Continuous Signed Distance Functions for 3D Vision

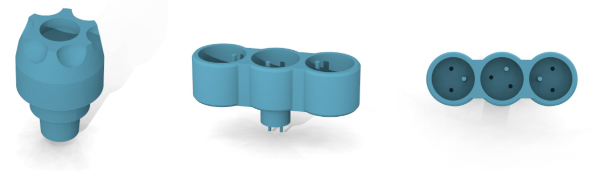
Simen Haugo, Annette Stahl, Edmund Brekke
Department of Engineering Cybernetics, NTNU
simen.haugo@ntnu.no, annette.stahl@ntnu.no, edmund.brekke@ntnu.no

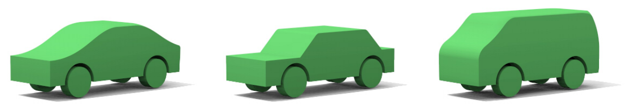
Abstract

We explore the use of continuous signed distance functions as an object representation for 3D vision. Popularized in procedural computer graphics, this representation defines 3D objects as geometric primitives combined with constructive solid geometry and transformed by nonlinear deformations, scaling, rotation or translation. Unlike its discretized counterpart, that has become important in dense 3D reconstruction, the continuous distance function is not stored as a sampled volume, but as a closed mathematical expression. We argue that this representation can have several benefits for 3D vision, such as being able to describe many classes of indoor and outdoor objects at the order of hundreds of bytes per class, getting parametrized shape variations for free. As a distance function, the representation also has useful computational aspects by defining, at each point in space, the direction and distance to the nearest surface, and whether a point is inside or outside the surface.

(a) Furniture formed by combining primitives

(b) Variations generated by adjusting parameters


(c) Mechanical parts with symmetry and repetition


(d) Cars formed by smoothly combining cubes and cylinders

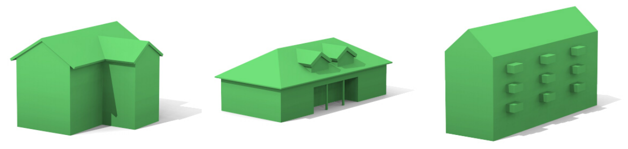

(e) Houses with symmetry, repetition and shape duplication

Figure 1: Objects modelled by combining geometric primitives with constructive solid geometry, enabling an expressive modelling language that can describe many man-made objects, and also exploit symmetries and repeated details. Each object is defined by a closed mathematical expression that can be stored in compiled bytecode in the order of 100 bytes of instructions, and can generate infinitely many variations by controlling the parameters that define the primitives and the operations combining them.

1. Introduction

An essential problem in computer vision is that of inferring a description of the 3D environment from cameras or other sensors. Solving this problem is key to programming robots that can act in and interact with the world around it. Object recognition and Simultaneous Localization and Mapping (SLAM) encompass attempts at solving this problem: SLAM aims to build a map of the environment and estimate the pose of the observer, while object recognition aims to recover the 3D pose and shape of known objects.

State of the art in SLAM represent the map at a low-level [6]: e.g. feature point clouds, meshes or occupancy grids. Such descriptions are useful for mapping algorithms, but fall short for further scene analysis. For this reason, there is a growing research focus on building high-level maps, i.e. by incorporating object recognition.

High-level maps provide valuable information that can feed back into the mapping process, as knowledge of objects and their behaviour informs how their geometry should be treated [4] (i.e. short- or long-term landmarks); prior knowledge about the scale and surfaces of objects can reduce noise or remove outliers [20], resolve scale ambiguity in monocular setups [27], identify occluded geometry [47], or compress the map [46]; higher-level landmarks can facilitate place recognition and loop-closure algorithms [6]; and imposing structural constraints can improve tracking accuracy or robustness in adverse conditions [13, 36]. A high-level description also facilitates tasks the user might have: the presence and location of objects is useful in path-planning, obstacle avoidance or interaction; shapes can be completed from partial measurements using prior knowledge, e.g. for urban reconstruction; or the scene type can be identified, such as "kitchen" or "highway".

High-level maps will be key to deploying robots: to program an autonomous car, it is not enough to have a point cloud of the immediate surroundings; to plan paths and obey traffic rules, we must know where the road, the lanes, and other people in traffic are. Such information can only be acquired by imposing our world knowledge into algorithms that are, otherwise, oblivious to these concepts.

However, the ideal representation of such knowledge is an open problem [6], as representations differ in their descriptive power, storage and pre-processing requirements, implementation complexity, suitability for inference, scalability to larger scenes, and adaptability to variations.

In computer graphics, *continuous signed distance functions* have become popular among the demoscene community for procedurally modelling scenes with a limited memory budget (Fig. 1). Although discrete signed distance functions are an established tool in computer vision for high-quality 3D reconstruction [16], its continuous counterpart has not received much attention despite having compelling properties:

- It is highly expressive: Simple and complex scenes can be defined with the same mathematical tools used by Computer Aided Design (CAD) engineers for decades, supporting also perfectly curved surfaces, like cylinders or spheres, and infinite geometry like planes or half-open cubes. With extra effort, image appearance can be included via procedural coloring, texture maps or other shading techniques.

- It is useful for processing: Being a signed distance function, the direction and distance to the nearest surface can be directly evaluated at any point in space, as well as whether the point is inside or outside the surface; information which would need expensive searches in meshes or point clouds, possibly through complicated volume acceleration structures. This information is of great use in many applications, such as robotic motion planning and optimization.

- It is cheap to store: A variety of surfaces can be exactly defined by mathematical expressions, using orders of magnitude less memory than their discrete counterpart and avoiding the need for compression [34, 7, 16].

- Surfaces can be updated at zero cost: Variations can be generated by adjusting the parameters that define the combination and transformation of primitives, thus avoiding an expensive process of updating a boundary representation or storing large additive structures.

- Models can be "written down on a napkin": Modelling can be done simply with a text editor, thus avoiding the need to install and learn special tools, and avoiding the complexity needed to import and process binary or text formats; all that is needed is to convert the mathematical functions to the programming language of choice.

In this paper we explore the use of continuous signed distance functions in three areas: (1) In modelling man-made objects; (2) In recovering the pose and shape of objects from a scene; (3) In improving the mapping process or informing the task at hand. We first give the reader some context by reviewing how distance functions have been used in related work, and where potential benefits arise in the problem of building high-level maps. We then review the conceptual tools for modelling objects with distance functions, before presenting application vignettes, supported by surveys and qualitative modelling studies, to suggest possible uses and limitations within 3D scene reconstruction and analysis.

2. Related work

In computer graphics, signed distance functions (SDF) have been used to model scenes with deformable nature-like geometry [15]; render high-quality text [28]; render shadows and other effects [55]; or make 3D content generation more accessible [31, 22]: in particular, Dreams by Media Molecule [22] is an art game that uses geometric primitives as "brushes" that the player can use to sculpt a scene. By representing the scene and the primitives as signed distance functions, they are able to perform incremental scene updates in real-time, which is difficult with triangle meshes.

In SLAM, signed distance functions have been useful for merging depth measurements into one cohesive surface: i.e. the *truncated signed distance function* (TSDF), popularized by Newcombe et al. [38], is considered to be important for dense 3D reconstruction [16], due to its benefits for modelling continuous surfaces, dealing with noise, and efficient incremental updates. An issue with TSDFs is their large storage requirement due to discretization [16]; although compression methods have been derived [7, 16], these come at the price of added complexity, inaccuracy and artifacts. Continuous distance functions could give the needed scalability, but have yet to be used in this domain.

Incorporating prior world knowledge to build high-level maps has been done by imposing assumptions on the scene type: i.e. containing mostly flat surfaces, cubes or vertical walls, or containing (partially) known objects; and then inferring its parameters: e.g. pose and shape parameters of objects, or the location of the vertical walls. For objects in particular many representations have been devised; balancing suitability for detection, parameter estimation and adaptability to variations in shape and appearance, with implementation complexity, storage requirements and use in further scene analysis. Some examples include 3D point clouds with associated image descriptors [27, 44], CAD meshes [35], deformable wireframes [10], and models built from hand-crafted or learned descriptors [57, 50].

Of particular relevance is the deformable SDF representation introduced by Prisacariu et al. [42, 21, 58]: storing objects as discrete SDFs with the space of shape variations built from examples. However, storing a single object class with 100 shape parameters in a 256^3 SDF, at 32-bit precision, requires 6.4 gigabyte of memory. While compression can reduce the same data down to 6 megabyte [42], this introduces artifacts and complexity, and decompression incurs a performance hit during parameter estimation. In contrast, continuous SDFs can describe articulation and variability at the order of hundreds of bytes, uncompressed.

Objects as volumetric primitives appears to be a resurging representation: Tulsiani et al. [53] convert detailed 3D objects into collections of cubes, and note that such a representation can be useful for shape manipulation and similarity reasoning. Similar to this paper, they use a continuous distance function to measure discrepancy between the assembled shape and the input shape. This paper takes the notion further and considers objects as collections of more diverse primitives using techniques from the CAD/CSG and demoscene communities.

3. Continuous Signed Distance Functions

3.1. Definition

The *distance function* [26] $f(p) : R^3 \to R$ of a set of surface points S is defined as the distance from p to the closest point in S:

$$f(p) = \min_{q \in S} ||p - q|| \tag{1}$$

where the surface itself is given by the *level-set* or *iso-surface* $S = \{p : f(p) = 0\}$, and the distance $|| \cdot ||$ is some metric on R^3. *Signed* distance functions encode which side a point is on by the sign of f, such as taking outside as positive and inside as negative. *Continuous* signed distance functions (CSDF) are represented by a closed expression, unlike their *discretized* counterpart, which are represented as sampled volumes.

A simple example of a continuous signed distance function is that of a sphere of radius r centered at the origin. For any point $p = (x, y, z) \in R^3$, the function $f : R^3 \to R$ gives the signed distance between p and the closest point on the surface of the sphere, and can be written under the Euclidean distance metric as:

$$f(p) = \sqrt{x^2 + y^2 + z^2} - r \tag{2}$$

The above is also an example of an *implicit surface*. While *parametric surfaces* or *boundary representations*, such as that of a triangle mesh or a spline patch, is defined by a function that, given parameters, produces a point in space, an implicit surface is defined by a function that, given a point in space, indicates whether the point is on the surface or not. In general, this function does not define a geometric distance. For example, we could have described the sphere as the level-set of

$$f(p) = x^2 + y^2 + z^2 - r^2 \tag{3}$$

which defines an algebraic distance. Implicit surfaces have a long history (see e.g. [26]), but the subset of implicit surfaces defined by signed distance functions have a number of useful properties for robotics, computer vision and graphics. For example, unlike boundary representations, it is trivial to determine whether a point is inside, on or outside a surface, as the function that gives this information is defined everywhere in space. Further, the gradient of $f(p)$ provides the surface normal if p is on the surface, and the direction towards the closest point on the surface otherwise. Finally, in the next sections, we will see that complicated surfaces can be described with constructive solid geometry.

3.2. Distance metrics

The Euclidean distance l_2 is often used because of its utility in many applications: i.e. in motion planning [41, 54] or in optimization [32]. For one, it is rotation invariant, meaning that shapes look the same after rotation. It is also smooth with respect to its variables, which is desirable when computing gradients. Other metrics can have advantages over the Euclidean norm [54]: i.e. the max-norm l_∞ can in some cases be cheaper or simpler to compute.

In the remainder of this paper, distance functions are assumed to use the Euclidean l_2 metric, keeping in mind that the results and techniques presented may not be valid under other metrics.

3.3. Modelling scenes

Although distance functions have been derived for common geometric primitives, such as the cube, sphere and cylinder (see [29] and Fig. 2), deriving the distance function by hand is not a strategy that scales to complicated scenes. Thankfully, by defining a membership predicate over the entire domain, implicit surfaces are compatible with boolean operators from *constructive solid geometry* (CSG). This allows us to combine primitives with the well-known set operations: union, intersection, complement and difference. For signed distance functions, these can be implemented with min and max functions (Fig. 3):

$$\textit{intersection}(S_1, S_2) := \max(f_{S_1}, f_{S_2}) \tag{4}$$
$$\textit{union}(S_1, S_2) := \min(f_{S_1}, f_{S_2}) \tag{5}$$
$$\textit{complement}(S) := -f_S \tag{6}$$
$$\textit{difference}(S_1, S_2) := \max(f_{S_1}, -f_{S_2}) \tag{7}$$

Although the resulting distance function is not strictly Euclidean (Fig. 9), it is often a good approximation close to the surface [26]. Other implementations of the set operations have been derived (see [24, 23]); these aim to strike a balance between accurately approximating the Euclidean distance and being differentiable everywhere (Fig. 10).

A variety of *domain* operations are also possible [45]. Shapes can be rotated and translated by applying the inverse transformation on the input domain (Fig. 4):

$$\textit{rotate}(S, R) := f_S(R^T p) \tag{8}$$
$$\textit{translate}(S, T) := f_S(p - T) \tag{9}$$

Rotation and translation are examples of *isometric* transformations, meaning they do not change the distance between two points. Examples of non-isometric transformations include twisting, blending or scaling. For example, an object can be scaled by applying the (inverse) scale to the input:

$$\textit{scale}(S, k) := f_S(p/k) \quad \text{(incorrect)}$$

but the resulting function no longer defines the distance in the original coordinate system. For uniform scaling this is fixed by scaling the result appropriately:

$$\textit{scale}(S, k) := f_S(p/k)k \tag{10}$$

Other deformations such as non-uniform scaling, twisting or blending cannot be repaired to recover a correct distance function (Fig. 7). However, scaling factors can be derived that ensure that the distance function never overestimates [29]. One could also approximate the distance by the first order expression [24, section 5.3]

$$\frac{f(p)}{||\nabla f(p)||} \tag{11}$$

which can also estimate the distance to a generic implicit surface. Since the error tends to grow larger further away from the surface, one could substitute the object with a proper bound, such as a simple bounding shape, for points outside so as to limit the approximation error.

Domain mirroring and repetition can describe visually complex objects by exploiting symmetry and duplicated details (Fig. 5 and Fig. 6). However, the resulting distance function may be discontinuous, in that there can be a jump at the interface between the repeated domains, which can cause distances to be overestimated (Fig. 8). This can be handled by ensuring that the object being repeated is symmetric about the plane orthogonal to the axis of repetition, such that distances are exactly equal at the interface.

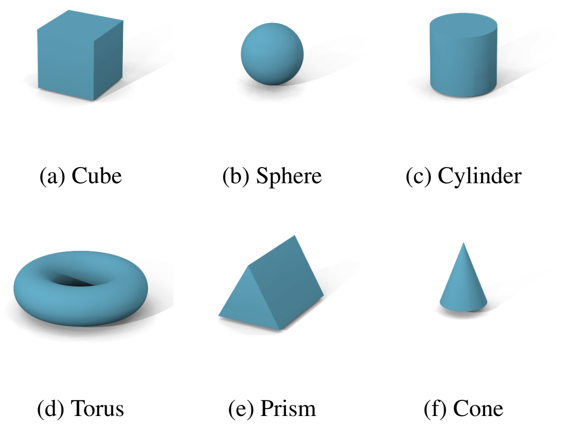

(a) Cube (b) Sphere (c) Cylinder
(d) Torus (e) Prism (f) Cone

Figure 2: Basic primitives (see [37, 29])

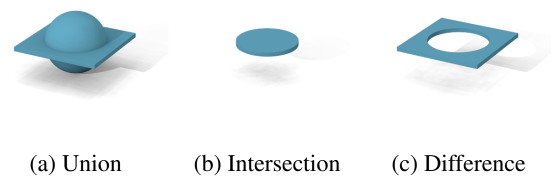

(a) Union (b) Intersection (c) Difference

Figure 3: Basic distance operations with min and max.

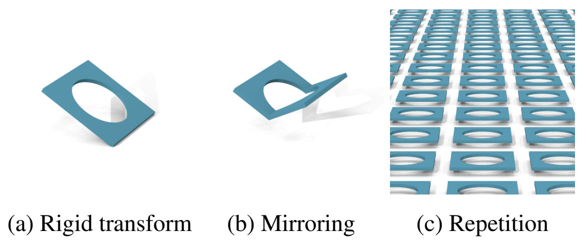

(a) Rigid transform (b) Mirroring (c) Repetition

Figure 4: Basic domain operations implemented with matrix- and vector multiplication and addition on the input points (a); taking the absolute value of one or more of the input coordinates (b); taking the modulus of one or more of the input coordinates (c)

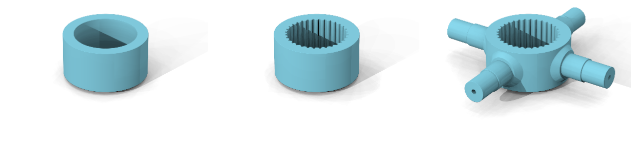

(a) Base (b) Notches (c) Beams

Figure 5: A cross shaft modelled by subtracting one cylinder from a larger cylinder to form the base; adding a tall, thin cylinder and repeating it with polar repetition to form the inner notches; adding cylindrical flanges with a chamfered union; and combining multiple cylinders and repeating them at 90 degree angles to form the beams.

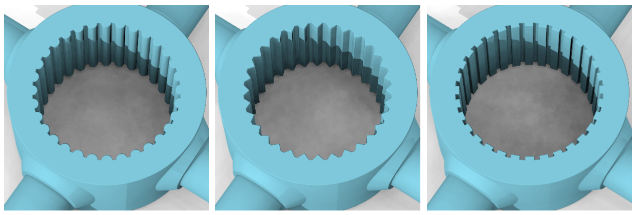
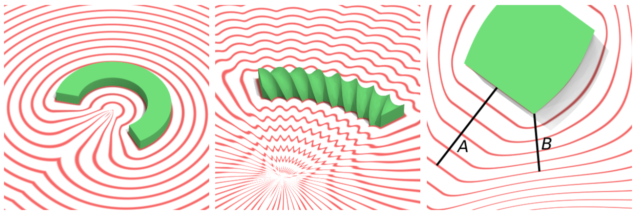
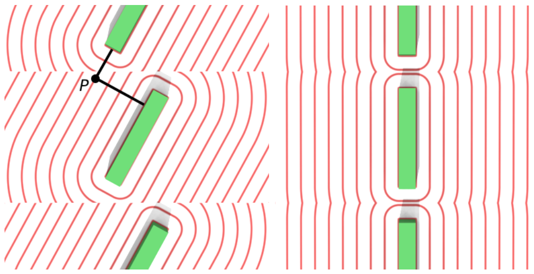

(a) Cylindrical notch (b) Prism notch (c) Cube notch

Figure 6: The visual complexity is detached from the structural complexity, in that there is only a single notch shape defined, which is repeated across the cylinder. This can greatly reduce modelling time and storage cost, while allowing changes to the notch to be automatically reflected everywhere.



(a) Bent cube (b) Screwed cube (c) Inflated cube

Figure 7: More complex shapes can be modelled with non-linear domain transformations, however, the resulting function will no longer be a valid distance function. The error can be visualized through the cut 2D distance field, by observing that isolines (red) no longer describe a constant distance to the surface: in (c) the lines A and B, indicating the shortest paths to the surface, are not the same length even they are on the same isoline. Note that this error tends to be smaller near surface: see the nearest isolines in (c).



(a) Discontinuous interface (b) Continuous interface

Figure 8: Repetition will in general not preserve continuity and can cause distances to be overestimated: On the left a rotated cube is repeated along one axis. A point P incorrectly evaluates the distance to be toward the center cube, even though the shortest distance is in fact up across the interface. On the right is shown a repeated cube where the distances are equal on both sides of the domain interface, and thus no overestimation is done.

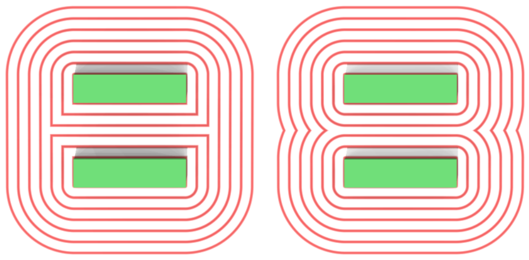

(a) Incorrect distance (b) Correct distance

Figure 9: The min or max of two Euclidean distance function do not always result in a Euclidean distance: On the left is shown a cube with a smaller cube subtracted. Note the incorrect distance near the corners of the inner region.

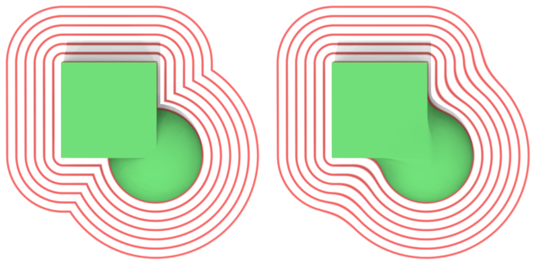

(a) Hard union (b) Soft union

Figure 10: The min or max of two distance functions does not always preserve smoothness. On the left (a) is shown the normal “hard” union (min) of two primitives. Note the kinks where the two sets of isolines meet, in contrast to the “soft” union on the right (b). Some implementations can be found in `hg_sdf` [37]. See also [23].

4. Applications

We now explore the use of CSDFs in three areas: (1) modelling objects; (2) pose and shape estimation; (3) improving mapping and scene analysis. Our discussion is based on surveys: looking for pipelines where the object representation could be replaced by CSDFs and considering the implications of doing so; and qualitative studies where we model selected objects by hand and discuss the expressive power, in terms of what kinds of objects and what level of detail is best suited for the representation, as well as the complexity involved in storing and operating on CSDF objects in a programming language.

4.1. Modelling tools

Text-editor. The most basic tool consists of a text-editor, in which code for the distance function is written by hand, accompanied by a GUI to visualize the scene. Examples of such tools can be found online[1], where the user can define a distance function in a C-like language and see the resulting scene live. Visualization can be done by sphere-tracing — an efficient ray tracing technique that exploits the distance function to traverse the space in step sizes guaranteed to never overstep the surface [29, 33] — or by rasterizing a mesh or point cloud obtained by iso-surface extraction [26].

Graphical editor. Some tools mimic the interface of traditional modelling software: Reiner et al. [45] made an editor with a live 3D view where primitives can be manipulated with a mouse and keyboard. They support several modes of manipulation, including rigid-body transformation, scaling, and adjusting properties, such as radius or height, with sliders; as well as the basic distance and domain operators. Code for the resulting distance function is automatically generated and exportable.

3D scanning is an alternative to manual modelling that has contributed to large object databases [11] in the form of meshes and point clouds. However, although CSDFs can be converted to a mesh or point cloud, generating a CSG structure from a boundary representation is ongoing research. Some recent work include Andrews et al. (2013) [1], who semi-automatically reverse-engineer meshes into a CSG tree; and Fayolle et al. (2016) [24], who detect primitives from a point cloud, and formulate an optimization problem to recover a CSG tree that combines the primitives to best fit the point cloud.

CAD software. CSDFs share the underlying modelling principle used by Computer Aided Design (CAD) tools. CAD tools represent models in terms of a tree of operations and primitives, which could possibly be exported to generate a CSDF expression. To keep the evaluation cost low, one could possibly remove details while the surface approximation error is below some threshold. However, we did not found any tools for doing this automatically, nor do all the shapes or operations employed in CAD software have simple distance functions: e.g. parametric surfaces.

4.2. Expressive power

Objects that consist of boolean operations between a few primitives are suited for precise modelling. Furniture, mechanical parts, houses and street signs, are among some objects that fall in this category. IKEA furniture is particularly suited (Fig. 1a), and sometimes come in variations that are parametrizable (Fig. 1b). Mechanical parts (obtained from [30]) tend to consist of a few primitives, often exhibiting symmetries or repeated details, and are likewise straightforward to model (Fig. 1c). Residential housing also appears to be suited for precise modelling (Fig. 1e).

Objects that consist of parametric surfaces, such as NURBS (used in car and motorcycle design), are impractical to model precisely. The closest distance to such surfaces do not, in general, have a closed-form solution but involve iterative root-finding or subdivision steps [51, 19]. Thus, although a precise model is possible with such algorithms (c.f. isogeometric analysis [14]), the resulting distance function can be prohibitively expensive to evaluate. If a coarser model is acceptable, a car chassis could be approximated with cubes and cylinders, and either a linear or rounded union operation (Fig. 1d).

Modelling the image appearance of objects can be done by defining a mapping from 3D points to color [45]. However, texture mapping requires parametrizing 3D space onto the surface, which is difficult in general [39, chapter 7].

4.3. Implementation complexity

Once an object has been defined in terms of its underlying primitives and operations, it must be made available in a programming language in order to use it in algorithms.

One way to define CSDF objects is programmatically as functions that accept a triplet of floating-point coordinates and return a floating-point distance (see e.g. [37]). This enables a uniform interface for algorithms that are agnostic to the particular object and are only concerned with the distance. Rigid-body transformation and scaling can be done by transforming input points before invoking the function, and scaling the return value appropriately. Most programming languages support such interfaces: e.g. function pointers in C or classes in C++. A more abstract way to store a CSDF is as a tree of primitives and operators [31, 22], sometimes called a scene graph [45].

The size of the generated bytecode for an object depends on many factors, such as the CPU instruction set and compiler settings, but can roughly be measured by the number of instructions. For example, counting addition and subtraction (`add`), multiplication (`mul`), division and modulo (`div`), function calls (`call`), and min, max and abs

[1] https://www.shadertoy.com/view/Xds3zN

(`branch`), a sphere (Eq. 3) can be written with three `adds`, three `muls`, and one `call` (square root). The instruction count for some more functions is shown in the table below: e.g. the cross shaft (Fig. 5) consisted of eight `calls`, five `min/maxs` and eight `adds`. If instructions are 32 bits long, the cross shaft could be stored in roughly 100 bytes (including constants and excluding called functions), which is arguably smaller than a triangle mesh of corresponding fidelity.

Function	`add`	`mul`	`div`	`call`	`branch`
`Sphere`	3	3	0	1	0
`Cylinder`	3	2	0	1	2
`Cube`	3	3	0	1	14
`CrossShaft`	8	0	0	9	5

Table 1: Instruction count for some primitives

For complex objects, the cost of evaluating the exact distance can be unwieldy. To mitigate this, one can use bounding volume hierarchies; substituting objects by simpler bounding shapes for points sufficiently far away [45].

4.4. Use in object detection methods

Once an object has been modelled, a natural goal is to recover its pose and possibly additional parameters (such as scale and deformation) from a scene. Recent object recovery pipelines tend to operate in two phases [50]: detection, where the category and coarse parameters of an object are retrieved among possibly many objects; and refinement, where the parameters are refined to best explain the observations. We now consider the first of these phases: how CSDF objects can be detected in a scene.

Detect constituent primitives. Extracting primitives, possibly only partially present, from point clouds is an important problem in surface reconstruction and reverse engineering [3, 24, 48, 52]. Since CSDFs consist of combinations of primitives, one could detect a complete object by its constituent parts, in a similar way as how a set of 2D image descriptors on a 3D object can enable recognizing a particular object and its pose.

Match features from sampled surface. Aligning two point clouds is a well-studied problem [32, 18] whose results could be used by point-sampling the CSDF surface [26]. The resulting point cloud could be registered against a target point cloud using standard methods: e.g. found in the Point Cloud Library [32]. For sparse point clouds, such as maps obtained from sparse SLAM methods, or self-similar objects, such as cylinders, keypoint matching may be difficult; see Cieslewski et al. (2016) [12] and Drost et al. [18] (2010) for a discussion.

Match volumetric features from SDF. While 3D keypoint descriptors have been of great use for registering point clouds, similar descriptors have been derived for registering SDF volumes directly [9], including generalized Harris corners, Shi-Tomasi descriptors and integral invariants. Such descriptors could be computed from a continuous SDF as well by sampling its domain in a volumetric grid.

Match image features from texture map. Despite the difficulty, in general, to obtain a 2D parametrization for texture mapping, if such a map nevertheless is in place, one can establish correspondences between pixels in the image and 3D coordinates on the object, and thereby recover the object using standard perspective-n-point methods [27, section 5.5]. This also enables the use of bag-of-words models, that have been useful in identifying a particular object from a database of candidates [27, section 5].

Use CSDFs only for refinement. If the above methods are impractical, one could use a seperate model for the detection step and only use a CSDF model for precise parameter estimation. There are a variety of pipelines that consist of a coarse detection step, such as proposing a 2D or 3D bounding box and an object category, followed by a refinement step, such as aligning a precise, deformable 3D model to images and depths. For an overview of such pipelines, see for example [10, 17, 25] for monocular images, [21] for stereo images, and [43, 50, 56] for recovering objects from depth and color images.

4.5. Use in refinement methods

Once the object class and a coarse estimate of its parameters has been detected, it is desirable to obtain more precise parameter estimates for accurate 3D analysis. In this section we look at how CSDFs can fit into precise parameter estimation methods for recovering objects and shape.

Cloud-cloud registration. Point cloud registration methods typically obtain a coarse alignment by matching 3D keypoints, and then refine the alignment with the Iterative Closest Point (ICP) algorithm [32]. Such methods could be used directly with a CSDF representation, by sampling its surface to generate a point cloud.

SDF-cloud registration. Some methods try to align SDFs directly to point clouds: Bylow et al. (2013) and Canelhas et al. (2013) [5, 8] represent a scene reconstructed from RGBD measurements as a TSDF, and estimate frame-to-frame camera motion by aligning the measured point cloud (obtained from the depth camera) against the current TSDF. The key idea being that, given a correct reconstruction and camera motion estimate, the measured point cloud, once transformed into the correct coordinate frame, should lie directly on the surface and have distances of zero. They express this quality metric as a cost function to be minimized with respect to the camera pose:

$$\arg\min_{R,T} \sum_p f\left(Rp + T\right)^2 \tag{12}$$

This defines an optimization problem, which can be (attempted) solved i.e. with the Gauss-Newton method [40].

A similar strategy can be used for CSDFs: the difference being that CSDFs evaluate the distances directly instead of trilinearly interpolating values in a grid. Aside from just rotation and translation, one could define an optimization problem over additional shape parameters:

$$\arg \min_{R,T,k,c_1,...,c_n} \sum_p (f(Rpk + T; c_1, ..., c_n)/k)^2 \qquad (13)$$

where $Rpk+T$ is the transformation from scene coordinates to scaled object and $c_1, ..., c_n$ are shape control parameters. The shape parameters could control the rotation and translation of smaller parts, to describe articulated objects, or they could control the parameters of the primitives involved.

SDF-RGBD registration. Relying solely on 3D data can be difficult when the scene reconstruction is sparse [49]. Hence, Dame et al. (2013) [17] include image appearance in a similar optimization problem, to obtain precise pose and shape of a deformable SDF. Their cost function both encourages 3D points to lie close to the SDF surface, and also maximizes the discrimination between image background and foreground. Such a method can be readily applied on CSDFs, using e.g. sphere-tracing to render the surface.

SDF-RGB registration. Marchand et al. (2016) [36] survey methods for object pose estimation from single images without depth, among them are optimization methods that penalize distances between a projected silhouette of the object and edges detected in the image. Such methods could be applicable to CSDFs, by rendering the CSDF with sphere-tracing to obtain its silhouette.

SDF-SDF registration. Slavcheva et al. (2016) [49] use TSDF maps for dense SLAM and estimate frame-to-frame camera motion by generating a TSDF from the incoming RGB-D frame, and then aligning both SDFs against each other. They use an optimization approach where the cost function includes the distance function discrepancy summed over a common domain and a term that encourages normals to be identical. A similar strategy could be used to align a CSDF object against a discrete SDF map.

4.6. Use in mapping and analysis

Once a high-level description of the scene is recovered, it is possible to exploit additional prior knowledge to improve the mapping process or better inform the task at hand:

Reconstruction noise can be reduced by encouraging the map to conform to object surfaces [2, 13, 44, 27]. The CSDF representation can provide performance benefits by providing the distance to the surface more efficiently than meshes or point clouds, and is compatible with SDF fusion methods (e.g. [17, 20]). On the other hand modelling detailed objects can be impractical (e.g. [27] and [44]), both in cost of evaluating the SDF and by means of acquiring it.

Motion planning algorithms often ask if a point is inside or outside the collision geometry [41]. The CSDF representation provides this information directly, and can thus be useful for such algorithms. Optimization-based algorithms can also use the gradient to "push" trajectories away from collision [41]. However, the direct evaluation cost might make continuous SDFs inferior to discretized SDFs, depending on the cost of compute versus memory access. This could be mitigated with bounding volume hierarchies [45].

5. Outlook

In this paper we have seen that Continuous Signed Distance Functions can be a powerful modelling tool, similar to that used in Computer Aided Design, with the ability to describe a variety of man-made objects, including perfectly curved surfaces, articulation and in-class variations, at considerably lower storage cost than meshes, point clouds, or discrete SDFs. This could be essential for scaling current 3D vision algorithms up to support the vast number of object classes found in real-life. By being a distance function, it also directly provides occupancy and the direction and distance to the nearest surface, which is useful in mapping and object recovery algorithms.

With even rudimentary tools, such as a text-editor, objects can be defined with Constructive Solid Geometry and rigid and nonlinear transformations. While tools for CSDF modelling are presently lacking, it seems feasible to convert the CSG tree obtained from CAD software to a CSDF expression, which would enable sophisticated tools. But, while CSDFs can be converted to meshes or point clouds, by isosurface extraction, going the other way is more difficult, which limits the use of existing object databases or 3D scanning to rapidly obtain models. There are however user-guided solutions that can assist the modelling process.

Some limitations prevent widespread adoption. For one, structurally complex objects or parametrized surfaces can be impractical to model precisely due to lack of tools, and the potentially large computational cost in evaluating their exact distance. This could be mitigated if crude approximations are tolerable, or by using bounding volume hierarchies at the expense of complexity. Second, the difficulty in acquiring a texture map over the surface of general objects prevents the use of established image-based methods for recognition and pose and shape estimation.

We are nonetheless excited about the potential scalability of this representation, and think it would be an interesting research direction to replace low-level maps altogether with CSG and CSDFs: e.g. in the form of a tree of operations and primitives. Unlike discrete SDF volumes, this requires considerably less storage, and inherits the benefits of distance functions. This appears to necessitate a means of inferring a CSG structure directly from images or depth measurements, which could be a task suited for machine learning.

References

[1] J. Andrews. User-guided inverse 3d modeling. 2013. 6

[2] M. Berger, A. Tagliasacchi, L. Seversky, P. Alliez, J. Levine, A. Sharf, and C. Silva. State of the art in surface reconstruction from point clouds. In *EUROGRAPHICS star reports*, volume 1, pages 161–185, 2014. 8

[3] M. Berger, A. Tagliasacchi, L. M. Seversky, P. Alliez, G. Guennebaud, J. A. Levine, A. Sharf, and C. T. Silva. A survey of surface reconstruction from point clouds. In *Computer Graphics Forum*. Wiley Online Library, 2016. 7

[4] J. Biswas and M. M. Veloso. Localization and navigation of the cobots over long-term deployments. *The International Journal of Robotics Research*, 32(14):1679–1694, 2013. 2

[5] E. Bylow, J. Sturm, C. Kerl, F. Kahl, and D. Cremers. Real-time camera tracking and 3d reconstruction using signed distance functions. In *Robotics: Science and Systems*, 2013. 7

[6] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016. 1, 2

[7] D. R. Canelhas, E. Schaffernicht, T. Stoyanov, A. J. Lilienthal, and A. J. Davison. An eigenshapes approach to compressed signed distance fields and their utility in robot mapping. *arXiv preprint arXiv:1609.02462*, 2016. 2

[8] D. R. Canelhas, T. Stoyanov, and A. J. Lilienthal. Sdf tracker: A parallel algorithm for on-line pose estimation and scene reconstruction from depth images. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 3671–3676. IEEE, 2013. 7

[9] D. R. Canelhas, T. Stoyanov, and A. J. Lilienthal. From feature detection in truncated signed distance fields to sparse stable scene graphs. *IEEE Robotics and Automation Letters*, 1(2):1148–1155, 2016. 7

[10] F. Chhaya, D. Reddy, S. Upadhyay, V. Chari, M. Z. Zia, and K. M. Krishna. Monocular reconstruction of vehicles: Combining slam with shape priors. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 5758–5765. IEEE, 2016. 3, 7

[11] S. Choi, Q.-Y. Zhou, S. Miller, and V. Koltun. A large dataset of object scans. *arXiv:1602.02481*, 2016. 6

[12] T. Cieslewski, E. Stumm, A. Gawel, M. Bosse, S. Lynen, and R. Siegwart. Point cloud descriptors for place recognition using sparse visual information. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 4830–4836. IEEE, 2016. 7

[13] A. Concha, W. Hussain, L. Montano, and J. Civera. Incorporating scene priors to dense monocular mapping. *Autonomous Robots*, 39(3):279–292, 2015. 2, 8

[14] J. A. Cottrell, T. J. Hughes, and Y. Bazilevs. *Isogeometric analysis: toward integration of CAD and FEA*. John Wiley & Sons, 2009. 6

[15] B. Cutler, J. Dorsey, L. McMillan, M. Müller, and R. Jagnow. A procedural approach to authoring solid models. In *ACM Transactions on Graphics (TOG)*, volume 21, pages 302–311. ACM, 2002. 2

[16] A. Dai, M. Nießner, M. Zollhöfer, S. Izadi, and C. Theobalt. Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface reintegration. *ACM Transactions on Graphics (TOG)*, 36(3):24, 2017. 2

[17] A. Dame, V. A. Prisacariu, C. Y. Ren, and I. Reid. Dense reconstruction using 3d object shape priors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1288–1295, 2013. 7, 8

[18] B. Drost, M. Ulrich, N. Navab, and S. Ilic. Model globally, match locally: Efficient and robust 3d object recognition. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 998–1005. Ieee, 2010. 7

[19] E. Dyllong and W. Luther. Distance calculation between a point and a nurbs surface. Technical report, DTIC Document, 2000. 6

[20] M. Dzitsiuk, J. Sturm, R. Maier, L. Ma, and D. Cremers. Denoising, stabilizing and completing 3d reconstructions on-the-go using plane priors. *arXiv preprint arXiv:1609.08267*, 2016. 2, 8

[21] F. Engelmann, J. Stückler, and B. Leibe. Joint object pose estimation and shape reconstruction in urban street scenes using 3d shape priors. In *German Conference on Pattern Recognition*, pages 219–230. Springer, 2016. 3, 7

[22] A. Evans. Learning from failure. Siggraph, 2015. 2, 6

[23] P.-A. Fayolle and A. Pasko. Distance to objects built with set operations in constructive solid modeling. In *Proceedings of the 13th International Conference on Humans and Computers*, pages 41–46. University of Aizu Press, 2010. 4, 5

[24] P.-A. Fayolle and A. Pasko. An evolutionary approach to the extraction of object construction trees from 3d point clouds. *Computer-Aided Design*, 74:1–17, 2016. 4, 6, 7

[25] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE transactions on pattern analysis and machine intelligence*, 32(9):1627–1645, 2010. 7

[26] S. F. Frisken and R. N. Perry. Designing with distance fields. In *ACM SIGGRAPH 2006 Courses*, pages 60–66. ACM, 2006. 3, 4, 6, 7

[27] D. Gálvez-López, M. Salas, J. D. Tardós, and J. Montiel. Real-time monocular object slam. *Robotics and Autonomous Systems*, 75:435–449, 2016. 2, 3, 7, 8

[28] C. Green. Improved alpha-tested magnification for vector textures and special effects. In *ACM SIGGRAPH 2007 courses*, pages 9–18. ACM, 2007. 2

[29] J. C. Hart. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, 12(10):527–545, 1996. 3, 4, 6

[30] T. Hodan, P. Haluza, Š. Obdržálek, J. Matas, M. Lourakis, and X. Zabulis. T-less: An rgb-d dataset for 6d pose estimation of texture-less objects. In *Applications of Computer Vision (WACV), 2017 IEEE Winter Conference on*, pages 880–888. IEEE, 2017. 6

[31] F. Hoenig. A road to 3d for everyone. GPU Technology Conference (GTC), 2017. 2, 6

[32] D. Holz, A. E. Ichim, F. Tombari, R. B. Rusu, and S. Behnke. Registration with the point cloud library: a modular framework for aligning in 3-d. *IEEE Robotics & Automation Magazine*, 22(4):110–124, 2015. 3, 7

[33] B. Keinert, H. Schäfer, J. Korndörfer, U. Ganse, and M. Stamminger. Enhanced sphere tracing. 2014. 6

[34] M. Klingensmith, I. Dryanovski, S. Srinivasa, and J. Xiao. Chisel: Real time large scale 3d reconstruction onboard a mobile device using spatially hashed signed distance fields. In *Robotics: Science and Systems*, 2015. 2

[35] J. J. Lim, H. Pirsiavash, and A. Torralba. Parsing ikea objects: Fine pose estimation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2992–2999, 2013. 3

[36] E. Marchand, H. Uchiyama, and F. Spindler. Pose estimation for augmented reality: a hands-on survey. 2016. 2, 8

[37] mercury. hg_sdf: A glsl library for building signed distance functions. http://mercury.sexy/hg_sdf/, 2016. [Online; accessed 27-May-2017]. 4, 5, 6

[38] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*, pages 127–136. IEEE, 2011. 2

[39] O. Nilsson. *Level-set methods and geodesic distance functions*. PhD thesis, Linköping Universisty Electronic Press, 2009. 6

[40] J. Nocedal and S. Wright. *Numerical optimization*. Springer Science & Business Media, 2006. 8

[41] H. Oleynikova, A. Millane, Z. Taylor, E. Galceran, J. Nieto, and R. Siegwart. Signed distance fields: A natural representation for both mapping and planning. In *RSS Workshop on Geometry and Beyond*, 2016. 3, 8

[42] V. A. Prisacariu, A. V. Segal, and I. Reid. Simultaneous monocular 2d segmentation, 3d pose recovery and 3d reconstruction. In *Asian Conference on Computer Vision*, pages 593–606. Springer, 2012. 3

[43] M. Rad and V. Lepetit. Bb8: A scalable, accurate, robust to partial occlusion method for predicting the 3d poses of challenging objects without using depth. *arXiv preprint arXiv:1703.10896*, 2017. 7

[44] D. Ramadasan, T. Chateau, and M. Chevaldonné. Dcslam: A dynamically constrained real-time slam. In *Image Processing (ICIP), 2015 IEEE International Conference on*, pages 1130–1134. IEEE, 2015. 3, 8

[45] T. Reiner, G. Mückl, and C. Dachsbacher. Interactive modeling of implicit surfaces using a direct visualization approach with signed distance functions. *Computers & Graphics*, 35(3):596–603, 2011. 4, 6, 7, 8

[46] R. B. Rusu. *Semantic 3D object maps for everyday robot manipulation*. Springer, 2013. 2

[47] M. Salas, W. Hussain, A. Concha, L. Montano, J. Civera, and J. Montiel. Layout aware visual tracking and mapping. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 149–156. IEEE, 2015. 2

[48] R. Schnabel, R. Wahl, and R. Klein. Efficient ransac for point-cloud shape detection. In *Computer graphics forum*, volume 26, pages 214–226. Wiley Online Library, 2007. 7

[49] M. Slavcheva, W. Kehl, N. Navab, and S. Ilic. Sdf-2-sdf: Highly accurate 3d object reconstruction. In *European Conference on Computer Vision*, pages 680–696. Springer, 2016. 8

[50] S. Song and J. Xiao. Deep sliding shapes for amodal 3d object detection in rgb-d images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 808–816, 2016. 3, 7

[51] J. B. Thomassen, P. H. Johansen, and T. Dokken. Closest points, moving surfaces, and algebraic geometry. *Mathematical methods for curves and surfaces: Tromsø*, pages 351–362, 2004. 6

[52] Z. Toony, D. Laurendeau, and C. Gagné. Pgp2x: Principal geometric primitives parameters extraction. In *GRAPP*, pages 81–93. Citeseer, 2015. 7

[53] S. Tulsiani, H. Su, L. J. Guibas, A. A. Efros, and J. Malik. Learning shape abstractions by assembling volumetric primitives. *arXiv preprint arXiv:1612.00404*, 2016. 3

[54] G. Varadhan, S. Krishnan, Y. J. Kim, S. Diggavi, and D. Manocha. Efficient max-norm distance computation and reliable voxelization. In *Symposium on geometry processing*, pages 116–126, 2003. 3

[55] D. Wright. Dynamic occlusion with signed distance fields. Siggraph, 2015. 2

[56] X. Zabulis, M. I. Lourakis, and P. Koutlemanis. Correspondence-free pose estimation for 3d objects from noisy depth data. *The Visual Computer*, pages 1–19, 2016. 7

[57] A. Zeng, S. Song, M. Niessner, M. Fisher, J. Xiao, and T. Funkhouser. 3dmatch: Learning local geometric descriptors from rgb-d reconstructions. *arXiv preprint arXiv:1603.08182*, 2016. 3

[58] S. Zheng, V. A. Prisacariu, M. Averkiou, M.-M. Cheng, N. J. Mitra, J. Shotton, P. H. Torr, and C. Rother. Object proposals estimation in depth image using compact 3d shape manifolds. In *German Conference on Pattern Recognition*, pages 196–208. Springer, 2015. 3