CEIG – Spanish Computer Graphics Conference (2019)
A. Jarabo and D. Casas (Editors)

Perfect Spatial Hashing for Point-cloud-to-mesh Registration

Daniel Mejia-Parra[1,2], Juan Lalinde-Pulido[3], Jairo R. Sánchez[†2], Oscar Ruiz-Salguero[1] and Jorge Posada[2]

[1]Laboratory of CAD CAM CAE, Universidad EAFIT, Colombia
[2]Vicomtech, España
[3]High Performance Computing Facility APOLO, Universidad EAFIT, Colombia

Abstract

Point-cloud-to-mesh registration estimates a rigid transformation that minimizes the distance between a point sample of a surface and a reference mesh of such a surface, both lying in different coordinate systems. Point-cloud-to-mesh-registration is an ubiquitous problem in medical imaging, CAD CAM CAE, reverse engineering, virtual reality and many other disciplines. Common registration methods include Iterative Closest Point (ICP), RANdom SAmple Consensus (RANSAC) and Normal Distribution Transform (NDT). These methods require to repeatedly estimate the distance between a point cloud and a mesh, which becomes computationally expensive as the point set sizes increase. To overcome this problem, this article presents the implementation of a Perfect Spatial Hashing for point-cloud-to-mesh registration. The complexity of the registration algorithm using Perfect Spatial Hashing is $\mathcal{O}(N_Y \times n)$ (N_Y: point cloud size, n: number of max. ICP iterations), compared to standard octrees and kd-trees (time complexity $\mathcal{O}(N_Y \log(N_T) \times n)$, N_T: reference mesh size). The cost of pre-processing is $\mathcal{O}(N_T + (N_H^3)^2)$ (N_H^3: Hash table size). The test results show convergence of the algorithm (error below 7e-05) for massive point clouds / reference meshes ($N_Y = 50k$ and $N_T = 28055k$, respectively). Future work includes GPU implementation of the algorithm for fast registration of massive point clouds.

CCS Concepts

•Theory of computation → Convex optimization; Computational geometry; •Computing methodologies → Mesh models; Point-based models; •Applied computing → Computer-aided design;

1. Introduction

Point set registration is ubiquitous in Reverse Engineering, Medical Imaging, Visual (Dimensional) Inspection, Robotics, among other disciplines.

Consider two point set samples of an object, each one conducted in its own coordinate system. The points in one set do not exactly correspond to object locations sampled in the other set. Moreover, parts of the object visible in one coordinate system may be unaccessible for sample in the other coordinate system (e.g. two clipped depth scans of the same object). The point set registration problem consists of finding a rigid transformation that rotates and translates one point set onto the other, producing the best possible matching between the transformed and the static point sets.

Point set registration is strongly qualified by the underlying structure of the point sets. Registration of surface point samples is very different from registration of point samples obtained from the interior of the same object (such as the volumetric point sets obtained from Computed Tomography Scans) [SK15]. It is an important advantage the fact that a 2-manifold structure (i.e. non self-intersecting surface) might be recognized as underlying the point sets. The present publication refers to registration between a point set which is optically sampled on an object surface vs. a triangular mesh (i.e. a planar triangular graph) obtained from a CAD representation of the object. The problem of point-cloud-to-mesh registration is relevant in CAD CAM CAE applications where the CAD (or triangular mesh) model of the object to register is known a priori. These applications include (but are not limited to) Dimensional Inspection [SSB18, MPSRS*19] and Robotic Bin Picking [BG10].

Within point-cloud-to-mesh registration, the sub-problem of point-cloud-to-mesh distance is central and heavily contributes to the computing expenditure. For the later problem, existing literature relies on spatial partition structures (such as octrees or kd-trees), which produce logarithmic search times. Given the massive amount of points of the sets to be registered, it is of interest to find a more economic strategy. Therefore, this manuscript presents the implementation of a point-cloud-to-mesh registration algorithm based on a Spatial Hashing data structure. This Spatial Hashing structure provides constant time access ($\mathcal{O}(1)$) to the list of close triangles to a given point **p**. Consequently, the point-cloud-to-mesh registration based on Perfect Spatial Hashing is significantly faster than its hierarchical-based counterparts for massive point sets.

† Corresponding Author

DOI: 10.2312/ceig.20191202

In this manuscript, Section 2 presents a literature review of relevant approaches. Section 3 conveys the methodology applied. Section 4 discusses the results obtained with several data sets. Section 5 concludes the manuscript and mentions possible related future enhancements to the present approach.

2. Literature Review

The problem of point cloud registration has gotten a lot of research interest due to its relevancy in many engineering areas. Refs. [PCS15, TCL*13] present a survey on point cloud registration algorithms. The Iterative Closest Point (ICP) algorithm is one of the most widely used method for mesh registration in such literature. The algorithm consists of computing the closest points (correspondences) between the point cloud to register and the reference mesh. Such a procedure is performed iteratively until a convergence criteria is met [BM92]. The ICP extends the quaternion method [Hor87] for correspondent point-to-point registration.

To avoid local minima, the ICP requires the point-cloud-to-register and the reference mesh to be locally close enough. User-assisted alignment of correspondences is used to compute a pre-registration of the point cloud, which is finally registered by the ICP [SSB18]. Other ICP variations include feature-based mesh registration, in which some key points are automatically matched between the point-cloud-to-register and the reference mesh [PCS15]. These feature-based registration methods rely on spherical harmonics [SLW02] or surface signatures [YF02].

The main problem with ICP registration is the computation of correspondences (set of closest points from the reference mesh to the point cloud to register). The most naive approach is the exhaustive search, which is quadratic in time complexity $\mathcal{O}(N_Y \times N_T)$ (N_Y is the point-cloud-to-register size and N_T is the number of triangles in the reference mesh). Thus, spatial partitions of the domain are usually used to reduce the computational cost of the registration. Approaches to such spatial partitions include kd-trees [WGG11], heuristic search [JH02], R-trees [GZZ*12] and octrees [EBN13], whose search complexity becomes $\mathcal{O}(N_Y \log(N_T))$. Refs. [DI13, DI18] use 1-D hash tables to index octree entries, reducing the octree search to $\mathcal{O}(N_Y \log(\log(N_T)))$. Ref. [YB07] computes a regular grid that encloses the reference mesh, reducing the registration search complexity to linear $\mathcal{O}(N_Y)$. However, this last approach demands excessive storage resources as the full rectangular grid needs to be stored.

Other algorithms for cloud-to-mesh registration have been presented in the literature. RANdom SAmple Consensus (RANSAC) is a registration algorithm which takes many different sets of samples from the point cloud to register, and then fits a different model to each of these sets. The algorithm returns the best fitted model according to the optimization criteria [FRS07]. The Normal Distribution Transform (NDT) algorithm computes a 3D grid enclosing the point cloud to register and the reference mesh, which are used to compute a spatial probability distribution function. The registration of the obtained probability functions is performed using the Hessian matrix method [UT11]. RANSAC and NDT methods have shown to perform faster than standard ICP methods. However, their result is non-deterministic and highly sensitive to algorithm parameters. A full review on mesh registration algorithms is presented in [PCS15, CCL*18].

2.1. Conclusions of the Literature Review

Current mesh registration algorithms rely on spatial partitions of the 3D domain to search the cloud-to-mesh closest points. Most of these algorithms are linear-logarithmic. Table 1 summarizes the mesh registration algorithms presented in the literature with their respective time complexity.

Table 1: *Summary of closest point search algorithms in the literature. N_Y is the point-cloud-to-register size and N_T is the reference mesh size.*

Reference	Computational Complexity
K-d tree search [WGG11]	$\mathcal{O}(N_Y \log(N_T))$
Heuristic search [JH02]	$\mathcal{O}(N_Y \log(N_T))$
R-tree search [GZZ*12]	$\mathcal{O}(N_Y \log(N_T))$
Octree search [EBN13]	$\mathcal{O}(N_Y \log(N_T))$
Hash-Octree search [DI13, DI18]	$\mathcal{O}(N_Y \log(\log(N_T)))$
Cubic grid search [YB07]	$\mathcal{O}(N_Y)$
Perfect Spatial Hash (this manuscript)	$\mathcal{O}(N_Y)$

To overcome these problems, this manuscript presents the integration and implementation of a Perfect Spatial Hashing [LH06] data structure into the ICP registration process. Given a point to be registered, the Perfect Spatial Hashing defines a hash function which returns the closest point from the reference mesh in constant time. As a consequence, the complexity of our registration algorithm is $\mathcal{O}(N_Y \times n)$, improving previous spatial partition approaches. In contrast to the discretization presented in [YB07], the Spatial Hash partition reduces significantly the storage requirements of the data structure, as the Hash table is optimized to reach the smallest size possible, at the cost of some pre-processing time.

3. Methodology

Given a point cloud to register $\mathbf{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{N_Y}\}$ and a reference triangle mesh $\mathcal{M} = (\mathcal{T}, \mathbf{P})$ ($\mathcal{T} = \{t_1, t_2, \ldots, t_{N_T}\}$, $\mathbf{P} = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_{N_P}\}$), the mesh registration problem consists of finding a rigid transformation (rotation $\mathbf{R} \in SO(3)$ and translation $\mathbf{p}_0 \in \mathbb{R}^3$) that minimizes the distance between the point cloud $\mathbf{Y}$ and the reference mesh $\mathcal{M}$:

$$\min_{\mathbf{R},\mathbf{p}_0} \sum_{i=1}^{N_Y} d(\mathbf{R}\mathbf{y}_i + \mathbf{p}_0, \mathcal{M})^2 \tag{1}$$

where $d(\mathbf{y}_i^*, \mathcal{M})$ is shortest distance between the registered point $\mathbf{y}_i^*$ and the mesh $\mathcal{M}$. The registered point cloud is the set of points $\mathbf{Y}^* = \{\mathbf{y}_1^*, \mathbf{y}_2^*, \ldots, \mathbf{y}_{N_Y}^*\}$ such that $\mathbf{y}_i^* = \mathbf{R}\mathbf{y}_i + \mathbf{p}_0$.

The following sections describe the Iterative Closest Point (ICP) algorithm [BM92] that solves the above minimization problem and the integration of Perfect Spatial Hashing [LH06] in the registration process.

3.1. Mesh Registration of Correspondences

Let $\mathbf{x}_i \in \mathcal{M}$ be the closest point to the registered point $\mathbf{y}_i^*$ (see Fig. 1). The set $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{N_Y}\}$ is a resample of $\mathcal{M}$, known as the set of correspondences of $\mathbf{Y}$. As a consequence, Eq. 1 becomes:

$$\min_{\mathbf{R},\mathbf{p}_0} \sum_{i=1}^{N_Y} \|\mathbf{R}\mathbf{y}_i + \mathbf{p}_0 - \mathbf{x}_i\|^2 \tag{2}$$

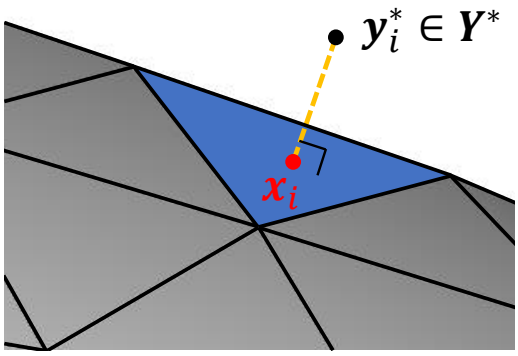


Figure 1: *Registered point $\mathbf{y}_i^*$ and its correspondent (closest) point $\mathbf{x}_i \in \mathcal{M}$. $\mathbf{x}_i$ does not belong to the original discretization of $\mathcal{M}$.*

It is worth noting that $\mathbf{X}$ and $\mathbf{Y}$ share the same number of points ($|\mathbf{X}| = |\mathbf{Y}| = N_Y$) and the set $\mathbf{X}$ does not contain the same points as the initial discretization of $\mathcal{M}$ (i.e. $\mathbf{X} \neq \mathbf{P}$). In addition, since the solution $\mathbf{Y}^*$ is an unknown of the problem, the set $\mathbf{X}$ is not known a priori. However, an estimation of $\mathbf{X}$ can be computed using the initial point set $\mathbf{Y}$. Such an estimation is discussed later in this section.

The minimization problem presented in Eq. 2 becomes the following maximization problem [BM92]:

$$\max_{\mathbf{R}} \sum_{i=1}^{N_Y} (\mathbf{y}_i - \mu_\mathbf{y})^T \mathbf{R} (\mathbf{x}_i - \mu_\mathbf{x}) \tag{3}$$

and the optimal solution to $\mathbf{p}_0$ becomes:

$$\mathbf{p}_0 = \mu_\mathbf{x} - \mathbf{R}\mu_\mathbf{y} \tag{4}$$

where $\mu_\mathbf{x} \in \mathbb{R}^3$ and $\mu_\mathbf{y} \in \mathbb{R}^3$ are the centroids of the point clouds $\mathbf{X}$ and $\mathbf{Y}$, respectively. Let $\mathbf{S}$ be the 3×3 cross-covariance matrix between $\mathbf{X}$ and $\mathbf{Y}$, defined as follows:

$$\mathbf{S} = \sum_{i=1}^{N_Y} (\mathbf{x}_i - \mu_\mathbf{x})(\mathbf{y}_i - \mu_\mathbf{y})^T \tag{5}$$

The rotation $\mathbf{R}$ can be expressed as a unit quaternion $\dot{\mathbf{q}} \in \mathbb{R}^4$, $\|\dot{\mathbf{q}}\| = 1$. Using quaternion algebra [Hor87], Eq. 3 becomes:

$$\max_{\|\dot{\mathbf{q}}\|=1} \sum_{i=1}^{N_Y} \dot{\mathbf{q}}^T \mathbf{Q}_i \dot{\mathbf{q}} = \max_{\|\dot{\mathbf{q}}\|=1} \dot{\mathbf{q}}^T \mathbf{Q} \dot{\mathbf{q}} \tag{6}$$

where $\dot{\mathbf{q}} \in \mathbb{R}^4$ is the unit quaternion ($\|\dot{\mathbf{q}}\| = 1$) representation of $\mathbf{R}$ and $\mathbf{Q}_i$ is the 4×4 symmetric matrix associated to the cross-covariance $(\mathbf{x}_i - \mu_\mathbf{x})(\mathbf{y}_i - \mu_\mathbf{y})^T$. The matrix $\mathbf{Q}$ ($\mathbf{Q} = \sum_i \mathbf{Q}_i$) is defined in terms of the cross-covariance matrix $\mathbf{S}$ as follows [Hor87]:

$$\mathbf{Q} = \begin{bmatrix} S_{00}+S_{11}+S_{22} & S_{12}-S_{21} & S_{20}-S_{02} & S_{01}-S_{10} \\ S_{12}-S_{21} & S_{00}-S_{11}-S_{22} & S_{01}+S_{10} & S_{02}+S_{20} \\ S_{20}-S_{02} & S_{01}+S_{10} & S_{11}-S_{22}-S_{00} & S_{12}+S_{21} \\ S_{01}-S_{10} & S_{02}+S_{20} & S_{12}+S_{21} & S_{22}-S_{00}-S_{11} \end{bmatrix} \tag{7}$$

Finally, Eq. 6 has the form of a Rayleigh quotient, thus becoming an eigenvector problem. The optimal rotation $\dot{\mathbf{q}}$ that registers the set of correspondences $\mathbf{X}, \mathbf{Y}$ is the eigenvector of the matrix $\mathbf{Q}$, corresponding to its largest eigenvalue.

3.2. Iterative Closest Point

As previously discussed, the set of correspondences $\mathbf{X}$ is not known a priori since the solution $\mathbf{y}_i^* = \mathbf{R}\mathbf{y}_i + \mathbf{p}_0$ is not known. The ICP algorithm [BM92] proposes to estimate a sequence of correspondences $\mathbf{X}^{(k)}$ based on a previous known point cloud $\mathbf{Y}^{(k-1)}$. The correspondent point $\mathbf{x}_i^{(k)} \in \mathcal{M}$ is the closest point in $\mathcal{M}$ to the point $\mathbf{y}_i^{(k-1)}$:

$$\mathbf{x}_i^{(k)} = \arg\min_{\mathbf{x} \in \mathcal{M}} \|\mathbf{x} - \mathbf{y}_i^{(k-1)}\| \tag{8}$$

In Eq. 8 it is reasonable to assume that $\|\mathbf{x}_i^{(k)} - \mathbf{y}_i^{(k-1)}\| < \Delta$, with $\Delta > 0$ being a distance threshold. This assumption means that the point cloud $\mathbf{Y}^{(k-1)}$ is locally close enough to the reference mesh $\mathcal{M}$ (i.e., $d(\mathbf{y}_i^{(k-1)}, \mathcal{M}) < \Delta$). Any point $\mathbf{y}_i^{(k)}$ not satisfying such assumption is discarded from $\mathbf{Y}^{(k-1)}$. Such an assumption is made in order to: (1) avoid falling in local minima and, (2) filter outliers from $\mathbf{Y}^{(k-1)}$ [BM92]. Other methods already presented in the literature can be used as a pre-processing to guarantee that most of the points in $\mathbf{Y}$ satisfy the previous assumption before our algorithm starts [SSB18].

With such a set of correspondences, it is possible to solve the optimization problem presented in Eq. 2, which becomes:

$$\min_{\mathbf{R}^{(k)},\mathbf{p}_0^{(k)}} \sum_{i=1}^{N_Y} \|\mathbf{R}^{(k)}\mathbf{y}_i^{(k-1)} + \mathbf{p}_0^{(k)} - \mathbf{x}_i^{(k)}\|^2 \tag{9}$$

where $\mathbf{R}^{(k)} \in SO(3)$, $\mathbf{p}_0^{(k)} \in \mathbb{R}^3$ originate the rigid transformation at the current iteration k. Finally, the point cloud $\mathbf{Y}^{(k)}$ is updated by using the obtained transformation:

$$\mathbf{y}_i^{(k)} = \mathbf{R}^{(k)}\mathbf{y}_i^{(k-1)} + \mathbf{p}_0^{(k)} \tag{10}$$

The sequences $\mathbf{Y}^{(k)}$, $\mathbf{R}^{(k)}$ and $\mathbf{p}_0^{(k)}$ have been proved to converge to the optimal solution $\mathbf{Y}^*$, $\mathbf{R}$ and $\mathbf{p}_0$, respectively [BM92]:

$$\begin{aligned} \mathbf{y}_i^* &= \lim_{n\to\infty} \mathbf{y}_i^{(n)} \\ \mathbf{R} &= \lim_{n\to\infty} \prod_{i=0}^{n} \mathbf{R}^{(k)} \\ \mathbf{p}_0 &= \lim_{n\to\infty} \left[\sum_{k_1=0}^{n-1} \left(\prod_{k_2=k_1+1}^{n} \mathbf{R}^{(k_2)} \right) \mathbf{p}_0^{(k_1)} \right] + \mathbf{p}_0^{(n)} \end{aligned} \tag{11}$$

The ICP works iterating over $k = 1, 2, \ldots, n$ for the previous sequences, until either one of the following criteria is satisfied:

1. Max. number of iterations n reached.
2. Approximation error below a given threshold $(\sum_i \frac{\|\mathbf{y}_i^{(k)} - \mathbf{y}_i^{(k-1)}\|^2}{N_Y} < \varepsilon)$

The algorithm is initialized from the original point cloud $\mathbf{Y}^{(0)} = \mathbf{Y}$, and the identity transformation $\mathbf{R}^{(0)} = \mathbf{I}_{3\times3}$, $\mathbf{p}_0^{(0)} = \mathbf{0}_{3\times1}$. Fig. 2 summarizes the mesh registration algorithm. The most expensive procedure in the ICP algorithm is the computation of the cloud-to-mesh distance (steps 4 and 5), which computed by an exhaustive search drives the complexity of the registration to $\mathcal{O}(N_Y \times N_T \times n)$, with N_Y being the point cloud size, N_T being the number of triangles in the mesh $\mathcal{M}$ and n being the maximum number of ICP iterations. It is common in the literature to use hierarchical partition structures (such as kd-trees and octrees) which improve such a search to $\mathcal{O}(N_Y \log(N_T) \times n)$. Our registration algorithm implements instead a Perfect Spatial Hashing strategy (step 1), whose search complexity is constant ($\mathcal{O}(1)$) [LH06]. As a consequence, the overall time complexity of our mesh registration algorithm becomes $\mathcal{O}(N_Y \times n)$. The following sections discuss the construction of the Spatial Perfect Hash and the distance computation.

3.3. Perfect Spatial Hash

Given a triangular mesh $\mathcal{M} \subset \mathbb{R}^3$, consider $\mathcal{V} \subset \mathcal{P}(\mathbb{R}^3)$ ($\mathcal{P}(\cdot)$ is the power set) as a rectangular prism, oriented along the coordinate axes, which contains $\mathcal{M}$ and is the union of small (disjoint) cubic cells (voxels v_{ijk}) of side length Δ (Fig. 3):

$$\mathcal{V} = \{v_{ijk} | i \in [0, N_V) \wedge j = [0, N_V) \wedge k \in [0, N_V)\} \quad (12)$$

where each voxel v_{ijk} is also oriented along the coordinate axes, and the interiors of two different voxels never intersect.

The size of the previous spatial partition is $|\mathcal{V}| = N_V^3$, with $i < N_V$, $j < N_V$ and $k < N_V$ being the 3D indices of each voxel. Define $D(v_{ijk})$ as the triangles of $\mathcal{M}$ that intersect v_{ijk} (Fig. 4), i.e.:

$$D(v_{ijk}) = \{t \in \mathcal{T} | t \cap v_{ijk} \neq \emptyset\} \quad (13)$$

Finally, the set $\mathcal{V}_M \subset \mathcal{V}$ is the set of voxels $v_{ijk} \in \mathcal{V}$ that intersect at least one triangle of $\mathcal{M}$, i.e. $\mathcal{V}_M = \{v_{ijk} \in \mathcal{V} | D(v_{ijk}) \neq \emptyset\}$. It is worth noting that the set size $|\mathcal{V}_M|$ is much smaller than the full grid size $|\mathcal{V}|$ (Fig. 3).

A Perfect Spatial Hash table $\mathbf{H} : \mathbb{N}^3 \rightarrow \mathcal{P}(\mathcal{T})$, is a 3D table with indices h_i, h_j, h_k. Each entry $\mathbf{H}[h_i, h_j, h_k]$ contains the set of triangles associated to the voxel $\mathbf{h}^{-1}(h_i, h_j, h_k)$, i.e.:

$$\mathbf{H}[\mathbf{h}(v_{ijk})] = \mathbf{H}[h_i, h_j, h_k] = D(v_{ijk}) \quad (14)$$

where $\mathbf{h} : \mathcal{V}_M \rightarrow \mathbb{N}^3$ is a function which takes a voxel v_{ijk} and returns its respective position indices h_i, h_j, h_k in the Hash table $\mathbf{H}$. $\mathbf{h}$ is known as the hash function of $\mathbf{H}$. The Perfect Spatial Hash is denoted as $(\mathbf{H}, \mathbf{h})$.

The objective of the Perfect Spatial Hash is to produce a table $\mathbf{H}$ which stores the information $\mathcal{V}_M$, and its respective hash function $\mathbf{h}$. A trivial hash function would be the identity function $\mathbf{h}(v_{ijk}) = [i, j, k]$ (implicitly used by [YB07]). However, such a function implies storing the full rectangular prism $\mathcal{V}$ in the table $\mathbf{H}$ ($|\mathbf{H}| = |\mathcal{V}| >> |\mathcal{V}_M|$), and the content of most of the table cells would be empty (most cells of $\mathcal{V}$ are empty, Fig. 3). Instead, the Perfect Spatial Hash [LH06] aims to produce the smallest table $\mathbf{H}$ possible able to store the set $\mathcal{V}_M$, such that $|\mathcal{V}_M| \leq |\mathbf{H}| << |\mathcal{V}|$ (ideally, $|\mathbf{H}| = |\mathcal{V}_M|$).

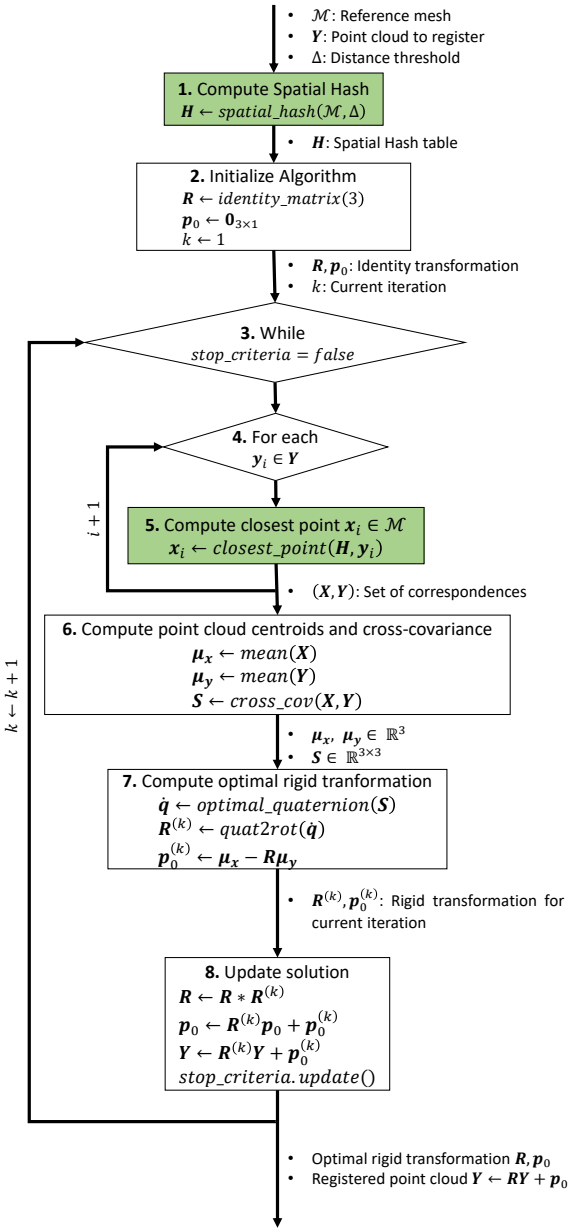


Figure 2: *Scheme of the Iterative Closest Point mesh registration algorithm. Our registration uses Perfect Spatial Hashing to compute the cloud-to-mesh distances.*

The Perfect Spatial Hashing $(\mathbf{H}, \mathbf{h})$ satisfies by definition the following conditions:

1. The function $\mathbf{h}$ is bijective. As a consequence, there are no col-

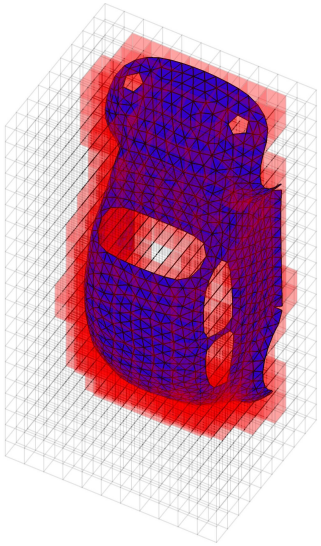


Figure 3: *Full min-max voxel set $\mathcal{V}$ (gray). Non triangle-empty voxel set $\mathcal{V}_M$ (red). Triangle mesh $\mathcal{M}$ (blue). $|\mathcal{V}_M| << |\mathcal{V}|$.*

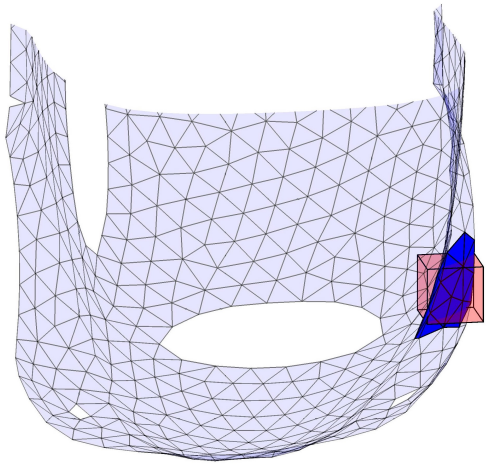


Figure 4: *Set of triangles $D(v_{ijk})$ (dark blue) that intersect the voxel v_{ijk} (red)*

lisions in the table $\mathbf{H}$ (i.e. different voxels in $\mathcal{V}_M$ never point to the same cell of $\mathbf{H}$).
2. The size of $\mathbf{H}$ is greater or equal than the size of $\mathcal{V}_M$ ($|\mathbf{H}| > |\mathcal{V}_M|$).

In addition, $(\mathbf{H},\mathbf{h})$ should satisfy (by construction) the following conditions:

1. The size of $\mathbf{H}$ is smaller than the size of $\mathcal{V}$ ($|\mathbf{H}| < N_V^3$).
2. Evaluation of the hash function $\mathbf{h}$ should be $\mathcal{O}(1)$.

The first step to build the Spatial Hash $(\mathbf{H},\mathbf{h})$ is to compute the table size $|\mathbf{H}| = N_H^3$, as the smallest table size able to store the set $\mathcal{V}_M$:

$$N_H = \arg\min_{N_H \in \mathbb{N}} |\mathcal{V}_M| \leq N_H^3 \tag{15}$$

The hash function $\mathbf{h}$ is then defined as a sum of an auxiliar function $\mathbf{f}$ and a displacement Φ [LH06]:

$$\mathbf{h}(v_{ijk}) = \mathbf{f}(v_{ijk}) + \Phi[g(v_{ijk})] \tag{16}$$

The auxiliar function $\mathbf{f} : \mathcal{V}_M \to \mathbb{N}^3$ is defined as:

$$\mathbf{f}(v_{ijk}) = [f_i, f_j, f_k] = [i, j, k] \mod N_H \tag{17}$$

By taking the modulo of each of the voxel indices, the values of the function $\mathbf{f}$ are guaranteed to never exceed the size of the Hash table $\mathbf{H}$ (i.e. $f_i < N_H$, $f_j < N_H$ and $f_k < N_H$). The function $\mathbf{f}$ is not bijective as $N_H \leq N_V$. As a consequence, an auxiliar 3D table Φ is computed as follows:

Let $\Phi \circ \mathbf{g} : V_M \to \mathbb{N}^3$ be an (auxiliar) 3D table of size $N_\Phi^3, N_\Phi \neq N_H$, and its corresponding auxiliar function $\mathbf{g} : \mathcal{V}_M \to \mathbb{N}^3$. The objective of the table $(\Phi,\mathbf{g})$ is to provide a translation term $\Phi[\mathbf{g}(v_{ijk})] = [\phi_i, \phi_j, \phi_k]$ such that $\mathbf{f}(v_{ijk}) + \Phi[\mathbf{g}(v_{ijk})]$ is bijective, guaranteeing that there are no collisions in $\mathbf{H}$.

Similar to the auxiliar function $\mathbf{f}$, the function $\mathbf{g}$ is defined as:

$$\mathbf{g}(v_{ijk}) = [g_i, g_j, g_k] = [i, j, k] \mod N_\Phi \tag{18}$$

where $g_i < N_\Phi$, $g_j < N_\Phi$ and $g_k < N_\Phi$ indicate the position of the voxel v_{ijk} in the auxiliar table Φ, i.e. $[\phi_i, \phi_j, \phi_k] = \Phi[g_i, g_j, g_k]$. It is worth noting that, by construction, $\mathbf{f} \neq \mathbf{g}$ (since $N_\Phi \neq N_H$).

Fig. 5 illustrates the aforementioned translation Φ. In the example, the non-empty voxels v_{11} and v_{33} map to the same $\mathbf{f}$ value. However, the same voxels map to a different $\mathbf{g}$ value. The Φ table stores the respective translations $\phi_{11} = [0,0]$ and $\phi_{33} = [1,1]$. The Perfect Hash Table presents no collisions as the hash function is bijective ($\mathbf{h}_{11} = [1,1]$, $\mathbf{h}_{33} = [0,0]$).

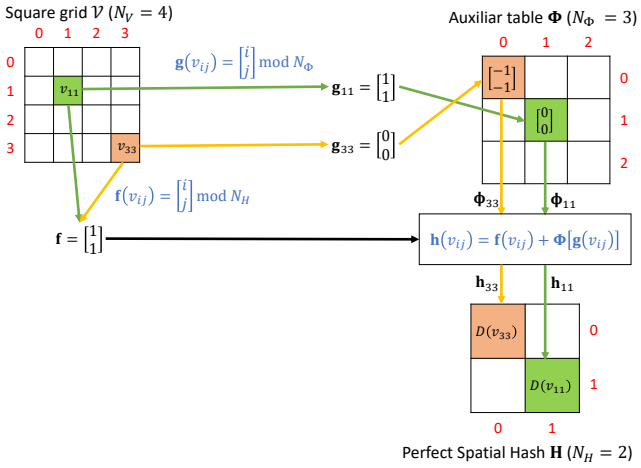


Figure 5: *Perfect Spatial Hash 2D example. The auxiliar function $\mathbf{f}$ is not bijective, but the Hash function $\mathbf{h}$ is.*

The table Φ and its size N_Φ is computed using an heuristic approach as described in Ref. [LH06], as follows:

1. Locate all collisions in $\mathbf{f}$.
2. Initialize the size of Φ as $N_\Phi \leftarrow \text{ceil}(\sqrt[3]{|\mathcal{V}_M|/6})$.

3. Initialize Φ as an empty N_Φ^3 3D table.
4. Locate all free indices of $\mathbf{f}$ (i.e. $\mathbf{f}(v_{ijk})$ is undefined).
5. For each collision $\mathbf{f}(v_{ijk})$, set $\Phi[\mathbf{g}(v_{ijk})]$ as $\mathbf{c} - \mathbf{f}(v_{ijk})$, where $\mathbf{c} = [c_i, c_j, c_k] \in \mathbb{N}^3$ is a free index in $\mathbf{f}$.
6. If there are no collisions in $\mathbf{f} + \Phi$, return Φ.
7. Otherwise, increase N_Φ and go to step 3.

In the previous heuristic, it is worth noting that there is no theoretical guarantee that the computed Perfect Spatial Hash ($\mathbf{H}$ and Φ) is smaller than the full grid $\mathcal{V}$. In fact, it is possible that $|\mathbf{H}| + |\Phi|$ is larger than $|\mathcal{V}|$. However, our experiments and the experiments presented in [LH06] have shown that the Perfect Spatial Hash is always smaller than the full grid discretization (i.e., $|\mathbf{H}| + |\Phi| < |\mathcal{V}|$).

After Φ, $\mathbf{h}$ and N_H have been computed, the table $\mathbf{H}$ is filled with the elements of the set $\mathcal{V}_M$. At this point, the function $\mathbf{h}$ is guaranteed to be bijective and as a consequence, $\mathbf{H}$ presents no collisions. Fig. 6 summarizes the algorithm to compute the Perfect Spatial Hashing. Note that if the reference mesh $\mathcal{M}$ slightly changes (due to a small rigid transformation or shape deformation), the Perfect Spatial Hash table changes dramatically, requiring to rebuild it from scratch. However, since our registration algorithm assumes that $\mathcal{M}$ does not change at any time, the aforementioned problem is out of the scope of this research.

For the computation of the set of voxels that intersect the triangulation (i.e. $\mathcal{V}_M$), our algorithm visits each triangle of the mesh as illustrated in steps 2-3 of Fig. 6. The triangle-voxel intersection for each $t_i \in \mathcal{T}$ is implemented as follows: (1) all the voxels that intersect the bounding box of t_i are identified and then, (2) all the voxels inside the bounding box, which also intersect the plane defined by t_i are kept, discarding the non-intersecting ones.

From the algorithm presented in Fig. 6, steps 2-3 are $\mathcal{O}(N_T)$, steps 7-8 are $\mathcal{O}((N_H^3)^2)$ and steps 10-11 are $\mathcal{O}(N_H^3)$. Therefore, the computational cost for the Perfect Spatial Hash construction is $\mathcal{O}(N_T + (N_H^3)^2)$. Such a cost becomes reasonable for large point cloud and reference mesh sizes as this pre-processing is performed only once. In addition, the storage complexity of the Perfect Spatial Hash is $\mathcal{O}(N_H^3 + N_\Phi^3)$, which is considerably less expensive than storing the full grid $\mathcal{O}(N_V^3)$ (such as in Ref. [YB07]).

3.4. Point-to-mesh Distance Computation

Given a point $\mathbf{y}_i \in \mathbf{Y}$, it is necessary to locate its closest point $\mathbf{x}_i \in \mathcal{M}$ (as per Eq. 8, Fig. 1). This problem is equivalent to find the closest triangle $t \in \mathcal{T}$ to $\mathbf{y}_i$, and then find the closest point $\mathbf{x}_i \in t$ to $\mathbf{y}_i$, as described below.

Given any triangle $t \in \mathcal{T}$, the distance from a point $\mathbf{y}_i \in \mathbf{Y}$ to t is defined as follows:

$$d(\mathbf{y}_i, t) = \min_{\alpha,\beta \in \mathbb{R}} \|\alpha\mathbf{q}_0 + \beta\mathbf{q}_1 + (1-\alpha-\beta)\mathbf{q}_2 - \mathbf{y}_i\| \quad s.t. \quad \alpha+\beta \leq 1, \quad \alpha,\beta \geq 0 \tag{19}$$

where $\mathbf{q}_0$, $\mathbf{q}_1$ and $\mathbf{q}_2$ are the vertices of the triangle t, and α, β, $(1-\alpha-\beta)$ are their corresponding barycentric coordinates, respectively. Therefore, the closest point $\mathbf{q}^* \in t$ to $\mathbf{y}_i$ is defined as the point $\mathbf{q}^* = \alpha\mathbf{q}_0 + \beta\mathbf{q}_1 + (1-\alpha-\beta)\mathbf{q}_2$ that minimizes Eq. (19). The closest point $\mathbf{x}_i \in \mathcal{M}$ to $\mathbf{y}_i$ is defined as:

$$\mathbf{x}_i = \arg \min_{\mathbf{q}^* \in \mathcal{M}} \|\mathbf{y}_i - \mathbf{q}^*\| \tag{20}$$

A naive evaluation of Eq. 20 requires searching the closest triangle t through the full mesh $\mathcal{M}$. However, the Perfect Spatial Hash $\mathbf{H}$ reduces such an evaluation by only requiring to evaluate triangles that are already close to $\mathbf{y}_i$. Let $v_{jkl} \in \mathcal{V}$ be the voxel that contains

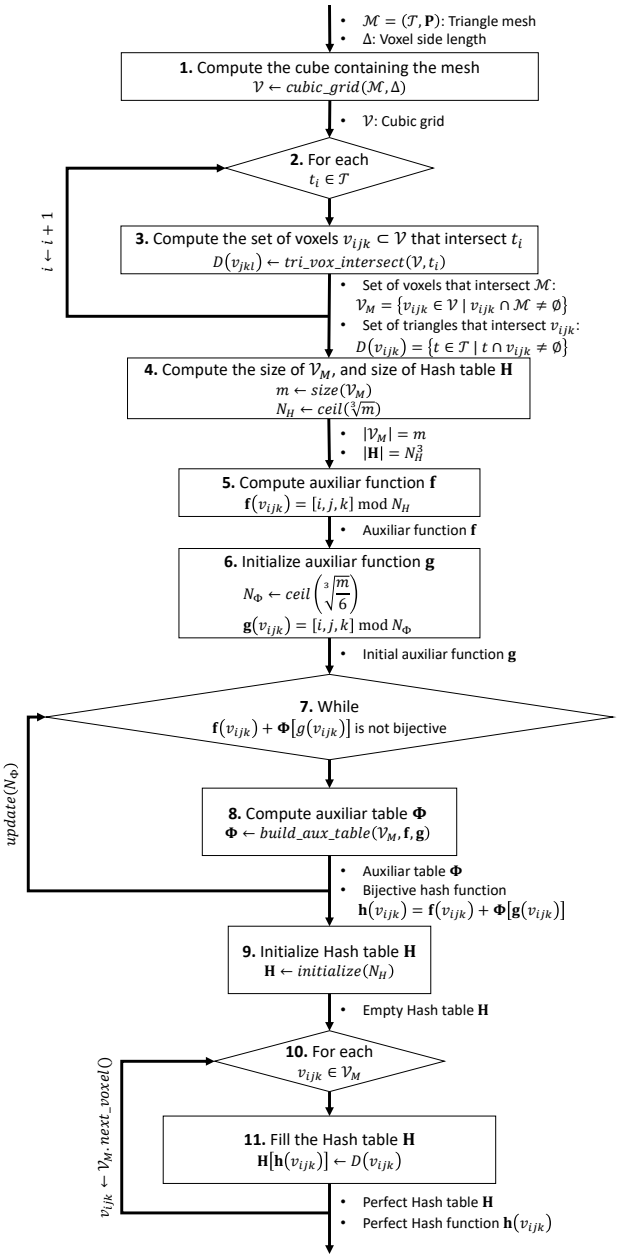


Figure 6: *Algorithm scheme for the construction of the Perfect Spatial Hash* $(\mathbf{H}, \mathbf{h})$

the point $\mathbf{y}_i$. The Hash cell $\mathbf{H}[\mathbf{h}(v_{jkl})]$ stores the set of triangles $D(v_{jkl})$ that intersect v_{jkl} (as illustrated in Fig. 4).

Let $B_{jkl} \subset \mathcal{V}$ be the set of adjacent voxels to v_{jkl} (v_{jkl} included). The set of closest triangles to $\mathbf{y}_i$ can be extracted from the intersection between B_{jkl} and $\mathcal{M}$, i.e. the set $\mathbf{H}[\mathbf{h}(B_{jkl})]$ (see Fig. 7). Therefore, Eq (20) is equivalent to:

$$\mathbf{x}_i = \arg \min_{\mathbf{q}^* \in \mathbf{H}[\mathbf{h}(B_{jkl})]} \|\mathbf{y}_i - \mathbf{q}^*\| \tag{21}$$

where clearly $|\mathbf{H}[\mathbf{h}(B_{jkl})]| << \mathcal{T}$. Since each voxel side size is Δ, the set B_{jkl} is guaranteed to contain a triangle whose distance to $\mathbf{y}_i$ is less than Δ (if such triangle exists in $\mathcal{M}$). It is worth noting that if such triangle does not exist, then $d(y_i, \mathcal{M}) > \Delta$, and the registration algorithm treats $\mathbf{y}_i$ as an outlier (as discussed at the beginning of Sect. 3.2) [BM92].

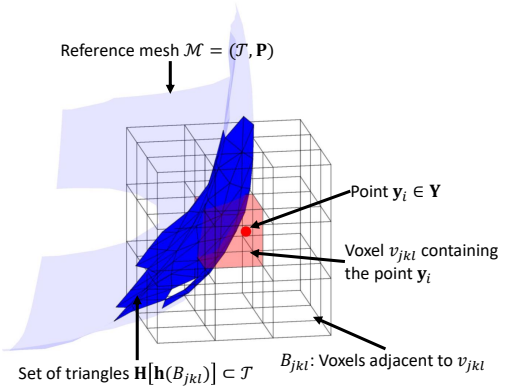


Figure 7: *The closest point of $\mathcal{M}$ to $\mathbf{y}_i$ is in the set B_{jkl} ($|B_{jkl}| << |\mathcal{T}|$). B_{jkl} is the set of triangles that intersect v_{jkl} and all its adjacent voxels.*

The algorithm for computing the closest point $\mathbf{x}_i$ is summarized as follows:

1. Compute the voxel v_{jkl} that contains the point to register $\mathbf{y}_i$ (i.e., $\mathbf{y}_i \in v_{jkl}$).
2. Compute the set of voxels B_{jkl}, adjacent to v_{jkl} (as illustrated in Fig. 7).
3. Compute the Hash indices $\mathbf{h}(B_{jkl})$ as per Eq. (16).
4. Extract from the Spatial Hash, the triangles $\mathbf{H}[\mathbf{h}(B_{jkl})]$ closest to $\mathbf{y}_i$ (Fig. 7).
5. Compute the closest triangle $t \in \mathbf{H}[\mathbf{h}(B_{jkl})]$ as per Eq. 19.
6. Compute $\mathbf{x}_i$ as per Eq. (21).

Since the evaluation of $\mathbf{h}$ in Eq. (16) and the access to the table $\mathbf{H}$ is $\mathcal{O}(1)$, the computational cost of the above algorithm is $\mathcal{O}(1)$.

4. Results

Four different models have been used to test our registration algorithm: Gargoyle, Dragon, Buddha and Lucy [CL96]. The point-cloud-to-register is extracted from the original model by computing a uniform re-sample of each model surface. Figs. 8(a), 8(c), 8(e) and 8(g) plot the unregistered point-clouds of each model, respectively. As mentioned in Sect. 3.2, the point-cloud-to-register should be close enough to the reference mesh to avoid falling into a local minima solution [BM92]. Figs. 8(b), 8(d), 8(f) and 8(h) plot the result of our registration process for each model, respectively. The registration algorithm minimizes the point-cloud-to-mesh distance as per Eq. (1).

Table 2 shows Spatial Hashing and ICP convergence results of our registration algorithm. The 4 point-clouds-to-register are of size $N_Y = 50k$, while the size of the reference meshes (N_T) is 20k, 871.4k, 1631.6k and 28055.7k for the Gargoyle, Dragon, Buddha and Lucy, respectively. The smallest Spatial Hash constructed is for the Gargoyle dataset, consisting of a $N_H^3 = 512$ Hash table and a $N_\Phi^3 = 1.3\text{k}^3$ auxiliar table, and the largest Spatial Hash is constructed for the Lucy ($N_H^3 = 5.8\text{k}^3$ Hash table and $N_\Phi^3 = 2.2\text{k}^3$ auxiliar table). The convergence error is measured as the difference between the last iteration and the previous iteration $\frac{\sum_i \|y_i^{(n)} - y_i^{(n-1)}\|^2}{N_Y}$, as discussed in Sect. 3.2. All the 4 test cases converge at 34, 19, 30 and 53 ICP iterations (n), respectively, with an error below 7e-05.

Table 2: *Perfect Spatial Hashing and ICP convergence results for the 4 datasets presented in Fig. 8*

Dataset	N_Y	N_T	N_H^3	N_Φ^3	n	$\frac{\sum_i \|\mathbf{y}_i^{(n)} - \mathbf{y}_i^{(n-1)}\|^2}{N_Y}$
Gargoyle	50k	20k	512	1.3k	34	6.20e-05
Dragon	50k	871.4k	2.1k	4.9k	19	5.87e-05
Buddha	50k	1631.6k	3.4k	1.3k	30	6.06e-05
Lucy	50k	28055.7k	5.8k	2.2k	53	5.97e-05

Table 3 presents the execution times for the registration of a $N_V^3 = 50$k point cloud to the Buddha mesh (Figs. 8(e), 8(f)). For a prism of size $N_V^3 = 4096$, the construction of the Hash table requires 0.032 minutes, while the ICP registration takes about 232.6 minutes to perform 31 iterations and converge to the solution. Increasing the prism resolution to $N_V^3 = 32.8k$, the construction of the Hash table requires 0.033 minutes while the registration takes 41.1 minutes to perform 30 iterations. In the case of a prism of size $N_V^3 = 2097.2k$, the construction of the Hash table and the mesh registration times are 0.06 and 2.6352 minutes, respectively, which is 15× faster than the $N_V^3 = 32.8k$ and 86× faster than the $N_V^3 = 4096$ test cases. Finally, the high resolution of the last test case ($N_V = 16777.2k$) implies that the voxel size Δ is significantly smaller than the average distance between the point cloud $\mathbf{Y}$ and the reference mesh $\mathcal{M}$, resulting in the registration algorithm exiting at 0 iterations without converging.

5. Conclusions

This manuscript presents the implementation of a Perfect Spatial Hash Hashing for point-cloud-to-mesh registration. The registration algorithm uses the Perfect Spatial Hashing data structure to aid the computation of point-to-mesh distance of the Iterative Closest Point (ICP) algorithm. Compared to standard spatial partition techniques (such as octrees and kd-trees), our algorithm reduces the closest-point-search complexity from logarithmic ($\mathcal{O}(\log(N_T))$, N_T: reference mesh size) to constant $\mathcal{O}(1)$ complexity. As a consequence, the cost of the mesh registration algorithm becomes $\mathcal{O}(N_Y \times n)$ (N_Y: point-cloud-to-register size, n: number of max. ICP iterations). The cost of pre-processing (pre-computation

Table 3: *Buddha dataset. Execution times for the construction of the Perfect Spatial Hashing* $(\mathbf{H},\mathbf{h})$ *and the registration of a* $N_Y = 50k$ *point cloud for different voxel resolutions* N_V*. Note that the performance for the registration significantly improves as* N_V *increases.*

N_V^3	N_H^3	N_Φ^3	n	Time to build $(\mathbf{H},\mathbf{h})$ (min)	Registration time (min)	Total time (min)
4096	343	216	31	0.0317	232.6	232.7
32.8k	1728	2197	30	0.0334	41.077	41.111
262.1k	8000	4913	32	0.0407	9.6378	9.6785
2097.2k	32.8k	19.7k	40	0.0587	2.6352	2.6939
16777.2k	140.6k	91.1k	0	0.2210	NA	NA

of the Perfect Spatial Hashing) is $\mathcal{O}(N_T + (N_H^3)^2)$ (N_H^3: Hash table size). Our algorithm is able to register a point cloud of size $N_Y = 50k$ against a mesh of size $N_T = 28055.7k$, converging with an error below 7e-05. We also show that the mesh registration algorithm improves significantly in performance as the Spatial Hashing resolution increases. However, if the voxel size Δ becomes too small (smaller than the average distance between the point cloud and the reference mesh), the registration algorithm fails.

5.1. Future Implementation on GPU

The main shortcoming of our point-cloud-to-mesh registration algorithm lies in the construction of the Perfect Spatial Hashing computational cost, as the worst case scenario complexity is squared in the size of the Hash table ($\mathcal{O}((N_H^3)^2)$, see Sect. 3.3). To mitigate this problem, we intend to implement Perfect Spatial Hash mesh registration in a Graphic Processing Unit (GPU) parallelization architecture. By taking advantage of Graphics Processing Units (GPUs), the Hash structure can be computed in a more efficient way, reducing the pre-processing time [LH06]. In addition, the independence in the computation of the closest point (Eq. (21)) between any two different points $\mathbf{y}_i, \mathbf{y}_j \in \mathbf{Y}$ permits an implementation following a highly parallelizable approach, resulting in fast registration of considerably larger point clouds.

Glossary

ICP: Iterative Closest Point.

$\mathcal{M}$: Triangular mesh $\mathcal{M} = (\mathcal{T},\mathbf{P})$ of a 2-manifold embedded in $\mathbb{R}^3$, defined by the triangle set $\mathcal{T} = \{t_1, t_2, \cdots, t_{N_T}\}$ and the point set $\mathbf{P} = \{\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_{N_P}\}$. $\mathcal{M}$ is the reference mesh for registration.

$\mathbf{Y}$: Point cloud to register $\mathbf{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_{N_Y}\}$. $\mathbf{Y}$ is a noisy sample of $\mathcal{M}$, conducted in an unknown coordinate system.

$\mathbf{R}, \mathbf{p}_0$: Rigid transformation $\mathbf{R} \in SO(3)$ (Special Orthogonal Group), $\mathbf{p}_0 \in \mathbb{R}^3$, that matches the coordinate system of $\mathbf{Y}$ to the coordinate system of $\mathcal{M}$.

$\mathbf{Y}^*$: Rigidly transformed point cloud $\mathbf{Y}^* = \{\mathbf{y}_1^*, \mathbf{y}_2^*, \cdots, \mathbf{y}_{N_Y}^*\}$, such that $\mathbf{y}_i^* = \mathbf{R}\mathbf{y}_i + \mathbf{p}_0$.

$\mathbf{X}$: Point cloud $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots \mathbf{x}_{N_Y}\}$ sampled from $\mathcal{M}$, such that $\mathbf{x}_i$ is the closest point in $\mathcal{M}$ to $\mathbf{y}_i$ ($|\mathbf{X}| = |\mathbf{Y}|$). $\mathbf{X}$ is the *set of correspondences* of $\mathbf{Y}$.

$\mu_\mathbf{x}$, $\mu_\mathbf{y}$: Centroids $\mu_\mathbf{x}, \mu_\mathbf{y} \in \mathbb{R}^3$ of the point sets $\mathbf{X}$ and $\mathbf{Y}$, respectively.

$\mathbf{S}$: 3×3 matrix of cross-covariances between $\mathbf{X}$ and $\mathbf{Y}$.

$\dot{\mathbf{q}}$: Unit quaternion $\dot{\mathbf{q}} \in \mathbb{R}^4$ ($\|\dot{\mathbf{q}}\| = 1$), equivalent to the rotation matrix $\mathbf{R}$.

$\mathbf{Y}^{(k)}, \mathbf{X}^{(k)}$: Values for the points sets $\mathbf{Y}$, $\mathbf{X}$ at the current ICP iteration k.

$\mathbf{R}^{(k)}, \mathbf{p}_0^{(k)}$: Values for the rigid transformation $\mathbf{R}$, $\mathbf{p}_0$ at the current ICP iteration k.

n: Maximal Number of iterations $n > 0$ allowed by the ICP algorithm.

Δ: Distance below which a point $\mathbf{y}_i \in \mathbf{Y}$ is not considered an outlier w.r.t. mesh $\mathcal{M}$ (i.e. $d(\mathbf{y}_i, \mathcal{M}) < \Delta$).

$\mathcal{P}(A)$: Power set of A, defined as all the subsets of A. $\mathcal{P}(A) = \{a | a \subset A\}$.

v_{ijk}: A cubic cell $(i, j, k) \in \mathbb{N}^3$, of side length Δ, oriented along the coordinate axes.

$\mathcal{V}$: Rectangular prism $\mathcal{V} \subset \mathcal{P}(\mathbb{R}^3)$ oriented along the coordinate axes, defined as a set of disjoint voxels v_{ijk} that build the bounding box of $\mathcal{M}$. $|\mathcal{V}| = N_V^3$.

$D(v_{ijk})$: Set of triangles in $\mathcal{M}$ that intersect voxel $v_{ijk} \in \mathcal{V}$. $D : \mathcal{V} \to \mathcal{P}(\mathcal{T})$.

$\mathcal{V}_M$: Set of voxels $v_{ijk} \in \mathcal{V}$ that intersect at least one triangle of $\mathcal{M}$ (i.e. $D(v_{ijk}) \neq \emptyset$).

$\mathbf{H}$: Perfect Spatial Hash table $\mathbf{H} : \mathbb{N}^3 \to \mathcal{P}(\mathcal{T})$. $\mathbf{H}$ is a 3D table where each entry $\mathbf{H}[h_i, h_j, h_k]$ stores a subset of triangles $D(v_{ijk})$. $|\mathbf{H}| = N_H^3$.

$\mathbf{h}$: (Bijective) Hash function $\mathbf{h} : \mathcal{V}_M \to \mathbb{N}^3$ of $\mathbf{H}$. $\mathbf{h}$ takes a voxel $v_{ijk} \in \mathcal{V}_M$ and returns the respective indices h_i, h_j, h_k in $\mathbf{H}$, such that $H[h_i, h_j, h_k] = D(v_{ijk})$.

$\mathbf{f}, \mathbf{g}$: Auxiliar functions $\mathbf{f}, \mathbf{g} : \mathcal{V}_M \to \mathbb{N}^3$ used by the function $\mathbf{h}$ to compute a bijective mapping.

Φ: Auxiliar 3D table table $\Phi : \mathbb{N}^3 \to \mathbb{N}^3$ used by the function $\mathbf{h}$ to compute a bijective mapping. $|\Phi| = N_\Phi^3$.

$\mathbf{q}_0, \mathbf{q}_1, \mathbf{q}_2$: Vertices of triangle $t_j \in \mathcal{T}$ with $\mathbf{q}_i \in \mathbf{P}$.

α, β: Barycentric coordinates on a triangle $t \in \mathcal{T}$ with $\alpha, \beta \geq 0$, and $\alpha + \beta \leq 1$.

B_{jkl}: Set $B_{jkl} \subset \mathcal{V}$ of all adjacent voxels to v_{jkl} (including v_{jkl}).

References

[BG10] BÖHNKE K., GOTTSCHEBER A.: Fast object registration and robotic bin picking. In *Research and Education in Robotics - EUROBOT 2009* (Berlin, Heidelberg, 2010), Gottscheber A., Obdržálek D., Schmidt C., (Eds.), Springer Berlin Heidelberg, pp. 23–37. 1

[BM92] BESL P. J., MCKAY N. D.: A method for registration of 3-d

shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence 14*, 2 (Feb 1992), 239–256. doi:10.1109/34.121791. 2, 3, 7

[CCL*18] CHENG L., CHEN S., LIU X., XU H., WU Y., LI M., CHEN Y.: Registration of laser scanning point clouds: A review. *Sensors (Basel) 18*, 5 (May 2018), 1641:1–1641:25. doi:10.3390/s18051641. 2

[CL96] CURLESS B., LEVOY M.: A volumetric method for building complex models from range images. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1996), SIGGRAPH '96, ACM, pp. 303–312. doi:10.1145/237170.237269. 7, 10

[DI13] DROST B., ILIC S.: A hierarchical voxel hash for fast 3d nearest neighbor lookup. In *Pattern Recognition* (Berlin, Heidelberg, 2013), Weickert J., Hein M., Schiele B., (Eds.), Springer Berlin Heidelberg, pp. 302–312. 2

[DI18] DROST B. H., ILIC S.: Almost constant-time 3d nearest-neighbor lookup using implicit octrees. *Machine Vision and Applications 29*, 2 (Feb 2018), 299–311. doi:10.1007/s00138-017-0889-4. 2

[EBN13] ELSEBERG J., BORRMANN D., NÜCHTER A.: One billion points in the cloud – an octree for efficient processing of 3d laser scans. *ISPRS Journal of Photogrammetry and Remote Sensing 76* (2013), 76–88. Terrestrial 3D modelling. doi:10.1016/j.isprsjprs.2012.10.004. 2

[FRS07] FONTANELLI D., RICCIATO L., SOATTO S.: A fast ransac-based registration algorithm for accurate localization in unknown environments using lidar measurements. In *2007 IEEE International Conference on Automation Science and Engineering* (Sep. 2007), pp. 597–602. doi:10.1109/COASE.2007.4341827. 2

[GZZ*12] GONG J., ZHU Q., ZHONG R., ZHANG Y., XIE X.: An efficient point cloud management method based on a 3d r-tree. *Photogrammetric Engineering & Remote Sensing 78*, 4 (2012), 373–381. doi:doi:10.14358/PERS.78.4.373. 2

[Hor87] HORN B. K. P.: Closed-form solution of absolute orientation using unit quaternions. *J. Opt. Soc. Am. A 4*, 4 (Apr 1987), 629–642. doi:10.1364/JOSAA.4.000629. 2, 3

[JH02] JOST T., HÜGLI H.: Fast icp algorithms for shape registration. In *Pattern Recognition* (Berlin, Heidelberg, 2002), Van Gool L., (Ed.), Springer Berlin Heidelberg, pp. 91–99. 2

[LH06] LEFEBVRE S., HOPPE H.: Perfect spatial hashing. *ACM Trans. Graph. 25*, 3 (July 2006), 579–588. doi:10.1145/1141911.1141926. 2, 4, 5, 6, 8

[MPSRS*19] MEJIA-PARRA D., SÁNCHEZ J. R., RUIZ-SALGUERO O., ALONSO M., IZAGUIRRE A., GIL E., PALOMAR J., POSADA J.: In-line dimensional inspection of warm-die forged revolution workpieces using 3d mesh reconstruction. *Applied Sciences 9*, 6 (2019). doi:10.3390/app9061069. 1

[PCS15] POMERLEAU F., COLAS F., SIEGWART R.: A review of point cloud registration algorithms for mobile robotics. *Found. Trends Robot 4*, 1 (May 2015), 1–104. doi:10.1561/2300000035. 2

[SK15] SAHILLIOĞLU Y., KAVAN L.: Skuller: A volumetric shape registration algorithm for modeling skull deformities. *Medical Image Analysis 23*, 1 (2015), 15–27. doi:https://doi.org/10.1016/j.media.2015.03.005. 1

[SLW02] SHARP G. C., LEE S. W., WEHE D. K.: Icp registration using invariant features. *IEEE Transactions on Pattern Analysis and Machine Intelligence 24*, 1 (Jan 2002), 90–102. doi:10.1109/34.982886. 2

[SSB18] SÁNCHEZ J. R., SEGURA Á., BARANDIARAN I.: Fast and accurate mesh registration applied to in-line dimensional inspection processes. *International Journal on Interactive Design and Manufacturing (IJIDeM) 12*, 3 (Aug 2018), 877–887. doi:10.1007/s12008-017-0449-1. 1, 2, 3

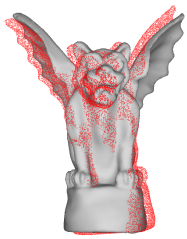
[TCL*13] TAM G. K. L., CHENG Z., LAI Y., LANGBEIN F. C., LIU Y., MARSHALL D., MARTIN R. R., SUN X., ROSIN P. L.: Registration of 3d point clouds and meshes: A survey from rigid to nonrigid. *IEEE Transactions on Visualization and Computer Graphics 19*, 7 (July 2013), 1199–1217. doi:10.1109/TVCG.2012.310. 2

[UT11] ULAS C., TEMELTAS H.: A 3d scan matching method based on multi-layered normal distribution transform. *IFAC Proceedings Volumes 44*, 1 (2011), 11602–11607. 18th IFAC World Congress. doi:10.3182/20110828-6-IT-1002.02865. 2

[WGG11] WU H., GUAN X., GONG J.: Parastream: A parallel streaming delaunay triangulation algorithm for lidar points on multicore architectures. *Computers & Geosciences 37*, 9 (2011), 1355–1363. doi:10.1016/j.cageo.2011.01.008. 2

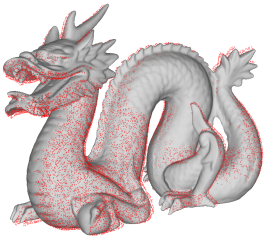
[YB07] YAN P., BOWYER K. W.: A fast algorithm for icp-based 3d shape biometrics. *Computer Vision and Image Understanding 107*, 3 (2007), 195–202. doi:10.1016/j.cviu.2006.11.001. 2, 4, 6

[YF02] YAMANY S. M., FARAG A. A.: Surface signatures: an orientation independent free-form surface representation scheme for the purpose of objects registration and matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence 24*, 8 (Aug 2002), 1105–1120. doi:10.1109/TPAMI.2002.1023806. 2

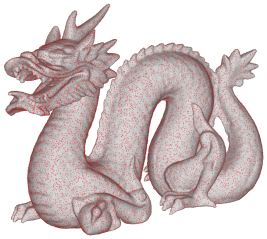


(a) Gargoyle mesh and unregistered point cloud

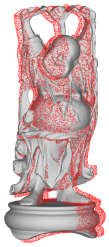
(b) Gargoyle point cloud registration



(c) Dragon mesh and unregistered point cloud



(d) Dragon point cloud registration



(e) Buddha mesh and unregistered point cloud

(f) Buddha point cloud registration

(g) Lucy mesh and unregistered point cloud

(h) Lucy point cloud registration

Figure 8: *Point-cloud-to-mesh registration of 4 different models: Gargoyle, Dragon, Buddha and Lucy [CL96]. The registration algorithm minimizes the cloud-to-mesh distance.*