A Survey of Voxel Interpolation Methods and an Evaluation of Their Impact on Volumetric Map-Based Visual Odometry

Daniel Ricão Canelhas*, Todor Stoyanov†, Achim J. Lilienthal†
*Univrses AB, Sweden †Center of Applied Autonomous Sensor Systems (AASS), Örebro University, Sweden

Abstract— **Voxel volumes are simple to implement and lend themselves to many of the tools and algorithms available for 2D images. However, the additional dimension of voxels may be costly to manage in memory when mapping large spaces at high resolutions. While lowering the resolution and using interpolation is common work-around, in the literature we often find that authors either use trilinear interpolation or nearest neighbors and rarely any of the intermediate options. This paper presents a survey of geometric interpolation methods for voxel-based map representations. In particular we study the truncated signed distance field (TSDF) and the impact of using fewer than 8 samples to perform interpolation within a depth-camera pose tracking and mapping scenario. We find that lowering the number of samples fetched to perform the interpolation results in performance similar to the commonly used trilinear interpolation method, but leads to higher frame-rates. We also report that lower bit-depth generally leads to performance degradation, though not as much as may be expected, with voxels containing as few as 3 bits sometimes resulting in adequate estimation of camera trajectories.**

I. Introduction

There are several options when choosing a map representation. A truncated signed distance field (TSDF) is one option that has proven to be useful for several tasks relevant to robotics, e.g. accurate real-time dense surface reconstruction [1], robust ego-motion estimation [2][3], SLAM [4][5], feature detection and description [6][7]. The TSDF is a voxel-based representation of 3D space that encodes surfaces implicitly by storing, at each voxel, the distance to the nearest surface. A sign is used to disambiguate between distances measured relative to the front and the back side of surfaces, with a distance of zero corresponding to the actual surface position.

Although using a TSDF as a common frame of reference for obtaining de-noised depth maps is a well-established method for accurate frame-to-model registration, direct alignment of depth maps against the full volumetric TSDF is a less common approach [2], [8]. The direct alignment method is often regarded as too costly to be used in real-time settings, partly due to its unpredictable memory-access pattern compounded by the assumed need for expensive tri-linear interpolation for numerical gradients and function values [9]. More generally, the large memory requirements of TSDFs (and other voxel-based maps) may be prohibitive. However, in the context of robot mapping and pose estimation, memory constraints may potentially be relaxed simply by using fewer bits per voxel. In this paper we attempt to quantify the memory requirement more precisely by evaluating how many bits per voxel are required for accurate pose estimation. Experimental evidence presented in this paper also supports the claim that the latency associated with memory access for interpolation can be reduced relative to that of the standard trilinear method. Our results are based on from surveying the different methods of voxel-based interpolation, from nearest-neighbor *extrapolation* to standard trilinear *interpolation*[1] in the context of the pose estimation problem by direct scan alignment to a volumetric map.

Deciding on what interpolation method to use is not a straightforward task, as it involves considering several application dependent criteria. For example, the run-time performance cost of querying a large number of voxel locations during e.g. scan-registration, rendering [10] or collision-free grasp planning [11] is generally dominated by the time taken to access the memory in which the voxels are stored, suggesting that the choice of interpolation method has a significant influence on performance. In one scenario, one may consider using trilinear interpolation to avoid the blocky artifacts caused by nearest neighbor extrapolation. Similarly, in another scenario one may needlessly accept low-quality nearest-neighbor extrapolation, because trilinear interpolation is deemed too expensive. One contribution of the present work is a survey of linear interpolation methods that lie *in between* these two extremes. We describe the methods in sufficient detail to enable a straightforward implementation of each. Another contribution is a set of performance evaluations of the SDF-Tracker algorithm, comparing trilinear, tetrahedral and nearest neighbor access, as well as experiments reducing the number of bits per voxel.

A. Organization of topics

In section II, we will review the prerequisite geometrical concepts that will provide us with a common notation for expressing different interpolation methods. In section III we describe the interpolation methods in detail and briefly discuss low-bit voxel representations. In section IV we describe the experimental set-up, followed by empirical results (in section V) with respect to tracking accuracy and run-time performance. Lastly, in section VI we present our conclusions.

II. Background

Consider the line segment joining nodes a and b in Fig. 1, assumed to be of unit length.

[1]Higher-order methods involving additional samples exist, but we will restrict our survey to linear methods with 8 samples or less in this work.

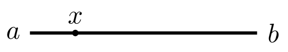

Fig. 1: Line segment a b, with an intermediate point x.

The value of some function $\phi(x)$ at an arbitrary point, x, along the line can be estimated via linear interpolation as:

$$\phi(x) \approx \phi(a) \cdot (b - x) + \phi(b) \cdot (x - a) \tag{1}$$

In essence, the weight given to the function value at point a is proportional to the length of the segment on the opposite side of the query point i.e, $(b - x)$, and vice-versa. For

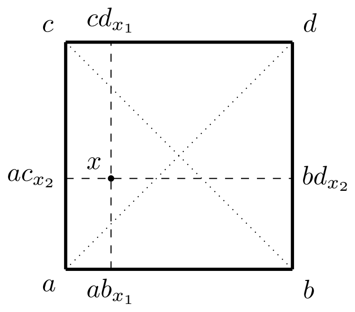

Fig. 2: bilinear interpolation offers the choice of linearly interpolating in x_1 first, yielding $\phi(cd_{x_1})$ and $\phi(ab_{x_1})$, and then obtaining $\phi(x)$ or, alternatively starting with x_2.

interpolation in a square, the bilinear approach is simply an extension of this logic into two dimensions. In other words, for a given query point x, shown in Fig. 2 to be in a unit square defined by vertices $a, b, c, d \in \mathbb{R}^2$, the function $\phi(x) : \mathbb{R}^2 \to \mathbb{R}$ can be approximated as:

$$\phi(cd_{x_1}) \approx \phi(c) \cdot (d - cd_{x_1}) + \phi(d) \cdot (cd_{x_1} - c) \tag{2}$$
$$\phi(ab_{x_1}) \approx \phi(a) \cdot (b - ab_{x_1}) + \phi(b) \cdot (ab_{x_1} - a) \tag{3}$$
$$\phi(x) \approx \phi(ab_{x_1}) \cdot (cd_{x_1} - x) + \phi(cd_{x_1}) \cdot (x - ab_{x_1}) \tag{4}$$

as long as the query point is within the domain defined by the vertices. Bilinear interpolation scales the contribution of the value at each vertex by the area of the rectangle formed between the query point and the diagonally opposite vertex. *As a general rule, linear interpolation in any number of dimensions can be performed by scaling the contribution of each vertex by the size of the simplex formed between the query point and the other vertices.* A square is not a simplex in $\mathbb{R}^2$, though; a triangle is. In the example shown in Fig.2, we see that the point x falls inside both triangles Δadc and Δabc. If we denote the areas of a generic triangle formed by vertices x, y, z as $\Delta^2 xyz$ we can compactly represent the *2-Simplex* interpolated value of $\phi(x)$ as:

$$\phi(x) \approx \phi(a) \cdot (\Delta^2 bcx) + \phi(b) \cdot (\Delta^2 axc) + \phi(c) \cdot (\Delta^2 abx). \tag{5}$$

Alternatively, using triangle Δadc:

$$\phi(x) \approx \phi(a) \cdot (\Delta^2 cxd) + \phi(d) \cdot (\Delta^2 axc) + \phi(c) \cdot (\Delta^2 adx). \tag{6}$$

The volume of any simplex (including the area of a triangle) with vertices $v_0, v_1, v_2, \ldots, v_n$ can be computed

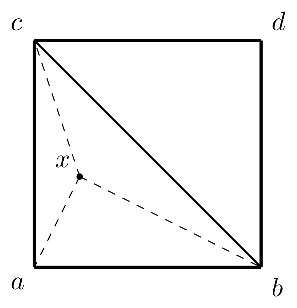

Fig. 3: 2-simplex interpolation offers the choice of linearly interpolating the value of $\phi(x)$ from one of two triangles. Since x is furthest from d, one may prefer the triangle Δabc over Δadc

using the following expression [12]:

$$\Delta^n v_0 \ldots v_n = \frac{1}{n!} |det(\begin{bmatrix} v_1 - v_0 & v_2 - v_0 & \ldots & v_n - v_0 \end{bmatrix})| \tag{7}$$

with n indicating the dimensionality i.e. 1 for a line, 2 for a triangle, 3 for a tetrahedron, and so on. By $det()$ we denote the determinant of the $n \times n$ matrix formed by concatenating the vertex differences as its columns. We now have the basic maths required to understand how the simplex interpolation and orthogonal interpolation options can be combined to produce several different ways of estimating the value of a function in between the discrete samples of a voxel grid.

III. Interpolation Methods and Their Relation to TSDF Reconstruction

In the following sections we will assume that we have a discrete lattice in 3D, with values sampled at the nodes as shown in Fig. 4. The node labels correspond to the following coordinates:

$$\begin{bmatrix} a & b & c & d & e & f & g & h \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix} \tag{8}$$

and we are interested in estimating the value of the underlying function at the real-valued 3D query point x whose components lie in the interval $[0, 1]$.

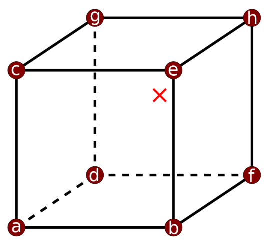

Fig. 4: A cube formed by 8 adjacent vertices, each containing a discrete sample of a real-valued field or function, e.g. a TSDF. The x indicates a point within the cube for which we desire an interpolated value

A. *Trilinear Interpolation*

Trilinear interpolation is done by first performing four linear interpolations between the eight vertices of the cube, as shown in Fig. 5(a). One then ends up with a square slice that embeds the query point. The subsequent steps, illustrated in Figs. 5(b), 5(c) are identical to the bilinear interpolation. The order in which the axes for interpolation are chosen is arbitrary. Formally, the interpolation for $f(x)$ is expressed

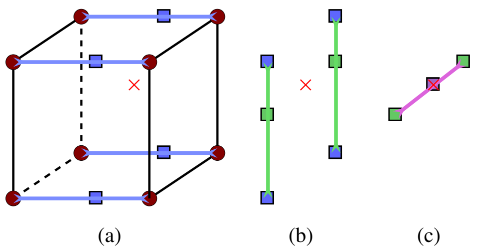

Fig. 5: Trilinear interpolation proceeds along each orthogonal axis in turn, first reducing the cube to a plane, then reducing the plane to a line, and lastly reducing the line to a single point.

by the steps:

$$\phi(p) \approx \phi(a) \cdot (b_1 - x_1) + \phi(b)x_1 \tag{9}$$
$$\phi(q) \approx \phi(c) \cdot (e_1 - x_1) + \phi(e)x_1 \tag{10}$$
$$\phi(r) \approx \phi(d) \cdot (f_1 - x_1) + \phi(f)x_1 \tag{11}$$
$$\phi(s) \approx \phi(g) \cdot (h_1 - x_1) + \phi(h)x_1 \tag{12}$$

abusing the notation somewhat to simultaneously define the interpolated locations as p, q, r, s and the estimated value of the function $\phi()$ at these locations. This step is followed by,

$$\phi(t) = \phi(p) \cdot (q_2 - x_2) + \phi(q)x_2 \tag{13}$$
$$\phi(u) = \phi(r) \cdot (s_2 - x_2) + \phi(s)x_2 \tag{14}$$

defining t, u as the interpolated points between p, q and r, s, respectively. Lastly,

$$\phi(x) = \phi(t) \cdot (u_3 - x_3) + \phi(u)x_3 \tag{15}$$

Although this process seems to collapse the dimensions of the problem one at a time, down to a point, it is mathematically equivalent to scaling the contribution of each vertex by the volume of the parallelepiped formed between the query point and the vertex diametrically opposed to the vertex in question. Graphics Processing Units (GPUs) that support 3D textures generally implement this process in hardware.

B. *Prismatic Interpolation*

Relative to trilinear interpolation, prismatic interpolation [13] reduces the number of necessary samples to six instead of eight. This method results from splitting the cube into two triangular prisms with right-angled triangles and selecting the prism containing the query point. The first step is to interpolate the function value on each of the triangular faces, e.g using the 2-simplex method detailed in the introduction (or other methods [14], [15]). The second step is a simple linear interpolation between the resulting values. In Fig. 6 we have assumed that 2-simplex interpolation is used and color-coded the vertices and triangles such that the contribution of each vertex is proportional to the area of the triangle with the same color. Since the equations are identical to those described in the introduction, we will omit the formalism here. Because the interpolation method chosen may be of higher order within the triangle, the ordering may play a role in the result. For instance, in applications where the function varies less in one dimension, it may be wise to leave it for the final step.

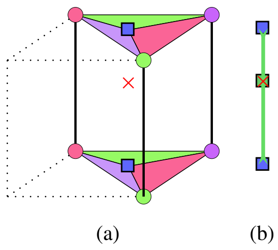

Fig. 6: Prismatic interpolation is a combination of 2-simplex interpolation, followed by linear interpolation.

C. *Pyramidal Interpolation*

When we remove another vertex: picking four on one face plus an additional vertex on the opposite face; we obtain a pyramid with a square base and four triangular sides [16]. Although the pyramid will need to be oriented in different ways to encompass the query point, we will refer to the single point on the non-basal face as the apex. In Fig. 7 we illustrate selecting the topmost four vertices as the base, and the bottom as the "apex". There are several options for how to obtain the interpolated value at the query point. Making reference to Fig. 7 (best viewed in color), these are:

- Find the line from the apex that intersects the base while passing through the query point (pink line). Perform bilinear interpolation on the square base to get the value at the intersection point, then linearly interpolate between the intersection and the apex for the final value.
- interpolate along each of the four edges connecting the apex to the base, to get a square slice (green edges) through the pyramid that embeds the query point. Then interpolate on the plane bilinearly for the final value.

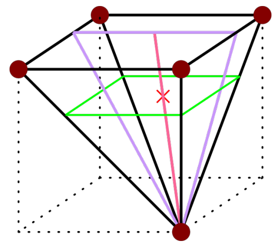

Fig. 7: Pyramidal interpolation can be done by reducing the cube to a line, square or triangle, and applying one of the 2D interpolation methods of choice

- Interpolate along two edges in the pyramid base, along a chosen dimension. This results in a line segment across the square base of the pyramid. Form a triangle with this line segment and the apex to get a triangle (drawn in purple) containing the query point. Interpolate for the final value using any choice of triangular interpolation method.

While pyramidal interpolation does result in one less memory look-up than the prismatic method, it leaves many options for the choice of which face to pick as the pyramid base and apex. Given that there are six different faces, and four possible apexes for each face, the number of cases are 24. Since this method samples a large proportion of vertices from a single side of the cube, it may result in more apparent discontinuities in the approximation of the function when transitioning from one pyramid to another.

D. *Tetrahedral Interpolation*

When memory access is expensive relative to computation it might be reasonable to look for methods that require as few samples as possible. Tetrahedral interpolation is the **interpolation** method that requires fewest samples [17]. With an additional check, it is also possible to identify the near-degenerate cases, in which a query point is very close to a face, edge or vertex of the tetrahedron. These checks can be done analytically, or by discretizing the sub-voxel space to some arbitrary level of precision to determine how many samples are needed to accurately estimate the value. For example, a query point exactly at the center of the cube could potentially be interpolated by picking any pair of diametrally opposed vertices and doing a single linear interpolation. Checking for near-degeneracy allows for accurate interpolation with even fewer than 4 samples, on average [18].

There are exactly thirteen ways to split a cube into tetrahedra, without counting reflections and rotations as unique cases [19]. Twelve of the splittings result in six rather thin and irregular shapes but a thirteenth, depicted in Fig. 8, results in only five, four of which are right-angled and the last being a regular tetrahedron. This last splitting is especially interesting for interpolation purposes, since determining which of the tetrahedra contains the query point is quite straightforward, so we will look at it a little closer. To construct the tetrahedral splitting of the cube, we will use the labels defined in Eq. (8) and define the tetrahedra using the following sets of vertices:

$$T_1 = \{a, b, c, d\}, \tag{16}$$

$$T_2 = \{e, b, c, h\}, \tag{17}$$

$$T_3 = \{g, c, h, d\}, \tag{18}$$

$$T_4 = \{f, b, d, h\}, \tag{19}$$

$$T_5 = \{b, c, d, h\} \tag{20}$$

Essentially, it picks a subset of four of the cube's vertices that are separated by diagonals of the faces of the cube, along with the three vertices joined to them, by edges of the cube. These form the first four 'corner' tetrahedra ($T_{1\ldots4}$).

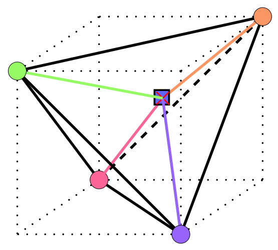

Fig. 8: Tetrahedral interpolation - the value at the query point marked by "X" is found by taking the sum of the values at each vertex in proportion to the volume of the tetrahedron formed by the remaining vertices and the query point itself. The value is normalized by dividing the result by the volume of the tetrahedron formed by the four vertices. Drawn in bold is tetrahedron T_5

The remaining volume is a central tetrahedron (T_5) with equilateral triangular faces of edge lengths equal to $\sqrt{2}$. This central remaining tetrahedron is composed of the vertices that were not part of the initial selection of four.

Determining into which tetrahedron a given query point x is found can be done by checking if the distance between the query point and any of the corners a, f, g, e is less than unit (by the L_1-norm). If not, the query point is found inside T_5. The interpolated value for x, when x falls into T_5 is given by:

$$\phi(x) \approx \frac{h\Delta^3 bdcx + b\Delta^3 hcdx + c\Delta^3 hdbx + d\Delta^3 bhcx}{\Delta^3 bdch} \tag{21}$$

This refers back to Eq. (7) for the general formula for the volume of a simplex and is analogous for interpolation in $T_{1\ldots4}$. However, it is worth noting that the equations for the volumes in these tetrahedra simplify greatly, due to their axis-aligned faces.

E. *Nearest Neighbor*

The last interpolation method that we will discuss is, strictly speaking, an **extrapolation** method. It entails to simply check which octant of the cube the query point is in and return the voxel value stored in that octant. It requires a single memory lookup.

$$\phi(x) \approx \phi(\begin{bmatrix} \lfloor x_1 + 0.5 \rfloor \\ \lfloor x_2 + 0.5 \rfloor \\ \lfloor x_3 + 0.5 \rfloor \end{bmatrix}) \tag{22}$$

F. *A note on gradient estimation*

In applications such as pose estimation, we are often interested not just in the function value $\phi(x)$ but also the gradient, $\nabla\phi(x)$. In TSDFs, the gradient provides an estimate for the orientation of surfaces. The most straight-forward method of numerically obtaining gradients is by finite differences. In this work we evaluate the performance of the SDF-Tracker algorithm by computing the gradients using central differences with the interpolation being tested. One exception is made for nearest neighbor extrapolation, where we additionally evaluate the performance obtained by alternating between forward or backward differences. This is done by fetching the three values connected to the nearest neighbor to the query point x, without leaving the bounding cube of eight. To exemplify, if x is nearest to a, we set $\phi(x) = \phi(a)$ and estimate the gradient $\nabla\phi(x) \approx \begin{bmatrix} \phi(b) - \phi(a) \\ \phi(c) - \phi(a) \\ \phi(d) - \phi(a) \end{bmatrix}$ however, if x is instead nearest to e.g. g, we set $\phi(x) = \phi(g)$ and estimate the gradient $\nabla\phi(x) \approx \begin{bmatrix} \phi(h) - \phi(g) \\ \phi(g) - \phi(c) \\ \phi(g) - \phi(d) \end{bmatrix}$. While this represents forward differencing in the first dimension, it results in backward differencing in the second and third.

G. Varying the bit-depth in Voxel representations

The TSDF representation used by the SDF-Tracker uses the incremental update method of Curless et al [20] which requires storing a signed distance value D along with a *weight* W that represents the confidence regarding the value in each voxel. As an additional experiment in resource limitation, D and W can be represented with a reduced number of bits relative to e.g. a 32-bit representation, leading to increasing quantization.

IV. Experimental Set-up

For our evaluations, we used a publicly available data-set published by the Computer Vision group at the Technical University of Munich (TUM) [21] consisting of recordings made with a hand-held depth-sensing camera that is tracked by an external motion capturing system to provide ground-truth poses with each image frame. The first frame is used to approximate a TSDF using line-of-sight distances and is incrementally updated by fusing further depth images into the volume. The pose of the camera is estimated by directly aligning the depth image vertices to obtain the least-squares sum of signed distances. This corresponds to the predicted surface location, implied by the TSDF.

Since we are interested in the lower limits of interpolation quality, we compare the trilinear interpolation scheme directly with the tetrahedral and nearest neighbor methods using both central differences and the alternating forward and backward differences for gradients, as described in Sec. III-F.

For each method under evaluation, we report the mean, median and variance for pose errors, relative to ground truth for each of 8 different depth image sequences in different categories. The error metrics used are the relative pose errors (RPE), measured between consecutive frames, as well as the absolute trajectory error (ATE). We also investigate the performance relative to the number of bits used to store the TSDF, using the same metrics.

Finally, we also report the run-time in terms of frames per second, using the SDF-tracker at different voxel volume sizes and input depth image resolutions for both nearest-neighbor and trilinear interpolation, running on an Intel i7-4770K CPU at 3.50GHz, without GPU acceleration.

V. Empirical Results

A. Interpolation

Since all the interpolation methods here are linear methods; as long as the underlying function is also linear, the interpolation is expected to be exact. For indoor environments, maps tend to be largely dominated by planar structures resulting in a TSDF that is, at least piecewise, well-approximated by linear functions. From the absolute trajectory error (ATE) in Table I we see that there is some apparent degradation in performance when using lower-quality interpolation and gradients, but looking at the relative pose errors (RPE) for translation and rotation, shown in Table III and Table II it is unclear where the differences are. In fact, under a pairwise Mann-Whitney U test [22], we find no statistically significant difference in performance between any of them, even though the mean for the nearest neighbor methods are clearly higher than the interpolation methods, in several cases. Due to the observed similarity in the performance between the trilinear and tetrahedral methods, we have omitted the methods with intermediate number of samples from the evaluation although we include their descriptions for the completeness of the survey section of this paper.

B. Bit Depth

Absolute Trajectory Error for tracking and mapping, using different numbers of bits per voxel to represent the TSDF shows more obvious differences. We see a general trend towards greater errors and variances as the number of bits is reduced, however this progression is not always monotonic for all depth image sequences. Failures to track the camera may lead to gross misalignments from which the tracking is unable to recover. The large errors are generally indicative of the severity or frequency of such failures. However, the progression in relative pose errors seen in Table II and Table III is more gradual, with errors becoming, on average, slightly larger as the number of bits is reduced.

Nonetheless, it is interesting that tracking the camera is still possible with with some errors as far as down to a 3-bit representation. At that level of quantization, the number of distinct values used to represent the distance field from D_{min} to D_{max} are only 7. Given that the distance field was truncated at the fixed values of $\pm 0.30m$, and the voxel size was of 0.02m, this results in severe stair-casing artifacts in the distance field, since several adjacent voxels take on the same value. We find that the tracking fails more often with fewer bits, leading to significantly lower performance in trajectory estimation. However, we note that the relative error between consecutive frames does not increase as much. This indicates that reducing the number of bits leads to an increased brittleness, causing few but devastating failures.

There is little difference in performance, from a reliability standpoint, when using more than 4 bits for representing the distance, and tracking will succeed and fail to similar extents. However, we can see that the average relative errors and variance of the error increase slightly each time one reduces the number of bits per voxel.

C. Run-time

What these tests do not reveal, since they process an offline data-set, is the impact that these lower-fidelity methods have

TABLE I: Absolute Trajectory Errors - (Rotation and Translation, combined)

Method \ Sequence	desk	desk2	teddy	plant	xyz
tri32	0.056 ± 0.059	0.171 ± 0.117	0.150 ± 0.076	0.089 ± 0.043	0.015 ± 0.008
tet32	0.058 ± 0.062	0.152 ± 0.096	0.158 ± 0.079	0.088 ± 0.042	0.016 ± 0.009
nn32c	0.048 ± 0.041	0.324 ± 0.154	0.186 ± 0.082	0.093 ± 0.038	0.016 ± 0.009
nn32fb	0.225 ± 0.085	0.234 ± 0.130	0.199 ± 0.101	0.107 ± 0.048	0.019 ± 0.011
tri16	0.045 ± 0.049	0.130 ± 0.104	0.387 ± 0.297	0.110 ± 0.075	0.016 ± 0.009
tri8	0.041 ± 0.038	0.233 ± 0.184	0.399 ± 0.307	0.104 ± 0.090	0.017 ± 0.009
tri4	0.142 ± 0.112	0.443 ± 0.202	0.428 ± 0.243	0.177 ± 0.052	0.017 ± 0.010
tri3	0.292 ± 0.167	0.227 ± 0.164	2.173 ± 0.993	0.976 ± 0.431	0.014 ± 0.008

TABLE II: Relative Pose Error (Rotation, in degrees)

Method \ Sequence	desk	desk2	teddy	plant	xyz
tri32	0.516 ± 0.427	0.652 ± 0.507	0.644 ± 0.526	0.548 ± 0.353	0.274 ± 0.204
tet32	0.516 ± 0.427	0.643 ± 0.490	0.649 ± 0.560	0.548 ± 0.353	0.274 ± 0.205
nn32c	0.509 ± 0.400	0.627 ± 0.448	0.620 ± 0.530	0.545 ± 0.348	0.274 ± 0.202
nn32fb	0.489 ± 0.368	0.630 ± 0.468	0.631 ± 0.497	0.540 ± 0.346	0.272 ± 0.199
tri16	0.527 ± 0.498	0.676 ± 0.582	0.670 ± 0.737	0.561 ± 0.418	0.275 ± 0.204
tri8	0.531 ± 0.493	0.674 ± 0.577	0.659 ± 0.590	0.564 ± 0.430	0.276 ± 0.208
tri4	0.535 ± 0.445	0.695 ± 0.598	0.693 ± 0.646	0.575 ± 0.412	0.280 ± 0.215
tri3	0.549 ± 0.465	0.713 ± 0.604	1.008 ± 2.254	0.608 ± 0.586	0.280 ± 0.216

TABLE III: Relative Pose Error (Translation, in meters)

Method \ Sequence	desk	desk2	teddy	plant	xyz
tri32	0.007 ± 0.007	0.008 ± 0.010	0.007 ± 0.011	0.006 ± 0.005	0.004 ± 0.003
tet32	0.007 ± 0.007	0.008 ± 0.010	0.007 ± 0.011	0.006 ± 0.005	0.004 ± 0.003
nn32c	0.007 ± 0.006	0.009 ± 0.010	0.007 ± 0.010	0.006 ± 0.005	0.005 ± 0.003
nn32fb	0.007 ± 0.006	0.009 ± 0.010	0.007 ± 0.010	0.006 ± 0.005	0.005 ± 0.003
tri16	0.007 ± 0.007	0.008 ± 0.010	0.008 ± 0.017	0.007 ± 0.006	0.005 ± 0.003
tri8	0.007 ± 0.006	0.009 ± 0.012	0.008 ± 0.016	0.007 ± 0.007	0.005 ± 0.003
tri4	0.008 ± 0.007	0.010 ± 0.012	0.012 ± 0.023	0.007 ± 0.008	0.005 ± 0.003
tri3	0.008 ± 0.009	0.011 ± 0.014	0.029 ± 0.113	0.010 ± 0.016	0.004 ± 0.003

In all tables the results are reported as mean $\pm$ standard deviation, using metrics described in [21]. The labels denote the following: triX – X-bit trilinear interpolation, using central differences for gradients; tet32 – 32-bit tetrahedral interpolation using central differences for gradients; nn32c – 32-bit nearest-neighbor extrapolation using central differences for gradients; nn32fb – 32-bit nearest-neighbor extrapolation using alternating forward and backward differences for gradients. Notable errors are highlighted in gray, but the reader is advised to be conservative in drawing conclusions about them due to the large variance.

on run-time efficiency. In Fig. 9 we can see that the SDF-tracker is bound by memory access latency, and that reducing the number of voxels that have to be retrieved has an impact on the frame-rate. In spite of the lower-quality estimate of the distances, due to the fact that direct image alignment is a *dense* method, the combined effect of evaluating thousands of samples and gradients evidently results in a reasonably accurate pose estimation. Since frame-rates can be increased if we accept lower-quality estimates of the TSDF, higher resolution depth images or frame-rates [23] are possible for the same number of voxels. On the machine used for testing, there were negligible performance gains in using the tetrahedral method compared to the trilinear approach. The lack in performance increase may be due to the way memory is fetched, incurring similar latency costs, even if fewer values are effectively used. Another possible culprit may be the branching used to determine which tetrahedron to use for evaluation. Since the tetrahedral method already had similar run-time performance to the trilinear approach, we saw no utility in evaluating additional solutions for this particular use-case.

VI. Conclusions

In this paper we have surveyed several interpolation methods that employ *fewer* than 8 samples to interpolate a value in a voxel grid. We have applied the cheapest of these to robot mapping, using the SDF-Tracker algorithm for rigid-body pose estimating of the sensor and incremental map building.

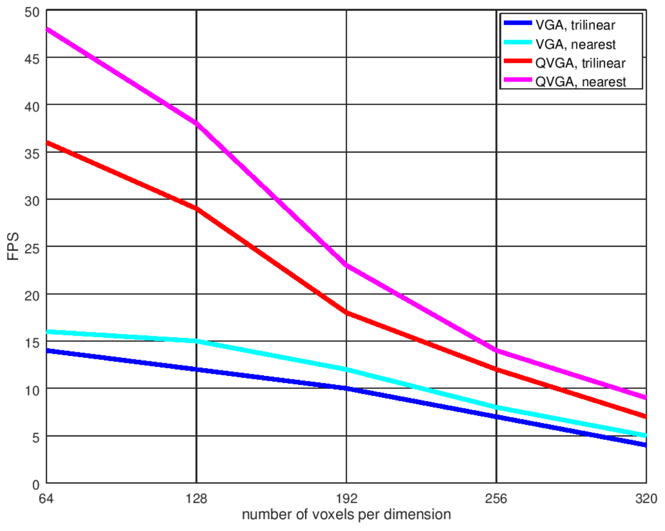

Fig. 9: Frame-rate plotted as a function of interpolation method, number of voxels and image sizes (VGA: 640×480 pixels, QVGA: 320×240 pixels).

Experiments showed that there is little loss in quality, but a noticeable gain in processing speed. We also evaluated the impact of reducing the number of bits stored in each voxel, to represent the distance field. Here, the results were clearer: fewer bits lead to poorer pose estimates.

Although the interaction between number of bits and interpolation scheme may be used in tandem, we chose to quantify each effect separately. Whether they are additive or multiplicative in terms of their adverse effects is an open question. However, an effort to extend this work further may instead proceed in a different direction, towards more sophisticated interpolation methods or more efficient memory layouts such as Z-order curves [24] to optimize cache coherence.

There are other volumetric representations that are not commonly used with interpolation between voxels, such as Occupancy grids [25], and others where trilinear interpolation tends to be applied, such as 3D-Normal Distributions Transforms [26]. In either case, we hope that this work has shown that there are viable alternatives in between expensive trilinear interpolation and low-quality nearest-neighbor extrapolation that may be worth investigating.

Acknowledgement

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732737 (Project ILIAD)

References

[1] R. A. Newcombe, A. J. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim, and A. Fitzgibbon, "KinectFusion: Real-time dense surface mapping and tracking," in *Mixed and Augmented Reality (ISMAR), 2011 10th IEEE International Symposium on*, 2011, pp. 127–136.

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

[3] T. Whelan, M. Kaess, M. Fallon, H. Johannsson, J. Leonard, and J. McDonald, "Kintinuous: Spatially extended KinectFusion," in *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, Sydney, Australia, Jul 2012.

[4] T. Whelan, M. Kaess, H. Johannsson, M. Fallon, J. Leonard, and J. McDonald, "Real-time large scale dense RGB-D SLAM with volumetric fusion," *Intl. J. of Robotics Research, IJRR*, 2014.

[5] A. Dai, M. Nießner, M. Zollhöfer, S. Izadi, and C. Theobalt, "BundleFusion: Real-time globally consistent 3D reconstruction using on-the-fly surface reintegration," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 3, p. 24, 2017.

[6] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3D shapenets: A deep representation for volumetric shapes," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1912–1920.

[7] D. R. Canelhas, T. Stoyanov, and A. J. Lilienthal, "Improved local shape feature stability through dense model tracking," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, 2013, pp. 3203–3209.

[8] E. Bylow, J. Sturm, C. Kerl, F. Kahl, and D. Cremers, "Real-time camera tracking and 3D reconstruction using signed distance functions," in *Robotics: Science and Systems (RSS), Online Proceedings*, vol. 9. Robotics: Science and Systems, 2013, p. 8.

[9] P. Stotko, "State of the art in real-time registration of RGB-D images," in *Central European Seminar on Computer Graphics for Students (CESCG 2016)*, Apr. 2016, supervisor: Tim Golla.

[10] J. C. Hart, "Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces," *The Visual Computer*, vol. 12, no. 10, pp. 527–545, 1996.

[11] T. Stoyanov, R. Krug, R. Muthusamy, and V. Kyrki, "Grasp envelopes: Extracting constraints on gripper postures from online reconstructed 3D models," in *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE, 2016, pp. 885–892.

[12] P. Stein, "A note on the volume of a simplex," *The American Mathematical Monthly*, vol. 73, no. 3, pp. 299–301, 1966.

[13] K. Kanamori, T. Fumoto, and H. Kotera, "A color transformation algorithm using prism interpolation," in *IS&T's 8th International Congress on Advances in NIP Technologies*, 1992, pp. 164–174.

[14] R. Barnhill, G. Birkhoff, and W. J. Gordon, "Smooth interpolation in triangles," *Journal of Approximation Theory*, vol. 8, no. 2, pp. 114–128, 1973.

[15] G. M. Nielson, "The side-vertex method for interpolation in triangles," *Journal of Approximation theory*, vol. 25, no. 4, pp. 318–336, 1979.

[16] P. E. Franklin, "Interpolation methods and apparatus," June 8 1982, uS Patent 4,334,240.

[17] K. Kanamori, H. Kawakami, and H. Kotera, "Novel color transformation algorithm and its applications," in *Electronic Imaging'90, Santa Clara, 11-16 Feb'92*. International Society for Optics and Photonics, 1990, pp. 272–281.

[18] P. Hemingway, "n-simplex interpolation," *HP Laboratories Bristol, HPL-2002-320*, pp. 1–8, 2002.

[19] E. Baumann, "Splitting a cube in tetrahedras," 2004, http://www.baumanneduard.ch/Splittingacubeintetrahedras2.htm, accessed 2017-03-09.

[20] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. ACM, 1996, pp. 303–312.

[21] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of RGB-D SLAM systems," in *Proc. of the IEEE Int. Conf. on Intelligent Robot Systems (IROS)*, 2012.

[22] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The annals of mathematical statistics*, pp. 50–60, 1947.

[23] A. Handa, R. A. Newcombe, A. Angeli, and A. J. Davison, "Real-time camera tracking: When is high frame-rate best?" in *European Conference on Computer Vision*. Springer, 2012, pp. 222–235.

[24] G. M. Morton, "A computer oriented geodetic data base and a new technique in file sequencing," 1966.

[25] A. Elfes, "Occupancy grids: A probabilistic framework for robot perception and navigation," 1989.

[26] M. Magnusson, "The three-dimensional normal-distributions transform: an efficient representation for registration, surface analysis, and loop detection," Ph.D. dissertation, Örebro universitet, 2009.