal runtime benefit in our test scenes but runs in constant time using the freely available local gradient information and thus cannot harm runtime performance.

6.5 Normals

The surface normals of an implicit volume are defined by the gradients of the field function ∇f. Our algorithm already bounds the local field gradient within each box and it therefore seems natural to re-used this information for rendering purposes, but we do not recommend this. As discussed in Section 4.6.3, we point-sample the field in the center of each indeterminate leaf box to counteract interval overapproximation and clean up discontinuities. This may, however, remove the surface layer of the octree and thus expose the underlying internal nodes of P_{in} whose gradients we evaluate not at points but over large spacetime volumes, making them unsuitable for rendering purposes. Figure 18 illustrates this effect. Without point-sampling this issue is largely resolved but we can now see the discontinuities in the field as discussed in Section 3.2.1. Our solution is to simply recalculate the normals per-pixel during the visualization pass via floating-point finite-difference of the implicit field. Keeter [2020] uses a similar strategy via per-pixel forward-mode automatic differentiation.

6.6 Ray Marching in Leaves

The raytracing techniques described in Section 2.1 can easily be used within the leaves of our octree to improve the visual quality of low-resolution pavings. By starting rays close to the surface and exploiting the freely-available local gradient information required for our main algorithm, we perform Lipschitz-corrected sphere-tracing using only ~3 march steps per box compared to the hundreds that are typically required for artifact-free results.

In practice, sphere-traced rays often pass through modulo-induced discontinuities in empty space without visual artifacts. Sphere-tracing the leaves of our octree therefore allows us to disable the point-sampling discussed in Section 3.2.1 to avoid the surface-layer stripping issue described in Section 6.5 *without* also visualizing the indeterminate leaf boxes around zero-straddling discontinuities in the field. Note that due to the non-conservative nature of our octree, parts of the interface between P'_{out} and P'_i where such rays start marching may lie inside the implicit volume. This is rarely an issue in practice. Figure 19 illustrates the resultant gain in visual quality.

Table 3. Performance metrics of our algorithm on our test scenes under a small global time-varying perturbation at various octree resolutions $(\text{Res})^3$ for various values of δ using Lipschitz correction as described in Section 6.1. For each configuration we report the average frametime in ms (Avg. FT), the total speedup factor (Speedup) over the animation, as well as the average per-frame speedup factor (Avg. Speedup).

	Res	δ	Avg. FT	Speedup	Avg. Speedup
Sponza	1024	0.0	1304.5	1	1
		0.01	330.12	3.95	4.09
		0.03	288.09	4.53	4.60
		0.06	270.17	4.83	4.87
	256	0.0	85.70	1	1
		0.01	43.05	1.99	2.00
		0.03	41.61	2.06	2.07
		0.06	40.81	2.10	2.10
Spaghet	1024	0.0	686.04	1	1
		0.01	546.75	1.25	1.26
		0.03	548.56	1.25	1.25
		0.06	542.32	1.27	1.27
	256	0.0	37.82	1	1
		0.01	36.95	1.02	1.02
		0.03	36.96	1.02	1.02
		0.06	37.02	1.02	1.02
Cradle	1024	0.0	218.85	1	1
		0.01	45.93	4.74	5.03
		0.03	36.27	6.01	6.42
		0.06	32.18	6.77	7.24
	256	0.0	20.38	1	1
		0.01	6.79	3.00	3.04
		0.03	6.27	3.25	3.30
		0.06	6.04	3.38	3.43
Piano	1024	0.0	540.87	1	1
		0.01	80.51	6.72	6.93
		0.03	80.43	6.72	6.94
		0.06	80.30	6.74	6.95
	256	0.0	39.32	1	1
		0.01	12.97	3.03	3.05
		0.03	12.90	3.05	3.07
		0.06	12.91	3.05	3.06

7 FUTURE WORK

We identify several opportunities for potential related research on animated temporally-coherent set inversion in computer graphics.

7.1 View-Dependence

For pure visualization tasks where a global discretization is not required, our algorithm can be further accelerated by introducing view-dependent effects like ray-guided subdivision [Crassin et al. 2009] where octree construction and maintenance is interleaved with raytracing passes that atomically accumulate ray/box intersections to determine the set of currently visible boxes that require further subdivsion. This effectively restricts the per-frame re-evaluations to only the unoccluded, visibly-moving surfaces.

The temporal error tolerance δ can likewise be made view dependant to update occluded, distant, or out-of-frustum boxes less frequently. By varying δ by the dot product of the viewing ray with the surface normal, we can also reduce temporal artifacts on object silhouettes where the are most noticeable. If combining SIVIA with raymarching as described in Section 6.6 then one may augment our algorithm to not subdivide the search space until all indeterminate boxes are sufficiently small, but until the Lipschitz constants within each box are bounded to a sufficient tightness, thereby allowing raymarchers to operate more efficiently.

7.2 Function Approximation

Merging our algorithm with the function approximation strategies described in Section 3.2.2 is a promising avenue for future research. Constructing such a combined algorithm however is highly non-trivial as a program specialized over a box $[\boldsymbol{B}]$ over a time interval $[[\boldsymbol{B}].t, t]$ cannot be used in its child box $[\boldsymbol{b}]$ over a potentially longer time interval $[[\boldsymbol{b}].t, t]$. Child programs are therefore not guaranteed to be subsets of their parent's programs and would need to temporally accumulate in some fashion.

7.3 Healing

The main limitation of our current algorithm is that high-error regions of the octree remain high-error indefinitely if no further motion is detected in the area. One would ideally wish to "heal" such regions over time by potentially dedicating some fraction of the per-frame runtime to re-evaluating high-error boxes in static regions where artifacts are most noticeable. This would enable the use of larger δ values with greater speedups at similar levels of perceived error. Localizing such artifacted regions however is highly nontrivial due to the non-hierarchical nature of $[\boldsymbol{b}].\delta$ described in Section 4.3.

7.4 Program Transformations

The interactive implicit modelling process involves not just the manipulation of object transforms and shape parameters but the addition and removal of shapes which discontinuously transform the computational graph of f. Our algorithm does not currently support temporally coherent updates in such cases and instead performs a full octree rebuild upon structural changes to f itself. For small localized transformations, one could in principle re-pave only the regions of space where the change in the field with respect to the modified subexpression is sufficiently high, but more extensive transformations may require more sophisticated program analysis.

An additional consideration is the time required to recompile f and upload the transformed instructions to the GPU. Our current workflow described in Section 4.6.4 is slow to update due to the LLVM compilation step. A dedicated compiler that directly outputs interval and autodiff code in WGSL would be ideal to speedup the workflow and sidestep the large interpreter overhead at runtime.

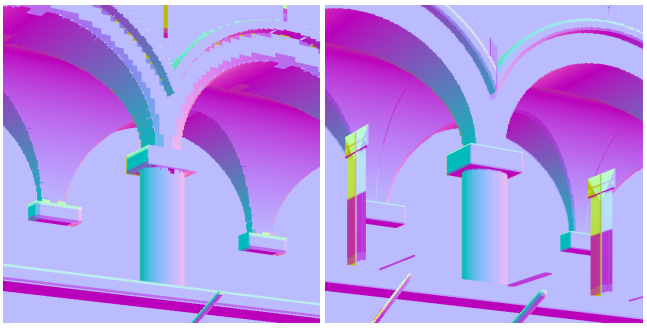


Fig. 18. Keeter-style point-sampling strips off the surface layer of the octree, revealing internal nodes with gradients unsuitable for rendering purposes (left). Disabling point-sampling (right) largely fixes this issue but reveals the discontinuities in the field as discussed in Section 3.2.1. Point-sampling the normals per-pixel at visualization time resolves this issue.

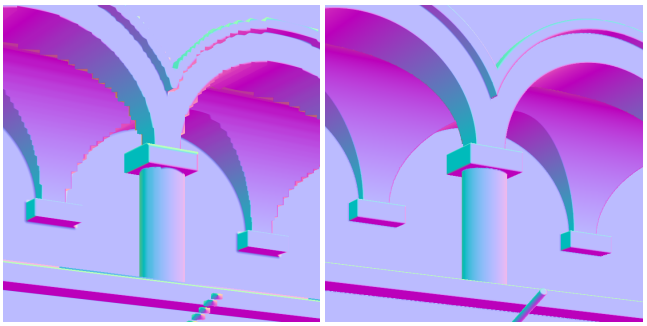


Fig. 19. Raymarching through the leaf boxes of a 256^3 octree (left) produces a smooth result (right) which is almost indistinguishable from a 1024^3 octree.

8 CONCLUSION

In summary, our work introduces a novel temporally-coherent SIVIA variant that allows us to trade temporal accuracy for speed. We apply our algorithm to the computer graphics problem of animated implicit surface rendering and achieve significant speedups in complex scenes with localized deformations commonly found in games and modelling applications where interactivity is required, a global discretization is desired, and bounded-error approximation is acceptable. We augment our algorithm to efficiently render bounded-error swept volumes and extend it via gradient-based contractions and raymarching through low-resolution voxelizations. Finally, we provide an in-depth discussion on state-of-the-art interval subdivision schemes in computer graphics from the set inversion perspective and outline promising directions for future research in this area.

ACKNOWLEDGMENTS

We'd like to thank Inigo Quilez, David P. Sanders, and Luc Jaulin for their inspiring work, as well as Joey Litalien, Michael Kenzel, Connor Fitzgerald, Kenny Erleben, and the anonymous reviewers for their help. We also acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC).

REFERENCES

T. Aila and S. Laine. 2009. Understanding the Efficiency of Ray Traversal on GPUs. In *Proceedings of the Conference on High Performance Graphics 2009* (New Orleans, Louisiana) *(HPG '09)*. Association for Computing Machinery, New York, NY, USA, 145–149. https://doi.org/10.1145/1572769.1572792

G. Allen. 2019. nTopology Modeling Technology. https://ntopology.com/resources/whitepaper-implicit-modeling-technology/. Accessed: 2022-09-01.

J. Amanatides and A. Woo. 1987. A Fast Voxel Traversal Algorithm for Ray Tracing. In *EG 1987-Technical Papers*. Eurographics Association, 3–10. https://doi.org/10.2312/egtp.19871000

W. Barth, R. Lieger, and M. Schindler. 1994. Ray tracing general parametric surfaces using interval arithmetic. *The Visual Computer* 10, 7 (01 Aug 1994), 363–371. https://doi.org/10.1007/BF01900662

F. Benhamou, F. Goualard, L. Granvilliers, and J.-F. Puget. 1999. Revising Hull and Box Consistency. In *INT. CONF. ON LOGIC PROGRAMMING*. MIT press, 230–244.

M. Berz and G. Hoffstätter. 1998. Computation and Application of Taylor Polynomials with Interval Remainder Bounds. *Reliable Computing* 4, 1 (Feb. 1998), 83–97.

A. Bouthors and M. Nesme. 2007. Twinned meshes for dynamic triangulation of implicit surfaces. In *Proceedings of Graphics Interface 2007* (Montréal, Québec, Canada) *(GI 2007)*. 3–9.

G. Chabert and L. Jaulin. 2009. Contractor programming. *Artificial Intelligence* 173, 11 (2009), 1079–1100. https://www.sciencedirect.com/science/article/pii/S0004370209000381

L. D. Comba and J. Stolfi. 1990. Affine Arithmetic and Its Applications to Computer Graphics.

C. Crassin, F. Neyret, S. Lefebvre, and E. Eisemann. 2009. GigaVoxels: Ray-Guided Streaming for Efficient and Detailed Voxel Rendering. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games* (Boston, Massachusetts) *(I3D '09)*. Association for Computing Machinery, New York, NY, USA, 15–22. https://doi.org/10.1145/1507149.1507152

A. De Cusatis, L. De Figueiredo, and M. Gattass. 1999. Interval methods for ray casting implicit surfaces with affine arithmetic. In *XII Brazilian Symposium on Computer Graphics and Image Processing (Cat. No.PR00481)*. 65–71. https://doi.org/10.1109/SIBGRA.1999.805711

F. De Goes and D. L. James. 2017. Regularized Kelvinlets: Sculpting Brushes Based on Fundamental Solutions of Elasticity. *ACM Trans. Graph.* 36, 4, Article 40 (jul 2017), 11 pages. https://doi.org/10.1145/3072959.3073595

B. Desrochers and L. Jaulin. 2017. Thick set inversion. *Artificial Intelligence* 249 (2017), 1–18. https://doi.org/10.1016/j.artint.2017.04.004

J. Deussen, J. Riehme, and U. Naumann. 2016. Automation of significance analyses with interval splitting. In *Parallel Computing: On the Road to Exascale*. IOS Press, 731–740.

J. Díaz, M. Sbert, J. Casellas, and I. i. A. Universitat de Girona. Departament d'Electrònica. 2008. *Improvements in the Ray Tracing of Implicit Surfaces Based on Interval Arithmetic*. Universitat de Girona. Escola Politècnica Superior. Departament d'Electrònica, Informàtica i Automàtica. https://books.google.dk/books?id=brsRzQEACAAJ

T. Duff. 1992. Interval Arithmetic Recursive Subdivision for Implicit Functions and Constructive Solid Geometry. *SIGGRAPH Comput. Graph.* 26, 2 (jul 1992), 131–138. https://doi.org/10.1145/142920.134027

O. Fryazinov, A. Pasko, and P. Comninos. 2010. Fast reliable interrogation of procedurally defined implicit surfaces using extended revised affine arithmetic. *Computers & Graphics* 34, 6 (2010), 708–718. https://doi.org/10.1016/j.cag.2010.07.003 Graphics for Serious Games Computer Graphics in Spain: a Selection of Papers from CEIG 2009 Selected Papers from the SIGGRAPH Asia Education Program.

E. Galin, E. Guérin, A. Paris, and A. Peytavie. 2020. Segment Tracing Using Local Lipschitz Bounds. *Computer Graphics Forum* 39, 2 (2020), 545–554.

M. Gleicher and M. Kass. 1992. An interval refinement technique for surface intersection. In *Proceedings of Graphics Interface '92* (Vancouver, British Columbia, Canada) *(GI '92)*. Canadian Human-Computer Communications Society, Toronto, Ontario, Canada, 242–249. http://graphicsinterface.org/wp-content/uploads/gi1992-28.pdf

O. Gourmel, A. Pajot, M. Paulin, L. Barthe, and P. Poulin. 2010. Fitted BVH for Fast Raytracing of Metaballs. *Computer Graphics Forum* 29, 2 (2010), 281–288. https://doi.org/10.1111/j.1467-8659.2009.01597.x

K. Gupta, J. A. Stuart, and J. D. Owens. 2012. A study of Persistent Threads style GPU programming for GPGPU workloads. In *2012 Innovative Parallel Computing (InPar)*. 1–14. https://doi.org/10.1109/InPar.2012.6339596

E. Hansen and R. Greenberg. 1983. An interval Newton method. *Appl. Math. Comput.* 12, 2 (1983), 89–98. https://doi.org/10.1016/0096-3003(83)90001-2

E. R. Hansen. 1992. *Global optimization using interval analysis*.

J. C. Hart. 1996. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer* 12, 10 (1996), 527–545.

J. C. Hart, E. Bachta, W. Jarosz, and T. Fleury. 2002. Using Particles to Sample and Control More Complex Implicit Surfaces. In *SMI '02: Proceedings of the Shape Modeling International 2002 (SMI'02)*. IEEE Computer Society, Washington, DC, USA, 129.

M. Innes. 2018. Don't unroll adjoint: Differentiating ssa-form programs. In *arXiv preprint arXiv:1810.07951*.

L. Jaulin, M. Kieffer, O. Didrit, and E. Walter. 2001. *Applied Interval Analysis with Examples in Parameter and State Estimation, Robust Control and Robotics*. Springer London Ltd. 398 pages. https://hal.archives-ouvertes.fr/hal-00845131

L. Jaulin and E. Walter. 1993. Set inversion via interval analysis for nonlinear bounded-error estimation. *Automatica* 29, 4 (1993), 1053–1064. https://doi.org/10.1016/0005-1098(93)90106-4

D. Jevans, B. Wyvill, and G. Wyvill. 1988. Speeding up 3-D animation for simulation. In *Proceedings of the Fourth SCS Multiconference on Multiprocessors and Array Processors*. 94–100.

T. Ju, F. Losasso, S. Schaefer, and J. Warren. 2002. Dual contouring of hermite data. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. 339–346.

M. J. Keeter. 2020. Massively Parallel Rendering of Complex Closed-Form Implicit Surfaces. *ACM Trans. Graph.* 39, 4, Article 141 (jul 2020), 10 pages. https://doi.org/10.1145/3386569.3392429

B. Kerbl, M. Kenzel, J. H. Mueller, D. Schmalstieg, and M. Steinberger. 2018. The Broker Queue: A Fast, Linearizable FIFO Queue for Fine-Granular Work Distribution on the GPU. In *Proceedings of the 2018 International Conference on Supercomputing* (Beijing, China) *(ICS '18)*. Association for Computing Machinery, New York, NY, USA, 76–85. https://doi.org/10.1145/3205289.3205291

A. Knoll, Y. Hijazi, A. Kensler, M. Schott, C. Hansen, and H. Hagen. 2009. Fast Ray Tracing of Arbitrary Implicit Surfaces with Interval and Affine Arithmetic. *Computer Graphics Forum* 28, 1 (2009), 26–40. https://doi.org/10.1111/j.1467-8659.2008.01189.x

J. Korndörfer, B. Keinert, U. Ganse, M. Sänger, S. Ley, K. Burkhardt, M. Spuler, and J. Heusipp. 2015. HG_SDF: A glsl library for building signed distance functions. https://mercury.sexy/hg_sdf. Accessed: 2021-07-28.

C. Lattner and V. Adve. 2004. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation *(CGO '04)*. IEEE Computer Society, USA, 75.

R. Levien. 2021. Prefix sum on portable compute shaders. https://raphlinus.github.io/gpu/2021/11/17/prefix-sum-portable.html

W. E. Lorensen and H. E. Cline. 1987. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '87)*. Association for Computing Machinery, New York, NY, USA, 163–169. https://doi.org/10.1145/37401.37422

A. Majercik, C. Crassin, P. Shirley, and M. McGuire. 2018. A Ray-Box Intersection Algorithm and Efficient Dynamic Voxel Rendering. *Journal of Computer Graphics Techniques (JCGT)* 7, 3 (20 September 2018), 66–81. http://jcgt.org/published/0007/03/04/

M. Merchante. 2018. Sponza pt2. Shadertoy. https://www.shadertoy.com/view/ltGcRW

A. Mercier-Aubin, A. Winter, D. I. W. Levin, and P. G. Kry. 2022. Adaptive Rigidification of Elastic Solids. *ACM Trans. Graph.* 41, 4, Article 71 (jul 2022), 11 pages. https://doi.org/10.1145/3528223.3530124

D. Merrill and M. Garland. 2016. *Single-pass parallel prefix scan with decoupled look-back*. Technical Report. NVIDIA, Tech. Rep. NVR-2016-002.

D. P. Mitchell. 1990. Robust Ray Intersection with Interval Arithmetic. In *Proceedings of Graphics Interface '90* (Halifax, Nova Scotia, Canada) *(GI '90)*. Canadian Man-Computer Communications Society, Toronto, Ontario, Canada, 68–74. http://graphicsinterface.org/wp-content/uploads/gi1990-8.pdf

R. Moore. 1966. *Interval Analysis*. Prentice-Hall.

NAGA 2023. NAGA: Universal Shader Translation in Rust. GitHub. https://github.com/gfx-rs/naga

A. Opalach and M.-P. Cani. 1997. Local Deformation for Animation of Implicit Surfaces. In *Spring Conference on Computer Graphics (SCCG)*.

I. Quilez. 2008. Rendering Worlds with Two Triangles with raytracing on the GPU in 4096 bytes. (2008). NVSCENE 08.

I. Quilez. 2013. Piano. Shadertoy. https://www.shadertoy.com/view/ldl3zN

D. Ratz. 1996. *An optimized interval slope arithmetic and its application*. Inst. für Angewandte Mathematik.

S. Redon, Y. J. Kim, M. C. Lin, and D. Manocha. 2005. Fast Continuous Collision Detection for Articulated Models. *Journal of Computing and Information Science in Engineering* 5, 2 (2005), 126–137. https://doi.org/10.1115/1.1884133

J. Riehme and U. Naumann. 2015. Significance Analysis for Numerical Models. In *1st WorkShop on Approximate Computing (WAPCO 2015)*.

D. P. Sanders. 2020. Fast global optimization on the GPU. JuliaCon 2020. https://live.juliacon.org/uploads/posters/8K8P7R.pdf

D. P. Sanders, L. Benet, L. Ferranti, K. Agarwal, B. Richard, J. Grawitter, E. Gupta, M. Forets, M. F. Herbst, yashrajgupta, E. Hanson, B. van Dyk, C. Rackauckas, R.

Vasani, S. Micluța-Câmpeanu, S. Olver, T. Koolen, C. Wormell, D. Karrasch, F. A. Vázquez, G. Dalle, J. Sarnoff, J. TagBot, K. O'Bryant, K. Carlsson, M. Piibeleht, M. Giordano, Ryan, R. Deits, and T. Holy. 2022. *JuliaIntervals/IntervalArithmetic.jl: v0.20.8.* https://doi.org/10.5281/zenodo.7257716

R. Schmidt. 2006. *Interactive Modeling with Implicit Surfaces.* Master's thesis. The University of Calgary.

R. Schmidt, B. Wyvill, and E. Galin. 2005. Interactive implicit modeling with hierarchical spatial caching. In *International Conference on Shape Modeling and Applications 2005 (SMI' 05).* 104–113. https://doi.org/10.1109/SMI.2005.25

S. Sellán, N. Aigerman, and A. Jacobson. 2021. Swept Volumes via Spacetime Numerical Continuation. *ACM Trans. Graph.* 40, 4, Article 55 (jul 2021), 11 pages. https://doi.org/10.1145/3450626.3459780

D. Seyb, A. Jacobson, D. Nowrouzezahrai, and W. Jarosz. 2019. Non-Linear Sphere Tracing for Rendering Deformed Signed Distance Fields. *ACM Trans. Graph.* 38, 6, Article 229 (nov 2019), 12 pages. https://doi.org/10.1145/3355089.3356502

N. Sharp and A. Jacobson. 2022. Spelunking the Deep: Guaranteed Queries on General Neural Implicit Surfaces via Range Analysis. *ACM Trans. Graph.* 41, 4, Article 107 (jul 2022), 16 pages. https://doi.org/10.1145/3528223.3530155

J. M. Snyder. 1992. Interval Analysis for Computer Graphics. *SIGGRAPH Comput. Graph.* 26, 2 (jul 1992), 121–130. https://doi.org/10.1145/142920.134024

J. M. Snyder, A. R. Woodbury, K. Fleischer, B. Currin, and A. H. Barr. 1993. Interval Methods for Multi-Point Collisions between Time-Dependent Curved Surfaces. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques* (Anaheim, CA) *(SIGGRAPH '93).* Association for Computing Machinery, New York, NY, USA, 321–334. https://doi.org/10.1145/166117.166158

J. Stam and R. Schmidt. 2011. On the Velocity of an Implicit Surface. *ACM Trans. Graph.* 30, 3, Article 21 (may 2011), 7 pages. https://doi.org/10.1145/1966394.1966400

M. Steinrucken. 2021. Newton's Cradle Tutorial. Shadertoy. https://www.shadertoy.com/view/sdsXWr

J. Tupper. 2001. Reliable Two-Dimensional Graphing Methods for Mathematical Formulae with Two Free Variables. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '01).* Association for Computing Machinery, New York, NY, USA, 77–86. https://doi.org/10.1145/383259.383267

V. Vassiliadis, J. Riehme, J. Deussen, K. Parasyris, C. D. Antonopoulos, N. Bellas, S. Lalis, and U. Naumann. 2016. Towards Automatic Significance Analysis for Approximate Computing. In *Proceedings of the 2016 International Symposium on Code Generation and Optimization* (Barcelona, Spain) *(CGO '16).* Association for Computing Machinery, New York, NY, USA, 182–193. https://doi.org/10.1145/2854038.2854058

X.-H. Vu, D. Sam-Haroud, and B. Faltings. 2009. Enhancing numerical constraint propagation using multiple inclusion representations. *Annals of Mathematics and Artificial Intelligence* 55, 3 (March 2009), 295.

B. Wang, Z. Ferguson, T. Schneider, X. Jiang, M. Attene, and D. Panozzo. 2021. A Large-Scale Benchmark and an Inclusion-Based Algorithm for Continuous Collision Detection. *ACM Trans. Graph.* 40, 5, Article 188 (sep 2021), 16 pages. https://doi.org/10.1145/3460775

F. White, M. Abbott, M. Zgubic, J. Revels, S. Axen, A. Arslan, S. Schaub, N. Robinson, Y. Ma, G. Dhingra, W. Tebbutt, D. Widmann, N. Heim, N. Schmitz, A. D. W. Rosemberg, C. Rackauckas, C. Lucibello, R. Heintzmann, frankschae, A. Noack, K. Fischer, A. Robson, J. F. de Cossio-Diaz, J. Ling, mattBrzezinski, R. Finnegan, A. Zhabinski, D. Wennberg, M. Besançon, and P. Vertechi. 2023. *JuliaDiff/ChainRules.jl: v1.48.0.* https://doi.org/10.5281/zenodo.7669643

A. P. Witkin and P. S. Heckbert. 1994. Using Particles to Sample and Control Implicit Surfaces. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '94).* 269–277. https://doi.org/10.1145/192161.192227

B. Wyvill, A. Guy, and E. Galin. 1999. Extending the CSG Tree. Warping, Blending and Boolean Operations in an Implicit Surface Modeling System. *Computer Graphics Forum* 18, 2 (1999), 149–158. https://doi.org/10.1111/1467-8659.00365

R. Young. 1931. The algebra of many-valued quantities. *Math. Ann.* 104 (1931), 260–290. http://eudml.org/doc/159462

yuntaRobo. 2020. cables2. Shadertoy. https://www.shadertoy.com/view/wlKXWc

X. Zhang, S. Redon, M. Lee, and Y. J. Kim. 2007. Continuous Collision Detection for Articulated Models Using Taylor Models and Temporal Culling. 26, 3 (jul 2007), 15–es. https://doi.org/10.1145/1276377.1276396