NTNU

Norwegian University of
Science and Technology

Simple Shapes for Localization and Mapping

Simen Haugo

Master of Science in Cybernetics and Robotics
Submission date: June 2017
Supervisor: Annette Stahl, ITK
Co-supervisor: Edmund Brekke, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Preface

This report is written as a compulsory part of the five-year MSc programme in Engineering Cybernetics at NTNU. It has intentionally been written with a high IPI (information per inch) in an attempt to maximize the likelihood of inspiring the reader with ideas.

Thanks to my supervisor Annette Stahl and to Edmund Førland Brekke for giving feedback along the way and reviewing earlier drafts of this report.

Simen Haugo
Trondheim, June 2017

ABSTRACT

About 540 million years ago, an astonishing event occurred that caused evolution to ramp up her production: the Cambrian explosion. It is believed that the evolution of eyes and, subsequently, the knowledge of where one is and what the world looks like, had much to do with it.

Simultaneous Localization and Mapping (SLAM) is an attempt to one-up nature using human ingenuity, but despite substantial advances in recent years, current solutions still fall short. For one, the representation of the information has a significant effect on the implementation difficulty of the tasks we want to perform. For example, 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 is, where the lanes are, and where other colleagues in the traffic are. Such information can only be acquired by imposing our world knowledge into the algorithms that are, otherwise, oblivious to these concepts.

However, the ideal representation for this knowledge is still an open problem. This report considers the problem of augmenting SLAM maps with fabricated objects, such as houses, cars, furniture, rooms or teacups. In particular, we consider how these objects, of which there is a tremendous variety, can be modelled. A good model representation should be expressive, so as to describe many object classes; adaptable, so as to describe unseen variations; scalable, so as to be stored at a low memory cost; and usable, in the operations which we wish to perform: such as detecting and recovering an object from the map, or exploiting the information that an object's existance provides.

Inspired by a procedural scene modelling technique in computer graphics, we investigate the use of continuous signed distance functions to model objects as geometric primitives, transformed and combined with well-known set operations from constructive solid geometry, and deformations, such as scaling, rotation and translation. Unlike their discretized counterpart, which although they have become an important tool in various fields, the continuous distance function is stored as a mathematical expression, formed by computing directly with analytic distance functions.

We show through experiments and surveys that this representation has several benefits: such as being nearly compatible with highly sophisticated CAD tools; being fit for modelling many indoor and outdoor objects with a few primitives; requiring orders of magnitude less memory than their discrete counterpart, without compression or loss of precision; and being able to describe an infinite variety of objects through controlling the parameters of the constituent operations and primitives. Being a signed distance function, it also inherits their compelling benefits: such as defining, at each point in space, the direction and distance to the nearest surface, and whether this point is inside or outside the surface.

SAMMENDRAG

Den kambriske eksplosjonen betegner den plutselige opprampingen av evolusjonen som skjedde omtrent 540 millioner år siden, som medførte et stort antall nye skapninger. Det er trodd at utviklingen av øyet og, derav, bevissthet om hvor en er og hvordan verden ser ut, hadde mye å gjøre med saken.

Simultaneous Localization and Mapping (SLAM) er menneskets forsøk på å gjøre det samme for våre maskiner, men til tross for store framskritt på dette problemet, så er nåværende løsninger fremdeles ikke gode nok. For eksempel, for å programmere an selvkjørende bil, er det ikke nok å ha en punktsky av dens umiddelbare omgivelser; for å planlegge kjøreruter og adlyde trafikkregler, må vi vite hvor veien og kjørefeltet er, samt hvor andre biler eller objekter befinner seg. Slik informasjon kan kun oppnås ved å tilføye vår kunnskap om kultur og vår verden inn i algoritmer som, ellers, ikke har noe forhold til slike konsepter.

Derimot er den idelle representasjonen for slik kjennskap fremdeles et åpent problem. Denne rapporten betrakter delproblemet om å tilføye til SLAM en modell av menneskeskapte objekter, slik som hus, biler, møbler, rom eller tekopper. En god modell bør kunne beskrive mange typer objekter; bør kunne beskrive usette variasjoner; bør ha lav lagringskostand; og bør være nyttig i algoritmene vi ønsker å benytte den til: f.eks. oppdage objekter fra rekonstruert geometri, eller utnytte objektets egenskaper for videre prosessering.

Inspirert av teknikker fra datagrafikk, ser vi på bruk av *continuous signed distance functions* for å modellere objekter som en kombinasjon av geometriske primitiver, satt sammen med *constructive solid geometry* og transformert med skalering, rotasjon og translasjon. I motsetning til deres diskretiserte motpart, som har blitt et viktig verktøy i mange områder, så er den kontinuerlige varianten lagret som et matematisk uttrykk, satt sammen av en håndfull analytiske funksjoner og operasjoner.

Gjennom eksperimenter og litteraturundersøkelser, viser vi at denne representasjonen har flere fordeler: slik som å være kompatibel med CAD verktøy av høy kvalitet; er passende for å modellere mange innendørs og utendørs objekter; trenger størrelsesordner mindre lagringsminne enn dens diskrete motpart, uten komprimering eller tap av presisjon; og har evne til å beskrive uendelig mange variasjoner ved å kontrollere parameterene til primitivene og operasjonene som bygger opp objektet. I tillegg arves de ettertraktede egenskapene til avstandsfunksjoner ellers: avstand og retning til nærmeste overflate, og om et punkt er innenfor eller utenfor en overflate, er informasjon som kjapt kan beregnes utifra definisjonen.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

READING GUIDE

Definitions

Term	*Meaning*
Reconstruction	Obtaining a 3D representation of a scene from images.
Pose	Rigid-body rotation and translation in 3D space.
Scale	Relative scale of an object in the reconstructed scene.
Shape	A specific instance within a class of parametrized object shapes.
Detection	Obtaining coarse estimates of some parameters of an object model.
Refinement	Obtaining precise estimates of every parameter of the object model.
Registration	Synonym of refinement.
Recovery	Determining the existence and precise parameters of an object.

Table 1: Frequently used words

Notations

Symbol	*Meaning*
p, q	Point in 3D space
P	Set of reconstructioned 3D points
S	Set of points on a surface
f	Signed distance function
f_S	Signed distance function of S
R, T	Rotation matrix and translation vector
k	Object scale

Table 2: Frequently used symbols

Abbreviations

Term	*Meaning*
SLAM	Simultaneous Localization and Mapping
SDF	Signed Distance Function
CSDF	Continuous Signed Distance Function
TSDF	Truncated Signed Distance Function
ICP	Iterative Closest Point
CSG	Constructive Solid Geometry
CAD	Computer Aided Design
GUI	Graphical User Interface
NURBS	Non-Uniform Rational B-Spline

Table 3: Frequently used abbreviations

1 INTRODUCTION

Simultaneous Localization and Mapping (SLAM) is the problem of building a model of the environment (the map), and the estimation of the pose of the sensor(s) moving within it. This is an important problem with applications in many areas, such as automotive, agriculture and inspection, or, beyond robotics, in content creation, helping the visually impaired, or facilitating search-and-rescue workers.

Attempts at solving this problem have predominantly represented the environment in terms of a fairly *low-level* description, such as a collection of 3D points, a triangle mesh or an occupancy grid. Although such descriptions are useful for underlying algorithms, they fall short for further scene analysis. An important focus in recent research has therefore been on building maps with *high-level* scene descriptions.

A high-level description has two-fold benefits. First, it facilitates the tasks that the user wants to perform: query the the presence and location of objects, such as for path-planning, obstacle avoidance or interaction; complete shapes from partial measurements, using prior knowledge about geometry, such as for quality control or urban reconstruction; or identify traversible terrain or scene category, such as being in a kitchen or on a highway, to inform planning algorithms.

It also provides valuable information that can feed back into the mapping process: objects and knowledge about their behaviour informs how their geometry should be treated: i.e. as short-term or long-term landmarks [4]; prior knowledge about the scale and surfaces of objects can be used to reduce noise or remove outliers [23], resolve the scale ambiguity in monocular setups [33], identify occluded geometry [58], or compress the map [57]; higher-level landmarks can facilitate place recognition and loop-closure algorithms [7]; and additional structure can impose constraints to improve tracking accuracy or robustness in adverse conditions [15, 44].

However, the ideal representation of higher-level structures is an open problem [7], as the representations differ in their descriptive power, storage and pre-processing requirements, modelling and processing complexity, scalability to larger scenes, and adaptability to variations. An additional trade-off has to be made as the properties associated with one representation can make it more desirable over another for certain algorithms.

Representing the map in terms of objects modelled as combinations of geometric primitives, such as cubes or cylinders, is an interesting direction that has not received much attention in SLAM [7], despite having compelling properties. For one, primitives enable very compact models that can still capture many elements in man-made environments. It also provides an easy way to discern scenes and objects: i.e. by looking at the parameters that define the primitives.

My focus in this report is on a geometric modelling tool called continuous signed distance functions (CSDF), that has been used extensively within computer graphics. It is based on a principle of combining distance functions to well-known geometric primitives using constructive solid geometry. I think this can be a useful tool for representing objects in SLAM maps for the following reasons:

It is highly expressive

Simple and complicated shapes can be defined by combining and transforming geometric primitives with constructive solid geometry: adding and subtracting shapes, applying rigid-body transformations or other deformations: chamfering, blending, twisting or repetition. Perfectly curved surfaces like cylinders or spheres, and infinite geometry like planes or half-open cubes can also be described. With extra effort, image appearance can be included with procedural coloring, texture maps or other shading techniques.

It has compelling properties

Being a signed distance function, CSDFs directly provide the direction and distance to the nearest surface from any point in space, as well as whether a 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 or optimization.

It is cheap to store

Unlike discretized SDFs that have been popular in many SLAM methods [19], CSDFs do not suffer from their scalability issues since the surface can be defined directly with a handful of mathematical expressions, thus requiring orders of magnitude less storage than uniform 3D grids and avoiding the need for complicated compression techniques [42, 9, 19].

Surfaces can be updated at zero cost

An infinite variety of shapes can be generated on the fly by adjusting parameters that control the combination and transformation of primitives, thus avoiding an expensive process of updating an explicit surface or volumetric grid.

Models can be written down on a napkin

Modelling can be done as simply as with a text editor, thereby avoiding the need to install special tools and learning how these operate, and possibly making it simpler to integrate a modelling tool into an application. Models can also be portable, in that there is no need to write exporters and importers to handle special binary or text formats; all that is needed is to convert the mathematical functions to the programming language of choice.

In this report I investigate continuous signed distance functions as an object representation for point cloud SLAM, and study their advantages and drawbacks in three areas:

- *Use in object modelling:* Are CSDFs useful for modelling objects found in man-made environments? To answer this question I model some indoor and outdoor objects, and study the authoring process and available tools, implementation complexity, scalability and compatibility with existing object databases.
- *Use in object recovery:* Once an object has been modelled, a natural goal is to obtain its pose and (optionally) additional shape parameters (like deformation or scale) in the scene. I study the applicability of CSDFs for detection (determining the presence and rough location of an object), and refinement (determining the precise shape, pose and scale of the object). I survey detection pipelines and discuss how CSDFs might fit in. I also implement an ICP-like refinement method and discuss problems that arise, such as implementing CSDFs in code, and the effect of scale ambiguity and sparsity.
- *Use in post-processing:* Once an object has been recovered, it becomes possible to exploit prior knowledge to improve the SLAM process or inform the task at hand. I survey relevant applications where the representation can have a significant impact, and study the benefits and drawbacks that CSDFs have over other representations.

Why these three? As I discuss in Section 2, one is not necessarily stuck in any particular representation as it is possible to convert meshes and point clouds to SDFs and vice versa. The question of interest is therefore which representation do we use for what? The properties of one representation might make it simpler or more efficient for one task over another: i.e. CSDFs might be suited for modelling, but less so for recovering objects. I think these three areas — modelling, recovery and post-processing — cover the main uses in which the choice of representation matters.

Why divide the recovery problem into detection and refinement? I think detection is a problem that greatly benefits from domain knowledge: such as image appearance of objects, possible objects in the scene, and so forth; ignoring knowledge that a user is likely to have would be needlessly limiting. Why, then, should we consider the refinement problem? After all, if detection is best left to the user, shouldn't they be recovering the pose themselves? I think this dichotomy is useful because I see it befitting recent object recovery pipelines: methods based on machine learning are getting good at predicting the category and rough location of an object in the scene, but, since these predictions alone fall short for further 3D interaction, they are followed by a fine alignment step against a 3D model of the object (see related work in [62]).

Why point cloud SLAM? I would like to study this representation in other SLAM architectures: meshes, voxel grids, planes, points and edges, or lines, are among a few of the map representations, beyond point clouds, that have been used in SLAM. But such a study has been deferred to future work, for now. Point cloud SLAM is nevertheless interesting on its own, as it can be more applicable on power-, weight- and size-constrained platforms.

1.1 Related work

The work presented in this thesis lies at the intersection of several domains: computer graphics and computer aided design; map representations and high-level scene descriptions in 3D reconstruction and SLAM; surface reconstruction and reverse engineering; motion planning and interaction. This section is meant to ground the work presented in a wider research context and give an impression of how it relates to existing work, and where the results might be useful.

Figure 1: An animated snail modelled by combining analytic distance functions to geometric primitives, such as extruded bezier curves (to form the body and antennae) and logarithmic spirals (to form the shell). The image was generated from a few hundred lines of code that define the distance function to the snail and leaf, and rendered by ray-tracing each pixel against the combined surface. Copyright Inigo Quilez.
Source: `http://iquilezles.org/www/articles/sdfmodeling/sdfmodeling.htm` (retrieved 2017).

Signed distance functions in computer graphics

Signed distance functions have become an important tool in robotics as well as many other fields. The inspiration for this thesis comes from their use in computer graphics, where they have been used extensively in 4k demos[1] to procedurally render complex scenes and effects on a limited memory budget (Figure 1); modelling scenes with deformable nature-like geometry [18]; or rendering high-quality text and 2D shapes [34]. Recent video game engines are also experimenting with signed distance functions for storing and rendering scenes: *Dreams* by Media Molecule[2] 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 at real-time framerate, which would have been difficult using triangle meshes. Moreover, the properties of distance functions make them useful for lighting and shading, as computing occlusions can be done efficiently.

[1] An animated music video generated entirely from a 4096 byte executable

[2] `http://www.mediamolecule.com/blog/article/siggraph_2015` (retrieved 2017)

Map representations in SLAM

Within 3D reconstruction and SLAM, signed distance functions have been discovered to be particularly useful for merging depth camera measurements into one cohesive surface. The *truncated signed distance function* (TSDF), popularized by Newcombe et al. [47], is currently considered to be an important tool for high-quality dense 3D reconstruction [19], due to its beneficial properties for modelling continuous surfaces, dealing with noise, and for doing incremental updates. While the continuous signed distance function (CSDF), considered in this thesis, is stored as a mathematical expression, a TSDF is discretized as a 3D grid, each cell being called a *voxel*. TSDFs are also *truncated*, meaning that distances beyond a certain threshold are considered invalid; which is a useful property for compression.

The terminology here can be confusing: the prevalent definition of TSDFs is *not* the Euclidean distance from a point in space to the nearest point on the surface, but rather the distance along the rays of the sensor towards the surface: the *projective* distance [47, 19, 6, 50, 61, 10]. The motivation behind this is that the projective distance is much faster to compute: given a set of 3D point measurements; each point can be transformed into the correct coordinate frame by simple rotation and translation; and the distance is readily computed as the vector length of each transformed point. Unlike computing the Euclidean SDF to a set of 3D points, computing the projective distance is trivially parallelizable and thus suited for GPU implementations. In contrast, the definition we will work with for CSDFs is that of the Euclidean distance.

An issue with TSDFs is that they do not scale well to larger scenes as they require tremendous storage [19]. Significant research has been put into reducing their storage cost using compression techniques [9, 19]. These come at the price of additional complexity, inaccuracy, or even artifacts. In contrast, CSDFs scale to simple and complex scenes at considerably lower storage cost, especially when the surfaces involved are well approximated by geometric primitives. To illustrate, consider a table consisting of a box and four cylinders; storing the distance function as discrete grid would arguably be wasting space when the distance can be readily computed from the primitives involved.

It would be interesting to study if CSDF expressions could be used to represent maps in SLAM: incrementally building a tree of CSG operations on simple volumetric primitives, resulting in a global signed distance function to the final surface. Similarily to TSDFs, the surface can be incrementally updated by adding and subtracting shapes, without the need to tessellate surfaces or handle topology changes; and the surface is continuous and watertight. Being a distance function, it readily provides alignment error information that can be used in motion estimation [6].

However, building a map in terms of CSG operations seems to necessitate a means of obtaining the operations and primitives from incoming measurements, such as images or depth scans, so that they can be merged into the global CSG structure. Since that might require a way of detecting and recovering individual objects, the focus in this thesis has been on just that, and a further study on CSDF/CSG as a complete map representation has been deferred to future work.

Incorporating prior world knowledge into SLAM

Incorporating prior world knowledge has been an important research topic within SLAM, as it can better inform the task at hand, as well as improve the mapping process itself. One way to impose prior knowledge is to assume a dominant structure: i.e. the presence of affine surfaces [51], planes [23, 41] or cubes [40]; the scene being a room [70, 58]; the scene being a corridor [68]; or a combination of these [15]. A description of the scene in terms of the assumed structure is then recovered: i.e. the orientation and location of planes or cubes; the 3D room layout; or the location of walls. Once in place, this higher-level description can facilitate further 3D reasoning and also improve the tracking and mapping process: i.e. increase robustness during motion with low parallax or in scenes with scarce texture, or reduce noise by encouraging points to conform to known flat surfaces.

However, these methods are perhaps too narrow: for example, it would be interesting to extend the work of Jiang et al. [40] to identify not only cubes, but also cylinders or other primitives; or the work of Pinies et al. and Dzitsiuk et al. [51, 23] to identify piecewise-planar structures whose parts are linked in a rigid manner, thus facilitating the detection of more specific objects; or the work of Zhang et al. and Salas et al. [70, 58] to indoor layouts beyond cubical rooms. The findings in this thesis may be useful for representing these structures in a uniform way.

Extracting objects from SLAM maps

While the above is useful for coarse scene analysis, tasks involving interacting with objects or avoiding obstacles benefit from a more specific scene understanding: i.e. the presence of objects such as furniture, vehicles, houses or teacups. Another way to impose prior knowledge into SLAM has therefore been to extract known objects from the map: for example, Galvez et al. (2016) [33] and Ramadasan et al. (2015) [55] scan real-life objects and store them in a database, from which they are recovered online with SLAM.

The findings in this thesis may be useful for representing objects at a lower storage cost and less complexity, and facilitate more powerful reasoning tools after extracting an object: for example, signed distance functions have been useful in motion planning [50], as they define, for every point in space, the distance from a point to a surface, or the direction one must travel to move away from obstacles. Such information can be difficult to obtain using point clouds or meshes, as they require expensive search methods.

Modelling deformable or articulated objects

The above methods demand that the object's shape appears exactly the same as when it was scanned; a limiting assumption, if the precise shape is unknown, such as for non-rigid objects, or if it is impractical to store precise models for each instance, such as cars. This has lead to methods that allow for some degree of variability. For example, Prisacariu et al. (2013) [53] try to recover a 2D segmentation, the 3D pose and the precise shape of an object observed in a single image. To constrain the problem, they create a deformable model of objects up-front, where the space of possible shapes is obtained by statistical analysis on a set of examples. These models are stored as discrete signed distance functions. Similar models have been used by Zheng et al. (2015) [72] and Engelmann et al. (2016) [26].

However, storing a single object class with 100 shape parameters in a $256 \times 256 \times 256$ SDF at 32-bit precision requires 6.4 gigabyte of memory. Since this scales rather poorly, Prisacariu et al. [53] store the SDF in a compressed format which reduces the same data down to 6 megabyte. However, compression introduces artifacts and additional complexity, and decompression incurs a significant performance hit during parameter estimation. In contrast, continuous SDFs can describe articulated and deformable objects at the order of hundreds of bytes whilst preserving the benefits inherent to distance functions, without complicated compression schemes, artifacts or loss of precision.

Surface reconstruction and reverse engineering

Point clouds arise in industrial settings where objects of interest are mechanical parts. However, point clouds obtained in this setting often contain holes due to specularities [3], and lack a higher-level description necessary for further tasks, such as grasp planning or surface inspection. Since mechanical parts tend to be formed with geometric primitives, there has been a focus on recovering primitives from point clouds; under the name of surface reconstruction or reverse engineering [3, 63, 60, 65].

The findings in this thesis can be useful for describing the large variety of objects in this domain at a low storage cost; with the ability to describe perfectly curved surfaces; and providing useful information such as the direction and distance to the closest point on a surface from any point in space.

Modelling houses for urban reconstruction

Urban reconstruction is the problem of acquiring 3D models of cities or environments [46]. Accurate 3D models can be beneficial for content creation, navigation through cities, preserving cultural heritage sites, autonomous driving, or all sorts of civil protection or emergency management. Many methods try to solve the problem by incorporating prior knowledge about possible 3D structure: such as the shapes of houses and buildings or the repetitive and symmetric look of facades. However, the optimal representation of buildings remains an open problem: the main challenge lies in the extreme variability and complexity of urban scenes [39].

The findings in this thesis could be useful for modelling the overwhelming variety of urban structures, with an efficient way of describing repeated details or symmetries, and providing useful alignment error information for fitting the models to 3D measurements.

1.2 Report structure

I have in this introduction explained my motivation for studying this subject, and I summarize my findings at the end of this report (Section 4). To allow for an easier understanding of the results, I introduce the most important concepts in Section 2. If the results entice you enough to want to use the ideas presented in your application — or I convince you otherwise — but you question the validity of the results and want to scrutinize my experimental methods, then you should read Section 3.

2 BACKGROUND

In this chapter I describe the concept of *continuous signed distance functions*, and illustrate how they can define simple and complex scenes using *constructive solid geometry* and Euclidean transformations. Along with a short overview of *simultaneous localization and mapping*, the chapter will have presented the basic theory necessary to understand the experiments and results in this report.

2.1 Continuous signed distance functions

Definition [30]: The *distance function* $f(p) : R^3 \rightarrow R$ of a set of surface points S is defined as the distance from p to the closest point in S:

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

the surface itself being given by the *level-set* or *iso-surface* $S = \{p : f(p) = 0\}$, and the distance $|| \cdot ||$ being some metric on R^3. *Signed* distance functions (SDF) 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 $f : R^3 \rightarrow R$ 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. General implicit surfaces have a long history (see e.g. [30] and references therein), but the particular 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.

2.1.1 Distance metrics

The Euclidean distance l_2 is frequently used because of its utility in many applications: i.e. in collision detection and motion planning [50, 66], or in cost functions for shape alignment [38]. 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 nice when computing gradients. Other metrics can have advantages over the Euclidean norm [66]: for example, the max-norm l_∞ can in many cases be cheaper to compute, and in some cases much easier as well.

When referred to in the remainder of this report, unless the text specifies otherwise, 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.

2.1.2 Describing scenes with distance functions

Although distance functions have been derived for common geometric primitives, such as the sphere, cube and cylinder (see [36] and Figure 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 (Figure 3):

$$\mathit{intersection}(S_1, S_2) := \max(f_{S_1}, f_{S_2}) \tag{4}$$

$$\mathit{union}(S_1, S_2) := \min(f_{S_1}, f_{S_2}) \tag{5}$$

$$\mathit{complement}(S) := -f_S \tag{6}$$

$$\mathit{difference}(S_1, S_2) := \max(f_{S_1}, -f_{S_2}) \tag{7}$$

Although the resulting distance function is not strictly Euclidean (Figure 9), it is often a good approximation close to the surface [30]. Other implementations of the set operations have been derived (see [28, 27]); these aim to strike a balance between accurately approximating the Euclidean distance and being differentiable everywhere.

A variety of *domain* operations are also possible [56]. Shapes can be rotated and translated with rigid-body transformations by applying the inverse transformation on the input domain (Figure 4):

$$\mathit{rotate}(S, R) := f_S(R^T p) \tag{8}$$

$$\mathit{translate}(S, T) := f_S(p - T) \tag{9}$$

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 domain like so:

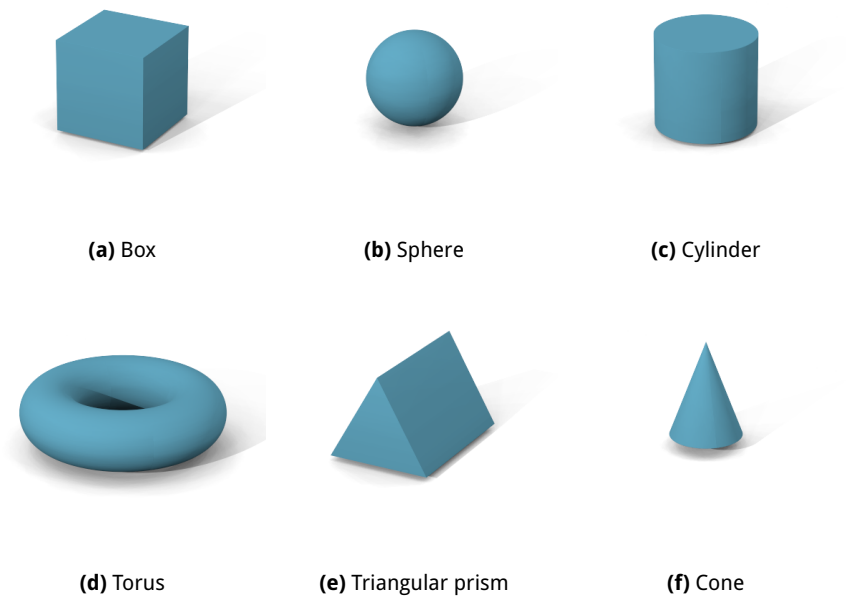
$$\mathit{scale}(S, k) := f_S(p/k) \quad \text{(incorrect)}$$

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:

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

Other deformations such as non-uniform scaling, twisting or blending cannot be repaired to recover the correct distance function (Figure 7). Scaling factors can be derived that nevertheless ensure that the distance function never overestimates [36].

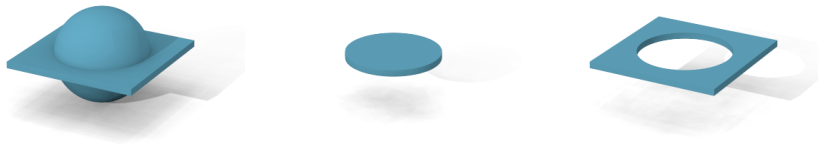
Domain mirroring and repetition can describe visually complex objects by exploiting symmetry and duplicated details (Figure 5 and Figure 6). However, the resulting distance function may no longer be continuous, in that there can be a jump at the interface between the repeated domains, which can cause distances to be overestimated (Figure 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) Box **(b)** Sphere **(c)** Cylinder

(d) Torus **(e)** Triangular prism **(f)** Cone

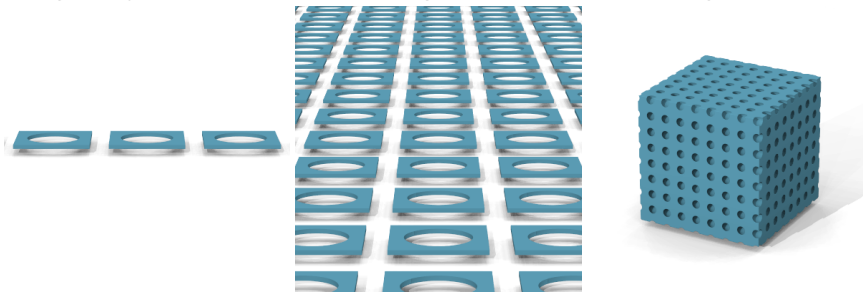
Figure 2: Basic primitives whose Euclidean distance can be computed exactly in closed-form. See the open-source `hg_sdf` library [45] for implementations of these primitives and approximations for other primitives.



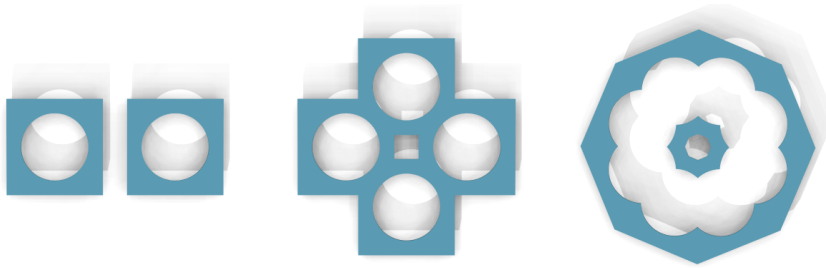
(a) Union **(b)** Intersection **(c)** Difference

Figure 3: Basic distance operations implemented with min and max functions.

(a) Rigid-body transformation **(b)** Mirroring about an axis **(c)** Mirroring about two axes



(d) Repeating in one axis **(e)** Repeating in two axes **(f)** Repeating in three axes

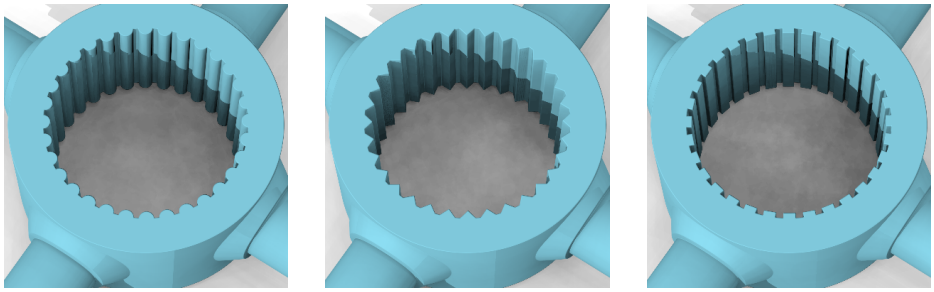


(g) Polar repetition (180°) **(h)** Polar repetition (90°) **(i)** Polar repetition (45°)

Figure 4: Basic domain operations implemented with matrix- and vector multiplication and addition on the input point (a); taking the absolute value of one or more of the input coordinates (b,c); taking the modulus of one or more of the input coordinates (d,e,f); and taking the modulus of the angle, obtained by atan, of two of the coordinates (g,h,i).

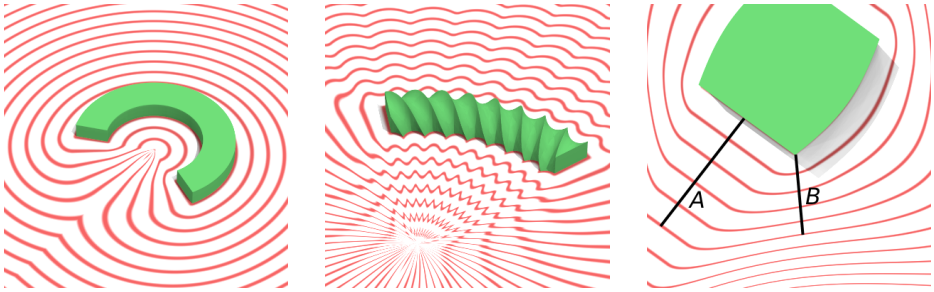
(a) Base **(b)** Notches **(c)** Flanges **(d)** Beams

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.



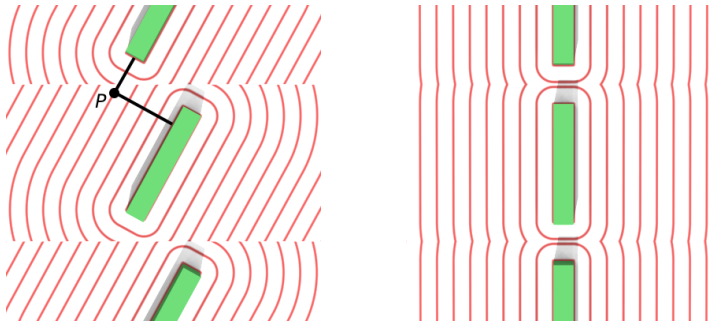
(a) Cylindrical notch **(b)** Prism notch **(c)** Cube notch

Figure 6: Changes to the notch are automatically reflected everywhere.



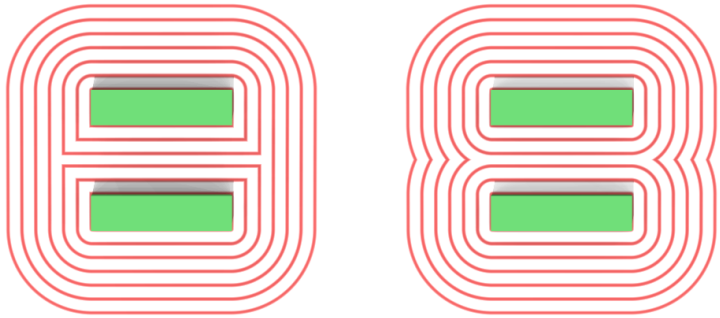
(a) Bent cube **(b)** Screwed and bent cube **(c)** Inflated cube

Figure 7: More complex shapes can be modelled with nonlinear 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).



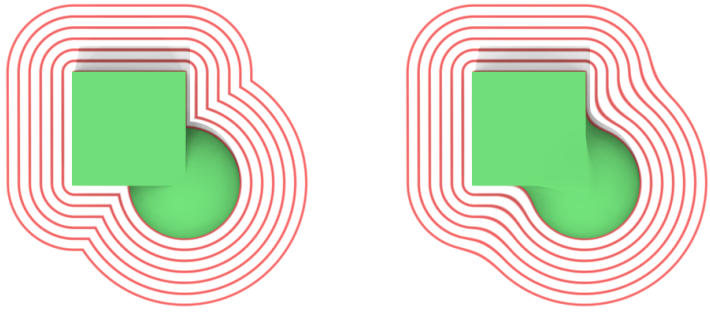
(a) Discontinuous interface **(b)** Continuous interface

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 Euclidean distance **(b)** Correct Euclidean 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` [45]. See also [27].

2.1.3 Converting between representations

It is sometimes necessary to convert one surface representation to another. Meshes and point clouds can be generated from signed distance functions (and a broader class of implicit functions) using iso-surface extraction methods, such as *marching cubes* or *surface nets*, or by point-sampling the domain and relaxing points onto the surface along gradient directions [30].

Although these representations could also be used to visualize the surface, *sphere tracing* is an alternative that directly renders the surface by intersecting pixel rays with the surface [36]. By exploiting the distance function properties, it can efficiently traverse the space in step sizes that are guaranteed to never overstep the surface, thus greatly increasing efficiency. Sphere tracing also enables point clouds to be extracted by sampling the surface from one or more viewpoints. This is done in [67], where they sphere trace a discretized SDF volume from three orthogonal directions, thus obtaining the point cloud that is visible in each view.

Finally, while *discrete* SDFs can be generated from meshes and point clouds using distance transform or level-set methods [30], generating a continuous SDF in the form of primitives and CSG operations is an open problem that we will get back to in Section 3, when we discuss the applicability of CSDFs for modelling.

2.1.4 Other considerations

Texture mapping. As has been done in the figures above, the surface can be procedurally colored, texture-mapped or shaded with computer graphics techniques by defining a mapping from 3D points to color [56]. This requires parametrizing 3D space onto the surface, which, unfortunately, can be very difficult [48, chapter 7].

Signed or unsigned. By removing the sign, we lose the ability to discern free space from occluded space, a property which is used in some CSG operations (such as difference), and which may otherwise be useful in other domains (such as motion planning [50]). Signed distance functions also remain monotonic across the surface interface, not having a kink like their unsigned counterpart. A kink could negatively affect the gradient computation [56], which is used in surface shading and gradient-based optimization algorithms. Removing the sign also makes it harder to extract a surface mesh [52].

Smoothness. Differentiability of the distance function is not preserved under all operators, such as min and max. This can have a negative effect when used in algorithms that compute the gradient, as it is not defined at junction points. A mitigation is to use smooth blending operators (Figure 10) or smooth primitives (such as spheres and rounded cylinders or cubes).

Computational cost. Complex distance functions can be expensive to evaluate. *Bounding volume hierarchies* [56] can mitigate this issue by substituting expensive geometry with a simpler distance bound, the result of which is that points far away from the surface can evaluate the cheap bound, while points near the surface evaluate the exact distance.

2.2 Simultaneous Localization and Mapping

Simultaneous Localization and Mapping (SLAM) is the problem of building a model of the environment (the map), and the estimation of the pose of the sensor(s) moving within it (see [7] and references therein to learn about SLAM). It has been done with many sensors, some notable ones being scanning lasers, depth cameras or digital color cameras; some recent ones being event-based cameras or light-field cameras. When cameras are involved, and high-quality mapping is of greater importance than the camera poses, the problem is called *Structure from Motion* (SfM) or *Multiview Stereo* (MVS) [32]. If localization is of greater importance than mapping, the problem is called *Visual Odometry* [59].

RGBD sensors (such as the *Microsoft Kinect*) measure color and depth, and are often used when dense maps are wanted, that is, surfaces without holes everywhere [19]. Sometimes RGB cameras are used, but dense maps are wanted anyway: recent work achieves this in real-time on a CPU [35] using clever segmentation and smoothing tricks. Of particular interest in this report, are SLAM methods that represent the map as a sparse point cloud: that is, as a set of independent points. In this chapter we will briefly define the output of such methods, and also define *SLAM as an optimization problem*. These definitions will be used when we discuss object recovery methods and post-processing.

2.2.1 Point cloud SLAM

Definition. The output of a *point cloud SLAM* method is atleast one image, and depth values or 3D coordinates associated with a subset of the pixels in atleast one of the images. This definition is compatible with recent sparse methods [25, 24], and incompatible with dense methods that use meshes, occupancy grids or discrete SDFs [19]. The availability of pixel depths is considered to be equivalent to the availability of 3D coordinates, since 3D points can be obtained by inverse projecting the pixels. The collection of 3D coordinates will be referred to as *the point cloud*: mathematically we will write this as the set

$$P = \{p \in R^3\}. \tag{11}$$

2.2.2 SLAM as an optimization problem

Iterative optimization has become a tool of choice for recent SLAM methods [7]. These methods recover 3D structure by defining a cost function based on *reprojection error*: that is, given an image formation model and an estimate of the map and camera poses, the reprojection error is the discrepency between the measured data and the predicted data. *Direct* methods define this error on the level of image pixel intensities, while *indirect* methods define a geometric error between point correspondences [24]. The best estimates of the map and the camera poses are obtained, after an appropriate initialization, by minimizing reprojection error jointly with respect to a parametrization of the camera poses and the map, a problem referred to as *bundle adjustment* [7] or *reprojection error minimization*. Aside from reprojection error, the cost function can be augmented with additional terms, such as to encourage spatial smoothness of the map or the camera trajectory, or to encourage points to conform to a surface [23, 7].

3 RESULTS

This chapter presents my attempts at answering the research questions posed in the introduction, the methods I used to obtain them, and what the precise goals of the studies were. First, the use of continuous signed distance functions for modelling objects is studied through a literature survey on available tools, and a qualitative experiment regarding the expressive power. Then, the use of CSDFs for recovering objects from a 3D reconstruction of a scene is studied through a survey on detection and refinement methods, and thereafter a qualitative experiment of an ICP-like refinement method. Finally, the benefits and drawbacks of CSDFs for post-processing is studied through a survey and discussion.

3.1 Ability to model real-life objects

The following aspects were considered:

Modelling tools: What tools are available for creating CSDF models?

Expressive power: What objects and level of detail is suited for CSDF modelling?

This section presents the methods used to answer these questions, followed by a survey of modelling tools, and a qualitative analysis of the expressive power.

3.1.1 Experiment setup

To assess the state of tools I searched the internet for queries including *signed distance function*, *sdf*, *tool*, *modelling*, *sculpting*, *edit*, *sphere tracing*, *raymarching*, along with terms related to CSG and CAD. I also searched Google Scholar for papers that cite the sphere-tracing paper of John C. Hart [36], which was an influential work that popularized SDFs in the demoscene community.

To assess the expressive power I looked at objects and judged to what degree I could model them within a reasonable timespan. The following classes of objects were considered:

Household objects: Objects were sampled from the IKEA furniture catalogue and the YCB dataset for robotic manipulation benchmarks (2015) [8].

Outdoor objects: Houses, vehicles and misc. objects were sampled from city photographs and the large dataset of object scans (2016) [13].

Mechanical parts: Objects were sampled from papers about textureless object recognition and the T-LESS RGB-D dataset [37].

The following is a qualitative analysis, as lengths and angles were not accurately modelled, nor was the model approximation error evaluated. The modelling process consisted of writing the distance function by hand, using photos for reference and the `hg_sdf` library [45] for CSDF primitives and operators, and visualizing the scene with a custom GUI.

3.1.2 Modelling tools

Text-editor. The most basic tool consists of a text-editor, in which code for the distance function is written by hand, and a GUI that visualizes the scene. Examples of such tools can be found on the ShaderToy website[3], where the user can implement a distance function in a C-like language and see the resulting scene rendered with realistic shading and lighting.

Graphical editor. Writing distance functions by hand can be time-consuming. Some tools therefore try to mimic the interface of traditional modelling software. Reiner et al. (2011) [56] 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 the primitive properties (i.e. a sphere's radius) with sliders; as well as the basic distance and space operators. Code for the resulting distance function is automatically generated and exportable. A similar tool can be found online[4].

3D scanning. An alternative to manual modelling is to use 3D scanning, which has contributed to large object databases [13] 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 in this direction has been done by Andrews et al. (2013) [1], who semi-automatically reverse-engineer meshes into a CSG tree; and Fayolle et al. (2016) [28], 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 are partially compatible with CAD software, since they share the principle of combining primitives with CSG. These tools usually represent a model in terms of a tree of CSG operations and primitives, which could possibly be exported to generate a CSDF expression. To prevent the CSDF expression from getting too complicated, one could possibly remove details until the surface approximation error surpasses some specified threshold. However, I have not found any existing methods for doing this, nor do all the shapes or operations employed in CAD software have a simple distance function: for example, some CAD models mix parametric surfaces with CSG primitives.

3.1.3 Expressive power

Furniture, mechanical parts and houses. Objects whose design consists of boolean operations on a few primitives were modelled in a few minutes to a seemingly precise level of detail. IKEA furniture was particularly suited (Figure 11), and sometimes came in variations that were parametrizable (Figure 12). Most of the mechanical parts were composed of a few primitives, and were likewise straightforward to model (Figure 13). Some of the parts consisted of ten or more primitives, and would be straightforward, but time-consuming, to model (Figure 14). The mechanical parts tended to be symmetric or contain repeated details, which was modelled with repetition operators. Residential housing were modelled with less than ten primitives, mostly rotated cubes and planes, and benefitted from mirroring and repetition operators (Figure 16).

[3] `https://www.shadertoy.com/view/Xds3zN` (Retrieved 2017)

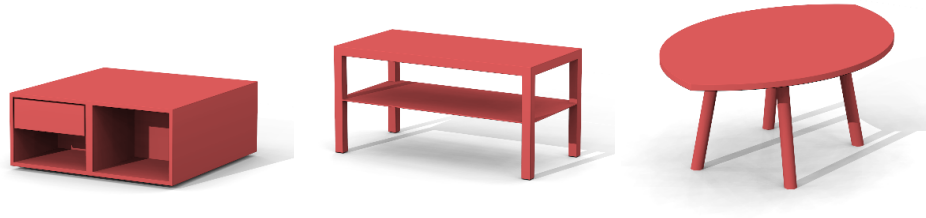
[4] `https://stephaneginier.com/archive/editSDF` (Retrieved 2016)

Cars and motorcycles. Objects that consist of parametric surfaces, such as NURBS, much used in car or 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 [64, 22]. Thus, although a precise model could be made with such algorithms, doing so can make the distance function expensive to evaluate. If a coarser model is acceptable, a car chassis could possibly be approximated with cubes and cylinders, and either a linear or rounded union operation (Figure 15).

(a) IKEA BOKSEL

(b) IKEA LACK

(c) IKEA STOCKHOLM



(d) A small cube, mirrored and translated to form two cubes, is subtracted from a larger cube; a small cube forms the tiny drawer.

(e) Two thin cubes form the top and bottom tables; a tall cube, mirrored and translated, forms the legs.

(f) One thin cylinder, mirrored and translated, forms the top table; one rotated and capped cylinder, mirrored and translated, forms the legs.

Figure 11: Table samples from IKEA / Stue (2016)

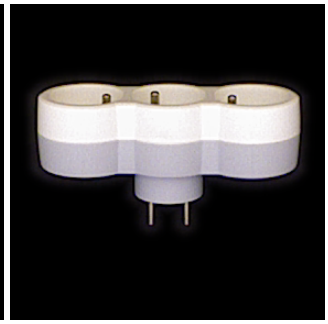
(a) Adjusting width

(b) Adjusting height

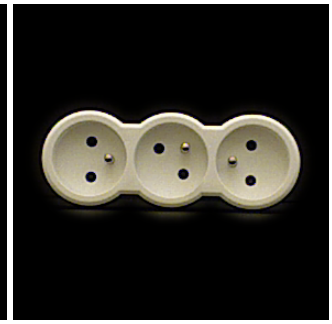
(c) Adjusting bottom table offset

Figure 12: Variations of the IKEA LACK generated by adjusting parameters.

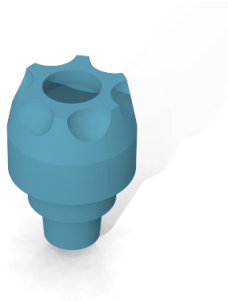
(a) T-LESS Object 02



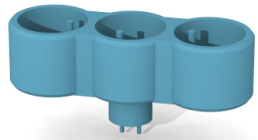
(b) T-LESS Object 23



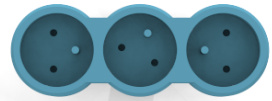
(c) T-LESS Object 23



(d) Cylinders and capped cones form the body; a thin cylinder with a spherical butt is subtracted from the top; repeated spheres subtracted around the top form the grooves.



(e) A hollow cylinder is repeated to form the sockets; Thin cylinders underneath form the plug; a box forms the flat junctions.

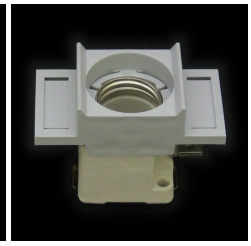


(f) The quotient from the domain repetition can be used to add instance-specific details, such as the rotation of the socket.

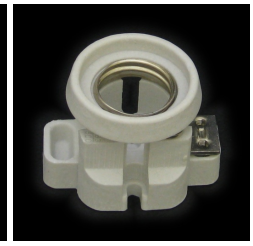
Figure 13: Mechanical part samples from T-LESS

(a) T-LESS Object 08

(b) T-LESS Object 09



(c) T-LESS Object 10



(d) T-LESS Object 12

Figure 14: Complicated mechanical parts

(a) A car formed by taking a rounded union between two cubes, and forming the wheels with cylinders.

(b) The same car with a linear ramp union.

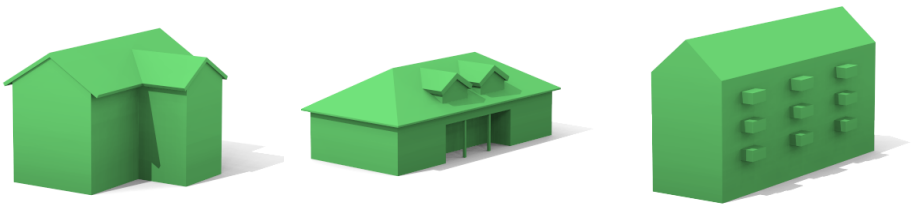
(c) A van formed by adjusting the length of the upper cube.

Figure 15: Cars

(a) HOF

(b) HVALSTAD

(c) A block in Norway



(d) A traditional house formed by intersecting a cube with two rotated planes; and duplicating and scaling that shape to form the smaller part.

(e) A villa formed by intersecting a cube with four rotated planes; reusing the base shape from (a) to form the top windows; and subtracting a cube to form the entrance.

(f) An apartment block formed by reusing the base shape from (a); adjusting its parameters to stretch it; and repeating cubes to form the balconies.

Figure 16: Residential housing. Photos (a) and (b) were sampled from BLINK HUS.

3.2 Compatibility with detection methods

I assume object recovery occurs in two stages: detection and refinement. The refinement step assumes an initial guess for the object is available, but that it might be inaccurate or wrong, so its aim is to validate the guess and produce higher accuracy estimates. Thus, the usefulness of CSDFs for object recovery depends on their usefulness in these two areas.

Deferring the refinement problem to the next section, the problem considered in this section is: Given a 3D reconstruction (images and point depths) and one or more objects modelled by CSDF expressions, determine the presence of a particular object in the reconstruction, and provide a coarse estimate of its parameters (atleast pose, but optionally also scale and shape). To study the applicability of the CSDF representation for this problem, I did a survey of detection methods, and point out how the representation fits into such methods.

The survey was done by searching for recent object detection papers that contain overviews of this large topic; searching for object-dataset papers and in particular how they obtain ground truths; searching for papers on point cloud segmentation, surface reconstruction, reverse engineering, and recovering primitives from point clouds; as well as papers related to SLAM with TSDFs, since the problem of estimating camera motion between a TSDF and a point cloud (or another TSDF) is similar to estimating the pose of an object.

3.2.1 Survey of object detection strategies

Detect constituent primitives

Extracting primitives from point clouds is an important research topic for surface reconstruction [3] and reverse engineering [28, 65]. Schnabel et al. (2007) [60] and Toony et al. (2015) [65] aim to extract volumetric primitives from point clouds, also when the primitives are only partially present (such as for objects that are combinations of primitives). 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 the existence of a particular object and its pose.

Match volumetric features from SDF

Aside from keypoint descriptors for point clouds, similar descriptors have been derived for matching two SDF volumes directly [11], such as generalized Harris corners, Shi-Tomasi descriptors and integral invariants. If the map were a SDF, i.e. obtained by computing the distance transform on a point cloud, and the object is a CSDF, then object features could possibly be extracted and matched against the map.

Match point cloud features from sampled surface

A vast body of literature can be found on the registration of two point clouds [38, 21] that we can take advantage of by sampling the CSDF surfaces to generate point clouds. These point clouds could then be registered against the map, using methods such as found in the Point Cloud Library [38]. These methods typically work by obtaining a coarse alignment by matching 3D keypoints, and then refining the alignment with ICP. For sparse point

clouds, such as obtained from sparse SLAM methods, or self-similar objects, such as cylinders, keypoint matching may be difficult; see Cieslewski et al. (2016) [14] and Drost et al. [21] (2010) for a discussion.

Match image features from texture map

A CSDF object can be augmented with texture mapping to model its image appearance, similar to textured meshes or point clouds. Once a texture is in place, with an associated mapping from 3D object coordinates to texture coordinates, 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 [33, section 5.5]. This also enables the use of bag-of-words models, that have been useful in identifying a particular objects from a database of candidates [33, section 5].

Use CSDFs only for refinement

For the above methods, it was assumed that the user had modelled objects in a CSDF representation with the hope of using solely these models for the entire recovery task. However, since these methods may be difficult to use, one could instead use a seperate model for the detection step, and only use the CSDF models 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. It is perhaps in these pipelines that CSDFs fit in most naturally, as a replacement of the precise 3D model. The amount of literature in this domain is large, but for an overview you may consult the following papers that use a coarse-to-fine pipeline: [12, 20, 29] for monocular images, [26] for stereo images, and [54, 62, 69] for recovering objects from depth with (maybe) color images.

3.3 Compatibility with refinement methods

Leading on the results from the previous section, the problem considered in this section is: Given a 3D reconstruction (images and point depths), an initial pose and (optionally) scale and shape parameters, find the model parameters that best explains the data according to some quality metric. To study the applicability of the CSDF representation to this problem I did a brief survey: looking at the literature found in the previous section and in particular those that have detection and refinement stages.

3.3.1 Strategies for precise parameter estimation

Cloud-cloud registration

Estimating a rigid-body transformation between two point clouds (also known as point cloud registration) is a well studied problem (see [38] for an overview). The strategy used in the Point Cloud Library (PCL) is to obtain an initial alignment by establishing 3D keypoint correspondences, and then refine the alignment using some variation of the Iterative Closest Point (ICP) algorithm. 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) [6, 10] represent a scene reconstructed from RGBD measurements as a TSDF; they 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 particular cost function comes from a probabilistic point of view: assuming the measurement noise in the depth camera is normally distributed and all pixels are independent and identically distributed, the likelihood of observing the measured point cloud, given a camera motion estimate, can be written as the product of normal distributions, and through seeking the maximum likelihood and taking the negative log-likelihood we get the sum of squared distances (see [6] for details).

This defines an optimization problem, which can be (attempted) solved in many ways[5]. The authors of [6] decided to use the Gauss-Newton method [49], which is an iterative algorithm that requires a sufficiently good starting guess. Other algorithms try to efficiently search the parameter space to obtain a solution no matter where it starts. The Gauss-Newton method requires the gradient of f at each point; but there are also algorithms that don't.

[5] Optimization is used in many domains, such as machine learning, robotics and economics. It can be difficult to get right, so here are some thousand pages about the subject [5, 49].

The concept behind their approach — measuring solution quality by analyzing the distance function evaluated at (transformed) point cloud points — is one that fits with CSDFs: the difference being that CSDFs evaluate the distances directly instead of trilinearly interpolating values in a grid. The approach also naturally begs to be extended: aside from just rotation and translation, we could try to recover scale and precise shape parameters by solving the following minimization problem:

$$\arg \min_{R,T,k,c_1,\ldots,c_n} \sum_p f\left(Rpk + T; c_1, \ldots, c_n\right)^2 \tag{13}$$

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

The above formulation is similar to ICP methods for point cloud registration: a difference being that ICP first needs to pair points before calculating the distance between them; another being that SDFs directly compute point-to-surface distances, whereas point pairs only gives point-to-point distances: variants of ICP, deemed as more robust, have been developed that do extra work to obtain point-to-plane or point-to-edge distances [38].

SDF-RGBD registration

The above method uses only 3D geometry, which, as noted by Slavcheva et al. (2016) [61], can be unreliable when the reconstruction is sparse. Dame et al. (2013) [20] also use an optimization approach, but include image appearance in their cost function, to determine precise pose, scale and shape parameters of a deformable SDF volume. Their cost function combines a term encouraging 3D map points to lie close to the SDF surface, and a term that maximizes the discrimination between a statistical background/foreground image model. Such a method could be readily applicable, by using sphere tracing to render the surface.

SDF-RGB registration

Some methods try to recover objects from a single image without depth information. Particularly, implicit representations have been used in model-based image segmentation [16], where the aim is to find the optimal parameters of the model to satisfy some statistical background/foreground discrimination. Marchand et al. (2016) [44] survey methods for object pose estimation from single images, among them are optimization methods that include in their cost function a term that penalizes distances between the 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) [61] use TSDF maps in dense SLAM and estimate frame-to-frame camera motion by generating a TSDF for the incoming RGB-D frame, and then aligning both SDFs with each other. Again, they use an optimization approach, where the cost function includes the distance function discrepency summed over the common domain and a term that encourages normals to be identical. This strategy could be extended to CSDFs, substituting their notion of a *scene* with an *object*.

3.4 Assessing CSDFs for ICP-like refinement

The survey in the previous section seemed to indicate that iterative closest point methods, or in general methods based on minimizing a sum of point-to-surface distances, is the dominant approach for precise parameter estimation. Motivated by this I investigated how CSDFs apply in an ICP-like refinement method. This resulted in an experiment where I implemented a method similar to [6], but extended to recover scale as well. I defer the recovery of shape variation parameters and the use of image data for alignment, to future work. My goals were to study the following aspects:

Implementation complexity and scalability

The difficulty in designing and maintaining the implementation is affected by the object representation: for example, meshes must be loaded and parsed into an appropriate structure for the algorithm, and will need additional code to extract information, such as searching through volume acceleration structures to obtain distances. How can CSDF objects be represented in a programming language, and how does it support the operations we need? How does it scale when we deal with many objects or very dense point clouds?

The remaining questions are specific to my personal implementation of the particular alignment method, for which the results may cast a light on the potential problems that arise when implementing a correspondence-free, shape-based alignment method.

Effect of object detail

In modelling a house there is obviously a choice that must be made as to how much detail is enough. On the coarsest level one might describe the entire thing as a cube; or if that's too crude: a cube and a prism; then one could include the slight indentation that is made as the roof meets the walls; going even further one could include the planks on the walls, the subtle chamfering around the windowsills, the knob on the front door, and so forth. How is the alignment affected by more or less model detail?

Effect of point cloud density

Point cloud density can vary based on the SLAM method (dense or sparse) and the object texture itself. I study the effect of density by attempting to recover objects of different texture, from a single flat color to a checkerboard.

Effect of clutter

Other unmodelled objects might surround the object of interest; what effect does this have, and how can it be mitigated?

Effect of (not) knowing scale

Monocular SLAM methods do not provide the map in real scale, which leads to an additional degree of freedom: the scale of the object relative to the map. If the scale were known, for example with stereo or depth sensors, the alignment might be easier. I study the effect of scale ambiguity by performing the alignment with and without scale freedom; obtaining the correct scale by hand when used.

3.4.1 Experiment setup

The ability to recover objects from a point cloud depends on what the point cloud looks like, and in particular, how it was obtained. I implemented a sparse direct SLAM method based on recent work [25, 24]. This allowed me to obtain point clouds that have similar noise and density properties as that of a modern sparse SLAM system.

I formulated the registration problem as: Given a (sub)set of points $p \in P$, a signed distance function $f(p) : R^3 \to R$, find the rigid-body transformation R,T and scale k such that the sum of squared distances between p and the surface,

$$E(R,T,k) = \sum_{p \in P} \left(\frac{f(Rpk + T)}{k} \right)^2 \tag{14}$$

is minimized. Note the division by k to ensure that distances are measured in scene coordinates instead of object coordinates. I solved the optimization problem using *Levenberg-Marquardt*, a gradient-based method similar to that in [6]. I found that I needed a good outlier rejection scheme and chose the *Tukey M-estimator* (see [71]). The gradient of f was computed by finite differences, but I could also have used *automatic differentiation* or derived analytic derivatives.

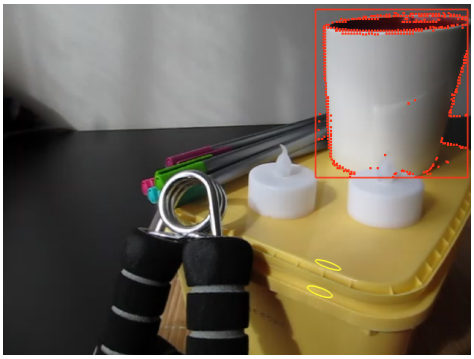
Some objects have degenerate degrees of freedom: a cylinder can spin around its axis of symmetry without affecting the cost. For these shapes we could parametrize the problem with fewer variables, but I decided to simply add a damping term to the redundant variables, effectively locking them in place and incurring a cost if they move.

The alignment algorithm was implemented in C++, representing objects as function pointers: the common interface being that an object takes an object-space coordinate and returns an object-space distance. This means that the alignment can be agnostic to the specific object that is being recovered.

I reconstructed real and synthetic scenes (Figure 18) and ran the alignment algorithm on objects that were modelled by hand (Figure 19). I initialized the pose and scale by selecting a bounding box in the image, computing the translation as the centroid of the contained 3D coordinates, and manually rotating and scaling the object close enough (see Figure 17). I also set the point cloud subset P equal to the points within the selected bounding box.

(a) **(b)** **(c)**

Figure 17: The alignment procedure consists of three steps: (a) Manually draw a 2D bounding box to select the point cloud subset and initialize the object translation, (b) Manually adjust scale and rotation, and finally (c) Run some iterations of ICP.



(a) Cup

(b) Candle

(c) Building

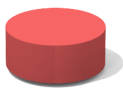
(d) Untextured box

(e) Textured box

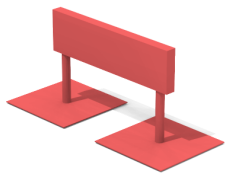
(f) Textured box corrupted by monkey head

Figure 18: Scenes used in the recovery experiment with point subset highlighted.

(a) Cup



(b) Candle



(c) Building

(d) Box

Figure 19: Models used in the recovery experiments

3.4.2 Experiment results

The following are my qualitative observations regarding the implementation and performance of the alignment algorithm.

Implementation complexity

I found that I could define CSDF objects under a *uniform* interface as functions that take a triplet of floating-point coordinates and return a floating-point distance. The ICP alignment algorithm can then be used with any object by taking a function pointer that adheres to the (mathematical) interface $f : R^3 \to R$. Alternatively, instead of C function pointers, one can achieve the same with *object oriented programming*, by letting specific objects be derived classes of base classes and making the alignment algorithm be a polymorphic procedure.

This interface supports the rigid-body transformation and scaling operation needed when evaluating the cost function, since they are domain operations that transform the input coordinate: in other words, the algorithms simply transforms each input point before invoking the distance function, and scales the resulting distance appropriately.

Scalability

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

$$\sqrt{x^2 + y^2 + z^2} - r$$

can be written with three `add`s, three `mul`s, and one `call` (square root). The instruction count for some more functions is shown in the table below. Besides the trivial sphere, the visually complex cross shaft (Figure 5) consisted of eight `call`s (six cylinders and two polar repetitions), five `min,max`s and eight `add`s. If each instruction is 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`	`min,max,abs,mod`
`Sphere`	3	3	0	1	0
`Cylinder`	3	2	0	1	2
`Box`	3	3	0	1	14
`RepeatPolar`	4	5	0	4	1
`CrossShaft`	8	0	0	9	5

Table 4: Rough instruction count for some primitives and operators

In their object recovery method [43], Lu Ma et al. represent deformable objects as discrete SDF grids, at a resolution of 512^3 cells, where each cell contains one 32-bit floating point number. A straightforward implementation (without compression) would then require

0.5 gigabyte per object. The object deformations are parametrized with principal components obtained from examples, with each component corresponding to one grid. Storing, for example, the five most important components would then require 2.5 gigabyte. In contrast, the IKEA table with adjustable parts (Figure 12) or the deformable car (Figure 15) can be stored in about 100 bytes each, although the type of variations is limited to what can be controlled by adjusting primitive and combination parameters.

Of course, size is not everything. The cost of evaluating the distance of complex objects can quickly become unwieldy, and one might be better off discretizing the CSDF. The performance difference between evaluating a discretized SDF and a continuous one depends on the relative latency of memory access versus compute instructions on the particular machine, the access pattern (i.e. number of cache misses), and the compression strategy used to store the SDF (i.e. if it is always stored in a compressed format).

Effect of point cloud density

In Figure 18d the cube was untextured, and the reconstruction only captured points on its edges. This made it difficult to align the object without landing on a false local minimum. The reason being that the number of inliers was too low, causing the outlier rejection scheme to be overly optimistic and allow for far-away outliers to be members. In contrast, figure Figure 18e shows a checkerboard textured cube, for which the alignment was more successful. The real life scenes were comparatively sparse, and the initialization had to be fairly tight (roughly within 5% of object dimension in translation, and within ± 5 degrees in rotation) to ensure correct alignment.

Effect of object detail

I found that alignment tended to be more successful when including the ground plane: the box top underneath the candle and the cup, and the ground below the pillars.

Effect of clutter

Clutter, such as the monkey head in Figure 18f, is the same as having outliers in the initial point subset. I found that the method still worked as long as the initial estimate was sufficiently close to the correct geometry, thus the outlier threshold was set sufficiently strict to reject the clutter. If, however, the number of outliers are comparatively many, the alignment may erroneously include the clutter.

Effect of (not) knowing scale

When the scale is unknown the alignment had a tendency to expand infinitely, when there was nothing preventing it from doing otherwise. In Figure 18e, the point cloud only covers half of the object surface causing the scale to be ill-constrained. Increasing the scale will still keep all the member points on the surface at the same distance, thus neither increasing nor decreasing the cost. However, due slight numerical imprecision in calculating derivatives and such, the cost alternatingly increases and decreases by a small amount by increasing the scale, causing jitter or inflation.

3.5 Use in post-processing

Once an object or a high-level description of the scene is recovered, it becomes possible to exploit additional prior knowledge to improve the mapping process or better inform the task at hand. The following is a survey of some relevant applications and what benefits or drawbacks continuous distance functions have over other scene representations (thus enabling a discussion of the applications of CSDFs in post-processing).

The survey was conducted by searching for papers citing Hart (1996) [36], often these are computer graphics papers; the papers of Curless and Levoy (1996) [17] and Frisken et al. (2000) [31], often cited by computer vision and robotics papers (dense reconstruction in particular); as well as the papers and included references of literature on SLAM and 3D reconstruction, and particularly, SLAM using high-level objects or distance functions.

3.5.1 Improving the mapping process

Noise reduction. In a survey on surface reconstruction from point clouds Berger et al. (2014) [2] review methods for imposing prior scene knowledge to reduce noise and fill holes. Among these is the detection and registration of primitives to the point cloud, and registration of general rigid- or deformable 3D objects. Once such an object has been recovered, a common way to incorporate its shape into the reconstruction is to add a *regularizer* to the reprojection error minimization problem [55, 33, 15]: i.e. add a term that penalizes discrepencies between the reconstruction and that of the object. Another method is to directly replace parts of the reconstruction with that of the model [23, 20]: i.e. fit objects (planes and deformable cars) to a TSDF reconstruction, and replace the TSDF values with that of the distance to the object.

CSDF models could in this case provide performance benefits, in that the distance to the object surface can be obtained more efficiently than for meshes ([55]) or point clouds ([33]), and is directly applicable to SDF fusion methods ([20, 23]). It could also reduce storage cost, in that a large variety of shapes can be modelled and variations explicitly parametrized, avoiding the scaling issues associated with detailed deformable models ([20]). On the other hand, in some cases the object is quite detailed, which requires an accurate model. Since modelling complex shapes can be impractical (both in cost of evaluating the SDF and by means of acquiring it), CSDFs may be unsuited for use cases like [33] and [55].

Tracking stability. In addition to reducing reconstruction noise, objects can help constrain the camera pose estimation as well [55, 33], which can be useful in adverse conditions, such as during motions with low parallax or in textureless areas [51]. Again, CSDFs can provide benefits to methods that use optimization, in that it can provide a point-to-surface distance error directly. On one extreme, object pose estimation can replace the localization part of a SLAM pipeline entirely, but such methods tend to rely on an image appearance model [44], which is difficult to combine with CSDFs (due to the need to parametrize 3D space onto the object surface).

Indoor occlusion culling. Indoor environments pose a problem for SLAM methods because significant occlusion can occur, i.e. when turning a corner, which can prevent underlying algorithms from determining point correspondences or computing accurate photoconsistency [32, 58]. Salas et al. (2015) [58] (and references therein) take advantage of a description of the indoor room layout, to determine which points in the map are currently visible, and can therefore avoid trying to establish point correspondences where there isn't any. As noted in their conclusion, they have only tested their approach in a simple setting of a single cubical room. CSDF may be useful for describing a larger variety of room shapes to a greater detail, and could enable the incorporation of simple approximations of furniture at a lower cost than that of triangle meshes or point clouds. However, the algorithms for estimating room layouts have hitherto not used a generic object-registration method, but instead identify various global and local structures (such as dominant perpendicular planes that make up the walls).

3.5.2 Informing the task at hand

Finally, we take a look at some applications where the CSDF representation may benefit the *task at hand*, that is, the task that the *user* of the SLAM system wishes to perform.

Scene understanding. Knowledge of where objects of interest are is greatly useful in many applications: in motion planning, calculating optimal grasps, or recording extreme sports. In many of these applications, a simple approximation of the shape is sufficient: such as approximating collision geometry with spheres and cubes [50] or approximating a graspable object with a cylinder [7]. A CSDF representation can allow for the description of simple objects with explicit shape parametrization, and can provide an easy way to discern objects from each other by looking at the constituent primitives and their parameters.

Motion planning. Motion planning algorithms often rely on being able to determine whether a point is inside or outside the collision geometry, that is, perform collision detection [50]. A CSDF representation provides a fast look-up of the occupancy with regard to simple or complex shapes, which can facilitate algorithms like A* or RRT. Moreover, optimization-based algorithms can use the gradient information, readily available, to "push" trajectories out of collision [50]. Again, however, the cost of increasing object complexity might make continuous SDFs inferior to discretized SDFs, the difference of which depends on the relative cost of compute (evaluating the distance) versus memory access (interpolating grid values). This could be mitigated with bounding volume hierarchies, at the expense of losing an accurate distance function globally (and instead only having accurate distances near surfaces).

4 CONCLUSION

4.1 Discussion of results

4.1.1 Ability to model real-life objects

Modelling tools

We have seen that, by virtue of being compatible constructive solid geometry and supporting a variety of transformations, continuous signed distance functions are compatible with a highly expressive modelling language similar to that of CAD software. With even very rudimentary tools, such as a text-editor and a visualization window, both simple and complex scenes can quickly be defined. Although currently available tools for creating CSDF expressions directly are limited to unreleased research projects and online demos, it could be possible to convert the CSG tree obtained from CAD software to a CSDF expression, which would enable the use of very sophisticated modelling tools.

On the other hand, although CSDFs can be converted to meshes or point clouds by surface extraction methods, thus being compatible with such pipelines, going the other way is still an ongoing research endeavour, which prevents the use of large object databases that already exist for meshes and point clouds, or the use of 3D scanning to obtain models from real life. This can make it difficult to acquire large sets of objects.

Expressive power

We have seen that objects whose design consists of combinations of a few primitives — such as mechanical parts, IKEA furniture or residential housing — can be modelled precisely with CSDFs. Moreover, unlike meshes, CSDFs can describe smooth surfaces without tessellation, as well as exploit domain repetition or mirroring to describe “template” - like details that can be changed in one place and reflected everywhere, thus greatly reducing storage cost and modelling time. Additionally, shape variations can be explicitly controlled with the parameters defining the primitives and the operations combining them, avoiding expensive surface updates or the need to store large data structures. This property can be of great use in tasks where the variety of shapes is tremendous, such as urban reconstruction, or recovering the shape and location of cars.

On the other hand, objects that consist of tens or hundreds of primitives will be unsuited, as the cost of evaluating their distance grows with complexity. Although this could be mitigated with bounding volume hierarchies, this comes at the cost of implementation complexity and the loss of a precise distance field everywhere. Also, objects that are designed with curves, such as NURBS surfaces, may be unsuited for precise modelling, as evaluating the distance to such surface require expensive iterative root-finding or subdivision methods.

4.1.2 Use in object recovery

Detection

Being an implicit representation, CSDFs are incompatible with correspondence-based registration methods, which are of great use in detecting and acquiring the coarse parameters of objects from images or point clouds. Although one could convert a CSDF to a mesh or point cloud, the difficulty in modelling textured image appearance limits their use to shape-based registration, or image-registration for simple-colored objects. Within surface reconstruction and reverse engineering literature we can find methods that try to detect (partial) primitives from point clouds. Such methods could be used to detect objects represented as CSDFs, by detecting their constituent primitives. These methods do, however, seem to require a sufficiently dense point cloud.

If a CSDF representation alone is insufficient for detection, it is possible to use a seperate representation for the detection step and reserve the CSDF representation for fine registration and post-processing. For example, we have seen recent object detection methods that use machine learning to build a 2D or 2D+3D appearance model, and provide a coarse 2D or 3D bounding box for objects.

Fine registration

We have seen that objects modelled with CSDFs are compatible with ICP-like point cloud registration methods, and can be more efficient than mesh-to-cloud or cloud-to-cloud registration where correspondences must be explicitly formed. Moreover, with explicit shape parametrization, CSDFs can be of great use in non-rigid or articulated object registration. However, again, the difficulty in modelling image appearance can make fine registration difficult, especially in sparse point clouds.

The objects I used in my experiments were relatively simple and their distance functions were (mostly) exact Euclidean distances. I did not investigate the effect of overestimation — such as caused by the min/max set operations, non-isometric deformations and non-symmetric repetition — when the resulting function is used as an alignment error. For example, Figure 8 shows how non-symmetric repetition can cause distances to be overestimated across domain interfaces, which could cause points to have higher alignment error than they actually do. These inaccuracies appear to be small when close to the surface, so one could possibly substitute an object with a proper bound when far away so as to limit the approximation error. One could also improve the distance estimate with the first order distance approximation

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

which can be used to estimate the distance to a generic implicit surface [28, section 5.3]. Neither did I investigate the effect of non-differentiability, such as caused by min/max or found inside a hard-edge cube.

4.1.3 Use in post-processing

We have seen that SDFs have properties that can be considered complementary to meshes and point clouds. They define, in all of space, the distance and direction to the nearest surface, the normal on the surface, as well as whether a point is on, outside or inside a shape. These properties are of great use in motion planning, where the need to reason about collision and nearby surfaces is used. CSDFs provide this information by evaluating a mathematical expression; discrete SDFs provide this information by trilinear interpolation; and meshes typically use raycasting through volume acceleration structures. However, the cost of evaluating this expression can grow unwieldy for complex shapes, whereas the cost of evaluating a discrete SDF stays the same. On the other hand, storing a CSDF can be cheaper and simpler than a compressed discretized SDF. These same properties make CSDFs compelling (or likewise unsuited) in optimization algorithms, such as used in *noise reduction*, where the distance to the surface is used as a cost and evaluated for many points.

Finally, the great variety of shapes, simple and complex, that can be described and explicitly parametrized with CSDFs make them compelling for many tasks. For example, in modelling indoor room layouts, where the knowledge of the room can provide occlusion or visibility queues for indoor mapping; or in general scene modelling.

4.2 Future work

It would be interesting to try to generate CSDF expressions from a mesh or a point cloud. If not fully automatically, perhaps semi-automatically by defining the desired primitives and operators and automatically estimating the best parameters. This would enable 3D scanning and the use of existing object databases. This is ongoing research, but recent work seems promising [28].

On a related note, it would be interesting to represent the entire SLAM map using CSG and CSDFs, perhaps in the form of a tree of operations and primitives. Such a representation would inherit the benefits of distance functions, which have been beneficial in high-quality 3D reconstruction, while requiring considerably less storage. This does seem to necessitate a means of obtaining primitives and "operations" from images or depth measurements; perhaps a task for *Deep-Learning* aficionados.

Fine pose estimation using only shape appears to be difficult, especially in sparse point clouds. Pose estimation in denser map representations would be an interesting future direction, as they could better constrain pose estimates. It would also be interesting to investigate use of image appearance, such as texture mapping or simple uniform colors, or to align against silhouette edges.

Denser maps could also enable estimation of even more parameters, such as those describing object shape variations. It would be interesting to more thoroughly investigate the use of CSDFs for recovering deformable objects. One way to generate variations is by controlling the parameters of the constituent primitives and the operations applied to them. One could possibly obtain from examples principal component-like vectors controlling these parameters.

REFERENCES

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

[2] Matthew Berger, Andrea Tagliasacchi, Lee Seversky, Pierre Alliez, Joshua Levine, Andrei Sharf, and Claudio Silva. State of the art in surface reconstruction from point clouds. In *EUROGRAPHICS star reports*, volume 1, pages 161–185, 2014.

[3] Matthew Berger, Andrea Tagliasacchi, Lee M Seversky, Pierre Alliez, Gael Guennebaud, Joshua A Levine, Andrei Sharf, and Claudio T Silva. A survey of surface reconstruction from point clouds. In *Computer Graphics Forum*. Wiley Online Library, 2016.

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

[5] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[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*, 2013.

[7] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John 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.

[8] Berk Calli, Aaron Walsman, Arjun Singh, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M Dollar. Benchmarking in manipulation research: The ycb object and model set and benchmarking protocols. *arXiv preprint arXiv:1502.03143*, 2015.

[9] Daniel R Canelhas, Erik Schaffernicht, Todor Stoyanov, Achim J Lilienthal, and Andrew J Davison. An eigenshapes approach to compressed signed distance fields and their utility in robot mapping. *arXiv preprint arXiv:1609.02462*, 2016.

[10] 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 *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 3671–3676. IEEE, 2013.

[11] Daniel R Canelhas, Todor Stoyanov, and Achim 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.

[12] Falak Chhaya, Dinesh Reddy, Sarthak Upadhyay, Visesh Chari, M Zeeshan Zia, and K Madhava 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.

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

[14] Titus Cieslewski, Elena Stumm, Abel Gawel, Mike Bosse, Simon Lynen, and Roland 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.

[15] Alejo Concha, Wajahat Hussain, Luis Montano, and Javier Civera. Incorporating scene priors to dense monocular mapping. *Autonomous Robots*, 39(3):279–292, 2015.

[16] Daniel Cremers. Image segmentation with shape priors: Explicit versus implicit representations. *Handbook of Mathematical Methods in Imaging*, pages 1909–1944, 2015.

[17] 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. ACM, 1996.

[18] Barbara Cutler, Julie Dorsey, Leonard McMillan, Matthias Müller, and Robert Jagnow. A procedural approach to authoring solid models. In *ACM Transactions on Graphics (TOG)*, volume 21, pages 302–311. ACM, 2002.

[19] Angela Dai, Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Christian Theobalt. Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface reintegration. *ACM Transactions on Graphics (TOG)*, 36(3):24, 2017.

[20] Amaury Dame, Victor A Prisacariu, Carl Y Ren, and Ian Reid. Dense reconstruction using 3d object shape priors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1288–1295, 2013.

[21] Bertram Drost, Markus Ulrich, Nassir Navab, and Slobodan 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.

[22] Eva Dyllong and Wolfram Luther. Distance calculation between a point and a nurbs surface. Technical report, DTIC Document, 2000.

[23] Maksym Dzitsiuk, Jürgen Sturm, Robert Maier, Lingni Ma, and Daniel Cremers. Denoising, stabilizing and completing 3d reconstructions on-the-go using plane priors. *arXiv preprint arXiv:1609.08267*, 2016.

[24] Jakob Engel, Vladlen Koltun, and Daniel Cremers. Direct sparse odometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[25] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *European Conference on Computer Vision*, pages 834–849. Springer, 2014.

[26] Francis Engelmann, Jörg Stückler, and Bastian 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.

[27] Pierre-Alain Fayolle and Alexander 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.

[28] Pierre-Alain Fayolle and Alexander Pasko. An evolutionary approach to the extraction of object construction trees from 3d point clouds. *Computer-Aided Design*, 74:1–17, 2016.

[29] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *IEEE transactions on pattern analysis and machine intelligence*, 32(9):1627–1645, 2010.

[30] Sarah F Frisken and Ronald N Perry. Designing with distance fields. In *ACM SIGGRAPH 2006 Courses*, pages 60–66. ACM, 2006.

[31] Sarah F Frisken, Ronald N Perry, Alyn P Rockwood, and Thouis R Jones. Adaptively sampled distance fields: A general representation of shape for computer graphics. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 249–254. ACM Press/Addison-Wesley Publishing Co., 2000.

[32] Yasutaka Furukawa, Carlos Hernández, et al. Multi-view stereo: A tutorial. *Foundations and Trends® in Computer Graphics and Vision*, 9(1-2):1–148, 2015.

[33] Dorian Gálvez-López, Marta Salas, Juan D Tardós, and JMM Montiel. Real-time monocular object slam. *Robotics and Autonomous Systems*, 75:435–449, 2016.

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

[35] W Nicholas Greene, Kyel Ok, Peter Lommel, and Nicholas Roy. Multi-level mapping: Real-time dense monocular slam. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 833–840. IEEE, 2016.

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

[37] Tomáš Hodan, Pavel Haluza, Štepán Obdržálek, Jirí Matas, Manolis Lourakis, and Xenophon 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.

[38] Dirk Holz, Alexandru E Ichim, Federico Tombari, Radu B Rusu, and Sven Behnke. Registration with the point cloud library: a modular framework for aligning in 3-d. *IEEE Robotics & Automation Magazine*, 22(4):110–124, 2015.

[39] Hai Huang and Helmut Mayer. Towards automatic large-scale 3d building reconstruction: Primitive decomposition and assembly. In *International Conference on Geographic Information Science*, pages 205–221. Springer, 2017.

[40] Hao Jiang and Jianxiong Xiao. A linear approach to matching cuboids in rgbd images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2171–2178, 2013.

[41] Michael Kaess. Simultaneous localization and mapping with infinite planes. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 4605–4611. IEEE, 2015.

[42] Matthew Klingensmith, Ivan Dryanovski, Siddhartha Srinivasa, and Jizhong Xiao. Chisel: Real time large scale 3d reconstruction onboard a mobile device using spatially hashed signed distance fields. In *Robotics: Science and Systems*, 2015.

[43] Lu Ma and Gabe Sibley. Unsupervised dense object discovery, detection, tracking and reconstruction. In *European Conference on Computer Vision*, pages 80–95. Springer, 2014.

[44] Eric Marchand, Hideaki Uchiyama, and Fabien Spindler. Pose estimation for augmented reality: a hands-on survey. 2016.

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

[46] Przemyslaw Musialski, Peter Wonka, Daniel G Aliaga, Michael Wimmer, L v Gool, and Werner Purgathofer. A survey of urban reconstruction. In *Computer graphics forum*, volume 32, pages 146–177. Wiley Online Library, 2013.

[47] 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 *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*, pages 127–136. IEEE, 2011.

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

[49] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[50] Helen Oleynikova, Alex Millane, Zachary Taylor, Enric Galceran, Juan Nieto, and Roland Siegwart. Signed distance fields: A natural representation for both mapping and planning. In *RSS Workshop on Geometry and Beyond*, 2016.

[51] Pedro Pinies, Lina Maria Paz, and Paul Newman. Dense mono reconstruction: Living with the pain of the plain plane. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 5226–5231. IEEE, 2015.

[52] Roi Poranne, Craig Gotsman, and Daniel Keren. 3d surface reconstruction using a generalized distance function. In *Computer Graphics Forum*, volume 29, pages 2479–2491. Wiley Online Library, 2010.

[53] Victor Adrian Prisacariu, Aleksandr V Segal, and Ian Reid. Simultaneous monocular 2d segmentation, 3d pose recovery and 3d reconstruction. In *Asian Conference on Computer Vision*, pages 593–606. Springer, 2012.

[54] Mahdi Rad and Vincent 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.

[55] Datta Ramadasan, Thierry Chateau, and Marc Chevaldonné. Dcslam: A dynamically constrained real-time slam. In *Image Processing (ICIP), 2015 IEEE International Conference on*, pages 1130–1134. IEEE, 2015.

[56] Tim Reiner, Gregor Mückl, and Carsten Dachsbacher. Interactive modeling of implicit surfaces using a direct visualization approach with signed distance functions. *Computers & Graphics*, 35(3):596–603, 2011.

[57] Radu Bogdan Rusu. *Semantic 3D object maps for everyday robot manipulation*. Springer, 2013.

[58] Marta Salas, Wajahat Hussain, Alejo Concha, Luis Montano, Javier Civera, and JMM Montiel. Layout aware visual tracking and mapping. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 149–156. IEEE, 2015.

[59] Davide Scaramuzza and Friedrich Fraundorfer. Visual odometry [tutorial]. *IEEE robotics & automation magazine*, 18(4):80–92, 2011.

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

[61] 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.

[62] Shuran Song and Jianxiong 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.

[63] Aksel Sveier. Primitive shape detection in point clouds. Master's thesis, NTNU, 2016.

[64] Jan B Thomassen, Pal H Johansen, and Tor Dokken. Closest points, moving surfaces, and algebraic geometry. *Mathematical methods for curves and surfaces: Tromsø*, pages 351–362, 2004.

[65] Zahra Toony, Denis Laurendeau, and Christian Gagné. Pgp2x: Principal geometric primitives parameters extraction. In *GRAPP*, pages 81–93. Citeseer, 2015.

[66] Gokul Varadhan, Shankar Krishnan, Young J Kim, Suhas Diggavi, and Dinesh Manocha. Efficient max-norm distance computation and reliable voxelization. In *Symposium on geometry processing*, pages 116–126, 2003.

[67] 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.

[68] Shichao Yang, Yu Song, Michael Kaess, and Sebastian Scherer. Pop-up slam: Semantic monocular plane slam for low-texture environments. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 1222–1229. IEEE, 2016.

[69] Xenophon Zabulis,Manolis IA Lourakis,and Panagiotis Koutlemanis. Correspondence-free pose estimation for 3d objects from noisy depth data. *The Visual Computer*, pages 1–19, 2016.

[70] Edward Zhang, Michael F Cohen, and Brian Curless. Emptying, refurnishing, and relighting indoor spaces. *ACM Transactions on Graphics (TOG)*, 35(6):174, 2016.

[71] Zhengyou Zhang. Parameter estimation techniques: A tutorial with application to conic fitting. *Image and vision Computing*, 15(1):59–76, 1997.

[72] Shuai Zheng, Victor Adrian Prisacariu, Melinos Averkiou, Ming-Ming Cheng, Niloy J Mitra, Jamie Shotton, Philip HS Torr, and Carsten Rother. Object proposals estimation in depth image using compact 3d shape manifolds. In *German Conference on Pattern Recognition*, pages 196–208. Springer, 2015.