Hybrid Rendering Pipeline for Selective Integration of Blended SDFs with Polygonal Geometry

A thesis submitted for the degree of Master of Science (MSc)

by

Anastasia Iosebadze

Faculty of Design, Informatics and Business,
Abertay University.

August, 2025

Table of Contents

Table of Figures

Table of Listings

Abstract

Triangle meshes remain the predominant geometry representation in graphics applications – most authoring tools and rendering pipelines are organized around them. Signed distance fields (SDFs), however, offer an alternative way to describe implicit surfaces, enabling compositional and procedural capabilities that are oftentimes cumbersome to implement with conventional meshes. These strengths come with the downside of more complex, performance-challenging render paths, making SDFs often confined to specific tasks and niche use cases.

This research aims to design, implement, and evaluate a pipeline for using SDFs in smooth surface blending that will allow SDF-based compositional operations to be extended to arbitrary 3D models without incurring the full performance and memory overhead usually associated with them. The core principle of the proposed pipeline would be to compose two different sources of geometry - triangles and SDF – in hopes of balancing out the advantages and setbacks each of them deliver.

Presented application was developed with Direct3D 12 and DirectX Raytracing and featured two concrete variants of this general idea – Hybrid-A, which is a more conventional approach centred around runtime sphere tracing, and Hybrid-B – a method derived from more modern ray tracing solutions. Both of these paths operated over triangle and SDF representations of scene instances and combined the two in a way that would still deliver dynamic blended surfaces and sharp authored details in relevant regions. Final outcomes were compared against each other, as well as with baseline SDF-only approach in terms of performance, memory, and visual fidelity.

Examining test results showed that hybrid pipelines improved both frame times and memory consumption of standard SDF-only method, a modern ray tracing-focused Hybrid-B path delivering up to 60% reductions in frame times with significantly smaller memory footprint. Comparing output images side-by-side revealed some variance in visual

features, but the gap was not overly pronounced and could be further reduced by adjusting configuration parameters.

Generally, the study shows that the concept of selective composition can deliver significant performance gains without sacrificing too much perceived quality, but more importantly, it opens a path to explore various additional SDF-based operations beyond just smooth blending discussed here, in this new context of hybrid combination framework.

1 Introduction

1.1 Background

For a long time, triangle meshes have been the default way to represent 3D geometry in real-time graphical applications. Throughout the years, engines and their rendering pipelines have been organized around this assumption, meaning common pipelines like rasterization or ray tracing were designed to process triangles efficiently. Similarly, asset creation workflows generally produce polygonal models that are triangulated during export or, alternatively, when they're processed as triangles when being imported into the engine. This kind of alignment between authoring tools and practical usage is what makes triangle meshes an established, predictable baseline for rendering geometry.

However, there are different approaches that exist alongside this baseline, one of which is signed distance fields (SDFs). An SDF represents a scalar field whose value at any point is the signed distance to the closest surface, meaning it can model a shape implicitly as the zero set of such field. This new form of representation introduces capabilities that are otherwise difficult to implement with conventional triangles. Using SDFs, tasks like constructive geometry, Boolean operations, procedural deformations, and smooth composition boil down to just algebraic calculations, while conventional representations would require complex methods like isosurface extraction, remeshing, etc. Additionally, field representations also have the benefit of allowing such operations to be done non-destructively.

Despite these additional features, SDFs still aren't a viable universal replacement for polygonal geometry in graphical applications. Rendering such implicit surfaces typically requires finding an analytical ray-based intersection, which is more expensive than simple triangles and could grow even costlier in cases like near-surface hits or traversing at grazing angles. Representing arbitrary artist-authored assets as SDFs poses yet another challenge. While it can be achieved by fitting a series of analytical primitives to existing object's shape, the process is far less standardised than conventional modelling and requires a whole new workflow and toolset. An alternative and much more streamline approach is volumetric discretisation – encoding distance field

values into finite texture data points. The simplicity of this method, however, comes with the downside that preservation of higher frequency details and sharp features requires bigger resolutions, incurring significant memory costs. In short, SDFs generally tend to counterbalance their compositional power with heavier runtime and/or memory overhead.

One way of addressing this downside is to utilize sparse data structures like hierarchical grids or sparse sets. They limit data storage to near-surface regions, reducing overall memory footprint without the loss of significant information. Additionally, these structures also act as traversal aids, allowing larger empty regions to be skipped entirely, while keeping relevant calculations local to the surface. These benefits are most prominent when the underlying surface is static or just slightly varying. Otherwise, if the geometry moves, deforms, or fuses frequently and drastically, the regions that have to be stored and refreshed change as well, requiring more data storage and naturally reintroducing some of the original memory consumption problems.
Blending functionality also adds another level of complexity to rendering process. In general, blending multiple objects' signed distance involves simply combining scalar values with smooth operator to get a new continuous surface. As previously stated, similar results with triangles would require complex surface reconstruction algorithms and potentially texture or normal reconciliation. However, even though SDFs offer a much more straightforward approach, there are still some practical challenges that must be addressed carefully. Mainly, smooth composition inevitably makes the implicit surface non-local, meaning the zero set becomes dependent on multiple neighbouring fields. This in turn results in modified surface moving beyond its original tight bounds as the instances in the scene move around and overlap. If implemented naively, added blend regions inflate original surface boundaries and introduce spatial dependency, increasing calculation costs, memory requirements and update logic.

The core motivation of this research is to address this exact challenge – incorporating extended blend regions into rendering pipeline without introducing overwhelming setbacks. The hybrid approach described below proposes to treat SDF as a selective overlay on its triangle baseline. The main idea is not to replace conventional meshes, but rather to accept SDF capabilities only where

they significantly change the surface and keep the polygonal representation where the two don't greatly deviate from one another. In regions where blend results do matter, their inherent smoothing effect allows the field to run at a coarser resolution, cutting both storage and evaluation costs. Essentially, it's a division of labour that enables SDF operations on arbitrary mesh assets, while keeping performance and memory overhead manageable. The rest of this paper examines whether such selective composition delivers the theorized benefits reliably and what potential compromises it may introduce in the process.

1.2 Research Question

Can a hybrid rendering pipeline that accepts blended SDF result only in regions where it significantly changes the surface, while defaulting to triangle geometry elsewhere, preserve original detail, produce smooth and continuous blends and achieve interactive frame rates with reasonable memory use?

1.3 Aims

To design, implement and evaluate a real-time hybrid pipeline that confines implicit surface evaluations to local, relevant regions and composes them with polygonal results in a way that respects original, authored detail. Additionally, to demonstrate that such selective application of blend effects still maintains visual quality while keeping runtime and memory costs typically associated with SDFs manageable.

1.4 Objectives

- Create separate pipelines for rendering polygonal geometry and their associated SDFs.
- Define a hybrid composition rule that specifies when to keep the triangle or SDF results and when to transition between them based on their difference. This policy should remain agnostic to distance data storage or calculation methods.
- Limit blended SDF evaluation to local contributors. Adopt a spatial filtering strategy that accommodates non-local nature of smooth composition without global iteration over the entire scene content.

- Use coarser signed distance grids since blends smooth out high frequency detail and let triangles represent unmodified regions. Validate that this doesn’t drastically diminish visual quality.
- Compare two different SDF rendering approaches - runtime evaluation and precomputed grids – and see how they adapt to dynamic blending requirement.
- Measure outcomes across multiple test scenarios. Examine close up snapshots for visual artefacts and increase scene complexity to probe method’s scalability.

2 Literature Review

2.1 Foundational Concepts

2.1.1 Signed Distance Fields (SDFs)

A signed distance field is a scalar field that encodes the shortest distance at each point in space to the nearest surface. The value is positive on the outside, negative inside, and zero on the surface itself. Such implicit representation allows compact encoding of potentially complex geometry, offering a continuous resolution-independent representation that is particularly well-suited for procedural modelling and constructive solid geometry (CSG).

6.6	5.9	5.3	4.7	4.3	3.9	3.6	3.5	3.5	3.6	3.9	4.3	4.7	5.3	5.9	6.6
5.9	5.2	4.5	3.9	3.4	3.0	2.7	2.5	2.5	2.7	3.0	3.4	3.9	4.5	5.2	5.9
5.3	4.5	3.8	3.1	2.5	2.0	1.7	1.5	1.5	1.7	2.0	2.5	3.1	3.8	4.5	5.3
4.7	3.9	3.1	2.4	1.7	1.1	0.7	0.5	0.5	0.7	1.1	1.7	2.4	3.1	3.9	4.7
4.3	3.4	2.5	1.7	0.9	0.3	-0.2	-0.5	-0.5	-0.2	0.3	0.9	1.7	2.5	3.4	4.3
3.9	3.0	2.0	1.1	0.3	-0.5	-1.1	-1.5	-1.5	-1.1	-0.5	0.3	1.1	2.0	3.0	3.9
3.6	2.7	1.7	0.7	-0.2	-1.1	-1.9	-2.4	-2.4	-1.9	-1.1	-0.2	0.7	1.7	2.7	3.6
3.5	2.5	1.5	0.5	-0.5	-1.5	-2.4	-3.3	-3.3	-2.4	-1.5	-0.5	0.5	1.5	2.5	3.5
3.5	2.5	1.5	0.5	-0.5	-1.5	-2.4	-3.3	-3.3	-2.4	-1.5	-0.5	0.5	1.5	2.5	3.5
3.6	2.7	1.7	0.7	-0.2	-1.1	-1.9	-2.4	-2.4	-1.9	-1.1	-0.2	0.7	1.7	2.7	3.6
3.9	3.0	2.0	1.1	0.3	-0.5	-1.1	-1.5	-1.5	-1.1	-0.5	0.3	1.1	2.0	3.0	3.9
4.3	3.4	2.5	1.7	0.9	0.3	-0.2	-0.5	-0.5	-0.2	0.3	0.9	1.7	2.5	3.4	4.3
4.7	3.9	3.1	2.4	1.7	1.1	0.7	0.5	0.5	0.7	1.1	1.7	2.4	3.1	3.9	4.7
5.3	4.5	3.8	3.1	2.5	2.0	1.7	1.5	1.5	1.7	2.0	2.5	3.1	3.8	4.5	5.3
5.9	5.2	4.5	3.9	3.4	3.0	2.7	2.5	2.5	2.7	3.0	3.4	3.9	4.5	5.2	5.9
6.6	5.9	5.3	4.7	4.3	3.9	3.6	3.5	3.5	3.6	3.9	4.3	4.7	5.3	5.9	6.6

Figure 1 – Signed distance values of the circle

SDFs provide multiple advantages in real-time graphics due to their ability to define shapes analytically, allowing for cheap inside/outside tests (negative scalar values immediately indicate a point inside the surface and vice versa), efficient collision detection, non-destructive surface modelling, and continuous gradient evaluation. They can be represented as mathematical functions (for known primitives like spheres, boxes, etc.), tree structures (CSG tree) that store these primitives along with operations specifying how to combine them, or discretized voxel grids that trade memory efficiency for flexibility to represent complex, scanned, or polygonal geometry.

2.1.2 Ray Marching and Sphere Tracing

Since SDFs provide a completely different representation of surfaces, standard rendering techniques that operate on triangle geometry aren't inherently

equipped to handle this field-based data. Instead, a different set of methods are needed to integrate SDFs into these common pipelines. One of such methods is ray marching – a general approach for finding surface intersections by stepping along the ray in discrete steps. It is a general algorithm that is incorporated into various rendering contexts including volumetric media, global illumination, soft shadow approximation, and the main point of interest for this research: implicit surface evaluation. In its most basic form, ray marching can be described as advancing a ray through space by small, fixed or adaptively chosen distances and sampling the associated field at each step to detect whether a stopping condition was satisfied or not.

Specifically in case of SDFs, a more efficient version of this approach is sphere tracing (Hart, 1995) – a modification that stems directly from the nature of distance fields. To reiterate, basic ray marching traverses along the line in small safe steps, but the process can be greatly accelerated if the step size is chosen adaptively based on some intrinsics. So, the problem essentially can be stated as maximizing step size $\|x - y\|$ while making sure that field values in that segment $|\mathrm{f}(\mathrm{x}), \mathrm{f}(\mathrm{y})|$ don't cross the zero-set $f^{-1}(0)$.

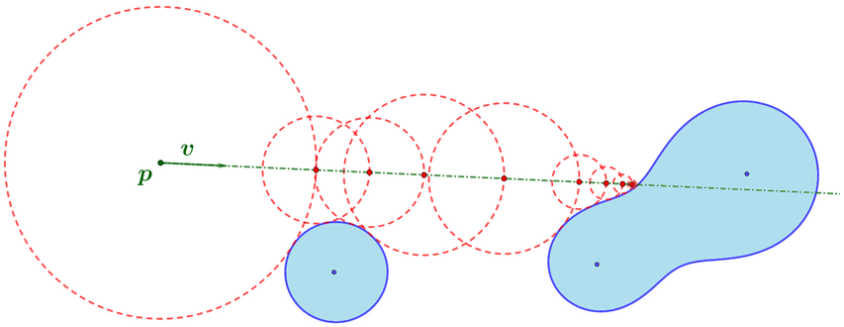

Figure 2 - Sphere tracing illustration (Bálint, Valasek, 2019). Radii of circles denote maximum safe distance, i.e. step size

Before deriving such maximum step size, it's useful to first dive deeper into the nature of signed distance fields. Even though the concept of SDFs was already described earlier, in a more formal definition, for a function to be considered a signed distance function of its implicit surface, it must satisfy the following condition:

$$|f(x)| \leq d\left(x, f^{-1}(0)\right) \tag{1}$$

where x is an arbitrary point and $f^{-1}(0)$ is the zero set of the surface. This expression shows that the values returned by this function are essentially a lower bound/underestimate of the distance to the surface.

As for finding the upper boundary of a safe stepping distance, in his paper about sphere tracing, Hart (1995) references the concept of Lipschitz continuity – a set of conditions that limit how fast a continuous function can change over the interval.

$$|f(x) - f(y)| \leq L \cdot \|x - y\| \tag{2}$$

Equation 2 - Lipschitz-continuity condition for function f with L constant

Hart (1995) showed that since true SDFs are Lipschitz functions (they're 1-Lipschitz continuous, something that can be derived from a basic triangle inequality), they could be turned into a signed distance bound of their respective implicit surface.

for any point $p \in f^{-1}(0)$,
$\|x - p\| \leq \|x - y\| + \|y - p\|$
substituting Equation (1) gives:
$|f(x)| \leq \|x - y\| + |f(y)|$
after rearranging:
$|f(x) - f(y)| \leq \|x - y\|,$
satisfying the original inequality in equation 2 with Lipschitz constant $L = 1$

Listing 1 - 1-Lipschitz continuity of signed distance fields

starting from the inequality in Listing 1, since the point of interest is distance to the closest surface, pick $y \in f^{-1}(0)$,
which makes $f(y) = 0$ and $\|x - y\| = d(x, f^{-1}(0))$
substituting everything yields:
$|f(x)| \leq L \cdot d(x, f^{-1}(0))$, or, after rearranging,
$$\frac{|f(x)|}{L} \leq d(x, f^{-1}(0))$$

Listing 2 - Derivation of signed distance boundary from Lipschitz-continuity

Listing 2 shows that $L^{-1} \cdot |f(x)|$ is a signed distance bound for any Lipschitz function f. Combining this with $L = 1$ from Listing 1's triangle inequality yields the final stepping distance for sphere tracing $stepSize = d(x, f^{-1}(0))$.
In a more intuitive manner, since the SDF function returns the shortest distance to its closest surface, it effectively defines a safe step size: the ray can be moved forward by at least that amount without intersecting any geometry.
Sphere tracing is widely used for its simplicity of implementation and robustness, when paired with appropriately defined constraints. However, its performance quickly degrades near the surface where distance values become

increasingly small, and the algorithm is forced to take many short steps. This problem is especially pronounced in high-frequency geometry or grazing viewing angles. Similarly, the number of steps required also increases with tighter surface hit thresholds, sacrificing performance for accuracy. There are different acceleration strategies that address this issue, a common one being segment tracing (Galin et al, 2020). This method is based on the observation that ray marching only needs the signed distance boundary along a specific ray, instead of global constant of $L = 1$(the nature of a fixed global constant can be observed in Figure 2, where step sizes are illustrated as circles instead of sections along the ray). So, by deriving step sizes from the rate of change of signed distance fields along specific ray direction, the technique can adjust traversal distance more aggressively in smooth regions and still avoid stepping over sharper features.

Even with improved iteration logic derived from all these advanced methods, ray marching is still not universally reliable and remains sensitive to how the underlying field is constructed, stored, and sampled. Most glaringly, it's still a numerical method of finding surface intersections, meaning that it inherently introduces some margin of error by having small thresholds for distance values that are accepted as actual surface hits. However, when paired with safeguards like step count limits, conservative thresholds or fallback strategies, it still remains a widely used technique in rendering.

2.1.3 Compositional Operations and Smooth Blending

A major strength of signed distance fields is their ability to be composed mathematically – basic operations like union, subtraction, intersection, etc., can be applied directly to the functional representation to combine, modify, and construct shapes. These operations can be layered, grouped and sorted into numerous different variations to build complex geometric structures – a technique known as constructive solid geometry (CSG).

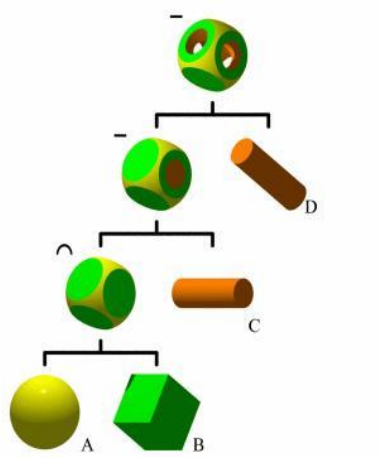

Figure 3 - A diagram of CSG tree. Leaf nodes represent analytical primitives, and the connections - combinatorial operations.

Additionally, beyond these set operations, since SDFs are scalar fields, they can also be manipulated using different mathematical techniques. For example, modifications commonly used in procedural content generation, such as noise, turbulence, displacement, or warping can directly be applied to the distance field, creating great visual diversity without changing the underlying surface representation.

An obvious downside of standard set operations is that they often result in sharp transitions between surfaces, making them appear visually discontinuous. One way to address this is to replace standard min/max functions with their respective smooth variants. These modifications provide a way to interpolate between distance values of two different objects over a specified blend region, producing a smoother, continuous transition.

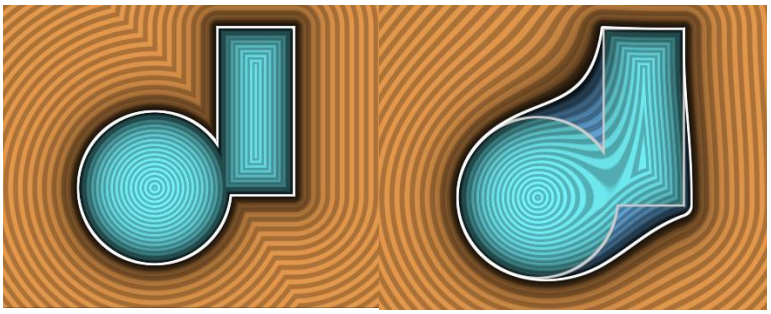

Figure 4-Demonstration of smoothing effect (Quilez, 2013). The image on the left is regular min(a,b), while the one on the right is quadratic smoothmin(a,b).

There are different families of such smoothing functions, one of the most widely used one being quadratic smooth minimum – a polynomial-based approximation that subtracts a quadratic blending term calculated from distance values from standard min() function to soften the transition between two fields.

It’s efficient and straightforward to implement, but due to it being a binary operation, must be applied sequentially when blending more than two objects, which can introduce order-dependent artefacts.

An alternative to general polynomial-based smoothing functions that addresses the previously stated shortcoming is log-sum-exp (LSE) – a method that combines multiple distance values in a more uniform manner. While it is less geometrically intuitive (blending radius is slightly more challenging to configure or visualize), it scales better with many inputs and often yields more predictable, order-independent results in multi-object blend scenarios.

2.2 Spatial Filtering of Blend Contributors

As discussed in previous section, surfaces created by combining multiple objects’ distance field values and smoothing functions can deliver dynamic, emergent, organic results. The process, however, also introduces a unique challenge: each object’s surface becomes no longer strictly defined by its own field but rather influenced by all the surrounding objects within a certain distance radius. A naïve implementation would, for each instance, iterate over all the other blendable objects in the scene, which would naturally become unscalable very quickly.

A common way to approach this is to implement some sort of spatial filtering mechanism – a system that closely resembles broad-band collision detection and neighbourhood search problems. The simplest solution is to partition the space into a uniform grid and use spatial hashing (Teschner et al, 2003) to assign objects to different cells based on their positions. Later at runtime, the queries can be limited to objects in the current cell and its immediate neighbours. This method is simple and parallelizable, but it does assume a relatively uniform distribution of instances and their sizes, meaning in complex scenes with significant variations in geometry, choosing parameters like cell size or maximum cell capacity could become quite challenging.

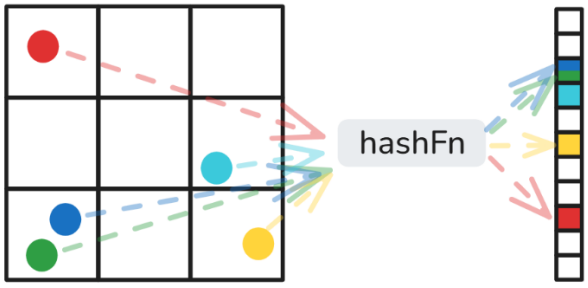

Figure 5 - Illustration of spatial hash grid. The space is divided into uniform grid cells and instance indices are written into a flattened list based on their hashed grid cell coordinates.

Hierarchical spatial hashing (Eitz, Lixu, 2007) builds on this principle but also tries to address the setbacks introduced with non-uniform geometry distributions. Instead of assigning all objects to the same uniform grid, it creates multiple spatial grids at different resolutions (cell sizes). Larger objects are then inserted into coarser grids, while smaller instances are assigned to finer ones. When querying a point, the algorithm searches relevant neighbouring cells across all grid levels and gathers contribution candidates. Such multi-resolution approach is a lot better at handling size variance, but in turn requires higher memory usage and more complex querying logic. Additionally, it still doesn't address the problems basic spatial hash grids have in case of irregular object distribution and spatial clustering.

Moving away from flat grid-based data, tree structures like bounding volume hierarchies (BVH) provide a more general, perhaps better scalable alternative (Karras, 2012). In these kinds of systems, objects are recursively grouped based on spatial proximity, with each group node having its own bounding volume. For runtime queries, the traversal begins at the root and descends only into branches whose bounds intersect the query region, reducing the number of checks (which was the initial general goal) without the constraints of fixed grid cell sizes. This added flexibility means that the hierarchical structure can easily adapt to either clustered or sparse object layouts and still provide the benefits of selective query refinement.

It's worth noting that even though the ray tracing frameworks like DXR internally construct BVHs for hardware-accelerated ray-object intersections, these acceleration structures are not directly accessible for manual traversal in

shader code. This means that in practice, a separate BVH needs to be constructed and traversed manually for just the purpose of proximity filtering.

2.3 Sparse Data Structures for SDF Representation

To reiterate, signed distance fields can represent complex surfaces with minimal compositional operations or simple texture-encoded data, but evaluating and rendering them efficiently, especially in dense and large-scale scenes poses significant challenges. One bit topic warranting closer inspection is how a dense 3D grid containing distance values can quickly become memory-heavy and computationally expensive when, in reality, most of the space is far outside or inside the surface and how various sparse and hierarchical data structures can address these problems.

2.3.1 Hierarchical Tree Structures and OpenVDB

OpenVDB (Museth, 2013) is a hierarchical data structure that organizes space into nested grids to allow efficient storage and traversal of level sets, SDFs, and volumetric data. The model itself closely resembles a B+-tree – a fixed-height, high-fanout structure that has a root, a couple internal levels and leaf brick nodes storing dense texture values. Nodes carry bitmasks to indicate child or value occupancy and at internal levels they can either reference their children or hold a tile (a uniform value representing the whole subregion of the node).

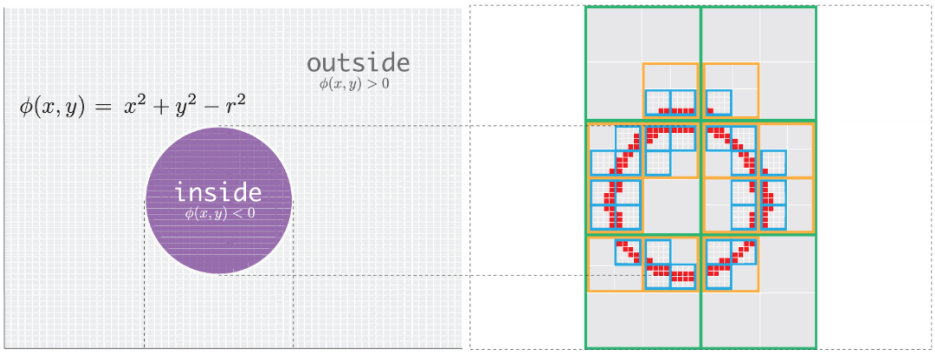

Figure 6 - Example of an OpenVDB structure (Museth, 2013). A An analytical equation of circle is discretized into a fine grid, out of which a hierarchical structure is built: smallest nodes are recursively merged at each upper level.

When using VDB for encoding implicit surfaces, the active nodes only store the narrow band (near-surface) data, populating the rest of the grid with tiles or background values. Not only does this approach greatly reduce the memory footprint (as only relevant surface data is being stored), but with the help of

hierarchical digital differential analyser (HDDA) algorithm, it also allows multi-level stepping that prunes large empty regions and quickly descends to leaf bricks that store relevant surface data.

For real-time rendering use cases, OpenVDB hierarchy is adapted into NanoVDB - a GPU-friendly storage layout designed specifically to accommodate graphical simulation and rendering tasks. This modification still preserves the benefits of the original solution like hierarchical skipping but avoids pointer-management costs on GPU hardware.

Another tree-like structure worth mentioning is a sparse voxel octree (SVO). While other sparse data models will be discussed in following sections, SVOs feature a hierarchical structure not dissimilar to VDB that also allows large empty region skipping. In contrast to OpenVDB's fixed height and number of layers, SVO is an adaptive octree where each node has only up to eight children but the depth in turn varies.

In short, these tree structures generally double as both a memory-saving representation and a spatial acceleration structure for ray marching, making them a powerful tool for optimizing sphere tracing for implicit surface rendering.

2.3.2 Sparse Voxel and Brick Sets

An alternative to constructing a tree hierarchy around the narrow band values is to just store these discrete patches of meaningful data in a flattened list. This is the principle of how sparse data sets operate.

A sparse voxel set (SVS) is the most straightforward implementation of this idea. It is a collection of voxels that each represent a cube formed by adjacent texels whose distance values lie within a specified surface threshold. These voxels are stored in a flat array, accompanied by an indexing structure like an indirection table. Since SVS is mostly used to represent only the zero-crossing region, both storage and evaluation costs scale with surface complexity.

Building off of this concept, a sparse brick set (SBS) stores small slices of local grids (e.g. 8x8x8 blocks), still only keeping distance values close to the surface. Compared to a voxel set, SBS support more coherent memory access, especially when sampling neighbouring values for computing gradients and normals. Because of its fixed-resolution brick structure, a simple offset-based

access pattern (like the indirection table mentioned for SVS) is still sufficient to sample relevant data and still retain GPU-friendly layout.

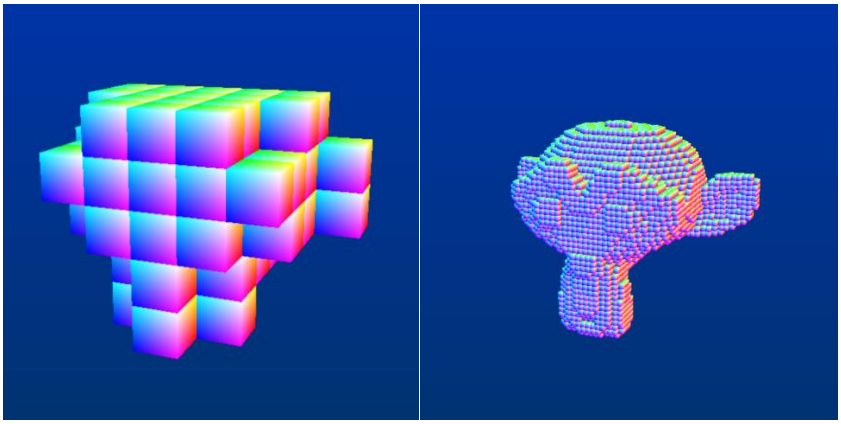

Figure 7 - Side-by-side comparison of SBS (on the left) and SVS (on the right).

Even though both of the discussed sparse sets decrease the memory footprint, due to their non-hierarchical nature, they lack built-in mechanisms for traversal and spatial skipping. Instead, they are typically used in conjunction with other acceleration structures like BVH or spatial hash grid that allow mapping query point's spatial positions to appropriate index and location in memory.

2.3.3 Traversal in Sparse Representations

Sparse representations already offer substantial memory savings by only including a relatively small portion of relevant near-surface data, but their benefits are even further amplified in ray tracing pipelines that already have internal spatial acceleration structures for empty space skipping and culling.

As previously stated, in hierarchical structures like OpenVDB, the tree layout itself is also an acceleration structure: its traversal algorithm operates by evaluating coarser regions and only stepping through finer grid details if the parent nodes contain possible surface data. Within a ray tracing context, the initial intersection is quickly resolved via framework's BVH and can be then followed by VDB's native traversal.

In contrast, flat representations like SVS or SBS lack internal hierarchy, but in turn, they integrate really well with existing spatial acceleration structures. Since each voxel or brick has a known spatial extent, they can be directly represented as axis-aligned bounding boxes (AABBs) and submitted to ray tracing pipeline as procedural geometry. The underlying API leverages hardware acceleration and handles all the details of efficient traversal, leaving a

custom intersection shader with the task of simply mapping a hit index with its associated data. This inherent compatibility between sparse representations and ray tracing acceleration structures enables rendering SVS/SBS of signed distance fields with increased memory and performance efficiency, without sacrificing the simplicity of implementation.

2.4 Rendering SDFs in Real Time

A commonly used approach to rendering signed distance fields is to wrap each representation into a bounding volume and perform sphere tracing within that region. It is a simple, flexible, and straightforward technique. More recently however, Söderlund, Evans, and Akenine-Möller (2022) proposed a new method that leverages the ray tracing pipeline for directly intersecting the voxels inside the SDF grids. This approach replaces sphere tracing with local surface evaluation, as intersection points between a ray and a surface region inside the voxel can be found through solving a polynomial equation, either analytically or numerically. The following sections discuss these approaches and their implications for dynamic, blended SDF content.

2.4.1 Bounding Volumes and Sphere Tracing

One of the most common methods for rendering SDFs is to associate each object with an axis-aligned bounding box (AABB) that completely covers the underlying surface. This bounding volume is used to register potential hits on its associated SDF, and the surface evaluation is then carried out with runtime sphere tracing until the process either converges on the surface or exits the bounding volume.

The underlying distance data, when leveraging this technique, could either be dynamically evaluated per ray step, or precomputed into a dense grid and simply sampled during the traversal process.

This pipeline is lightweight and flexible, but still dependent on how well the AABB fits the underlying surface: looser fit would better accommodate surface-deforming operations, but in turn would introduce substantial amount of potentially empty space, causing unnecessary calculations. Specifically for blending or similar significant surface deformations, a bounding volume would be conservatively padded to ensure all possible extrusions would still fall under

it, leading to even more wasted ray steps in regions farther away from the actual geometry.

2.4.2 Ray Tracing SDF Grids

A recent method proposed by Söderlund, Evans, and Akenine-Möller (2022) replaces traditional sphere tracing with fully ray-traced solution that operates over precomputed SDF grids. Rather than assigning one large bounding box to each object, the distance field is decomposed into a sparse set of near-surface nodes (voxels or bricks), each covering a small portion of the space where the surface intersection might occur. These nodes are then passed to ray tracing pipeline where a bounding volume hierarchy is automatically constructed over the dataset.

During rendering, when a ray intersects one of these smaller bounding volumes, the intersection point is narrowed down to a specific voxel, where distance data is evaluated locally. Instead of iterative stepping sphere tracing relies on, the shader retrieves eight corner values of the surface voxel and performs trilinear interpolation to represent the scalar field at any point inside the voxel. The main idea of this method is the fact that when this interpolated SDF is evaluated against an intersecting ray segment, the equation becomes a cubic polynomial in ray parameter t. The exact surface intersection where the interpolated value equals zero can then be found using a standard root-finding algorithm.

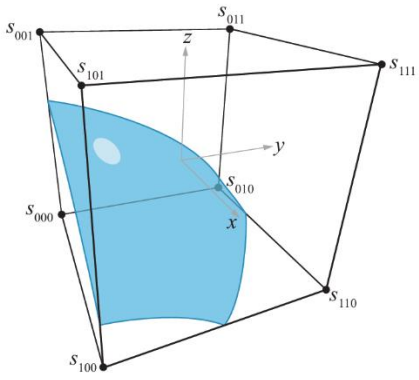

Figure 8 - Voxel with signed distance values s_{ijk} (Söderlund, Evans, and Akenine-Möller, 2022). Possible surface inside the voxel shown in blue.

On top of eliminating the need for iterative stepping, because the bricks are narrow band by design, the ray is routed directly to relevant surface regions and the intersection is resolved inside a small, bounded domain, leading to

substantial performance gains. Moreover, since the method leverages hardware acceleration, BVH traversal is already highly optimized, without the need for manual interference.

It's worth noting that while this method was shown to be highly efficient for static surfaces, it still presents unique challenges when used with dynamic or blended SDFs – a topic that is addressed in later sections.

2.5 Blending Operations in Ray-Based Pipelines

While sparse grids and BVH-accelerated ray tracing provide substantial performance gains for static signed distance fields, these advantages slightly break down when adding blend operations. Smooth composition discussed throughout previous sections introduces non-locality – the surface of a blended SDF is no longer defined strictly by its local values and bounds. Instead, new, modified surfaces emerge from the interaction between multiple overlapping fields. This insight challenges the central assumption that each AABB, brick, or voxel corresponds to a self-contained surface region data.

So, with this additional requirement in mind, per-node ray intersection and numerical solving need to be adjusted accordingly. For starters, multiple SDFs need to be queried, composed, and evaluated per node, which means that even with sparse near-surface representation, the shader must now perform dynamic lookups across multiple nearby instances. The task is further complicated by the fact that such blending is often dynamic, so the resulting field is being continuously modified and updated by objects moving through the scene.

In the basic sphere-tracing-inside-the-bounding-volume approach, the problem is addressed by inflating the AABB to cover the entire possible blend region. The rendering process itself remains the same: a ray enters a padded region and runtime evaluation combines nearby SDFs using the chosen blending function. It’s a natural, straightforward extension that doesn’t introduce significant implementation overhead, but it does increase the computational cost even further, which was already relatively high compared to more modern alternatives.

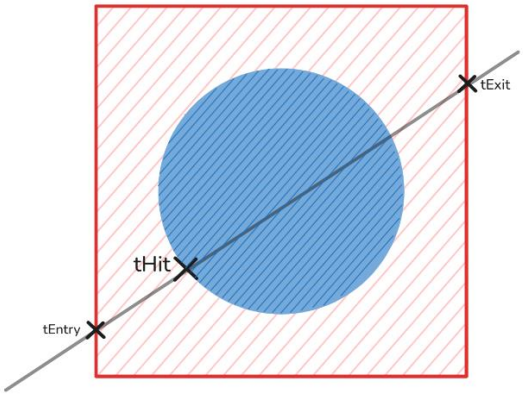

Figure 9 - Illustration of a ray intersecting a padded AABB over the initial surface (in blue).

The situation is more complicated for sparse grid ray tracing. Narrow band voxel or brick sets work well for static surfaces, as they can be extracted at startup and evaluated in small localized regions, but with blending involved, the surface will most likely shift or extend beyond the original narrow band, creating the need to regenerate the narrow band based on this newly blended field and possibly refit the BVH to match the updated layout. This could add a significant cost and partially diminish the original benefits of the entire approach, unless the recalculation and update strategies were considered and planned out carefully.

In general, both sphere tracing and ray-traced grids offer the potential for blending support, but the new ability doesn't come without additional setbacks. Despite that, the challenges discussed mostly applied to broad families of methods, so specific implementations from each category could still be specifically well-equipped to handle this added level of complexity. As such, it would be interesting to evaluate how well each type of strategy handled dynamic composition and whether the benefits of ray tracing acceleration would persist under the condition of blending as well.

2.6 Limitations of Representing Arbitrary Geometry with SDFs

Signed distance fields offer clear advantages when it comes to smooth blending and procedural modelling as their analytical form and composability make them very well-suited for these types of operations. However, their usefulness as a general-purpose geometry representation is certainly less apparent, especially when applied to complex and detailed 3D models.

Analytical SDFs and CSG trees are lightweight and precise, representing objects using simple mathematical primitives and combinatorial operations. That being said, representing high-detail objects with CSG quickly becomes difficult and time-consuming as the complexity of the target model increases. Conventional modelling workflows aren't primarily built around this format, so constructing detailed geometry purely from SDF primitives and Boolean operations is generally quite inefficient compared to triangle alternatives. As a result, such methods and workflows remain niche and are rarely integrated within traditional asset pipelines.

On the other hand, grid-based SDFs tend to offer a more practical alternative. The process of converting triangle meshes into volumetric grids is a lot more straightforward, as there are various tools, solutions, and algorithms for this exact task. But the solution does come with its own trade-offs: representing fine features like sharp edges or thin geometry requires higher resolution grids, which can quickly increase memory usage and sampling cost. More importantly, these grids are ultimately still approximations of the original surface, meaning some level of loss of detail or smoothing is almost inevitable, and especially in lower-resolution fields. This setback could become even more pronounced depending on how the rendering process calculates surface intersections, as techniques like sphere tracing inherently introduce some additional numerical error. So, even though SDF grids are quite straightforward to incorporate into existing asset and render pipelines, discretizing continuous data into finite grids makes the field resolution-dependent, more susceptible to numerical error, potentially memory-intensive, and ultimately still lossy compared to the original surface.

Neural SDFs provide one potential solution to loss of detail associated with SDF grids. Their principle lies in encoding complex shapes into compact neural networks that are trained to approximate signed distance functions. These representations can achieve impressive compression rates, smooth reconstructions, and manageable training times, even with high-resolution original surface data. NVIDIA's Instant-NGP (Müller et al, 2022) is one good example of such solutions – it offers efficient training pipeline and detailed geometry reconstruction with relatively low memory overhead. It's curious to see how this specific implementation also uses multi-resolution hash encoding

(similar to hierarchical hash grids discussed earlier in collision detection and proximity filtering context).

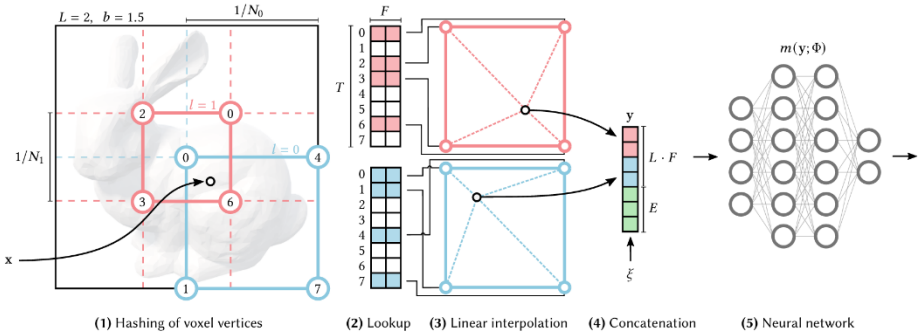

Figure 10 - Illustration of multi-resolution hash encoding of model features (Müller et al, 2022).

The main disadvantage of neural SDFs becomes apparent when trying to use them in real-time applications. At runtime, each distance query now involves evaluating a multi-level perceptron, making the process significantly slower than traditional sampling. Even with optimizations, per-query cost remains high compared to direct texture access or analytical SDF evaluation utilized in previously discussed approaches. As a result, while neural SDFs are promising for offline applications, they are currently still impractical for real-time rendering tasks.

Taking all this into consideration, using SDFs exclusively for all geometry seems inefficient, which is why this work focuses on a hybrid selective approach. Specifically in case of blending, if a given instance is isolated, it's probably preferable to render it as the original triangle mesh as that representation preserves surface detail, renders efficiently and avoids the overhead of distance field evaluation altogether. By contrast, when some regions of the surface deform significantly due to blending caused by other objects' proximity, smoothing operations blur high-frequency detail anyway, so using a lower resolution SDF grid would be both visually acceptable and cost-efficient.

2.7 Summary

This chapter reviewed the core concepts, representations, and rendering techniques associated with signed distance fields and their role in real-time rendering of arbitrary geometry. While analytical SDFs show to offer compact, resolution-independent representations, their applications are mostly limited to

simple primitives and procedural modelling workflows. By contrast, complex imported assets are better handled by grid-based SDFs that, while typically carrying memory and precision trade-offs, still enable simple and fast encoding of initial polygonal data.

In terms of rendering strategies, there are several approaches that make grid-based SDFs feasible in real-time contexts, some of which include hierarchical data structures with optimized traversal logic, or sparse sets that leverage GPU-accelerated ray traversal. These provide improvements in data storage and surface evaluation process but often rely on the assumption that the content is static and defined with localized boundaries – conditions that are no longer given under surface-deforming operations like blending. Supporting these kinds of complex operations where surface definitions are dependent on multiple overlapping fields is no longer just about compact storage and efficient ray traversal, but also about the ability to accommodate introduced cross-dependencies and dynamically changing boundaries. But, at the same time, incorporating all this additional work universally for all dynamic geometry could be inefficient, especially when they wouldn't deliver significant visual differences to justify the added complexity.

The work presented in this research builds on the strengths and limitations discussed above and proposes a hybrid strategy: use SDFs only in regions where blending operations significantly alter the surface. Because the blending operation itself naturally smooths out high-frequency detail, a coarser SDF resolution could be sufficient, reducing memory overhead of having high-resolution grid representations. Conversely, in unmodified regions, the process can fall back to the original triangle mesh, preserving original geometric detail. By enabling blending functionality of arbitrary models without the cost of dense distance field value storage and evaluation, this compositing approach aims to balance the two challenges repeatedly brought up throughout this chapter: visual fidelity and performance.

3 Methodology

Building on the constraints and opportunities identified in previous sections, this chapter describes a proposed hybrid rendering pipeline that composes smoothly blended signed distance fields with conventional triangle meshes in real-time rendering applications. The design aims to retain mesh sharpness where implicit blending doesn't significantly alter the surface, while still enabling continuous, organic deformations where it does.

The system is implemented in C++ with Direct3D 12 and DirectX Raytracing, leveraging hardware acceleration structures for ray traversal. Rendering standard triangle-based meshes remains unmodified from conventional methods, while SDF objects are handled by a mix of triangle and procedural geometry rendering techniques. This compositional nature already creates the possibility of incorporating various different approaches, but to keep the scope manageable, the present work focuses on two concrete pipelines which are:

- Hybrid-A – AABB-bounded runtime sphere tracing.
  Each SDF object is assigned a procedural axis-aligned bounding box within which the surface is evaluated by sphere tracing algorithm. This approach is conceptually simple and flexible, seamlessly accommodating all the established techniques of procedural geometry rendering, while also extending nicely to meet additional blending functionality requirement.
- Hybrid-B – Sparse brick set with analytic per-voxel intersection.
  For each SDF instance, near-surface bricks are extracted and submitted to DXR as multiple small AABBs. When a ray enters a brick, a relevant zero-surface voxel is determined through digital differential analyser (DDA algorithm). The local field inside that voxel is then treated as a trilinearly interpolated scalar and a ray-surface intersection is solved as a cubic polynomial, avoiding iterative ray marching altogether.

Both paths also use the same proximity BVH to limit blending evaluations to nearby contributors, and share the composition policy that decides, per ray, whether to keep the triangle hit, switch to SDF hit, or interpolate between them over a short transitional range.

To accommodate smooth blending, both paths are integrated with conservative padding representing maximum blend radius: in Hybrid-A, per-instance AABBs are inflated so that the intersection shader is invoked across the entire potential blend region; in Hybrid-B the narrow band is “thickened” so that the brick set covers not only zero-crossing, but also an outwards-extended region that could house any surface deformations.

3.1 Scene Representation and Acceleration Structures

3.1.1 Object Classes and Data Management

The system provides two different types of scene objects: triangle objects which are conventional meshes represented by vertex and index buffers, and hybrid objects that carry both triangle and SDF data – they have all the polygonal properties listed above plus and additional procedural AABB (or a list of AABBs) to trigger custom intersection logic for the SDF portion of the rendering pipeline. Hybrid objects are also further divided into sub-types (e.g. sphere, box, model) and carry optional handle to SDF texture data. These primitive types are later used during distance evaluations to choose between different analytical distance functions or texture sampling.

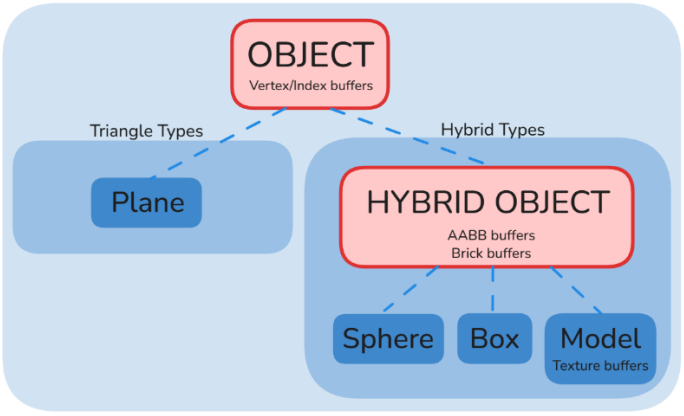

Figure 11-Object class structure. Abstract interfaces are denoted with red colour.

These objects encapsulate shared immutable resources, so the program needs an additional instance class to allow adding distinct items into the scene. Each such instance holds a reference to its associated object, as well as additional per-placement state like transformation matrices, or other SDF-related parameters. Acceleration structures also mirror this standard separation pattern: each object type owns a bottom-level acceleration structure that is built once

and reused, and each instance is associated with a single top-level acceleration structure rebuilt per-frame. Instance-specific transformation matrices are used not only during TLAS creation, but also for transforming sampling points back and forth between world- and local-space coordinates, as the distance functions are written in object's local space, but a common coordinate system is required to correctly compare and blend multiple instances' values. As a sidenote, to prevent potential distortion caused by these transformations, the system only allows uniform scaling for hybrid object instances.

3.1.2 Object Masks and Ray Routing

To easily rout rays through different content types, the program utilizes DXR's instance masks. A default triangle-only mask identifies conventional geometry, and hybrid instances contribute two TLAS entries: one with a (different) triangle mask (for their underlying polygonal mesh), and another with a procedural mask (for their AABB). For each frame of rendering, ray generation shader launches primary rays that test only against conventional triangle and hybrid procedural entries. When a ray intersects a hybrid procedural entry, a custom intersection shader is invoked, where an inline ray query is issued with a mask that restricts traversal to triangle geometry strictly associated with hybrid instances. This entire process manages to cleanly separate SDF intersection path from the polygonal baseline, yielding two candidate hits for hybrid instances' surface composition, all without interfering with the standard triangle rendering pipeline at all.

3.1.3 Spatial Filtering for Blending

The addition of blending features implies that each instance's surface is now influenced by other nearby objects. To avoid sampling every other object in the scene, a separate proximity hierarchy is built over padded instance bounds. This tree structure is then used during newly modified blended surface calculations. The mentioned padding matches the maximum blend influence, meaning that if current instance's AABB overlaps with any of proximity hierarchy entries, it implies a possible contribution to the blend calculations. This custom BVH is independent of the DXR accelerations structures used for visibility and is more inspired from broad-phase collision detection problems.

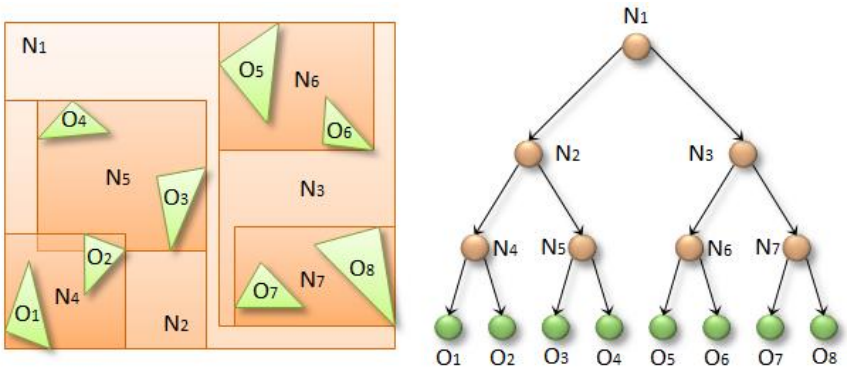

Figure 12-Illustration of BVH over a set of scene objects (Karras, 2012).

For the same task, uniform grids and spatial hashing were considered only in passing: with varied instance sizes and non-uniform layouts they would either admit multiple false positives or require very fine cell sizes. Conversely, the BVH adapts more cleanly to such variations and still manages to keep the construction and query logic mostly streamline, deterministic and reusable by both hybrid rendering paths.

The construction of the BVH happens on the CPU and iterates over the entire lies of scene objects. Firstly, for each instance, a world-space AABB is computed based on its uniform scale and placement. The builder then partitions this set of AABBs in top-down order: at each step it computes the bounding box of the current subset and if there's only one entry left, emits a leaf node. Otherwise, the subset is split at the median along the node's longest axis, child indices are sorted by the centroid on that axis and divided into left and right halves, before recursing again. This longest-axis median split yields a balanced tree with $O(n \log n)$ build time and good spatial locality. In terms of storage, the nodes are written to a contiguous vector and each reference a range from second index buffer (essentially an indirection table), making the whole structure GPU-friendly.

Regarding updates, just like top level acceleration structures, the proximity tree is also rebuilt on every frame to account for instances moving around in the scene at runtime. Generally, with large input data, the task would be delegated to the GPU, with Morton codes and parallel sorting, but since the test scenes in this project were kept relatively simple, a CPU-side rebuild for even hundreds of instances still proved to be inexpensive and allowed to keep the code path much simpler.

In terms of using this information during rendering, this same proximity hierarchy serves both hybrid rendering paths, with identical sampling logic: when a neighbourhood data is needed, a single query based on current instance's AABB is issued, which triggers a small, iterative depth-first walk with a fixed-size traversal stack allocated by the shader. During this process, each node is tested with a simple AABB-to-AABB overlap: non-overlapping subtrees are quickly pruned, and overlapping ones descended until a leaf is reached. Leaves then append their instance IDs to a small local array whose size is capped with maximum amount of blend contributors, and the resulting list is reused throughout either compute or intersection shaders for blended distance calculations.

Overall, this proximity BVH reduces blend evaluation from scene-wide traversal to a short list of overlapping instances, ensuring that distance sampling and composition are restricted to contributors that can actually affect the local surface field.

3.2 Blending and Composition

3.2.1 Smoothed Distance Calculations

Querying SDF value at a point p proceeds as follows: first, the proximity BVH is asked for the small set of instances close enough to have significant contribution. The world-space query point is then transformed into each contributor's local space: primitives are evaluated with their closed-form distance functions, while models sample their associated textures. Each local result is scaled back to world units using instance's (uniform) scale, so the values are comparable, and finally the per-contributor distances are combined with a smooth minimum function.

Probably the most common variant of such smoothing functions is the quadratic $smoothMin$ with a configurable radius r. It behaves like a regular $\min(a, b)$ everywhere except from where $|a - b| < r$, where it subtracts a small quadratic term to soften the transition between two fields.

```
float smoothMin(float a, float b, float r) {
    float h = max(r-abs(a-b),0.0);
    return min(a, b) - h*h/r;
}
```

Listing 3 - Smooth minimum function.

This characteristic is very useful, as surfaces far from an interaction region are not being perturbed, but the trade-off is that the operator is binary, meaning it needs to be applied sequentially in case of more than two inputs. This can introduce order-dependent results, which can be especially visible with contributors with varying geometry and dimensions. To address this shortcoming, a second variation of smooth minimum function was also implemented – the logarithmic sum of exponential distances (log-sum-exp, LSE).

```
expSum = 0
for each instance : filteredInstances
    distance = mapDistance(p)
    expSum += exp(-blendFactor * distance)

blendedDistance = -log(expSum) / blendFactor
```

Listing 4 - Pseudo-code of LSE. While iterating over all relevant instances, it accumulates the sum of individual distances and returns back the logarithm.

This version combines any number of inputs in one pass, removing order sensitivity present in polynomial variant, but due to its exponential nature, it also does slightly distort the field everywhere. The influence, however, decays really quickly with distance, so its effect outside the blend zone is negligible in practice.

Normals of newly blended surfaces are also computed alongside distance values. When collecting data from all nearby instances, analytic primitives provide exact local-space normal and texture-based contributors use central difference method with small step size. All normal are transformed back to world-space and blended with weights consistent with the chosen distance operator (same interpolation factor used by the quadratic form, or softmax-style weights for LSE) and finally normalised. This keeps shading coherent with dynamic distance composition.

3.2.2 Triangle-SDF Composition Policy

During ray tracing, each hybrid object ends up producing two candidate intersections per ray: a polygonal hit (from the inline triangle query) and an SDF hit (from either sphere tracing or brick-based polynomial solver). The two are compared along the ray parameter t: if only one of them is present, that his is automatically accepted. Otherwise, if both have returned valid results, a policy based on their separation Δt determines the output. When Δt is below a lower threshold, the triangle result is preferred to preserve mesh sharpness. Likewise, when Δt exceeds an upper threshold, the SDF result is selected, as the blending process has significantly deformed the surface. Between these thresholds a smooth interpolation produces an “in-between” transition.

3.3 Hybrid-A Pipeline: AABB-Bounded Runtime Sphere Tracing

The first hybrid rendering path uses a single AABB per hybrid instance to bound all potential SDF influence, including a conservative padded region that covers the maximum blend radius. For convenience, imported meshes are normalised to a unit cube size (approximately $[-0.4, 0.4]^3$ with ~ 0.1 padding in each direction) and transformation matrices are later used to scale and place them in world space.

When a ray registers a hit with the bounding AABB, the entry-exit interval is computed with the standard slab method, yielding a local-space t_{enter} and t_{exit} parameters. All subsequent work is bounded by this ray segment, ensuring the tracing is limited to its intended region.

Before dealing with SDF-related logic, the program issues an inline ray query restricted to the triangle content of hybrid instances and stores mesh hit point data, if the intersection falls within the AABB bounds. This acts as a baseline for later SDF calculations.

Contributors for SDF evaluation are gathered once per AABB hit as well – a proximity query is issued against the custom BVH whose leaves are the padded instance AABBs. The resulting set is capped to a reasonable upper limit and reused for the remainder of the intersection program.

Next, the execution moves to the main logic of this approach: sphere tracing proceeds over the aforementioned clamped ray segment transformed in

world space. At any parameter t, the world-space sample point $p(t)$ is evaluated against the filtered contributors: $p(t)$ is transformed into each instance's local space, the distance to the surface is obtained either analytically or by texture lookup, and the value is rescaled back to world units via instance's uniform scaling parameter. Individual distances are then combined by the chosen smooth operator outlined in earlier sections, and the normal is calculated in similar fashion. The resulting field value denotes the maximum safe distance for traversal and is therefore used as the next step size in ray marching loop.

```
for each instance : filteredInstances
    instanceData = getInstanceData(instanceIdx)
    objectData = getObjectData(instanceData.objectIdx)
    pLocal = transform(pWorld, instanceData.worldInv)

    switch(objectData.primitiveType)
            case box: distance = sdBox(pLocal)
            case sphere: distance = sdSphere(pLocal)
            case model: distance = sdTexture(pLocal)

    distance *= instanceData.scale
    blendedDistance = smoothMin(distance)
```

Listing 5 - Pseudo-code of distance mapping function.

Sphere tracing has several exit conditions and safeguards in place. First, if the ray parameter t advances beyond t_{exit}, the loop terminates, and the segment is reported as a miss. Second, if the composed distance falls below a small tolerance ε, the current point is accepted as a surface hit, avoiding further iterations with near-zero contributions. Finally, a fixed iteration cap bounds the number of iterations per ray, so that the loop doesn't run for an unreasonable number of steps. If any of these conditions are met, sphere tracing logic is terminated, and the entire SDF portion of the surface data is reported as a miss. After all these steps, the program has two hit candidates for the same ray segment: a triangle hit (if any), and an SDF hit (if any). Their combination follows the composing policy defined in 3.2.2 (thresholded selection with an optional interpolation in the transitional region) and is applied here without further modification.

3.4 Hybrid-B Pipeline: Sparse Brick Set with Per-Voxel Analytical Solver

The second hybrid rendering path directly relies on recent ray-traced SDF grid techniques introduced by Söderlund, Evans, and Akenine-Möller (2022). Rather than marching within a single loose bounding volume, a narrow band of candidate bricks is extracted: both those that contain the zero-crossing and those that cover the padded blend region around the surface. These bricks are then submitted as a set of AABBs to DXR and hardware-accelerated BVH traversal routes rays straight to these smaller relevant volumes. Within each intersected brick the surface is resolved without sphere tracing, and rather by solving the cubic polynomial created by trilinear interpolation of the eight corner samples of the relevant voxel.

3.4.1 Candidate Brick Layout and Extraction

Each hybrid instance starts out from a dense SDF data in object's local space. The spatial volume represented by this texture is partitioned into bricks that cover 9x9x9 voxels. Because a voxel is defined by 2x2x2 values, each brick ends up storing 10x10x10 texels so that all corners of child voxels lie within a single brick. Conceptually, the outermost texel layer in a brick overlaps with similar layer of the neighbouring brick, which in practice is addressed by duplicating border values, so that each brick can be evaluated in isolation. Several variations of sparse data sets were prototyped before arriving at the final layout. The first implementation didn't even use SBS and instead utilized a sparse voxel set scheme with one AABB per voxel. This structure was the easiest to implement and based on prior reports [reference here], should have yielded fastest render times in most cases. In static, non-blended scenes, the results did match the reports shown in the paper, but adding dynamic composition drastically changed the image. Enabling blending operations required submitting all voxels in the padded area to DXR, most of which did not actually contain the surface at a given frame. That meant DXR invoked a very high number of intersections only to reject them, causing high overdraw rates and performance regression. One attempt to remedy the overdraw problem was to assign individual bottom-level acceleration structure to each unique instance and rebuild them every frame with surface-containing voxels filtered out during

the compute pre-pass. The cost of frequent and per-instance BLAS rebuilds (since lending causes the surface to dynamically expand or shrink, the number of surface voxels is also changing, so just refitting the BLAS would not suffice), however, was too high to outweigh the benefits of having only surface voxels to traverse in ray tracing pass.

The next step was to switch to sparse brick sets – a solution that is present in the final version of the project. Bricks are obviously larger than single voxels, so the newly extracted padded band contains far fewer AABBs and the overdraw from non-contribution regions is correspondingly smaller. A per-frame compute pass populates a compact brick-mask buffer that marks which bricks actually do contain the zero-crossing. Crucially, bottom-level acceleration structures are not rebuilt per-frame and per-instance. Instead, the system reverts back to the common practice of having one shared BLAS per object type. During ray tracing, every candidate brick may still trigger the intersection program, but the shader immediately consults the mask buffer and exits early for inactive bricks. In practice, this leaves a much smaller amount of overdraw with lower computational cost compared to previous approach of per-frame BLAS rebuilds, and rays are still routed to relevant bricks efficiently through DXR.

3.4.2 Per-Brick Update Pass

To keep the narrow band consistent with dynamic blends, surface values are refreshed each frame by a compute pass that operates brick-wise. All bricks from all hybrid instances are first enumerated into a single flat buffer, each record storing brick's local data as well as any auxiliary indices to map it back to its instance or associated texture slice. Additionally, each hybrid instance also owns its own Texture2DArray where one 10x10x10 slice encodes a single brick's distance data.

The compute dispatch launches one thread group per brick, with a 10x10x10 thread layout, meaning each thread updates one texel in the corresponding brick slice. The distance at that point is evaluated using the same mapping function as with Hybrid-A: nearby contributors are gathered via a proximity query, each contributor is sampled, and the set of distances are

composed by the smooth-minimum operator. The resulting value, along with the calculated normal, is written back to the brick texture slice.

Because one thread group operates over the entire brick, group-shared state can be used to detect whether both positive and negative distance values occur within that brick – indicating surface crossing. After a group barrier, a single thread commits brick's active status (whether it contains zero-crossing or not) to a brick-mask buffer, which is later sampled by the intersection shader, enabling an immediate traversal termination for bricks not containing the surface. This specific pattern took inspiration from earlier discussed OpenVDB solution, where irrelevant nodes are marked as empty tiles to quickly skip over them during ray marching.

In short, this compute pre-pass aims to reduce the cost of distance mapping function: each unique texel containing a distance value is evaluated once per frame instead of many times per ray, and inactive bricks are filtered with a single mask buffer during rendering to still allow fast empty space skipping.

3.4.3 Per-Brick Intersection: DDA Traversal and Cubic Solve

As already discussed, when DXR reports an intersection with brick's AABB, the shader first checks the per-instance brick mask. If the brick is marked inactive, the intersection is rejected immediately. Otherwise, the ray segment clipped to the brick's $[t_{entry}, t_{exit}]$ interval is traversed with a 3D DDA over brick's 9x9x9 voxels. At each DDA step, the shader fetches 8 corner distance values of the current voxel from this brick's Texture2DArray using just two $gather()$ operations. A candidate cell is identified when the minimum and maximum of these samples have opposite signs, indicating a surface crossing within the voxel.

Inside a candidate voxel, the local scalar field $S(x, y, z)$ is the trilinear interpolation of the 8 corner samples, with x, y, z in $[0, 1]$ range. Additionally, along the clipped ray segment, $x(t)$, $y(t)$, $z(t)$ are functions of parameter t. Substituting them into $S(x, y, z)$ and equating it to 0, since the aim is to find an intersection point where distance to the surface is zero, yields a cubic equation of t over the $[t_{voxelEntry}, t_{voxelExit}]$ interval. The cubic coefficients are formed

directly from the 8 corner values and the ray equation, as described in the paper about ray tracing SDF grids by Söderlund, Evans, and Akenine-Möller (2022).

$$f(x,y,z) =$$
$$(1-z)\Big((1-y)\big((1-x)s_{000} + xs_{100}\big) + y\big((1-x)s_{010} + xs_{110}\big)\Big)$$
$$+ z\Big((1-y)\big((1-x)s_{001} + xs_{101}\big) + y\big((1-x)s_{011} + xs_{111}\big)\Big),$$

$$(o_z + td_z)\,(k_4 + k_5(o_x + td_x) + k_6(o_y + td_y) + k_7(o_x + td_x)(o_y + td_y))$$
$$- \Big(k_0 + k_1(o_x + td_x) + k_2\big(o_y + td_y\big) + k_3(o_x + td_x)\big(o_y + td_y\big)\Big) = 0,$$

$$c_3t^3 + c_2t^2 + c_1t + c_0 = 0$$

Listing 6 - Derivation of the cubic polynomial describing ray-surface intersection inside the voxel (Söderlund, Evans, and Akenine-Möller (2022).

The resulting cubic polynomial is then solved by a numerical approach: its derivative (a quadratic function) is computed and any real roots inside the interval are found, in order to partition the $[t_{voxelEntry}, t_{voxelExit}]$ into up to 3 sub-intervals. These sub-intervals are examined in sorted order and the first one with opposite signs at its endpoints is taken as the bracket. After picking the sub-interval, bisection is executed to refine the root until either the interval length falls below a small tolerance value, or a fixed iteration limit is reached. In both cases the midpoint of bisected interval edge values is accepted as the solution.

Finally, once the cubic equation for parameter t is solved, the result is used to reconstruct an actual hit point position and voxel uvw's to trilinearly interpolate per-texel normals stored at voxel corners.
If no candidate voxel yields a valid bracket before the DDA exits the brick, the entire brick intersection is ignored/rejected.

3.5 Testing and Evaluation

The evaluation is designed to test the proposed hybrid composition method in general and to compare two concrete implementations of it against each other and the baseline SDF-only approach. In both hybrid paths, a coarse signed distance field is composited with object's triangle mesh, the difference lying in how the SDF hit is obtained. Hybrid-A uses a single padded AABB per instance with runtime sphere tracing inside that boundary, while Hybrid-B follows the ray

traced SDF grids approach: submitting near-surface bricks to DXR and resolving the intersection analytically per voxel. The questions are whether the hybrid image preserves mesh sharpness where SDF influence is negligible, produces smooth blends where the surface is significantly altered, and maintains interactive frame times.

Each test run of hybrid pipelines produces multiple intermediary images: triangles-only (polygonal baseline), SDF-only (coarse distance field grid), and the final composited version of the two. This breakdown allows for closer inspection of the main premise of this research – the composition process. The images are visually evaluated side-by-side, expecting to see the hybrid result match the polygonal mesh where the smooth blend operation makes no material difference, and follow the SDF where blending meaningfully changes the surface. The images of final outcomes are also compared against the high-resolution SDF-only render – something that would be a direct straightforward alternative to the general premise of hybrid composition – to see what different visual features they might have.

In terms of scene configurations, instance numbers are gradually increased to gauge scalability of each approach, while the camera remains positioned in a way that captures the entirety of instance grid, ensuring consistent ray workload.

After visual analysis, performance is also reported by examining frame times and memory usage. The parameters are captured with Nsight Systems and Nsight Graphics running on NVIDIA GeForce RTX 5080 Laptop GPU with vertical sync disabled and timings averaged over 500 frame period. For Hybrid-A pipeline, the computational cost would mostly correspond to time spent in the AABB intersection program that performs sphere tracing. For Hybrid-B, SDF evaluation cost would include the per-frame compute pass that updates distance values, and the brick intersection shader that performs DDA and cubic polynomial solve.

Alongside frame times, memory usage is also tracked. For both paths the report includes the size of dense 64^3 SDF textures and proximity BVH. For Hybrid-B it additionally looks at the combined size of the Texture2DArrays that store precomputed brick values and the brick-data related buffers.

4 Results

This chapter reports the outcomes of the evaluation described at the end of previous section. There are two main goals: first, to gauge the usefulness of the hybrid approach by comparing it against the SDF-only pipeline with high-resolution textures and common rendering techniques; and second, to assess the proposed hybrid composition and compare two distinct implementations of that general approach. All captures are produced under identical conditions for either close-up or large-layout scenes. Unless stated otherwise, SDF grids are 64^3 resolution dense textures and Hybrid-B pipeline partitions them into 10^3 texel slices.

4.1 Baseline Fidelity: SDF-only Approach Across Resolutions

Before introducing the hybrid compositor, it's useful to observe the behaviour of SDF-only rendering at several grid resolutions and compare them visually with the source triangle mesh. Rendering methods chosen for this test follow conventional approaches: polygonal geometry is ray traced and SDF textures are sphere traced at multiple resolutions.

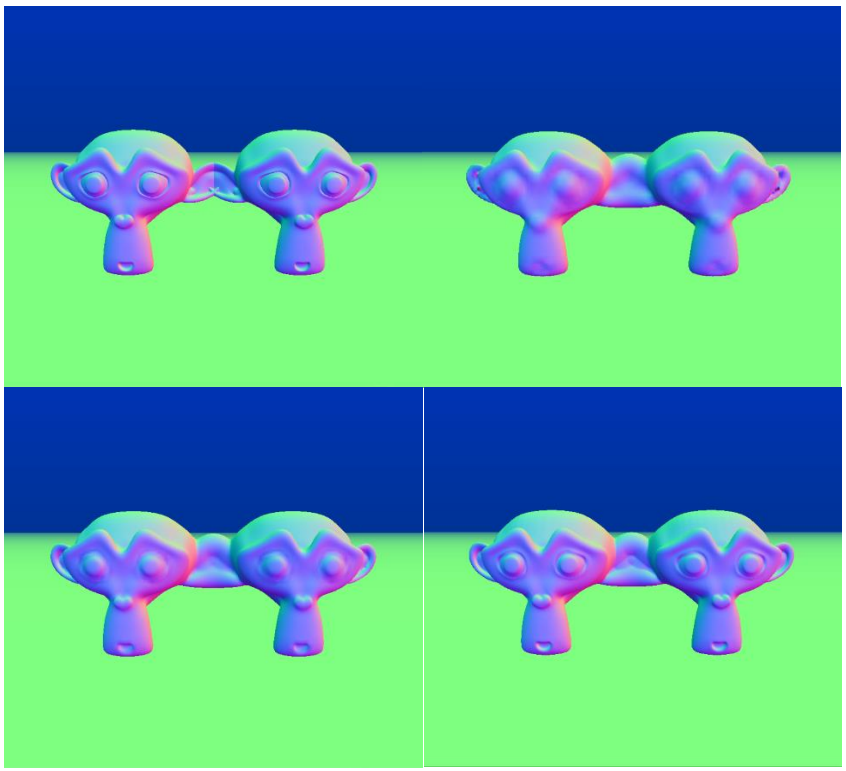

Figure 13 - SDF resolution comparison with triangle baseline (top-left corner).

4.2 Decomposing the Hybrid Pipeline

To make the contribution of each component explicit, the hybrid renderer presents multiple intermediary images in separate passes. For each scenario the system outputs: (i) triangle-only baseline, (ii) coarse SDF rendered with sphere tracing, (iii) coarse SDF computed by DDA and cubic polynomial, and (iv) hybrid compositions produced with Hybrid-A (v) and Hybrid-B (vi) pipelines. Side-by-side frames allow direct inspection of how the compositor behaves in regions where the SDF is close to the mesh and where blending genuinely alters and extends the original shape.

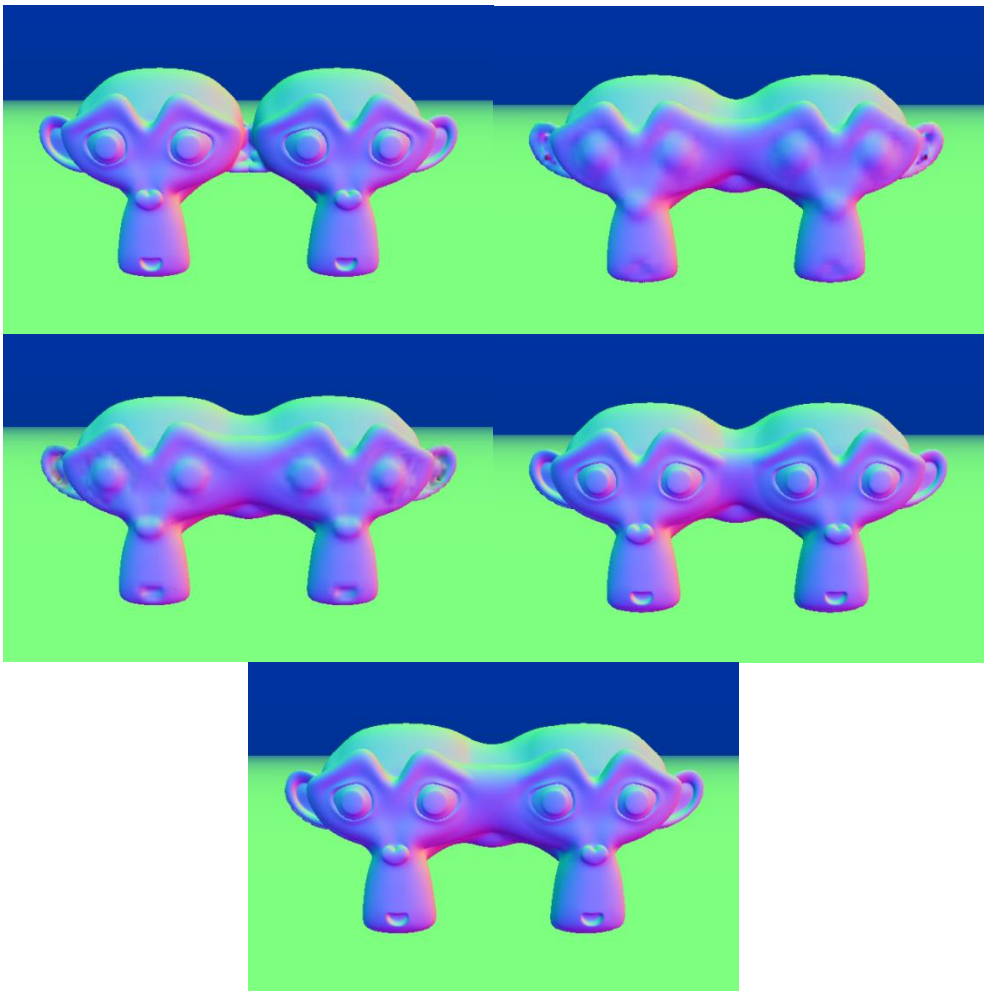

Figure 14 - Hybrid process breakdown. Images from top to bottom, left to right are triangle-only, Hybrid-A SDF-only, Hybrid-B, SDF-only, Hybrid-A final composite, Hybrid-B final composite.

4.3 Visual Comparison Against a High-Resolution SDF

The proposed hybrid pipeline's main point is to use coarse SDFs where blending is active and fall back to triangle geometry to preserve mesh detail elsewhere. To gauge how close this comes to an SDF-only baseline, each

scene is also rendered with a high-resolution SDF-only pass. Figure 15 demonstrates the difference between this high-resolution SDF and two hybrid results.

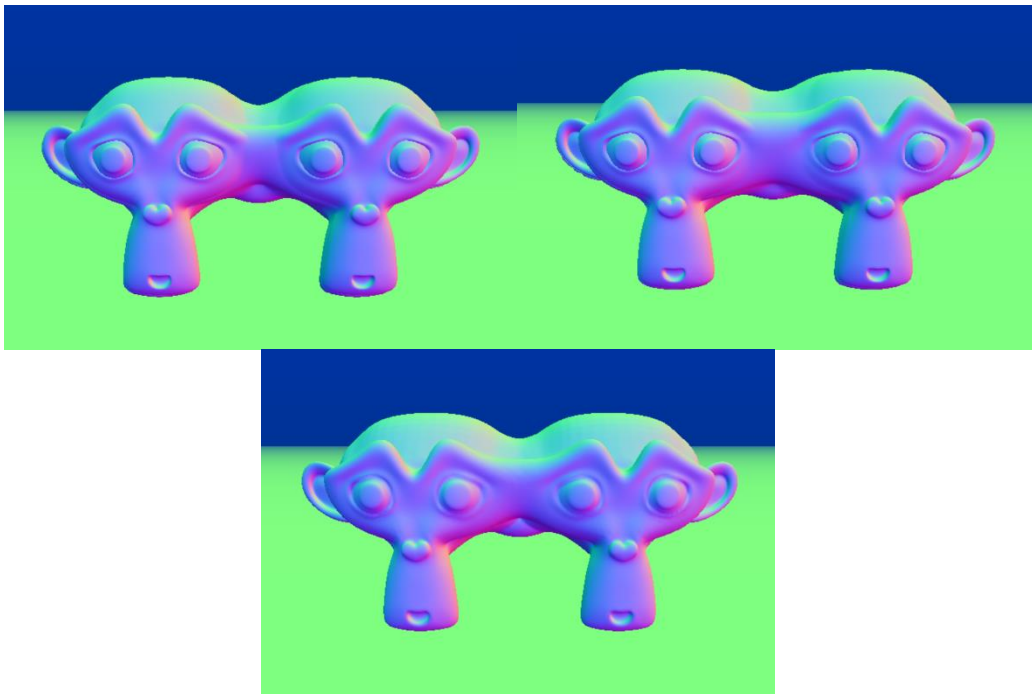

Figure 15 - Direct comparison of hybrid pipeline results (first row, left to right: Hybrid-A, Hybrid-B) with SDF-only baseline (bottom row).

4.4 Performance

Frame time is the primary performance measure for the test runs. For each scenario, a close-up view of the scene is framed so that the current grid of instances fills the image, while the camera remains fixed. Using Nsight Systems, each run starts with a 10-second warm-up and proceeds to capture 500 consecutive frames, reporting average frame time in milliseconds and the corresponding FPS. Two separate variants (plus the baseline) are recorded under identical conditions: Hybrid-A (loose AABB with runtime sphere tracing) and Hybrid-B (sparse brick set with DDA and per-voxel cubic solve). All method's timings cover the ray tracing and intersection program cost, and Hybrid-B's also include the per-frame brick update compute pass.

To observe scalability, the number of instances is increased across runs and the camera is adjusted to keep the grid filling the frame at a similar close-up, so that the ray workload remains comparable.

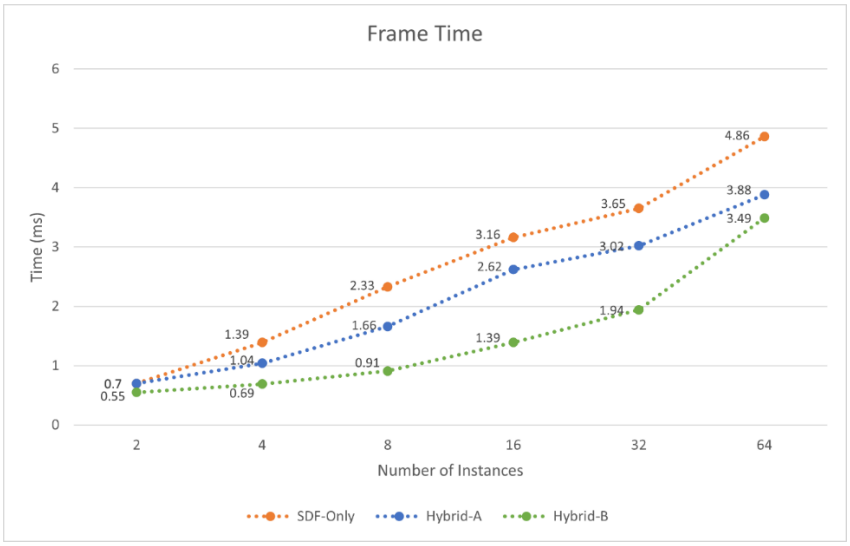

Figure 16 - Reported frame times of baseline and 2 hybrid approaches.

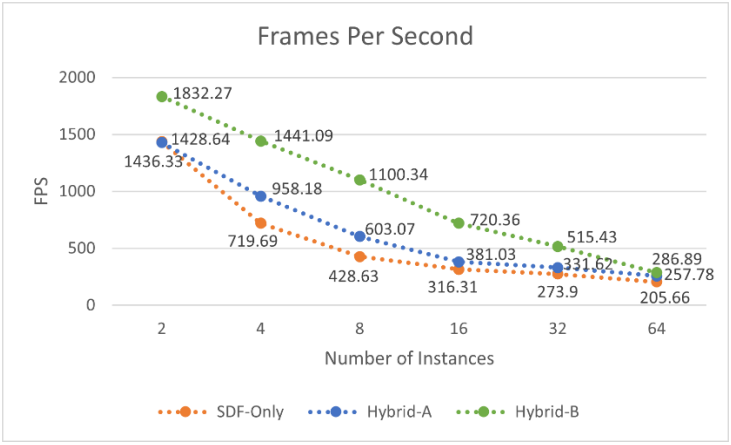

Figure 17 - Reported frames per second of baseline and 2 hybrid approaches.

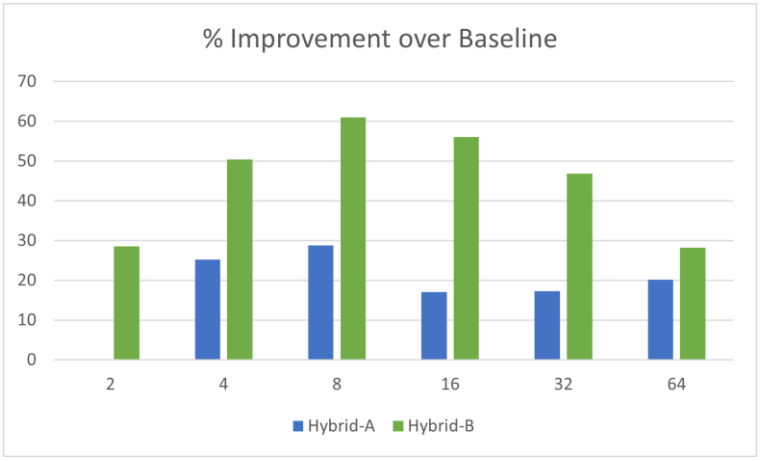

Figure 18 - Calculated percentage improvements of hybrid pipelines over the baseline approach.

Figures 16 and 17 show how reported frame times and corresponding FPS over increasing number of instances and figure 18 additionally displays relative frame time improvements of hybrid pipelines over the baseline SDF-only approach.

4.5 Memory Footprint

Memory is reported in a similar manner as frame times covered in previous section. Baseline SDF-only and Hybrid-A methods both have a fixed cost of allocated dense textures, while Hybrid-B additionally requires new Texture2DArray per instance, plus brick metadata and mask buffers.

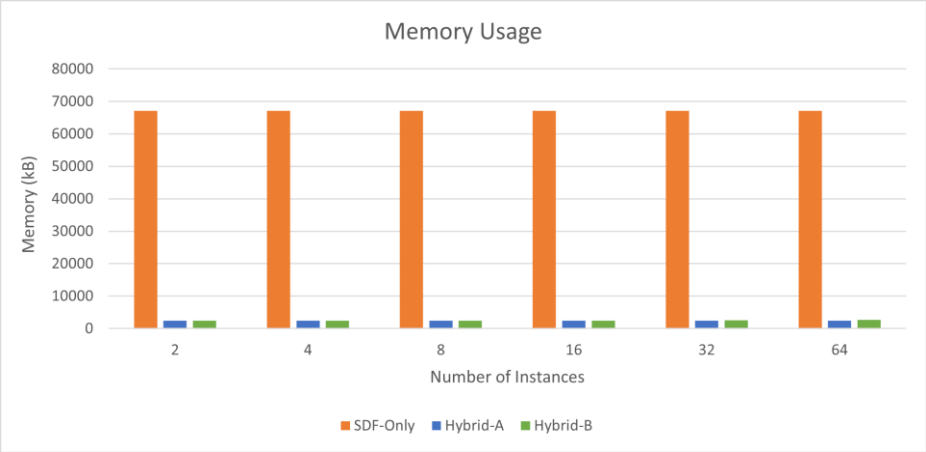

Figure 19 - Total memory consumption of each pipeline.

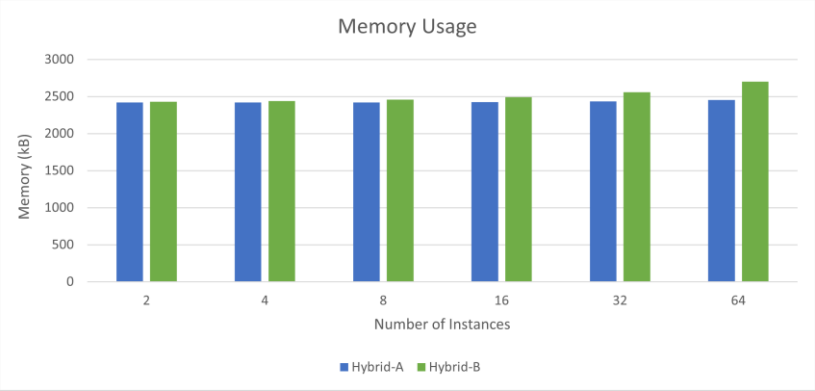

Figure 20 - Total memory consumption of hybrid pipelines.

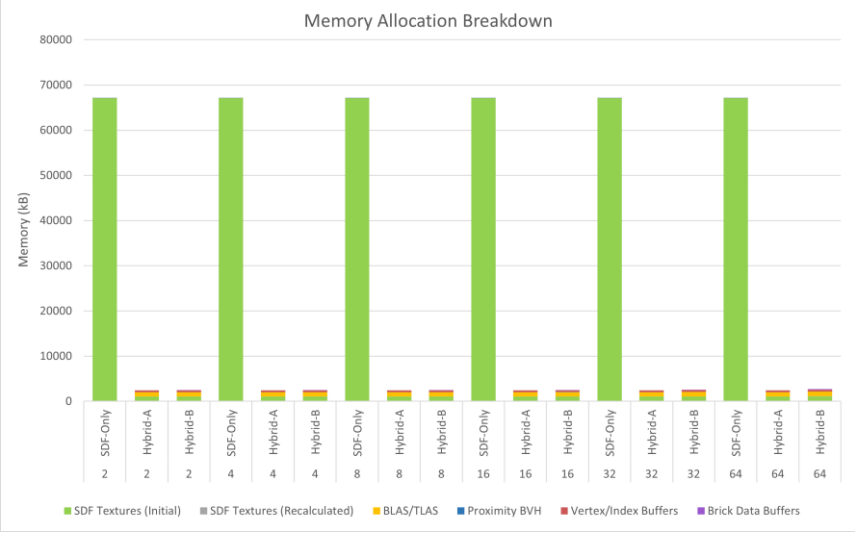

Figure 21 - Memory allocation breakdown of each separate approach.

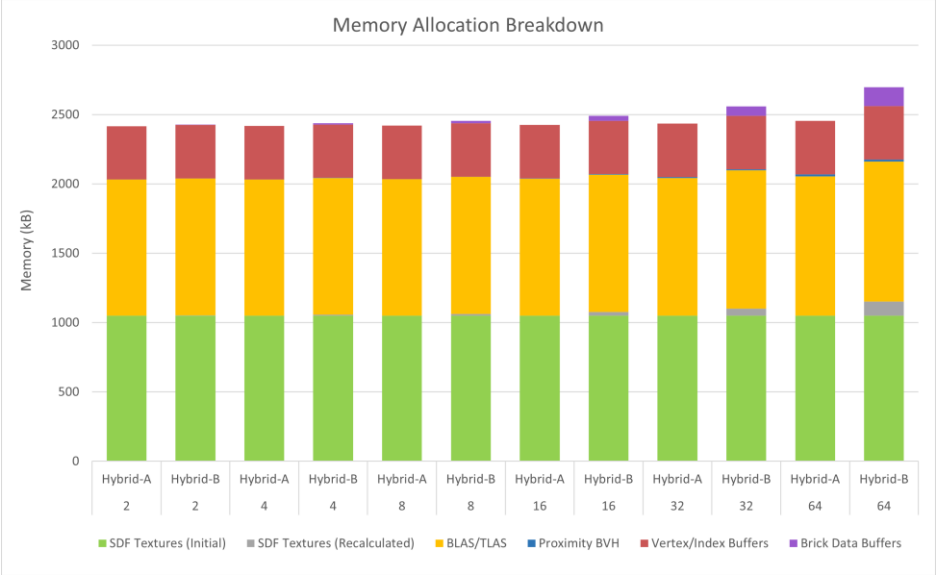

Figure 22 - Memory allocation breakdown of hybrid pipelines.

Since the gap between the SDF-only and hybrid pipelines was so significant, alongside figures 19 and 21 describing total memory allocations and breakdowns of all 3 approaches, hybrid pipeline data was plotted separately in figures 20 and 22 as well.

4.6 Summary

Taken together, these figures serve three purposes. First, they evaluate hybrid pipeline's contribution by comparing it to SDF-only baseline. Second, they make the composition itself explicit by presenting the decomposed passes (triangle-only and coarse SDF-only) alongside the hybrid result, showing where and how two distinct sources of geometry are combined. Third, they evaluate two implementations of SDF evaluation stage (pipelines Hybrid-A and Hybrid-B) against each other in terms of frame time, memory overhead, and scalability. Next chapter will discuss the trends visible in these materials in greater detail.

5 Discussion

The general aim of this project was to propose a new hybrid rendering pipeline for dynamically blended signed distance fields that addresses limitations of standard SDF-only approach (visual fidelity at sharp features, memory overhead, frame times). Accordingly, the evaluation of proposed approaches can be broken down into two main goals: observe visual difference between conventional SDF-only pipeline and the proposed hybrid approach that composites SDFs with triangle meshes, and to examine specific implementations of this general compositional method themselves – what techniques can they utilize and how they perform against each other, as well as compared to SDF-only baseline.

5.1 Grid Resolution Study: SDF-Only

Before moving onto examining hybrid pipelines in detail, it's useful to first look at SDF-only representations of the same model under different resolutions. Looking at figure 13, it's clear how lower resolutions struggle to represent distance and normal values for smaller crevices and thin geometry. The issue is especially prominent at model's eyes, mouth and ears. At 64^3 resolution, these features are pretty smoothed out, with some minor normal artefacts visible as well. Increasing the resolution naturally improves these issues, but even the higher-end 256^3 still seems to exhibit some blurriness around finer details when compared side-by-side with its associated triangle geometry.

Better visual fidelity at higher resolutions comes with clear costs, as the memory required by dense 3D textures scales cubically with linear resolution. A jump from 64^3(~262k texels) to 256^3(~16.8M texels) grid for example increases the storage by 64x. With 32-bit floats, the shifts is roughly ~1.0MB to ~64.0MB just for SDF alone. Sparse or hierarchical data structures (like OpenVDB for example) could reduce this memory overhead, but the benefits would be slightly diminished by the fact that they system would need to store not only surface but also padded region values as well.

The purpose of this test scenario was to demonstrate first hand the increasing cost of dense texture representations, but also additionally to choose appropriate grid resolutions for further investigation. Moving forward, the hybrid

pipelines use coarser 64^3 textures for distance data, while SDF-only baseline cases operate over finer 256^3 grids.

5.2 Visual Analysis: Hybrid Composition

As described in earlier chapters, hybrid pipelines composite two different sources of geometry: smoothly blended SDF values and underlying triangle meshes. Figure 14 shows both these intermediary steps and final results. As intended, the composition policy falls back to conventional mesh representation where there are no blend contributions: the end results seem to have retained all their original detail in regions away from each other. Conversely, where they intersect, the blended surface result is preferred.

Visual differences between the end results of Hybrid-A and Hybrid-B paths seem to be minor. Since Hybrid-B uses per-voxel traversal and numerical cubic solver, opposed to Hybrid-A's ray marching, it ends up having slightly "tighter" surface boundaries, but across multiple viewpoints and frames, the divergence is still pretty subtle.

By contrast, comparing hybrid pipelines' results with baseline SDF-only approach shows more significant changes. Looking at blend regions specifically, since textures in SDF-only approach have higher resolution, they retain more details there – an occurrence that can be best observed near thin geometry. Hybrid methods by comparison intentionally use coarser SDF grids, so their blended surfaces don't carry over as many prominent features present in original triangle mesh. By contrast, bigger and less detailed features hardly change between the two – coarse SDFs don't lose out much detail in those regions, so their blends come very close to the original baseline. Such results were anticipated and make sense – the key question is whether the extra detail retained in baseline results materially changes the perceived quality of smooth blends, something that would probably be determined per-use case. Additionally, it's worth noting that the variance can also depend on blending function configuration: smoother coefficients and larger blend zones will further blur out finer detail, bringing SDF-only results even closer to hybrid pipelines' images.

One consistent artefact in hybrids' images is a slightly more abrupt transition from detailed triangle geometry to smoothed out blended surface. This

can best be observed right at the edges of blend regions where two different sources of geometry meet. SDF-only baseline is more consistent in such cases, since it only ever has one source of geometry and method for rendering it, while hybrid methods' compositional nature makes the transition between two representations a bit more apparent. However, as discussed earlier, composition policy can be altered by modifying different threshold parameters to make the switch more gradual.

5.3 Performance and Memory

To measure how each method performed, frame times and memory usage were recorded. Figure 16 shows average frame times recorded for baseline and two hybrid approaches. The data was collected under same scene and camera setup and averaged over 500 frames after 10second warm-up. The amount of instances in the scene was gradually increased to also observe the scalability of each approach.

Figure 18 shows how both hybrid approaches manage to decrease frame times of SDF-only pipeline, Hybrid-B brick-solve method delivering the biggest improvement of 60.9% for mid-scale scene of 16 instance grid. Hybrid-A's improvement peaked at the same scene set up, delivering 28.8% speed up. Briefly looking at total memory allocation charts, the difference is more drastic between the baseline and proposed methods, with SDF-only approach having to allocate over 67MB just for the dense 3D texture. Even if the rest of the requirements are minimal - it doesn't need vertex/index buffers at all and the BVH is only constructed out of singular AABBs of instances, making it pretty small as well – the cost is still quite significant.

Both performance and memory shortcomings of baseline approach could be addressed by implementing VDB-like hierarchical structure. It would decrease memory consumption and help skipping empty regions inside the bounding AABB. The benefits would be slightly diminished by the fact that not only near-surface but also padded region values would need to be kept, but it would still be a step up. Similar improvements would apply to coarse SDF textures used in hybrid pipelines as initial distance data.

Calling back to visual evaluation of these method's results, hybrid pipeline's base principle of having coarser SDF grids for evaluation did mean

sacrificing small details in blend regions, but seemed to still retain sharpness elsewhere (triangle fallback) while delivering significant improvements in both render times and memory allocation.

5.4 Scaling Behaviour

It's worth taking a closer look at how these metrics scale with increasing instances in test scenes. Both SDF-only baseline and Hybrid-A use real-time sphere tracing, so it makes sense to see their frame times increase with more instances in the viewport, while the memory remains mostly constant (SDF textures allocate some initial fixed amount of memory and are shared between instances, so the only things that grow in memory with scene complexity are TLAS and BVH, but the change is minimal relative to other resources). It's worth noting that frame times are also affected by camera position/viewport – throughout test scenarios, camera angles were adjusted in a way that the entire grid was visible from a slightly top-down angle but keeping only a handful of instances in view instead, with close-up camera positions would reduce ray workload and therefore render times.

Hybrid-B approach deviates from these observations. As discussed in the methodology chapter, this pipeline has a separate compute pass that re-calculates distance values for the sparse brick set. Later during main ray tracing pass, instead of runtime stepping, the shader program just samples these new updated values and calculates exact surface point via interpolation. This means that while the ray tracing cost is still partially dependent on the viewport, the additional compute pass introduces a cost of updating all distance values that is not view dependent, i.e. doesn't make use of frustum or occlusion culling. The instances still share the same bottom level acceleration structure, but the actual distance values associated with bricks need to be separate per instance, meaning the compute pass workload directly depends on the amount of instances in the scene: every single brick of every instance has one thread group of 10x10x10 threads dispatched. The results can be observed in figure 16 that shows how frame times increase at a roughly similar rate for real-time pipelines but for Hybrid-B the growth starts out slow and picks up more from mid-scale scenes (of about 8-16 instances). Figure 18 reinforces this observation, as it shows good initial percentage differences between Hybrid-B

and baseline, but the difference starts to shrink back down as the instance count keeps growing. By comparison, Hybrid-A's percentage values seem to be steadier across varying instance numbers and the frame time graph looks to be following baseline a bit more closely. Looking at memory, it also reflects similar pattern – separate Texture2DArrays are allocated per instance, so the memory associated with that and the additional brick data and mask buffer grows with instance count as well. Even for maximum number considered in these scenarios (64), the overhead doesn't seem to be as considerable as other resources like initial dense textures, or vertex buffers and their acceleration structures, as shown in figures 21 and 22 that describe the breakdown of all different resources allocating memory. Still, in Hybrid-B's case, there's a clear dependency between frame times/memory and the amount of instances in the scene, something that will most likely become more apparent with even more complex scenes with lots of different objects.

5.5 Summary

To summarize, investigated test scenarios showed some visual differences between the high-resolution SDF-only render and the hybrid approaches proposed in this research. The differences can mostly be seen in blend zones that involve finer high-frequency details, where high-resolution SDFs preserve more information, but conversely, hybrid approaches feature better visual fidelity in non-blend areas as they fall back to the original triangle geometry. In terms of performance, both hybrid methods deliver improvements in overall frame times and memory consumption, with more drastic differences shown in the latter department. The gap would probably be reduced by using sparse hierarchical structures for storing distance values, instead of standard dense textures, but as previously stated, the original benefits would be slightly diminished, as outer padded region data for blend zones would also have to be stored on top of near-surface values, and, additionally, these benefits would apply to hybrid approaches as well, as they also make use of dense textures for their initial SDF data.

It's notable that, even with added overhead of separate compute pass and per-instance textures, the benefits of ray tracing SDF grids described by [paper name here] still carry over to dynamically blended surfaces, as Hybrid-B

shows bigger reductions in render times, with not that big of a memory overhead, compared to Hybrid-A's sphere tracing approach. The original method assumed static SDF grids that could be shared across instances, which had to be replaced with per-instance distance data and runtime recalculations for updated blend results. It was interesting to see that even with this significant modification, the ray tracing method still outperformed its sphere tracing alternative.

Overall, these test results managed to demonstrate that the general idea of compositing two different sources of geometry is not only viable in real time, but can also improve resource usage and rendering speed at least in these studied scenarios.

6 Conclusion and Future Work

6.1 Conclusion

The main goal of this project was to propose an alternative hybrid approach to rendering blended signed distance fields of arbitrary models. The idea stemmed from the practical observation that while SDFs enable operations like smooth composition, generating them for arbitrary artist-authored models is not so straightforward: dense 3D textures require higher resolutions to preserve finer detail and analytic/constructive methods (like CSG), while greatly reducing memory footprint, demand additional authoring pipelines that are less standard than conventional triangle mesh workflows. Despite the trade-offs, however, SDFs make compositional or procedural operations straightforward in a way that regular polygonal methods struggle to match, which motivated exploring a hybrid direction.

A central insight of this research was that, as previously stated, SDF textures require higher memory overhead to maintain visual fidelity, but, when used specifically for smooth blending, the operation itself already blurs out high frequency detail in blend regions, so, in active blend areas, might not be that important. However, even with this assumption, the regions where blend operation doesn't deform the surface should still remain sharp and close to the original authored asset. These two key conditions were the foundation of the newly proposed strategy: use coarse SDF textures to calculate the dynamically composed surface in blend regions and just fall back to polygonal representation elsewhere.

Two different implementations – Hybrid-A (runtime sphere tracing) and Hybrid-B (grid ray tracing with per-voxel cubic solve) - of this hybrid idea were proposed and compared against the baseline SDF-only method. The first aspect of the results to inspect was visual fidelity – how and where the two types of end results differed from each other. While there was some divergence in blend regions – higher resolution SDFs preserving more detail even with smooth blend operation – the differences would be strongly dependent on the blend function parameters and the resolution of coarse texture of initial SDF values hybrid methods used. Performance-wise, overall frame times and memory consumptions were observed, and both hybrid methods delivered an

improvement in those aspects, a more recent method of ray tracing grids and solving interpolation equation (Söderlund, Evans, and Akenine-Möller, 2022) showing best results, even after needing significant modifications to handle dynamic blending use case.

6.2 Future Work

Reflecting on overall implementation – one thing that can be said for is that this general idea of hybrid pipeline creates room for a multitude of techniques and data structures. Even presented implementations already presenting a clear scope for improvement. One such technique was continuously mentioned throughout this paper – hierarchical data structure like OpenVDB. While the algorithms taking part in the proposed implementations often took inspiration from VDB architecture, the exact method was never completely implemented to replace the use of dense 3D textures.

Another good place for improvement would be Hybrid-B pipeline and its brick value recalculation process. As already discussed, the computational overhead of that pipeline is directly tied to number of instances and their bricks, as a separate thread group is dispatched per unique brick. The compute pass can't really make use of frustum or occlusion culling and ends up updating all the values of all the bricks, so a smarter way to schedule those updates would be good for performance. An idea was to borrow the concept from temporal techniques often seen in GI or anti-aliasing: using previous frame's data for next frame calculations. This means during ray tracing pass, surface bricks and their immediate neighbours would be recorded and possibly re-formatted by stream compaction algorithms on the GPU. Then, the next frame's compute pass would only issue thread groups for these relevant bricks, reducing the workload. Alternatively, brick value updates could also be tied to changes in the proximity BVH.

Yet another common technique that could be implemented into the pipeline would be multi-resolution distance fields via multiple LOD/mips for reducing the work done for instances further away from the camera. This could be applied both to initial source SDFs and dynamically calculated brick values – although the latter would be a bit more challenging as thread dispatch and indexing logic would have to dynamically adapt to varying brick resolutions.

The composition policy for two different sources of geometry also left some room for experimentation. The one implemented was pretty straightforward and featured essentially an interpolation of triangle and SDF results. It was already effective and allowed minor adjustments via threshold and blend factors, but alternative parameters or methods could be explored as well.

Finally, perhaps biggest and most exciting prospect would be to take the idea of hybrid rendering and use it to implement other SDF features. This research focused on smooth blend operation, but as discussed in the literature review chapter, there are many more visually interesting operations SDFs enable. It would be interesting to see how the general concept of hybrid composition applies to them and what modifications the basic techniques like sphere tracing or sparse grids and per-voxel surface intersection solvers would require to accommodate these new features.

7 References

Eitz, M., Lixu, G. (2007) ‘Hierarchical Spatial Hashing for Real-time Collision Detection’, *IEEE International Conference on Shape Modeling and Applications 2007 (SMI '07).*

Galin, E., et al (2020) ‘Segment Tracing Using Local Lipschitz Bounds’, *Computer Graphics Forum.*

Hart, J. (1995) ‘Sphere Tracing: A Geometric Method for the Antialiased Ray Tracing of Implicit Surfaces’, *The Visual Computer.*

Karras, T. (2012) *Thinking Parallel, Part II: Tree Traversal on the GPU*. Available at: https://developer.nvidia.com/blog/thinking-parallel-part-ii-tree-traversal-gpu/ (Accessed: 7 July 2025).

Müller, T. et al (2022) ‘Instant Neural Graphics Primitives with a Multiresolution Hash Encoding’, *ACM Trans*.

Quilez, I. (2013) *smooth minimum*. Available at: https://iquilezles.org/articles/smin/ (Accessed: 25 May 2025).

Söderlund, H., Evans, A., Akenine-Möller, T. (2022) ‘Ray Tracing of Signed Distance Function Grids’, *Journal of Computer Graphics Techniques,* Vol. 11, No. 3.

Teschner, M., et al (2003) ‘Optimized Spatial Hashing for Collision Detection of Deformable Objects’, *Vision, Modeling, Visualization VMV 2003.*

8 Appendices

8.1 Complete Test Results

Frame Times					
Instance Count	Method	Avg. ms.	Min ms.	Max ms.	FPS
2	SDF-Only	0.7	0.29	1.61	1436.33
2	Hybrid-A	0.7	0.44	1.76	1428.64
2	Hybrid-B	0.55	0.29	1.32	1832.27
4	SDF-Only	1.39	0.94	1.39	719.69
4	Hybrid-A	1.04	0.66	1.04	958.18
4	Hybrid-B	0.69	0.29	0.69	1441.09
8	SDF-Only	2.33	2	2.64	428.63
8	Hybrid-A	1.66	1.23	2.03	603.07
8	Hybrid-B	0.91	0.64	1.58	1100.34
16	SDF-Only	3.16	2.55	3.53	316.31
16	Hybrid-A	2.62	2.08	3.13	381.03
16	Hybrid-B	1.39	1	2.34	720.36
32	SDF-Only	3.65	2.91	4.1	273.9
32	Hybrid-A	3.02	2.41	3.35	331.62
32	Hybrid-B	1.94	1.62	2.27	515.43
64	SDF-Only	4.86	4.26	5.84	205.66
64	Hybrid-A	3.88	3.24	4.37	257.78
64	Hybrid-B	3.49	3.21	3.75	286.89

Memory Consumption							
Instance Count	Method	SDF Textures (Initial)	SDF Textures (Recalculated)	BLAS/TLAS	Proximity BVH	Vertex/Index Buffers	Brick Data Buffers
2	SDF-Only	67100	0	3.45	0.352	0	0
2	Hybrid-A	1050	0	981.58	0.352	385	0
2	Hybrid-B	1050	3.2	986.32	0.352	385	4.276
4	SDF-Only	67100	0	3.58	0.8	0	0
4	Hybrid-A	1050	0	981.84	0.8	385	0
4	Hybrid-B	1050	6.4	986.58	0.8	385	8.56
8	SDF-Only	67100	0	3.84	1.696	0	0
8	Hybrid-A	1050	0	983.89	1.696	385	0
8	Hybrid-B	1050	12.8	988.63	1.696	385	17.12
16	SDF-Only	67100	0	5.89	3.492	0	0
16	Hybrid-A	1050	0	986.96	3.492	385	0
16	Hybrid-B	1050	25.6	991.7	3.492	385	34.25
32	SDF-Only	67100	0	8.96	7.07	0	0
32	Hybrid-A	1050	0	992.95	7.07	385	0
32	Hybrid-B	1050	51.2	997.69	7.07	385	68.5
64	SDF-Only	67100	0	14.95	14.25	0	0
64	Hybrid-A	1050	0	1,004.85	14.25	385	0
64	Hybrid-B	1050	102.4	1,009.59	14.25	385	137.4

8.2 Complete Test Scenarios