Gradient-SDF: A Semi-Implicit Surface Representation for 3D Reconstruction

Christiane Sommer* Lu Sang* David Schubert Daniel Cremers
Technical University of Munich
Computer Vision Group
{c.sommer, lu.sang, d.schubert, cremers}@tum.de

Abstract

We present Gradient-SDF*, a novel representation for 3D geometry that combines the advantages of implict and explicit representations. By storing at every voxel both the signed distance field as well as its gradient vector field, we enhance the capability of implicit representations with approaches originally formulated for explicit surfaces. As concrete examples, we show that (1) the Gradient-SDF allows us to perform direct SDF tracking from depth images, using efficient storage schemes like hash maps, and that (2) the Gradient-SDF representation enables us to perform photometric bundle adjustment directly in a voxel representation (without transforming into a point cloud or mesh), naturally a fully implicit optimization of geometry and camera poses and easy geometry upsampling. Experimental results confirm that this leads to significantly sharper reconstructions. Since the overall SDF voxel structure is still respected, the proposed Gradient-SDF is equally suited for (GPU) parallelization as related approaches.*

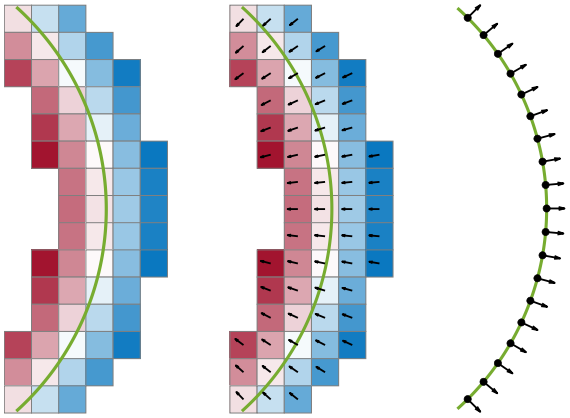

Figure 1. Our Gradient-SDF (*middle*) is a hybrid representation between standard signed distance fields stored in a voxel grid (*left*) and the explicit geometry representation using surfels (*right*): while we inherit the implicit nature of standard SDF voxels, we store gradients per voxel, which is similar to the surface normal property of a surfel. This combines the advantages of implicit representations, such as the possibility for direct SDF tracking, with those of explicit ones, for instance the possibility to perform bundle adjustment.

1. Introduction

The representation of 3D geometry in computer vision is a long-studied research topic. Mathematically, a surface is a 2D manifold embedded in $\mathbb{R}^3$. However, when it comes to implementing this on a computer, the question of discretization comes up: how can we represent a surface with possibly infinite amount of detail and large extent with a finite amount of memory and variables with finite precision? Different answers to this question exist, and which one is most suitable highly depends on the concrete problem one wants to solve.

On the one hand, there are *explicit representations*, such as point clouds, surfel clouds or polygon meshes. They directly sample points on the surface together with additional information like surface normals or point radius (for surfels), or connectivity of points (for meshes). This is useful for applications such as bundle adjustment, where surface points are reprojected into different camera frames.

On the other hand, we have *implicit representations*, that take a different approach: the surface is encoded implicitly by assigning each point $\mathbf{p}$ in the ambient space $\mathbb{R}^3$ a scalar value, such as a binary occupancy, or a (signed) distance to the nearest surface point. *Signed distance fields* (SDFs) have some useful properties, for instance, unlike explicit representations, they allow for changes of surface topology, and they can be updated very easily. There are different ways to store SDFs, the more traditional one being a voxel grid, where 3D space is partitioned into voxels, *i.e.*, cubes of a given size. Each voxel contains the SDF value of its center point, sometimes truncated to a certain value. Such voxel grids can be stored either densely, or sparsely using *e.g.* oc-

* These authors contributed equally.

trees or hash maps. Extracting a surface from an implicit representation requires an additional step, marching cubes being a popular choice [17]. Only recently, implicit parameterizations of implicit representations have become more popular. Neural networks regress the SDF value for a surface at any point, and the set of network weights uniquely characterizes the surface geometry. While these *doubly implicit* representations are very elegant as they theoretically provide an infinite level of detail, updating the geometry is less trivial than for voxel-based implicit representations, and surface extraction often requires voxelization.

In this work, we propose a hybrid between explicit and implicit representations called *Gradient-SDF*: we use a voxel-based implicit SDF representation, and augment it with the SDF gradient. To summarize, we propose the following contributions:

- We propose Gradient-SDF as an implicit geometry representation with explicit features. It exploits first-order Taylor expansion to perform interpolation without accessing several voxels.
- We prove that our stored Gradient-SDF vectors are significantly more accurate than gradients obtained by standard finite difference schemes.
- We show theoretically and experimentally how Gradient-SDF can be used in a depth-based tracking and mapping system, where efficient storage in a hash map is combined with direct SDF tracking.
- We provide a formulation for photometric bundle adjustment (BA) on our implicit voxel-based Gradient-SDF representation and evaluate the benefits of this.

2. Related Work

ICP tracking and SDF-based mapping In KinectFusion [19], the depth camera is tracked by the iterative closest point (ICP) algorithm, where the incoming depth map point cloud is registered to a point cloud extracted from the current SDF model via raycasting. Many methods build on this idea and focus on improving different aspects, most notably on reducing the high memory requirements imposed by the need for a volumetric voxel grid. The voxel hashing approach set the basis for lower memory requirements [20], and others followed and further improved this [12, 13, 32]. In TextureFusion [16], the voxel hashing data structure is augmented by a texture tile in order to compute high-resolution texture within an RGB-D scanning setup. While these KinectFusion-like approaches work very well and can be efficiently implemented on a GPU, they switch between different representations of 3D geometry: the voxel grid stores the SDF and is used for mapping, and a 3D point cloud obtained by raycasting is used for tracking.

SDF-based tracking and mapping Few works have addressed this issue of representation switching, most notably Bylow *et al*. [6] and Canelhas *et al*. [7], who directly minimize a sum of squared SDFs to estimate the camera pose, rather than converting the SDF to a point cloud. Slavcheva *et al*. [24] also convert the input depth map to a volumetric SDF prior to camera pose estimation. Both types of approaches converge better than KinectFusion-like methods, while avoiding the change of representation. However, they heavily rely on the voxel being stored cohesively in memory, as the tracking step involves interpolation and gradient computation of the SDF, which needs to access at least eight voxels for one single lookup.

Surfel-based tracking and mapping Surfel-based methods can also achieve impressive results for 3D tracking and mapping from depth images. Keller *et al*. [14] use surfels, *i.e*., points together with their normals and some additional properties, to reconstruct 3D scenes from depth sensors. To find surfels that correspond to pixels in an incoming image, they use index maps and align the incoming depth map with a virtual model depth map using ICP. One particular advantage of explicit representations is the fact that dense photometric bundle adjustment [10] can be easily integrated into a tracking and mapping pipeline, a recent example being BAD SLAM [22]. Thanks to SurfelMeshing [23], it is even possible to extract meshes from a surfel representation, using a lazily updated octree to store and easily access surfels.

Neural networks for tracking and mapping In the last years, neural networks have been used increasingly for tracking and mapping, as the representation of geometry using learned parameters has proven very successful [4, 11, 18, 21]. In RoutedFusion [30], the SDF is still stored as discrete voxel grid values, but the SDF update for a new incoming depth image is learned. The later NeuralFusion work [31] also represents geometry in a latent space. Similar approaches are also taken in more recent work [2, 28].

3. Geometry Representations for 3D Vision

3.1. Surfels or voxels?

SDFs defined on a voxel grid have been used for decades to store and efficiently update 3D geometry [9]. A subset Ω of $\mathbb{R}^3$ is subdivided into a discrete set of voxels with positions $\mathbf{v}_j \in \mathbb{R}^3$, each of which stores the distance ψ_j to its closest surface point, see also Figure 1 (*left*). The sign indicates if the voxel location is free space ($\psi_j < 0$) or inside an object ($\psi_j > 0$). Such a free-space classification is useful for collision avoidance in navigation tasks. Given an SDF representation and a new distance estimate $d(\mathbf{v}_j)$ for a

voxel located at $\mathbf{v}_j$, ψ_j is easily updated:

$$\psi_j \leftarrow \frac{W_j \psi_j + w(\mathbf{v}_j) d(\mathbf{v}_j)}{W_j + w(\mathbf{v}_j)}\,, \tag{1}$$

$$W_j \leftarrow W_j + w(\mathbf{v}_j)\,, \tag{2}$$

with $w(\mathbf{v}_j)$ a weight indicating how reliable the estimate is, and W_j the current weight estimate. The set of voxels $\mathbf{v}_j$ is usually arranged on a regular 3D grid, $\mathbf{v}_j = v_s \mathbf{m}_j$ with voxel size v_s and $\mathbf{m}_j \in \mathbb{Z}^3$. If stored volumetrically, ψ_j and W_j can be accessed very quickly, but memory grows cubically with the scene size. The memory footprint can be significantly reduced using hierarchical tree structures [25, 26] or hash maps [20]. For those, however, certain operations in the SDF, such as tri-linear interpolation or gradient computation (*i.e.*, normal estimation) can become costly, as multiple voxels need to be accessed. This is an issue, even if voxel blocks as in [20] are used: for a voxel block of size 8^3, only 6^3 voxels lie fully inside the block. For these, neighbors can be accessed similarly to a volumetrically stored SDF. However, for the 57.8% of voxels ($8^3 - 6^3$) in the block that have neighbors outside their block, neighbor lookup still means additional hash table lookups. Furthermore, to fully exploit the regular structure inside blocks, we would have to introduce a distinction between voxel types (fully inside/face/edge/corner). In addition, in hash maps it is possible that not all eight voxel corners exist, meaning we need rules to perform the interpolation given only a subset of neighbors.

On the other side of the spectrum, there are *surfels*: the surface is explicitly represented by a set of points $\mathbf{p}$ on the surface, together with surface normals $\mathbf{n}$, see Figure 1 (*right*). Surfels may have more properties, such as a radius, some visual descriptor, or timestamps [22], that can be used to define update rules in specific tracking or mapping applications. Surfel location and normal can be updated similarly to (1) in a running weighted average fashion. Normal estimation in a surfel representation is trivial, as normals are stored inside the surfel data structure. However, computing the distance to the closest surface point for a given point $\mathbf{p} \in \mathbb{R}^3$ can become quite costly, as it typically involves a nearest-neighbor search. Furthermore, surfel representations don't have an easy way to classify non-surface points in $\mathbb{R}^3$, so they cannot be used for tasks in which we want to distinguish between free and occupied space.

3.2. Gradient-SDF: the best of both worlds

We propose *Gradient-SDF* as a hybrid solution that combines the best of the implicit voxel world and the explicit surfel world: we augment the voxel structure by an additional 3D vector, namely a scaled gradient $\mathbf{g}_j$ of the SDF at that point.

This proposed data structure is visualized in Figure 1. For a signed distance function, the gradient at a point $\mathbf{p}$ is equal to the inwards-pointing surface normal at the closest surface point, and the negative of the outwards-pointing surface normal. Thus, similarly to the update in (1), $\mathbf{g}_j$ can be updated in a straightforward way:

$$\mathbf{g}_j \leftarrow \mathbf{g}_j + w(\mathbf{v}_j)\mathbf{n}(\mathbf{v}_j)\,. \tag{3}$$

In most applications, normals are already computed from the incoming data (*e.g.*, depth maps) for filtering or rendering, so the computation of $\mathbf{n}(\mathbf{v}_j)$ does not introduce any computational overhead. We normalize the weighted sum $\mathbf{g}_j$ to get the actual gradient estimate $\hat{\mathbf{g}}_j$ at $\mathbf{v}_j$.

Storing the gradients together with the distances allows for easy computation of the closest surface point $\mathbf{p}_s$ of a voxel $\mathbf{v}_j$:

$$\mathbf{p}_s(\mathbf{v}_j) = \mathbf{v}_j - \psi_j \hat{\mathbf{g}}_j\,. \tag{4}$$

As each $\mathbf{v}_j$ can be uniquely mapped to a surfel with point $\mathbf{p}_s(\mathbf{v}_j)$ and normal $-\hat{\mathbf{g}}_j$, we can interpret our storage scheme as a *voxelized way to store surfels*.

Just like traditional voxel SDFs, our Gradient-SDF can be stored either volumetrically, or using optimized structures like trees or hash maps. Since memory per voxel is increased by the use of Gradient-SDF, we focus our analysis on sparse voxel storage schemes. Combining a voxel representation with components of a surfel representation (namely, the SDF gradient/surface normal), Gradient-SDF overcomes the issues that pure voxel or surfel representations may have, in particular when voxels are stored sparsely. We demonstrate this on a range of example applications.

4. Example Applications in 3D Vision

4.1. Camera tracking using depth images

To find the rigid body transformation $(R, \mathbf{t})$ of an incoming point cloud with points $\mathbf{p}_k$ to a global surface model $\mathcal{S}$, we aim to minimize a weighted least squares energy

$$E(R, \mathbf{t}) = \sum_k w_k d_{\mathcal{S}}(R\mathbf{p}_k + \mathbf{t})^2\,, \tag{5}$$

where w_k is a weight and $d_{\mathcal{S}}(\mathbf{p})$ denotes the (possibly signed) distance from the point $\mathbf{p}$ to the surface:

$$|d_{\mathcal{S}}(\mathbf{p})| = \min_{\mathbf{p}_s \in \mathcal{S}} \|\mathbf{p} - \mathbf{p}_s\|\,. \tag{6}$$

An energy of this form is usually minimized using Gauss-Newton or Levenberg-Marquardt optimization, which need $d_{\mathcal{S}}$ and $\nabla d_{\mathcal{S}}$ in every iteration of the algorithm. To emphasize the benefits of Gradient-SDF, we briefly review how the two most common approaches—the iterative closest point (ICP) algorithm, and direct SDF tracking—estimate these quantities.

ICP-based tracking If the surface $\mathcal{S}$ is represented by a point or surfel cloud, *i.e.*, using a discrete set of points $\mathbf{q}_l \in \mathcal{S}$, possibly with normals $\mathbf{n}_l$, the distance $d_\mathcal{S}$ and its gradient can be estimated either using the (unsigned) *point-to-point* approximation

$$d_\mathcal{S}^{\text{pt-pt}}(\mathbf{p}) = \|\mathbf{p} - \mathbf{q}_{l^*}\|\,, \tag{7}$$

$$\nabla d_\mathcal{S}^{\text{pt-pt}}(\mathbf{p}) = \frac{\mathbf{p} - \mathbf{q}_{l^*}}{\|\mathbf{p} - \mathbf{q}_{l^*}\|}\,, \tag{8}$$

or the (signed) *point-to-plane* approximation

$$d_\mathcal{S}^{\text{pt-pl}}(\mathbf{p}) = \mathbf{n}_{l^*}^\top(\mathbf{p} - \mathbf{q}_{l^*})\,, \tag{9}$$

$$\nabla d_\mathcal{S}^{\text{pt-pl}}(\mathbf{p}) = \mathbf{n}_{l^*}\,, \tag{10}$$

with

$$l^* = \arg\min_l \|\mathbf{p} - \mathbf{q}_l\|\,. \tag{11}$$

Both require a nearest neighbor search for each evaluation of $d_\mathcal{S}$ and $\nabla d_\mathcal{S}$, which can be implemented using a kD-tree, or more efficiently by searching the pixel neighborhood in the depth image. In approaches like [19, 20], where a discrete SDF representation is used to store the global 3D model, each camera pose estimation step needs to convert the SDF representation to a point cloud in order to apply ICP, which is inconsistent and not elegant.

Direct SDF tracking Approaches like [6, 7], by contrast, directly use the SDF voxels $(\mathbf{v}_j, \psi_j)$ to approximate $d_\mathcal{S}$ using interpolation:

$$d_\mathcal{S}^{\text{sdf}}(\mathbf{p}) = \sum_{\mathbf{v}_j \in \mathcal{N}(\mathbf{p})} \psi_j \omega(\mathbf{p}, \mathbf{v}_j)\,, \tag{12}$$

where $\mathcal{N}(\mathbf{p})$ is a neighborhood of $\mathbf{p}$, and $\omega(\mathbf{p}, \mathbf{v}_j)$ are interpolation coefficients. For tri-linear interpolation, $|\mathcal{N}(\mathbf{p})| = 8$. The gradient of $d_\mathcal{S}^{\text{sdf}}$ can be computed by finite differences over the regular grid of sample points $\mathbf{v}_j$:

$$\nabla d_\mathcal{S}^{\text{sdf}}(\mathbf{p}) = \sum_{\mathbf{v}_j \in \mathcal{N}'(\mathbf{p})} \psi_j \begin{pmatrix} \tau_x(\mathbf{p}, \mathbf{v}_j) \\ \tau_y(\mathbf{p}, \mathbf{v}_j) \\ \tau_z(\mathbf{p}, \mathbf{v}_j) \end{pmatrix}. \tag{13}$$

The coefficients $\tau_{x,y,z}(\mathbf{p}, \mathbf{v}_j)$ and the neighborhood $\mathcal{N}'(\mathbf{p})$ depend on which type of interpolation and which type of finite difference scheme is chosen.

The advantage of such direct approaches over ICP-based ones is that the same volumetric representation that is used for mapping can also be used for pose estimation without any conversion, resulting in a very elegant solution that is easy to implement. However, to evaluate (12) and (13) at least 8 voxels need to be read in order to get $d_\mathcal{S}^{\text{sdf}}$ and its gradient. This is most efficient when voxels are stored contiguously in memory, which restricts the reconstruction volume to small to medium sizes (typically at most 512^3).

Tracking using Gradient-SDF With our data structure, we can easily approximate both $d_\mathcal{S}$ and $\nabla d_\mathcal{S}$ with only one single voxel look-up, using a first-order Taylor expansion:

$$d_\mathcal{S}^{\text{our}}(\mathbf{p}) = \psi_0 + (\mathbf{p} - \mathbf{v}_{j^*})^\top \hat{\mathbf{g}}_{j^*}\,, \tag{14}$$

$$\nabla d_\mathcal{S}^{\text{our}}(\mathbf{p}) = \hat{\mathbf{g}}_{j^*}\,, \tag{15}$$

$$j^* = \arg\min_j \|\mathbf{p} - \mathbf{v}_j\|\,. \tag{16}$$

This looks very similar to the ICP-based formulation, but in our case j^* can be computed without any neighbor search simply by rounding $\mathbf{p}/v_s$, as we know that the $\mathbf{v}_j$ are sampled on a regular grid in $\mathbb{R}^3$.

As a consequence, contiguous memory storage that is so beneficial for volumetric direct SDF tracking approaches is no longer as important, and we can use a hash map instead to compactly store our voxels, while still staying within one geometry representation. This allows us to store larger volumes just like in [20], where voxels far from the surface (*i.e.* with zero weight) are not explicitly stored.

4.2. Pose optimization and bundle adjustment

Typically, bundle adjustment is performed on a sparse set of points [1, 29], as computational cost grows with the number of points. In online approaches, also the number of cameras is usually limited to a sliding window. With the introduction of BAD SLAM [22], these limitations are lifted: the use of depth data for bundle adjustment together with some smart optimization allows for actually performing bundle adjustment on a more dense level. Naturally the question comes up if bundle adjustment can also be performed in implicit dense geometry representations such as signed distance fields. This is where Gradient-SDF comes in handy: while it is very hard to come up with a meaningful bundle adjustment energy in standard SDF representations, we can exploit (4) to define points on the surface for which we want to adjust bundles. Together with the finding of [22] that optimization can be limited to the normal direction, we can set up an *implicit photometric BA* cost:

$$E(\{R_i, \mathbf{t}_i\}, \psi) = \sum_{i,j,c} \nu_{ij} \Phi\big(I_{ij}^c - \tfrac{1}{N_j} \sum_k \nu_{kj} I_{kj}^c\big)\,, \tag{17}$$

where ν_{ij} denotes the visibility of voxel $\mathbf{v}_j$ in frame i ($N_j = \sum_i \nu_{ij}$), $c \in \{\mathrm{r}, \mathrm{g}, \mathrm{b}\}$, and Φ is a robust cost function. I_{ij}^c is given by

$$I_{ij}^c(\{R_i, \mathbf{t}_i\}, \psi_j) = I_i^c\left(\pi(R_i^\top(\mathbf{v}_j - \psi_j \hat{\mathbf{g}}_j - \mathbf{t}_i))\right)\,, \tag{18}$$

with π the perspective projection from $\mathbb{R}^3$ to the image domain. In the optimization, we abstract the original meaning of $\hat{\mathbf{g}}_j$ as gradient of ψ_j, and keep it fixed while changing ψ_j. Following [22], for strongly connected problems, the optimization of poses and distances can be performed

alternatingly to reduce computational cost. We can limit the pose optimization part to voxels that actually contain surface points to reduce computations. Depending on the scene, this approach can be further accelerated by limiting the optimization to the camera poses $(R_i, \mathbf{t}_i)$. Rather than projecting voxel centers $\mathbf{v}_j$ into the RGB images I_i (as done in *e.g.* [6, 20]), we project the real surface point $\mathbf{v}_j - \psi_j \hat{\mathbf{g}}_j$. This is not easily possible in a standard SDF representation, and while the effect for simple BA optimization is small, it opens up new applications such as high-resolution surface optimization using shading and lighting information to improve approaches like [5]. Gradient-SDF thus allows for (photometric) bundle adjustment in an implicit voxel-based representation.

Different cost formulations exist for photometric BA, and we chose (17) which is similar to [10] rather than a sum of pairwise squared intensity differences, because we appreciate the natural interpretation that this provides: for $\Phi(r) = r^2$, the energy E can be rewritten as

$$E(\{R_i, \mathbf{t}_i\}, \psi) = \sum_{j,c} N_j \cdot \operatorname*{Var}_i(\{I_{ij}^c\}) \,, \tag{19}$$

a weighted sum of the intensity variance of each voxel's closest surface point. As we do not store the color per voxel in contrast to [22], (17) couples all camera poses, making pose optimization quadratically dependent on the number of frames used. We lift this limitation by decoupling the original energy cost and simultaneously minimizing

$$E_i(R_i, \mathbf{t}_i, \psi) = \sum_{j,c} \nu_{ij} \Phi\big(I_{ij}^c - \tfrac{1}{N_j} \sum_k \nu_{kj} I_{kj}^c\big) \,. \tag{20}$$

This makes the pose optimization step linear in the number of frames. The simultaneous (rather than alternating) minimization of pose energies means that even though each of the energies contains an average of all residuals I_{ij}^c, this average does not impose any additional computation in practice. We show in the evaluation that the simplification introduced in (20) has nearly no effect on results. After minimizing E w.r.t. ψ_j and $(R_i, \mathbf{t}_i)$, we can compute the color ρ_j^c at voxel $\mathbf{v}_j$ as the mean

$$\rho_j^c = \frac{1}{N_j} \sum_i \nu_{ij} I_{ij}^c \tag{21}$$

Since we have the projection to the closest surface point implicitly encoded in the cost function (17), and do not store an explicit estimate of voxel color, we avoid the correspondence estimation between geometry and texture that is used in TextureFusion [16].

4.3. Surface extraction from a Gradient-SDF

In order to eventually extract a surface from the implicit TSDF representation, we have two choices: we can extract a set of surfels, or we run marching cubes to extract a mesh. We discuss both approaches and present ways to efficiently implement them given our specific data structure.

Oriented point cloud extraction A very fast way to extract a surface representation from our gradient-augmented SDF representation is to extract surface points $\mathbf{p}_s(\mathbf{v}_j)$ together with their normals $-\hat{\mathbf{g}}_j$ from all voxels that have $|\psi_j \hat{\mathbf{g}}_j| \leq \frac{v_s}{2}$ (component-wise). This results in a homogeneously sampled, consistently oriented point cloud of resolution $\frac{v_s}{2}$. Re-sampling for more regularly distributed surfels such as in [32] is not necessary in our Gradient-SDF representation. Regular geometry upsampling of this point cloud is also very easy: we subdivide each voxel into four subvoxels, determine their distance using the Taylor expansion (14), and then extract surfels from subvoxels with $|\psi_j \hat{\mathbf{g}}_j| \leq \frac{v_s}{4}$ instead.

Layered Marching Cubes for mesh extraction Extracting a mesh rather than a set of surfels is also easy for an implicit SDF representation like ours: we can use the well-known marching cubes (MC) algorithm [17]. In the case where voxels are not stored volumetrically in memory, but in a hash map, we can traverse the hash map and check for each voxel if the eight corners of the cube next to it (going in positive x-, y- and z-direction) are allocated. If yes, we can extract the corresponding triangle face from the MC lookup table. This, however, requires a complete re-implementation of the meshing algorithm. We use a different approach that can be more seamlessly integrated in an existing MC implementation: in our *layered marching cubes*, we first extract the minimum and maximum x, y and z coordinates of all voxels to obtain an axis-aligned bounding box. Then, starting from (without loss of generality) minimal z, we allocate memory for two voxel layers of size $(x_{\max} - x_{\min}) \times (y_{\max} - y_{\min})$, and fill it with the first two z-layers for weights, colors, and distance values. We now apply Marching Cubes on the layer interface. After this, we re-fill the first layer with the next z-values and proceed. This way, we avoid a cubic memory usage.

5. Evaluation

To demonstrate the potential of Gradient-SDF, we analyse our stored gradients and show two example applications where we use Gradient-SDF to store the underlying 3D geometry: a simple tracking and mapping system using depth images, and photometric bundle adjustment together with subsampling on a Gradient-SDF initialized from our tracking system.

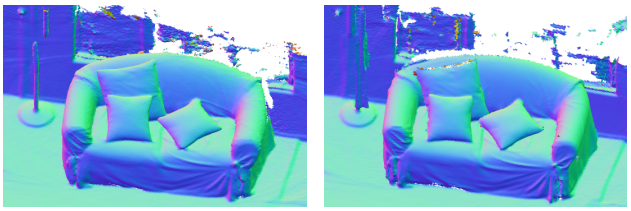

Figure 2. Qualitative reconstruction results for SDF Tracker [7] with $512^3 = 1.34 \times 10^8$ voxels (*left*) and our tracker using a Gradient-SDF hash map (*right*) on the *00577_sofa* sequence from [8]. Shown are the meshes after running marching cubes. The visual quality of results is largely comparable, but our hash map-based method needs more than $20\times$ less memory than the dense storage of [7], despite having three more entries per voxel.

5.1. Implementation

Our code is available on `https://github.com/c-sommer/gradient-sdf`. Our data structure is implemented in C++ with single-precision `float`s, and our `GradSdfVoxels` are stored in a hash map, one voxel per entry. We found that for a CPU implementation, the difference in performance compared to hashing blocks of size 8^3 is marginal. For pose optimization both in the tracker and in the bundle adjustment, we use Gauss-Newton and solve the resulting linear system using Cholesky decomposition. Optimizing the BA cost (20), we do not store the three variables for voxel color, which is in contrast to [6,20]. All experiments were performed on an Intel Xeon CPU @ 3.60 GHz, using OpenMP with four threads, and no GPU.

5.2. Gradient quality on synthetic data

Setup We start our evaluation with an analysis of the gradients $\hat{\mathbf{g}}_j$ that we store in our voxels, and compare them and finite difference gradients to ground truth gradients. Five random spheres with different radii are randomly rendered into a sequence of depth images augmented with Kinect-like sensor noise [15]. We perform (Gradient-)SDF fusion according to the formulas in (1)–(3) using ground truth poses. For a sphere with center $\mathbf{c}$, the (ground truth) SDF gradient at point $\mathbf{p}$ is $\frac{\mathbf{p}-\mathbf{c}}{\|\mathbf{p}-\mathbf{c}\|}$. Furthermore, we have the stored gradient and we can compute a central finite difference gradient from the accumulated SDF.

Results Figure 3 shows the angular deviation of gradient vectors to ground truth, for our stored gradients and for finite differences. Our gradients are clearly much closer to the ground truth ones—both for surface voxels, and throughout the whole truncation region. Additional visualizations in the supplement show that our gradient estimate is much smoother than the one using finite differences. Thus, not only does Gradient-SDF enable easier interpolation of $d_S(\mathbf{p})$, it also produces much better gradient estimates than a standard finite difference scheme.

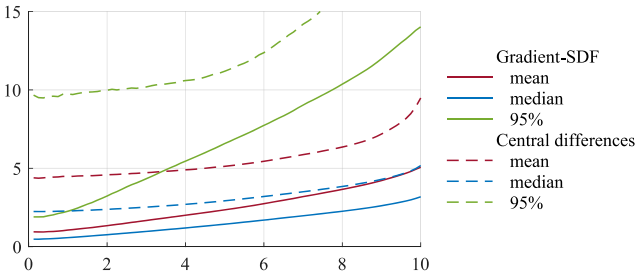

Figure 3. Quality of gradient estimates. For all voxels closer than x voxels to the surface, the y-value of the curves specify mean, median and 95^{th} percentile of the angular deviation from ground truth gradients in degrees. *Solid lines* are Gradient-SDF vectors, and *dashed lines* central finite differences. Our gradients are significantly more accurate than those computed using finite differences, *e.g.* the mean angular deviation of voxels within $10v_s$ from the surface is nearly twice as big for central differences ($9.49°$) as for our stored gradients ($5.07°$).

	KinFu [19]	SDF-Fusion [6]	SDF tracking hash map	Gradient-SDF hash map (ours)
fr1/desk	6.8	3.5	3.9	5.6
fr1/desk2	63.5	6.2	6.6	6.6
fr1/xyz	2.5	2.3	1.7	2.0
fr1/rpy	8.1	4.2	3.9	4.9
fr1/plant	28.1	4.3	5.5	11.2
fr1/teddy	33.7	8.0	10.1	11.3
fr3/household	6.1	4.0	3.8	5.2

Table 1. RMSE of the absolute trajectory error (ATE) in cm on sequences from [27], errors above 25 cm are marked red. On *fr1/floor*, all methods have an ATE above 50 cm, thus we exluded that sequence. While very slow, in terms of ATE our baseline implementation of direct SDF tracking using a hash map is on par with [6]. Gradient-SDF is much faster and still consistently outperforms KinectFusion and is comparable to standard direct SDF tracking. Results for KinFu and the direct tracker are taken from [6].

5.3. Camera tracking using depth images

Our first real-world application is a 3D scanning system that uses depth images to track the camera and build up an implicit 3D model, just like KinectFusion or direct SDF tracking.

Setup We take the general system setup from [6], with a linear weight, and cut off depth values at 3.5m. We choose

a voxel size of $v_s = 2\,\text{cm}$ and truncated at $5v_s$. Since a hash map representation does not allow for a straightforward voxel-wise SDF update [20], we update voxels based on depth image pixels: for each pixel, we update all voxels along the viewing ray that are within the truncation distance. In case the pixel size at the given depth is larger than one voxel, this can lead to some voxels not being updated, thus it is important to not choose the voxel size too small. Normals are estimated using the FALS method from [3], and only points whose angle between normal and viewing ray is less than $75°$ are integrated into the final Gradient-SDF. To have a fair comparison against KinectFusion and direct SDF tracking, we do not use the color values as in [20]. In order to evaluate the benefits compared to a direct SDF tracker with a hash map implementation, we implemented such a tracker as a baseline. This hash map SDF tracker stores voxels sparsely, but still uses tri-linear interpolation to evaluate $d_{\mathcal{S}}$ and $\nabla d_{\mathcal{S}}$, which means each function/gradient evaluation needs eight hash table lookups.

Results In Table 1, we summarize results on the TUM RGB-D dataset [27]. To make this quantitative evaluation reproducible, we switched off OpenMP to generate the table. [6, 19] on average perform best for a 512^3 voxel reconstruction volume [6], so we report numbers for that setting. In terms of average pose error, using a Gradient-SDF and interpolating using Taylor expansion rather than tri-linear interpolation is comparable to the other methods, while being superior in terms of representation consistency for sparse storage schemes. In addition, we show qualitative results in Figure 2 and the supplement. For this qualitative comparison, we use the open-sourced SDF tracker code with a volume of 512^3 voxels, and the same parameters as our Gradient-SDF tracker. The SDF tracker implementation has been extended to give access to the SDF volume, which allows us to run marching cubes on the raw data.

Runtime and memory Pose estimation takes about 30–40 ms per frame, compared to 100–120 ms for the hash map direct SDF tracker we implemented as baseline. This clearly shows the superiority of Taylor interpolation over tri-linear for sparse storage schemes. Integration into the SDF volume takes on average 80 ms and is about 7% (5 ms) slower than the baseline implementation which does not integrate the gradients. Thus, the gains in tracking more than compensate for the marginal overhead introduced in the mapping phase, and a GPU implementation of Gradient-SDF has the potential to run in real time.

The dense reconstruction volume of [6] and [19] consists of 512^3 voxels, which means memory for 1.34×10^8 voxels and thus 2.68×10^8 floating point variables (weights and distances per voxel) is required. This is in contrast to on average 2×10^6 voxels for hash map-based storage. Our baseline hash map direct SDF tracker thus needs about 4×10^6 floats. Gradient-SDF stores gradients in addition, so it has in total five values per voxel, resulting in about 10^7 variables. Despite one single voxel being $2.5\times$ larger than for a standard SDF, in total this is still more than $20\times$ less memory than volumetric storage. Additionally, for volumetric storage we need to know in advance where the reconstruction volume shall be placed, making those approaches less flexible in unknown environments.

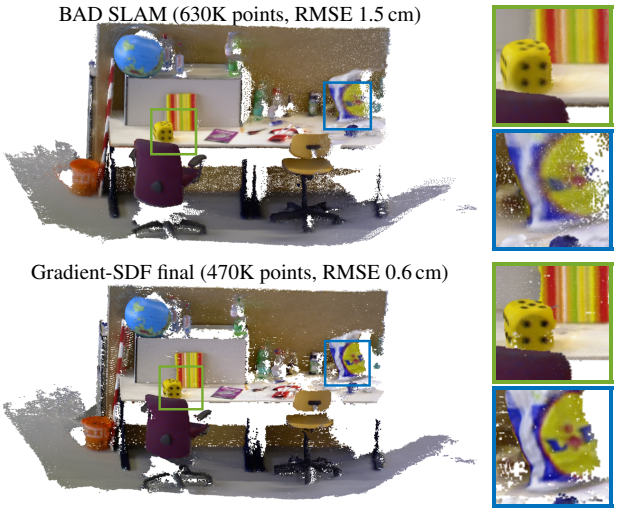

Figure 4. Colored point cloud produced by BAD SLAM (*top*), and after optimization of the BA cost on our Gradient-SDF (*bottom*), *fr3/long_office_household* [27]: the very low root mean squared pose error becomes apparent on the reprojected average colors of the surface points, for our Gradient-SDF even more than for BAD SLAM.

5.4. Bundle adjustment and pose optimization

Our second set of experiments demonstrates the power of our implicit photometric bundle adjustment/pose optimization formulation: on different sequences from [27], we minimize the bundle adjustment cost from (17) for 30 keyframes over 10 seconds, *i.e.* a keyframe ratio of 10%, as in BAD SLAM, and a regularizer weight of 0.01cm^{-2}. The initial geometry is obtained from our depth-based tracker.

Quantitative results All results are summarized in Table 2: We first perform photometric BA by minimizing (17). Poses improve substantially after optimization, while geometry only changes marginally, which we attribute to the good initial estimate that our depth tracker provides. Thus, we run a version with pose-only optimization, which is faster and achieves on-par pose errors, plus looks equally good visually (note that we do not have ground truth for geometry). We perform the same set of experiments for pose optimization decoupled as in (20), and find that we get nearly the

	fr1/xyz	fr1/teddy	fr2/xyz	fr2/rpy	fr3/household
before optimization	2.8	6.8	1.8	2.7	2.1
full BA, poses coupled	1.0	4.4	1.0	2.5	0.6
pose only, poses coupled	1.0	4.3	1.1	2.4	0.7
full BA, decoupled	1.2	4.6	1.0	2.5	0.6
pose only, decoupled	1.1	4.3	1.0	2.4	0.6
BAD SLAM [22]	1.8	3.7	1.3	0.9	1.5

Table 2. RMSE [cm] of translation pose error on 30 keyframes, sequences from [27]: even with a number of computational optimizations (see main text), we are able to keep the error in our implicit bundle adjustment very low compared to the initial error after depth tracking. For reference, we provide numbers for BAD SLAM as an example of an explicit dense bundle adjustment approach. Overall, our errors are comparable to those of BAD SLAM.

same improvement on poses, while keeping the computational complexity linear in the number of keyframes. With these optimizations, in total we can minimize the BA cost in 20–30 ms per iteration and pose using a single-threaded CPU implementation, meaning there is good potential for real-time capability on a GPU. As surfel-based reference method, we run BAD SLAM and use their pose estimates for quantitative evaluation.

Qualitative results We show qualitative results for our photometric optimization (using decoupled poses) and BAD SLAM's built-in surface reconstruction, again with the 30 keyframes out of 300 frames, in Figures 4 (*fr3/household*) and 5 (*fr1/teddy*). We extract the geometry in double resolution as outlined in 4.3 to get a denser point cloud that has a number of points comparable to the BAD SLAM result. For other sequences, see supplement. The better poses and distances are adjusted, the lower the variance of the mean reprojected color in (21) will be, and thus the sharper the texture. For this reason, we show colored point clouds for qualitative visualization. Since we have depth and color input decoupled – one is used for the tracker only, one for the photometric optimization, we nowhere assume synchronized data and can obtain very sharp textures even for unsynchronized datasets like TUM RGB-D [27]. This is in contrast to BAD SLAM and many other RGB-D scanning systems, which assume synchronized data in their model.

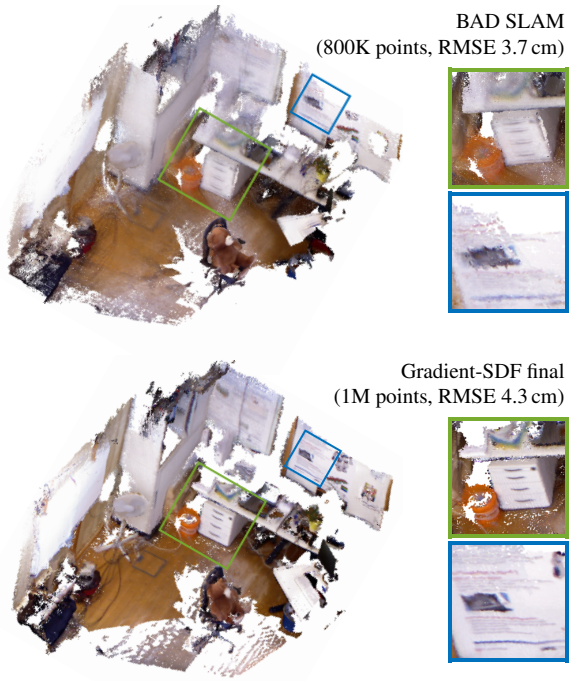

Figure 5. Colored point cloud produced by BAD SLAM (*top*), and after optimization of the BA cost on our Gradient-SDF (*bottom*), *fr1/teddy* [27]: despite slightly larger root mean squared pose error, our final texture is much sharper than that of BAD SLAM.

6. Discussion

While the memory overhead compared to hash map-based SDFs without gradients is not critical for a typical indoor scene, we still aim to reduce memory. Our next step is thus to exploit the fact that unit norm gradients are in $\mathbb{S}^2$. We will investigate how to parameterize $\hat{\mathbf{g}}_j$ with two variables and still keep the update rule simple. The shown applications of Gradient-SDF are exemplary, the representation can be used to store any 3D geometry. Thus, we kept our set of experiments small and instead focused on the theoretical description of how Gradient-SDF can be used. In the future we will explore the benefits of our Gradient-SDF for tracking and mapping in neural geometry representations.

Conclusion We proposed Gradient-SDF as a hybrid representation for 3D geometry that combines the advantages of explicit and implicit representations. By enhancing the classical implicit signed distance function (SDF) with the gradient value, we achieve capacities of explicit representations including direct photometric bundle adjustment. In several experiments we demonstrate these advantages—in particular, our gradients are much more accurate than finite difference approximations, and 3D scanning using Gradient-SDF produces impressively sharp reconstructions.

References

[1] Sameer Agarwal, Noah Snavely, Steven M Seitz, and Richard Szeliski. Bundle adjustment in the large. In *European conference on computer vision*, pages 29–42. Springer, 2010.

[2] Dejan Azinović, Ricardo Martin-Brualla, Dan B Goldman, Matthias Nießner, and Justus Thies. Neural rgb-d surface reconstruction. *arXiv preprint arXiv:2104.04532*, 2021.

[3] Hernan Badino, Daniel Huber, Yongwoon Park, and Takeo Kanade. Fast and accurate computation of surface normals from range images. In *2011 IEEE International Conference on Robotics and Automation*, pages 3084–3091. IEEE, 2011.

[4] Ma Baorui, Han Zhizhong, Liu Yu-shen, and Zwicker Matthias. Neural-Pull: Learning signed distance functions from point clouds by learning to pull space onto surfaces. In *International Conference on Machine Learning (ICML)*, 2021.

[5] Erik Bylow, Robert Maier, Fredrik Kahl, and Carl Olsson. Combining depth fusion and photometric stereo for fine-detailed 3d models. In *Scandinavian Conference on Image Analysis*, pages 261–274. Springer, 2019.

[6] Erik Bylow, Jürgen Sturm, Christian Kerl, Fredrik Kahl, and Daniel Cremers. Real-time camera tracking and 3D reconstruction using signed distance functions. In *Robotics: Science and Systems*, volume 2, page 2, 2013.

[7] Daniel R Canelhas, Todor Stoyanov, and Achim J Lilienthal. SDF tracker: A parallel algorithm for on-line pose estimation and scene reconstruction from depth images. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3671–3676. IEEE, 2013.

[8] Sungjoon Choi, Qian-Yi Zhou, Stephen Miller, and Vladlen Koltun. A large dataset of object scans. *arXiv:1602.02481*, 2016.

[9] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 303–312, 1996.

[10] Amaël Delaunoy and Marc Pollefeys. Photometric bundle adjustment for dense multi-view 3d modeling. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1486–1493, 2014.

[11] Amos Gropp, Lior Yariv, Niv Haim, Matan Atzmon, and Yaron Lipman. Implicit geometric regularization for learning shapes. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 3789–3799. PMLR, 2020.

[12] O. Kahler, V. A. Prisacariu, C. Y. Ren, X. Sun, P. H. S Torr, and D. W. Murray. Very High Frame Rate Volumetric Integration of Depth Images on Mobile Device. *IEEE Transactions on Visualization and Computer Graphics*, 22(11), 2015.

[13] Olaf Kähler, Victor Adrian Prisacariu, Julien P. C. Valentin, and David W. Murray. Hierarchical voxel block hashing for efficient integration of depth images. *IEEE Robotics and Automation Letters*, 1(1):192–197, 2016.

[14] Maik Keller, Damien Lefloch, Martin Lambers, Shahram Izadi, Tim Weyrich, and Andreas Kolb. Real-time 3d reconstruction in dynamic scenes using point-based fusion. In *3DV 2013*, pages 1–8, 2013.

[15] Kourosh Khoshelham and Sander Oude Elberink. Accuracy and resolution of Kinect depth data for indoor mapping applications. *Sensors*, 12(2):1437–1454, 2012.

[16] Joo Ho Lee, Hyunho Ha, Yue Dong, Xin Tong, and Min H Kim. Texturefusion: High-quality texture acquisition for real-time rgb-d scanning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1272–1280, 2020.

[17] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. *ACM siggraph computer graphics*, 21(4):163–169, 1987.

[18] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4460–4470, 2019.

[19] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. KinectFusion: Real-time dense surface mapping and tracking. In *10th IEEE International Symposium on Mixed and Augmented Reality*, pages 127–136. IEEE, 2011.

[20] Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Marc Stamminger. Real-time 3D reconstruction at scale using voxel hashing. *ACM Transactions on Graphics (ToG)*, 32(6):1–11, 2013.

[21] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 165–174, 2019.

[22] Thomas Schöps, Torsten Sattler, and Marc Pollefeys. Bad slam: Bundle adjusted direct rgb-d slam. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 134–144, 2019.

[23] Thomas Schöps, Torsten Sattler, and Marc Pollefeys. Surfelmeshing: Online surfel-based mesh reconstruction. *IEEE transactions on pattern analysis and machine intelligence*, 42(10):2494–2507, 2019.

[24] Miroslava Slavcheva, Wadim Kehl, Nassir Navab, and Slobodan Ilic. Sdf-2-sdf: Highly accurate 3d object reconstruction. In *European Conference on Computer Vision*, pages 680–696. Springer, 2016.

[25] Frank Steinbrücker, Christian Kerl, Daniel Cremers, and Jurgen Sturm. Large-scale multi-resolution surface reconstruction from rgb-d sequences. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3264–3271, 2013.

[26] Frank Steinbrücker, Jürgen Sturm, and Daniel Cremers. Volumetric 3d mapping in real-time on a cpu. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2021–2028. IEEE, 2014.

[27] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of rgb-d slam systems. In *2012 IEEE/RSJ international*

conference on intelligent robots and systems, pages 573–580. IEEE, 2012.

[28] Edgar Sucar, Shikun Liu, Joseph Ortiz, and Andrew J Davison. iMAP: Implicit mapping and positioning in real-time. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6229–6238, 2021.

[29] Bill Triggs, Philip F McLauchlan, Richard I Hartley, and Andrew W Fitzgibbon. Bundle adjustment—a modern synthesis. In *International workshop on vision algorithms*, pages 298–372. Springer, 1999.

[30] Silvan Weder, Johannes Schönberger, Marc Pollefeys, and Martin R Oswald. Routedfusion: Learning real-time depth map fusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4887–4897, 2020.

[31] Silvan Weder, Johannes L Schönberger, Marc Pollefeys, and Martin R Oswald. Neuralfusion: Online depth fusion in latent space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3162–3172, 2021.

[32] Thomas Whelan, Michael Kaess, Hordur Johannsson, Maurice Fallon, John J Leonard, and John McDonald. Real-time large-scale dense rgb-d slam with volumetric fusion. *The International Journal of Robotics Research*, 34(4-5):598–626, 2015.