Registration and fusion of large scale three-dimensional models

Tim Lenertz

July 17, 2015

Contents

1. Introduction

2. State of the Art

2.1. 3D Scanning

The general goal of a 3D scanning project is to create a digital model of a physical three-dimensional object. Different methodologies exist for all sizes and types of objects. This section will give an overview of the process of 3D scanning, including the technology involved in obtaining 3D data, the operations that are performed to process it, and the final results that a 3D scanning project aims to obtain.

Possibly the most simple case is to scan the surface of a single, solid object. Examples include artefacts such as the commonly used *Stanford Bunny*, models of teeth or bones for manufacturing of prosthesis, small archeological artefacts, etc. The object is idealised mathematically as a single closed two-dimensional surface embedded in three-dimensional space.

For larger of more complex objects, or objects embedded in a more complex environment, it becomes harder to delimit the targeted content, and to filter it out from the raw scans. This is for instance the case for buildings, archeological sites or rock formations. It may be desirable to have a higher resolution for specific parts of the scene.

In these cases, data is typically collected using a combination of 3D scanning and photogrammetry. 3D scanners emit light and detect the reflections from the object, in order to record a set of three-dimensional coordinates of points that lie on the object's surfaces. Photogrammetry consists of taking multiple photographic pictures of the object, and algorithmically recovering depth information by comparing photos from different camera poses. Both of these techniques yield a *range image* or *depth map*, which is a projected two-dimensional image of the object from a given view point where each pixel contains depth information. Post-processing of the 3D scans largely consists in filtering the raw data and combining range images and photographs from different view points, in order to obtain a 3D model of the entire object. This resulting 3D model is an approximation of the real object's surfaces, typically in the form of a set of unconnected points (*point cloud*), or a *mesh* of vertices forming triangular faces.

In other contexts, specifically in medical imaging, the goal is not to scan surfaces, but the whole inside of an object. Here techniques such as *computed tomography* and *magnetic resonance imaging* are used, and a volumetric model of the object in the form of *voxel data* is produced. A *voxel* is the three-dimensional equivalent of a *pixel*.

For this paper, the scanner object is modeled to be an ensemble of continuous surfaces, and the point clouds a discrete set of points from those surfaces.

2.1.1. Typical workflow

2.1.2. Point cloud

Data obtained from 3D scanning is recorded in the form of a *point cloud*. An unorganized point cloud is defined as a set $P = \{(x_i, y_i, z_i)\}$ of *points*, each of which have spatial coordinates set in a coordinate system specific for this scan. Each point can be attributed with additional information, such as an RGB

color, a scalar values, or the normal vector of the surface at that point. Laser scanners usually collect an intensity value that records the strength of the reflected light beam from that point.

Additional information can be contained in the ordering of the points in the set. Laser scanners probe their field of view by sending out rays in different directions in a well-defined order. Typically the elevation and azimuth angles are gradually incremented and reset in a line-by-line manner, forming a two-dimensional grid in the field of view. Knowing the width and height of this grid, the ordered point cloud corresponds to a *range image*: Each pixel in the range image corresponds to either a point $p_i \in P$, or an invalid point ϵ, in case when no reflected ray was received in the direction it was pointing at. The range image can be described as the function $\mathbb{N}^2 \to P \cup \{\epsilon\}$. In the case of a stationary laser scanner, the pixel coordinates would map to the azimuth and elevation angles of the point in spherical coordinates. The exact nature of this mapping is determined by the scanner, and it not necessarily a linear mapping. However it is such that a small difference in the pixel coordinates corresponds to a small difference in azimuth and elevation, which can for example be used to find neighbouring points efficiently.

Point clouds hold no information about the connectivity of the points that form a surface of the object. Points that are considered to be samples of a surface are called *inliers*. Each inlier has a certain *error*, as a result of the limited precision of the scanner, and due to properties of the material. Points that do form part of the object surface are called *outliers*. They may be the result of unwanted content that got scanned along with the targeted object, or any other *noise* data that appears during the processing pipeline.

Representing real objects using point clouds can be considered a two-fold modeling of physical reality: First it is assumed that the object is a set of continuous, solid surfaces. This disregards material properties such as surface reflectance and transparency, fine-scale texture of the surface and the resulting effects on light reflection, small scale motions of the object. Within this model point attributes such as a surface normal vector and a color are defined, and points are classified as inliers or outliers. Then this surfaces model gets represented by means of a sparse set of points, which introduces additional considerations such as the distribution, density and uncertainty of the points.

This notion is vague, and can be inappropriate when the real object is too complex to be modeled that way. For example for brick wall covered with climbing plants, scanned at low resolution, and possibly moving during the scan, there is not enough information available in the scan to represent the surfaces of the individual leaves. When instead considering the wall as one plane, the vegetation gets represented as an error value in subsequent processing.

2.1.3. 3D scanner technology

In general 3D scanners are devices which probe their physical environment in order to collect information about the shape and possibly appearance of objects.

Laser rangefinder

A laser rangefinder is a device which uses a laser beam to find the distance of a physical object. It consists of a laser and an optical sensor pointing in the same direction. A laser beam is sent out to the object, and the *time-of-flight* is measured until the reflected beam is received by the optical sensor. The distance of the object can then be deduced by measuring either *time of flight* or the *phase shift*.

For the former case, a short laser pulse is send out, and the time Δt until the reflected light hits the sensor is measured. The distance of the object can then be calculated as $d = \frac{c\,\Delta t}{2}$, where c is the speed of light. Using this procedure long distance measurements of up to 10 kilometers can be taken, at a very high rate. However due to the high speed of light, the precision is limited. Obtaining measurements

accurate within more than a few centimeters requires creating very short laser pulses and precise time measurement.

An alternate method is to use a continuous light beam instead of a short pulse, and measure the phase shift between the emitted and the received signal. This allows for measuring the distance in a range within the wavelength of the emitted light. By sampling the cross-correlation of both signals at different time offsets, a value for the phase shift can be obtained.

Time-of-flight scanner

Time-of-flight scanners are based on a laser range finder. A 3D scan is performed sequentially measuring the distances in different directions in the field of view. The beam is oriented by rotating the rangefinder or using a mirror system. A *range image* is thus obtained. In spherical coordinates, the radius of each point corresponds to the distance measured by the rangefinder, whereas the azimuth and elevation depends on the direction of the beam.

These devices are also called *Lidar* scanners.

Time-of-flight scanners are *long-range scanners*. They may operate over distances of several kilometers. However due to the high speed of light, they have a relatively low accuracy on the order of millimeters.

Triangulation scanner

Another laser-based technique for recording 3D data is to find the spatial location of the laser dot by triangulation. As with the time-of-flight scanner, a laser beam is sequentially projected in different directions onto the object. But instead of measuring the return time of the beam, a it used to track the location of the laser dot on the object surface. The projected two-dimensional position of the laser dot is thus known from both the view point of the laser and the view point of the camera, and the pose of the camera relative to the laser is fixed. These three pieces of information fully determine the three-dimensional location of the laser dot on the object surface.

2.1.4. Photogrammetry

2.2. Operations on point clouds

This section describes some of the operations that are applied to point cloud data during the post-processing of 3D scans. The point cloud is regarded as an unordered set $P = \{(x_i, y_i, z_i)\}$. In the next section data structures used to lay the point set out in memory are considered.

2.2.1. Cropping

Cropping means to simply remove the points from P that lie outside some geometric region. This region may be a bounding box, a view frustum of a camera or other. It is typically performed manually using point cloud software as a first step in post-processing the 3D scans.

2.2.2. Fusion

Since point clouds are unordered point sets, fusing two point clouds corresponds to taking the union of the two sets.

This creates some redundancy in the overlapping parts of the point clouds. Techniques exist to refine the resulting distribution and density of points, as described for example in [Kyöstilä *et al.* , 2013].

2.2.3. Projection

Generating a virtual *range image* from the point cloud. As described above, a range image contains only the part of the object that is visible from a given viewpoint. Each point in the range image corresponds to a two-dimensional coordinate on a pixel grid. In range images produced by real 3D scanners, the precise mapping from pixel coordinates to the azimuth and elevation components of the point's spherical coordinates may be unknown.

Here, projection essentially means to simulate the operation on a 3D scanner, with the point cloud replacing the real object. The range image obtained from projecting a point cloud using should ideally be the same as the one obtained from scanning the real object with a scanner that has the same pose and coordinate mapping parameters. However the point cloud contains only a sparse set of point representing the surfaces, and holds no direct connectivity information.

Let $w, h \in \mathbb{N}$ be the width and height of the image. The aim of a projection algorithm is to implement a function $r : [0, w[_{\mathbb{N}} \times [0, h[_{\mathbb{N}} \to P \cup \{\epsilon\}$, which associates to each image pixel a point from the point cloud P, or the invalid point ϵ.

Let $\mathbf{proj}_C : \mathbb{R}^3 \to \mathbb{N}^2 \cup \{\epsilon\}$ be the function used to map 3D coordinates to pixel coordinates. C represents the pose and parameters of the virtual camera. So $(0, 0) \leq \mathbf{proj}(\cdot) < (w, h)$. For each image coordinate (x_i, y_i), the region of space where points would map onto that pixel is given by $\{p \in \mathbb{R}^3 : \mathbf{proj}_C(p) = (x_i, y_i)\}$. In the case of perspective projection, it will have the shape of a thin square-base frustum extending from the camera point to infinity. In orthogonal projection, is instead corresponds to a thin cuboid. The discrete subset of points from the point cloud P which lie in that region is given by $P_{(x_i,y_i)} = \{p \in P : \mathbf{proj}_C(p) = (x_i, y_i)\}$.

Figure **??** shows the situation in 2D for a perspective position. Two surface parts of the object lie inside the frustum. The further one is called A and the closer one B.

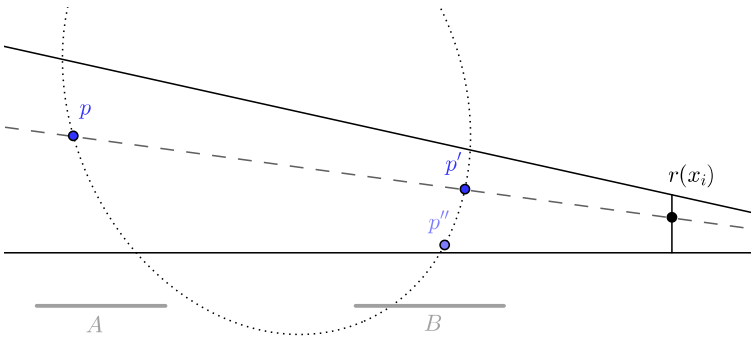

Figure 2.1.: Illustration of point cloud to range image projection in 2D

A simple approach to define the function r is to let $r(x_i, y_i)$ be the point in $P_{(x_i,y_i)}$ that is closest to the camera, or ϵ when $P_{(x_i,y_i)}$ is empty. If $P_{(x_i,y_i)}$ was not a discrete set, but instead an uncountable set of surface points, it would be guaranteed that the chosen point p if not occluded by another point p' lying in front of it, because p' would necessarily also be in $P_{(x_i,y_i)}$.

Choosing the closest point is not necessarily the best approach. Another point p'' might be a better candidate, when evaluating additional attributes of the points. Also the closest point may be an outlier located in front of B. Another approach can be to group several points and generate a new one. The

important thing is that the point lies on surface B.

But because $P_{(x_i,y_i)}$ are discrete sets, if the point density is not high enough, it can occur that no point in it lies on surface B, and so instead one in A would be chosen.

2.2.4. Meshing

2.2.5. Downsampling

2.2.6. Filtering

2.2.7. Registration

Given two point clouds P and Q that represent the same object, find a translation and rotation that will put both point clouds into the same coordinate system, and thus align the corresponding parts. P and Q are be taken from different poses, so different parts of the object are occluded in the two point clouds.

For example to obtain a full point cloud of a solid object, it needs to be scanned from different sides. Before it can be merged into a full point cloud, the precise relative poses of the scans need to be known. The aim of a registration algorithm is to compute an estimation of those poses.

Many different methods exist. A large survey of registration methods is given in chapter 3.

2.2.8. Image-to-cloud registration

2.3. Data structures for point clouds

Even though a point cloud is defined as an unordered set of points, for it to be processed algorithmically the data needs to be laid out in a certain way in computer memory. The lack of an order relation between the points provides for great flexibility. At the same time, the structure of a point cloud is that of a sparse discrete set of points embedded in three-dimensional continuous space, whereas computer memory is a dense, discrete array of bits.

Compromises need to be made in laying out the data in a way such that the required operations can be performed efficiently.

2.4. Preliminary mathematics

This section presents some mathematical results and methods that are used in the text to follow.

2.4.1. Transformation matrices and homogeneous coordinates

Positions in three-dimensional space can be represented using cartesian coordinates. Let O be an origin point in space, and let $\vec{i}, \vec{j}, \vec{k}$ be three orthogonal vectors of norm 1. Then $\vec{p} = (x, y, z)$ represents the position $O + x\,\vec{i} + y\,\vec{j} + z\,\vec{k}$. The vectors $\vec{o}, \vec{i}, \vec{j}, \vec{k}$ define the coordinate system.

A point cloud as defined as a set of points, where each one has a position and possible additional attributes. The positions in one point cloud are all set in the same coordinate system. When the points are attributes with normal vectors, or other vector-type attributes, this is also true for them. Applying a transformation $\mathbf{T}$ to a point cloud means applying that transformation to the position, and if applicable to any vector-type attributes of it.

Classification

The transformations used here are classified as follows:

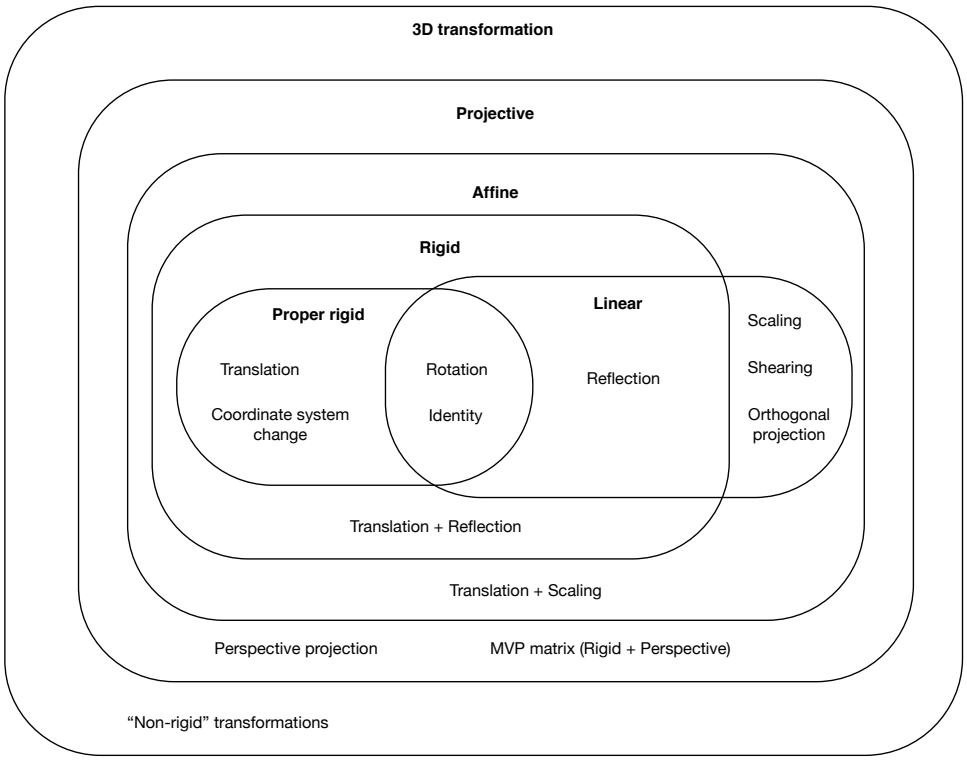

Figure 2.2.: Venn diagram of three-dimensional transformations

Linear transformations correspond to a linear recombination of the three coordinates, and can be expressed using a 3×3 transformation matrix. *Affine* transformations are linear transformation with an additional translation. A *rigid* transformation is an affine transformation where the linear part corresponds to an orthogonal matrix. It preserves the distance between every pair of points. *Proper rigid* transformations exclude shearing, and consist of a rotation and a translation. They correspond to the movement an object can make in three-dimensional space without altering its shape. A *projective* transformation is an affine transformation, followed by a division of the three coordinates by one same linear combination of the coordinates. It can for instance express perspective projections. Additionally, the term *non-rigid* transformation is sometimes used in the context of point cloud registration, to express any transformation (not necessarily affine) that alters the shape of the model.

Transformations can be combined into a conjunction of transformations, which would be classified into the lowest common class of its components. For instance a translation followed by a reflection would be a rigid transformation that is neither linear nor proper rigid. Conjunctions of transformations are in general not commutative, but all possible orders of application belong to the same class.

Linear transformation

A linear transformation is one where each coordinate of a point is mapped to a linear combination of the three coordinates. As such it can be expressed using a 3×3 matrix, and applying the transformation

corresponds to multiplying this matrix T by column vector formed by the point coordinates.

$$\begin{bmatrix} t_{1,1} & t_{1,2} & t_{1,3} \\ t_{2,1} & t_{2,2} & t_{2,3} \\ t_{3,1} & t_{3,2} & t_{3,3} \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} t_{1,1}\, x + t_{1,2}\, y + t_{1,3}\, z \\ t_{2,1}\, x + t_{2,2}\, y + t_{2,3}\, z \\ t_{3,1}\, x + t_{3,2}\, y + t_{3,3}\, z \end{bmatrix} \tag{2.1}$$

So in a transformation $\vec{p'} = \mathbf{T}\,\vec{p}$, $t_{i,j}$ indicates the coefficient of the j-th coordinate of $\vec{p}$ in the i-th coordinate of $\vec{p'}$. As a consequence the origin $(0, 0, 0)$ is a fixed point in linear transformation. Linear transformations can for example express a rotation, a shearing, a reflection, or an orthogonal projection. Because of the associativity of matrix multiplication, the composition of linear transformations can be expressed in one single linear transformation matrix. First applying $\mathbf{T_1}$ followed by $\mathbf{T_2}$ is the same as applying $\mathbf{T_2} \times \mathbf{T_1}$, because

$$\vec{p'} = \mathbf{T_2}\,(\mathbf{T_1}\,\vec{p}) = (\mathbf{T_2}\,\mathbf{T_1})\,\vec{p} \tag{2.2}$$

Also, the inverse transformation is expressed by the inverse of the transformation matrix $\mathbf{T^{-1}}$. As mentioned, the conjunction of two linear transformations is non-commutative, just like the underlying matrix multiplication.

Rotation

In two dimensional euclidian geometry, a rotation around the origin point $(0, 0)$ with angle θ is expressed by the linear transformation matrix

$$\mathbf{R}_\theta = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{2.3}$$

This can be derived using the geometric definition of the trigonometric functions. In three-dimensional additionally a plane of rotation, of equivalently an axis of rotation needs to be specified. The linear transformation matrices for the *elemental rotations* around the x, y or z axis can be deduced from 2.3 by keeping one coordinate unchanged and applying the 2D rotation on the plane formed by the other two:

$$\mathbf{R}^x_\theta = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \qquad \mathbf{R}^y_\theta = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \qquad \mathbf{R}^z_\theta = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

Euler angles

One common way to express a three-dimensional rotation is using *Euler angles*, that is, the angles by which to rotate around those three axis. For example, (ϕ, θ, ψ) may express the rotation $\mathbf{R}^x_\phi\,\mathbf{R}^y_\theta\,\mathbf{R}^z_\psi$. Because of the non-commutativity, many different conventions are possible: The three elementary rotations can be applied in different orders, such as *x-y-z*, *z-y-x* or *x-y-z*. Also combinations of only two elementary rotations like *x-y-x* or *z-x-z* are possible, where the same axis used for the first and third rotation. In expression like $\mathbf{R}^x_\phi\,\mathbf{R}^y_\theta\,\mathbf{R}^z_\psi$, the three elementary rotations are *intrinsic*: The second rotation rotates around the new y axis, which has already been displaced from the first rotation. In an *explicit* conventions, all three rotations instead take place in a fixed coordinate system.

The terms *yaw*, *pitch* and *roll* are commonly used to name the three rotation angles. Given an object pointing into a certain direction in space, such as a camera or a moving vehicle, *roll* refers to a rotation around the axis in which it is pointing. *Pitch* is a rotation around its relative horizontal axis, and *yaw* around its relative vertice axis. For example, a *pitch* rotation of an aircraft would make it fly upwards or downwards.

Besides the many possible conventions, another problem with Euler angles is the *gimbal lock* phenomenon, where two components can end up representing a rotation around the same axis.

Because of gimbal lock, and because the angles are taken modulo 2π, small changes in a rotation do not always correspond to small changes in the Euler angles. This makes them unsuited for processing rotations algorithmically and for solving optimisation problems. Instead, a good alternative to directly working with rotation matrices is to use quaternions, which express the rotation using 4 real values.

Quaternions

...

Rotation around an axis

Rotation between two vectors

Affine transformation

Translations are not linear transformations, because they add a constant term to each coordinate of $\vec{p'}$. An *affine* transformation corresponds to a linear transformation followed by a translation: $\vec{p'} = \mathbf{T}\,\vec{p} + \vec{t}$. Let $\vec{p'} = \mathbf{T_1}\,\vec{p} + \vec{t_1}$ and $\vec{p''} = \mathbf{T_2}\,\vec{p'} + \vec{t_2}$ be the expressions of two subsequent affine transformation to apply to $\vec{p}$.

$$\vec{p''} = \mathbf{T_2}\,(\mathbf{T_1}\,\vec{p} + \vec{t_1}) + \vec{t_2} = (\mathbf{T_2}\,\mathbf{T_1})\,\vec{p} + (\mathbf{T_1}\vec{t_1} + \vec{t_2}) \tag{2.5}$$

So a composition of multiple affine transformations is still an affine transformation, but the translation vectors cannot be simply added up. It can be seen that the composition is again non-commutative, except for the case when $\mathbf{T_2} = \mathbf{T_1} = \mathbf{I}$, that is, when both transformations are translations only.

Also if the translation component is applied *before* the linear component (instead of *after* it), it represents a different affine transformation:

$$\mathbf{T}\,(\vec{p} + \vec{t}) = \mathbf{T}\,\vec{p} + \mathbf{T}\,\vec{t} \neq \mathbf{T}\,\vec{p} + \vec{t} \tag{2.6}$$

Rigid transformation

The subclass of affine transformation where $\mathbf{T}$ is an orthogonal matrix are the rigid transformations. By definition, its row vectors, and its column vectors are orthogonal unit vectors. Orthogonal matrices preserve the dot product of vectors, and as a consequence distances and angles between any two pairs of points are preserved. Orthogonal matrices represent either a rotation or a reflection, and its determinant $\det(\mathbf{T})$ is always either $+1$ for a rotation or -1 for a reflection. This is a necessary but not sufficient condition, as there are also non-orthogonal matrices with determinant $+1$ or -1. So a rigid transformation can be a translation, rotation, reflection, or any composition thereof.

A proper rigid transformation is defined as a rigid transformation for which $\mathbf{T}$ is a rotation matrix. It always consists of a rotation and a translation, and any composition of proper rigid matrices can be reformulated as a single rotation and translation:

$$\mathbf{R_2}\,(\mathbf{R_1}\,\vec{p} + \vec{t_1}) + \vec{t_2} = (\mathbf{R_2}\,\mathbf{R_1})\,\vec{p} + (\mathbf{R_2}\,\vec{t_1} + \vec{t_2}) \tag{2.7}$$

Where $\mathbf{R_2}\,\mathbf{R_1}$ is also an orthogonal matrix, it being the composition of two rotations.

Proper rigid transformations correspond to the ways a real object can be moved in three-dimensional space without altering its shape. As such, they can be used to represent the position and orientation of an object.

In the following text, the term "rigid transformation" will always refer to a proper rigid transformation.

In most use cases involving three-dimensional geometry, non-proper rigid transformations are rarely used since they change the handedness of an object, which is in reality not possible in three-dimensional space. In fact, with a fourth spatial dimension it would be possible with a 4D rotation that temporarily pulls the object out of the three-dimensional subspace. This process would not alter the object's internal structure. But while both the start and end state would be possible in 3D space, the intermediary steps are not, and movements in reality are always continuous.

Homogeneous coordinates

In order to simplify calculations and notation, it is natural to use *homogeneous coordinates*. They allow for affine and projective transformations to be written using a single 4×4 matrix.

Formally, a vector $\vec{\hat{v}}$ in homogeneous coordinates corresponds to a vector $\vec{v}$ in cartesian coordinates iff

$$\vec{\hat{v}} = [x, y, z, w]^\intercal \qquad \text{and} \qquad \vec{c} = \left[\frac{x}{w}, \frac{y}{w}, \frac{z}{w}\right]^\intercal \tag{2.8}$$

The conversion from homogeneous to cartesian coordinates consists of an element-wise division by the fourth component w of the vector itself. This non-linear operation cannot be represented by cartesian matrix multiplications alone. The equivalent in homogeneous coordinates of an affine transformation given by the linear transformation $\mathbf{L}$[1], and the translation vector $\vec{t}$, where the linear transformation is applied before the translation, is the 4×4 matrix

$$\hat{\mathbf{T}} = \begin{bmatrix} l_{1,1} & l_{1,2} & l_{1,3} & t_1 \\ l_{2,1} & l_{2,2} & l_{2,3} & t_2 \\ l_{3,1} & l_{3,2} & l_{3,3} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.9}$$

A distinction is made between positions and vectors: When transforming a position, with the linear transformation is applied, followed by the translation. For vectors on the other hand, only the linear transformation is applied. A vector does not have a position in space, but only a direction and norm. Its coordinates represent the position of its end point, assuming that its start point is at $[0, 0, 0]^\intercal$. For example when applying a rigid transformation to a point cloud where points are attributed with surface normal vectors, the point positions should get rotated and then translated. The normal vectors should be rotated the same way, but no translation vector should be added to them.

Positions are converted into homogeneous coordinates by adding an extra component $w = 1$, whereas for vectors an extra component $w = 0$ is added. The three first components x, y, z remain the same. It can be seen that

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \times \begin{bmatrix} l_{1,1} & l_{1,2} & l_{1,3} & t_1 \\ l_{2,1} & l_{2,2} & l_{2,3} & t_2 \\ l_{3,1} & l_{3,2} & l_{3,3} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} l_{1,1}\, x + l_{1,2}\, y + l_{1,3}\, z + t_1 \\ l_{2,1}\, x + l_{2,2}\, y + l_{2,3}\, z + t_2 \\ l_{3,1}\, x + l_{3,2}\, y + l_{3,3}\, z + t_3 \\ 1 \end{bmatrix} \hat{=} \begin{bmatrix} l_{1,1}\, x + l_{1,2}\, y + l_{1,3}\, z + t_1 \\ l_{2,1}\, x + l_{2,2}\, y + l_{2,3}\, z + t_2 \\ l_{3,1}\, x + l_{3,2}\, y + l_{3,3}\, z + t_3 \end{bmatrix} \tag{2.10}$$

In the case where $w = 0$, the components of $\vec{t}$ get removed. In the case of affine transformations, the last row of the homogeneous matrix is always $[0, 0, 0, 1]$, and no division takes place. For perspective projections, a division by the z component will compute foreshortening.

Because here an affine or projective transformation is applied by single matrix multiplication, is follows that any composition of those matrices can be represented using the matrix product of multiple 4×4 matrices.

[1] Using $\mathbf{L}$ here instead of $\mathbf{T}$ to have different lowercase letters.

Pose

This position and orientation of an object in space is called its *pose*. A pose is also a cartesian coordinate system. Coordinates as well as other poses are always defined relative to such a coordinate system.

In any sufficiently complex three-dimensional environment, it becomes natural to define the positions and orientations of object relative to each other, and not in a single global coordinate system. 3D modelling software such as *Blender* typically implement a tree-structure of objects, in which the pose of each object is defined relative to its parent. A similar software system was written for this project.

The problem of 3D scan registration treated in this paper is precisely to find the pose from which a scan was taken, relative to another scan, and the output of any point cloud registration algorithm is a rigid transformation matrix.

A pose corresponds to an orthonormal coordinate system which is defined by tuple $S = \langle O, \vec{i}, \vec{j}, \vec{k} \rangle$ where O is the origin point, and $\vec{i}, \vec{j}, \vec{k}$ three orthogonal vectors with $\|\vec{i}\| = \|\vec{j}\| = \|\vec{k}\| = 1$. The origin point and the vectors of a coordinate system are defined within its *parent* coordinate system. The root coordinate system in that tree is called *world space*. For an *identity* coordinate system, $O = (0, 0, 0)$, $\vec{i} = [1, 0, 0]^\intercal$, $\vec{j} = [0, 1, 0]^\intercal$ and $\vec{k} = [0, 0, 1]^\intercal$.

A pose can also be expressed using the *transform-to-parent* rigid transformation $\mathbf{M}$, which transforms the coordinates of points or vectors expressed in its coordinate system S_i, into the coordinates of the same point of vector in its parent coordinate system S_{i-1}. Equivalently, the *transform-from-parent* rigid transformation is its inverse $\mathbf{M^{-1}}$. Then

$$O_i = \mathbf{M^{-1}}\, O_{i-1} \qquad \vec{i_i} = \mathbf{M^{-1}}\, \vec{i_{i-1}} \qquad \vec{j_i} = \mathbf{M^{-1}}\, \vec{j_{i-1}} \qquad \vec{k_i} = \mathbf{M^{-1}}\, \vec{k_{i-1}} \tag{2.11}$$

where the distinction between positions and vectors using homogeneous coordinates is made.

...

2.4.2. Least squares method

The least squares method is a way to calculate an approximate solution to an overdetermined system. For example the equation $y = ax + b$ of a line that crosses two points (x_1, y_1) and (x_2, y_2) can easily be calculated by solving a linear system with 2 equations and 2 unknowns. But if there are 3 or more points, there is no solution unless the points are perfectly aligned. Here a least squares solution would be the line which minimizes the sum of the squares of the distances of the points to itself.

Formally, let $\{(x_i, y_i)\}$ be a set of n data points. x_i are independent variables and y_i dependent variables found by observation. Both x_i and y_i can be vectors. For example the problem of fitting a plane to a set of 3D points may be modelled with x_i being a data point's X coordinate, and $\vec{y_i}$ that data point's Y and Z coordinates.

The goal is to find a vector of parameters β that define a model $\hat{y}_i = f_\beta(x_i)$. Because in general there is no f_β for which $\forall i, \hat{y}_i = y_i$, one searches a solution that minimizes the sum of the squared residuals:

$$\hat{\beta} = \arg\min_\beta S \qquad \text{where} \qquad S = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{2.12}$$

For the example where the model is a line, one can define $\beta = (a, b)$ and $f_\beta(x) = ax + b$. Then the residual would be the vertical distance of a point to a line. It is also possible to define a linear expression of f_β by which the orthogonal distances of the points to the lines is minimised. Many similar geometrical fitting problems can be solved in this manner, such as fitting a plane, circle, sphere, or other object to a set of points. [**?**]

Linear least-squares

In general, a solution for least square problems can be computed by setting the gradient S to zero. Since the terms y_i are constant, this becomes for each component β_j:

$$\frac{\partial S}{\partial \beta_j} = -2 \sum_{i=1}^{n} \frac{\partial f_\beta(x)}{\partial \beta_j} = 0 \tag{2.13}$$

If $f_{\vec{\beta}}$ is a linear combination of the parameters $\vec{\beta}$, then a closed form expression for the solution exists. Let there be n data points (x, y), where x are vectors of m components, and y real numbers. (Here the subscript refers to the component of the vector x, not the index of the data point). β will also have m components. The function f_β is of the form

$$f_\beta(x) = \beta_1 \, x_1 + \beta_2 \, x_2 + \cdots + \beta_m \, x_m \tag{2.14}$$

For the example of fitting a line to a set of 2D points, one would add a second component to the dependent variables which is always set to 1, so as to get back to $f_\beta(x) = ax + b$.

The overdetermined system to solve can be written in matrix form

$$\mathbf{X} \, \hat{\beta} = \vec{y} \tag{2.15}$$

where $\mathbf{X}$ is a nm matrix. Each row corresponds to a data point, and each component per row corresponds to the j-th component x_j of the dependent variable of that data point. The sum of squares S to minimize becomes

$$S = \sum_{i=1}^{n} \|y_i - \sum_{j=1}^{m} X_{i,j} \beta_j\|^2 = \|y - \mathbf{X} \, \vec{\beta}\|^2 \tag{2.16}$$

From 2.14, the partial derivative of a residual becomes

$$\frac{\partial (y_i - f_\beta(x))}{\partial \beta_j} = -X_{i,j} \tag{2.17}$$

Putting 2.17 into 2.13 and rearranging the expression, one obtains the *normal equations*:

$$\sum_{i=1}^{n} \sum_{k=1}^{m} X_{ij} \, X_{ik} \, \hat{\beta}_k = \sum_{i=1}^{n} X_{ij} \, y_i \qquad \text{for} \qquad j = 1, 2, \ldots, m \tag{2.18}$$

Written in matrix form:

$$(\mathbf{X}^\intercal \, \mathbf{X}) \, \hat{\beta} = \mathbf{X}^\intercal \, \vec{y} \tag{2.19}$$

And so the parameters $\hat{\beta}$ for which $f_{\hat{\beta}}(x)$ is a least squares solution to a linear system can be computed using the closed form expression

$$\hat{\beta} = (\mathbf{X}^\intercal \, \mathbf{X})^{-1} \, \mathbf{X}^\intercal \, \vec{y} \tag{2.20}$$

Non-linear least squares

When f_β is not a linear combination of $x_1, x_2, \ldots, x_m$ a closed form solution is generally not possible. Instead it can be computed using the *Gauss-Newton algorithm*, by iterative numerical approximation.

Weights

The least squares methods finds a solution for which the squared residual $(y_i - \hat{y}_i)^2$ is minimized and evenly distributed among all data points. Weights can be added to introduce bias towards higher weight data points, letting their residual become smaller and the others' larger.

For each data point a *weight* $w_i \in R$ is defined. When $w_i = k \times w_j$ for $k \in N$, the effect is the same as adding k times the data point (x_i, y_i). With weights, the sum to minimize is

$$S = \sum_{i=1}^{n} w_i \left(y_i - \hat{y}_i\right)^2 \tag{2.21}$$

By distributivity, when all w_i become λw_i, S becomes λS, which doesn't change the minimization problem. Supposing that w_i are rational numbers (which can approximate real numbers to an arbitrary precision), then there exists λ such that $\lambda w_i \in N$. This justifies the previous statement about data points duplication.

2.4.3. Point set alignment using singular value decomposition

For point cloud registration, corresponding points $(\vec{p_i}, \vec{q_i})$ of two point clouds are used to find a rigid transformation $\mathbf{M}$ that aligns the two point clouds. When there are exactly three pairs of points, an exact solution can be found using vector algebra. [**?**] For more than three points, a least squares solution can be found.

Here $f_\beta(\vec{q}) = \mathbf{M}\,\vec{q} = \mathbf{R}\,\vec{q} + \vec{t}$ is the function that applies the sought after rigid transformation. The parameter vector β has 6 components, three for the translation and three for the rotation. $\mathbf{R}$ is the rotation matrix and $\vec{t}$ the translation vector. The independent variable is $\vec{q_i}$, a 3-vector with the coordinates of a point. The dependent variable is $\vec{p_i}$, also a 3-vector with the coordinates of the corresponding point from the other point cloud.

Because the independent variables are vectors, the residuals to minimize are the squares of the norms of the vector differences. The sum S to minimize becomes

$$S = \sum_{i=1}^{n} w_i \left\|\vec{p_i} - (\mathbf{R}\,\vec{q_i} + \vec{t})\right\|^2 \tag{2.22}$$

The assumption is that both point sets $\{p_i\}$ and $\{q_i\}$ have approximately the same constellation but are in a different pose. For registration, first they need to be translated so as to make one common point coincide, and then they can be aligned by rotating one point set around that point. As a common point, the centroids of both point clouds are chosen:

$$\vec{c_p} = \frac{1}{N}\sum_{i=1}^{n} \vec{p_i} \qquad \text{and} \qquad \vec{c_q} = \frac{1}{N}\sum_{i=1}^{n} \vec{q_i} \tag{2.23}$$

Because the constellations are not exactly equal, this choice evenly distributes the error. Also, when the shape that the points form around the centroid point is roughly that of a sphere (their distances from the centroid do not vary too much), the rotating around the centroid will move each point by about the same amount, again evening out the error.

A rotation matrix $\mathbf{R}$ expressed a rotation around the origin. In order to simplify the problem, both point clouds are first translated to put the centroid at origin. So $\forall i$ one defines

$$\vec{p_i'} = \vec{p_i} - \vec{c_p} \qquad \text{and} \qquad \vec{q_i'} = \vec{q_i} - \vec{c_q} \tag{2.24}$$

and the minimization problem becomes

$$S' = \sum_{i=1}^{n} w_i \, \|\vec{p'_i} - \mathbf{R}\,\vec{q'_i}\|^2 \tag{2.25}$$

Since $\vec{t}$ was applied after $\mathbf{R}$, this does not change the meaning of $\mathbf{R}$.

Different approaches exist for computing $\mathbf{R}$ in constant time, that is without the need for iterative numerical approximation. [**?**] describes a solution whereby $\mathbf{R}$ is represented as a unit quaternion. This was used in the original description of the ICP algorithm (see) in [Besl & McKay, 1992]. [Lorusso *et al.* , 1995] gives a comparison of four methods, also in the context of the ICP algorithm. Probably the most commonly implemented approach is the following, which is based on singular value decomposition.

Using some matrix algebra, 2.25 becomes

$$S' = \sum_{i=1}^{n} w_i \, \|\vec{p'_i}\|^2 + w_i \, \|\vec{q'_i}\|^2 - 2\, w_i \, \vec{q'_i}^{\intercal}\, \mathbf{R}\, \vec{p'_i} \tag{2.26}$$

The first and second terms do not depend on $\mathbf{R}$, hence the problem is reduced to maximizing $\sum_i w_i \, \vec{q'_i}^{\intercal}\, \mathbf{R}\, \vec{p'_i}$.

Let $\mathbf{W} = \mathrm{diag}(w_1, w_2, \ldots, w_n)$, and let $\mathbf{P}$ and $\mathbf{Q}$ be $3 \times n$ matrices composed of the n column-vectors. One has

$$\mathbf{W}\,\mathbf{Q}^{\intercal} = \begin{bmatrix} - & w_1\, \vec{q'_1}^{\intercal} & - \\ - & w_2\, \vec{q'_2}^{\intercal} & - \\ & \vdots & \\ - & w_n\, \vec{q'_n}^{\intercal} & - \end{bmatrix} \qquad \text{and} \qquad \mathbf{R}\,\mathbf{P} = \begin{bmatrix} | & | & & | \\ \mathbf{R}\,\vec{p'_1} & \mathbf{R}\,\vec{p'_2} & \ldots & \mathbf{R}\,\vec{p'_n} \\ | & | & & | \end{bmatrix} \tag{2.27}$$

It can be seen that

$$\sum_{i=1}^{n} w_i \, \vec{q'_i}^{\intercal}\, \mathbf{R}\, \vec{p'_i} = \mathrm{tr}(\mathbf{W}\,\mathbf{Q}^{\intercal}\,\mathbf{R}\,\mathbf{P}) \tag{2.28}$$

Where the trace tr is the sum of the diagonal entries of a matrix. Because $\mathbf{AB} = (\mathbf{BA})^{\intercal}$ and the trace is invariant under the transpose,

$$\mathrm{tr}((\mathbf{W}\,\mathbf{Q}^{\intercal})\,(\mathbf{R}\,\mathbf{P})) = \mathrm{tr}((\mathbf{R}\,\mathbf{P})\,(\mathbf{W}\,\mathbf{Q}^{\intercal})) = \mathrm{tr}(\mathbf{K}\,\mathbf{R}) \tag{2.29}$$

This $n \times n$ *covariance matrix*[2] $\mathbf{K}$ can be computed directly from the input data points:

$$\mathbf{K} = \mathbf{P}\,\mathbf{W}\,\mathbf{Q}^{\intercal} = \sum_{i=1}^{n} w_i \, \vec{p'_i}\, \vec{q'_i}^{\intercal} \tag{2.30}$$

Here the *singular value decomposition* of $\mathbf{K}$ is taken:

$$\mathbf{K} = \mathbf{U}\,\mathbf{\Sigma}\,\mathbf{V}^{\intercal} \tag{2.31}$$

Now $\mathbf{U}$, $\mathbf{V}$ and also the unknown rotation matrix $\mathbf{R}$ are orthogonal matrices, and $\mathbf{\Sigma}$ is a non-negative diagonal matrix. Again using the properties of the tr, it can be rewritten

$$\mathrm{tr}(\mathbf{K}\,\mathbf{R}) = \mathrm{tr}(\mathbf{\Sigma}\,\mathbf{V}^{\intercal}\,\mathbf{R}\,\mathbf{U}) \tag{2.32}$$

Now $\mathbf{N} = \mathbf{V}^{\intercal}\mathbf{R}\mathbf{U}$ is also an orthogonal matrix, and so all $\|n_{i,j}\| \leq 1$. The trace $\mathrm{tr}(\mathbf{\Sigma N}) = \sum_i \sigma_i n_{ii}$ is maximal when $\mathbf{N}$ is such that $\forall i, n_{ii} = 1$, i.e. when $\mathbf{N} = \mathbf{I}$.

[2] Also called *correlation* matrix.

Solving for $\mathbf{R}$, one obtains:

$$\mathbf{N} = \mathbf{V}^\intercal \mathbf{R} \mathbf{U} = \mathbf{I} \Rightarrow \mathbf{V} = \mathbf{R} \mathbf{U} \Rightarrow \mathbf{R} = \mathbf{V} \mathbf{U}^\intercal \tag{2.33}$$

The resulting $\mathbf{R}$ is always an orthogonal matrix, but not necessarily a rotation matrix. If $det(\mathbf{R}) = -1$ it is a reflection instead. This is obtained when the point sets are such that a reflection aligns them better than a rotation.

To always find a rotation a rectification step needs to be added to the algorithm: When $det(\mathbf{R}) = -1$, instead choose $\mathbf{R}'$ for which $det(\mathbf{R}') = 1$ and S' attains the next best possible local maximum. $\mathbf{R}'$ is obtained by replacing the entries of $\mathbf{R}$ in the third column by their opposite.

This shows how the rotation matrix $\mathbf{R}$ which minimizes S' can be computed from the singular value decomposition of the covariance matrix $\mathbf{K}$. The point sets $\{\vec{p_i'}\}$ and $\{\vec{q_i'}\}$ have been translated to put their centroids $\vec{c_p}$ and $\vec{c_q}$ to the origin. Since adding a translational component to a rigid transformation does not change its rotational component, $\mathbf{R}$ is also valid on the original point sets. It remains to calculate the translation vector $\vec{t}$.

On the original point sets, $(\mathbf{R}, \vec{t})$ should solve $\vec{p_i} = \mathbf{R}\,\vec{q_i} + \vec{t}$ in a least squares sense. One has $\vec{p_i'} = \mathbf{R}\,\vec{q_i'}$, where $\vec{p_i'} = \vec{p_i} - \vec{c_p}$ and $\vec{q_i'} = \vec{q_i} - \vec{c_p}$. Putting this together and solving for $\vec{t}$, one obtains

$$\vec{t} = \vec{c_p} - \mathbf{R}\,\vec{c_q} \tag{2.34}$$

2.4.4. Information theory

2.4.5. Gaussian distribution

2.4.6. Parzen window

2.4.7. RANSAC

2.5. Preliminary computer science

2.5.1. Octree

2.5.2. KdTree

3. Point Cloud Registration

The main subject of this paper is two register two or more point clouds. The goal of registration is to find the rigid transformation(s) that will put the point clouds in one common coordinate system. Applying this transformation will make the point clouds overlap, making surfaces of the object that occur in both overlap. The term "alignment" is used interchangeably with "registration", though the former emphasizes that *surfaces* are being aligned.

For registration to be meaningful, the hypothesis is made that the point clouds do represent scans of the same object, and that the *true* rigid transformation(s) that align them exist. For the case of two 3D scans of a stationary object, this true transformation would be the pose of the first scan relative to that of the second scan. The goal of registration algorithms is then to find an estimation of the true transformation. Except for artificially generated testing data, the true transformation is usually unknown or known only at a limited precision.

This chapter consists of a classification and survey of existing registration methods.

3.1. Introduction

Pair-wise registration algorithms take one *fixed* point cloud P and one *loose* point cloud Q as input. The output is a rigid transformation M that brings Q into the coordinate system of P. During the algorithm P is left unchanged, while Q is being moved around. Thus implementations can preprocess P once to allow for more efficient computation, for example by representing it in a KdTree, so that closest points can be computed more quickly. The representation of Q on the other hand may be modified on the fly, for example by applying a partial transformation to the points' coordinates at each iteration of the algorithm.

When the goal is to register multiple point clouds $\{P_i\}$, a pair-wise algorithm would be applied multiple times to different pairs (P_i, P_j). Depending on how these pairs are chosen, this leads to an accumulation of the registration error. *Multi-view* registration algorithms instead directly operate on a set of point clouds to align, and aim to evenly distribute the registration error. The output is a set of rigid transformations $\{\mathbf{M_i}\}$, one for each input point cloud. (If one point cloud P_f is chosen as fixed, then $\mathbf{M_f} = \mathbf{I}$)

A primary distinction is made depending on whether an initial estimation for $\mathbf{M}$ is already known. This may come for example from manual alignment made in a 3D visualizer application, or from sensors attached to the scanners that detect its pose. *Coarse registration* algorithms make no such assumption and use a representation of the point cloud that is invariant to its current pose relative to the other one, to deduce an approximate alignment. *Fine registration* algorithms assume that the point clouds are already approximatively registered, and improve upon the alignment by bringing corresponding parts closer together. The goal is to obtain the most accurate solution possible. Some methods can also yield a fine registration, without initial registration.

In the most usual case the point clouds to register already have the same scale, so $\mathbf{M}$ is a proper rigid transformation. When this is not the case, for example when one of the point clouds was taken using photogrammetry and the real scale is unknown, the goal would additionally be to find a scaling factor.

$\mathbf{M}$ would then be a composition of a proper rigid transformation followed by a scaling transformation.

There is also the notion of *non-rigid registration*, which is a generalization where the loose point clouds can also be deformed. The output is no longer a single (rigid or otherwise) transformation matrix. Depending on the deformation model used, it may instead be a per-point displacement field, or a rigid transformation combined with a modification of a skeleton that parametrizes the deformation. It is used to to register point clouds depicting one same object that can change shape in some limited way between the scans, for instance a waving flag, a human face or in medical imaging an internal organ. This this paper only rigid registration is considered.

3.2. Robustness

The ideal case for registration of two point clouds P and Q would be if $P = \{\mathbf{M}p : p \in P\}$, that is they both contain exactly the same constellation of points, with Q being transformed by precisely the *true* rigid transformation $\mathbf{M}$. Here one unique (unless the model has precise rotation symmetry) rigid transformation $\mathbf{M}$ exists which makes P and Q coincide, and it is possible for a trivial algorithm to compute $\mathbf{M}$ at an arbitrary level of precision, taking only P and Q as input.

However in practice P and Q will differ in many more ways than the rigid transformation. A registration algorithm is deemed *robust* if it continues to deliver good results when the disparities between the point clouds get progressively worse.

The disparities can for example be the following:

Points dispersion The points of a point cloud constitute a discrete subset of the surfaces that are represented. Unless P and Q were algorithmically generated from one scanner output file, those subsets will be different but still approximate the same surface. The lower the point density, the stronger this disparity becomes.

Points on a planar surfaces will typically be dispersed approximatively on a quadrilateral grid, when scanned using a laser scanner that proceeds in sequential scan-lines. It becomes a square grid on surfaces facing the scanner. The more the surface is oblique, the wider the rectangles get and the density gets lower.

When large objects are scanner, the points density will naturally get lower on surface locations further away from the scanner.

Noise One or both point clouds can contain outlier points that do not lie on any relevant surface of the model. They result from points that were scanner from the environment but are not part of the model surfaces, scanner errors, or artifacts from prior processing.

A good registration algorithm should be insensitive to outlier points, or be able to identify them and sort them out before continuing. The *noise-to-signal* ratio can be defined as the number of outlier points divided by the number of inlier points.

Bounds The two point clouds contain only points within given geometric bounds, either due to the limited range and field of view of the scanner, or from having been cropped out of a larger point cloud in preprocessing.

While a minimal bounding box, frustum, etc. can be computed from a point cloud such that all its points are within it, the actual bounding region is usually not known. That is, if there is no point in a particular location, it is not immediately known whether this is because there is no object

surface at that location of because the location is out of bounds. (For range images, the field of view is known.)

But all points that are not within the intersection of those bounding regions of P and Q will have no corresponding point in the other point cloud, and so they need to be treated by the algorithm the same way as outliers.

Occlusion The scanner can only record surface points that are visible from its pose. So occluded areas will not be covered by the point cloud. Because no surface connectivity information is recorded, and sometimes the scanner position in the point cloud coordinate system is unknown, it becomes hard to tell what regions are occluded. Moreover because the surfaces are only partially covered, reconstructing them becomes a harder problem.

When registering two scans taken from different poses, their region of overlap becomes limited to the areas that are not occluded in either point cloud. Point outside it will have not corresponding point in the other cloud. But they still hold information about the underlying surface that is represented by both P and Q.

3.3. Fine registration

As stated, fine registration algorithms take as input two (or more) point clouds that are already approximatively aligned, and then improve their alignment as much as possible. The core observation is that corresponding points in the two point clouds are already close to each other in the current alignment.

3.3.1. Iterative Closest Point

The most well-known fine registration algorithm is Iterative Closest Point (ICP), first described in [Besl & McKay, 1992] and in [Che, 1991]. It is a pair-wise algorithm, though multi-view versions of it also are also possible [**?**].

The algorithm chooses *point correspondences* ($p \in P, q \in Q$), where q is the point closest to p with the current alignment. Using the assumption that P and Q are already roughly aligned, those correspondences approximatively correspond to real corresponding point in both representations of the object. Then a rigid transformation is applied to Q which minimizes the distances $d(p, q)$ for all corresponding point pairs in a least squares sense. The process is repeated iteratively.

Different terms for the fixed and loose point clouds are frequently used in the literature. For example the fixed point cloud may be called "model", "target" or "reference", the loose one may be called "data" or "source". In this text the terms "fixed" and "loose" are used.

ICP framework

There are many possible variations of ICP algorithms [Rusinkiewicz & Levoy, 2001]. For all of them the following 6 steps are performed at each iteration:

Selection Select a subset of points from P and/or Q to consider. In the simplest case all points are considered. Alternatives include the selection of a random subset with a given *downsampling ratio*. The decision to include or reject a point, or the probability of including a given point, may depend on a given metric, such as its distance to the center, distance to the closest neighbour, orientation of normal vector, and others. Here the selected subsets are called P^* and Q^* respectively.

Correspondence Build correspondence pairs (p_i, q_i) using the selected points. The *closest point criterion* is to choose for each $p \in P^*$ the point $q \in Q^*$ (or the other way) whose Euclidian distance $\|p - q\|$ to it is minimal. Here p_i and q_j denote corresponding points when $i = j$. It is not necessarily a one-to-one mapping: A point from one cloud may correspond to multiple point from the other.

There are other strategies for choosing point correspondences. When the normal vectors $\vec{n_p}$ are known, one possibility is to choose $q \in Q^*$ that is closest to the ray pointing out of p in the direction of its normal vector $\vec{n_p}$. Also it can be useful to only consider points in Q^* that satisfy certain constraints in function of p, such as a similar normal vector orientation, color, or other.

It is also not necessary for both P and Q to be point clouds. For example Q may instead be defined using a parametric surface. For each q a corresponding point q is then computed from p, instead of chosen from a finite set.

Finding correspondences is typically the most computationally intensive operation in the ICP iterations. One way to optimize closest point finding it is to use an appropriate data structure for Q, for example a KdTree. Also if Q is available as a range image with known camera parameters, p can be projected onto the 2D range image. Then the search can be limited to a certain radius surrounding the projection of p in image space.

Rejection Some correspondence pairs may be rejected afterwards. For example those where the distance is above a certain threshold value.

Weighting Optionally weights may be associated with the correspondences. Unless the correspondences perfectly match real corresponding points in both clouds, a rigid transformation that makes P^* and Q^* coincide is impossible. Defining weights introduces a bias in the distribution of the remaining error: The transformation will move correspondences with higher weight closer together than those of lower weight.

Error estimation An expression of the registration error $e(\mathbf{M})$ in function of the correspondences $\{(p, q)\}$ and the rigid transformation $\mathbf{M}$. $e(\mathbf{M}) = 0$ when P^* and $\mathbf{M}\,Q^*$ perfectly coincide. This value can only be reached in theoretical settings where the correspondence pairs have been set to be equal to real corresponding points on the object. Instead the algorithm will compute the transformation $\arg\min_{\mathbf{M}} e(\mathbf{M})$ that minimizes the error. When $e(\mathbf{M})$ is below a predefined threshold value, the algorithm stops and the registration is considered successful. A common problem is that $e(\mathbf{M})$ may have local minima, which can lead the algorithm to converge towards an incorrect registration.

For *point-to-point* ICP, the weighted sum of squared Euclidian distances of corresponding points is used:

$$e(\mathbf{M}) = \frac{1}{W} \sum_i w_i \, \|p_i - \mathbf{M}\, q_i\|^2$$

where $W = \sum_i w_i$. For *point-to-plane* ICP, instead the distances from q to the tangent plane of p are used. This requires knowledge of the normal vectors $\vec{n_p}$ associated with the points p. It is computed using the dot product of $\vec{n_p}$ and $\vec{p} - \vec{q}$:

$$e(\mathbf{M}) = \frac{1}{W} \sum_i w_i \, \vec{n_{p_i}} (p_i - \mathbf{M}\, q_i)$$

Intuitively, with this metric, $e(\mathbf{M})$ remains unchanged when two parallel surfaces "slide" along each other. With point-to-point, the different distributions of points on the two surfaces can lead to local minima. Point-to-plane typically increases convergence speed and robustness, but is more computationally expensive.

Several other error metrics to minimize have been developed, such as Generalized ICP [Segal *et al.* , 2009], Sparse ICP [Bouaziz *et al.* , 2013], and metrics that include attributes of the points such as its color value.

Minimization Finally $\mathbf{M}$ for which $e(\mathbf{M})$ is minimal is computed. For the point-to-point error metric it is a least squares problem, and a closed-form solution is possible as for detailed in section 2.4.3. A comparison of four methods is made in [Lorusso *et al.* , 1995]. It is concluded that the difference between the different methods are small.

For the point-to-plane metric, a similar solution is also possible by linearizing the problem using the small-angle approximation of trigonometric functions. [Che, 1991] For this the assumption is made that incremental rotations are small, which hold when the point clouds start out approximately aligned and converge to their optimal alignment.

There are also extrapolation methods that make an estimation of $\mathbf{M}$ based on previous iterations in order to improve efficiency, and methods that introduce randomisation to avoid convergence to a local minimum. [Rusinkiewicz & Levoy, 2001]

Any iterative algorithm that follows these steps is said to be in the *ICP framework*. [**?**] The original description [Besl & McKay, 1992] of ICP selects correspondences using the closest-point-criterion, applies a point-to-point error metric. [Che, 1991] describes a point-to-plane variant.

Convergence

ICP is based on estimating correspondences for the points of P in Q. If the best possible correspondences were known to start with, point cloud alignment could be solved in one iteration using a least-squares solution as described in 2.4.3. ICP instead uses the fact that P and Q are already approximately aligned to take approximate correspondences. Then Q is moved as if those correspondences were true correspondences, resulting in an improved alignment of P and Q. The convergence of ICP towards the optimal alignment depends on two hypothesis:

1. When the alignment of P and Q is more accurate, the computed correspondences become closer to true correspondences.

2. Solving the transformation for approximate correspondences results in an improved alignment.

The figure shows a fixed point cloud P, and a loose point cloud Q at two different alignments, Q_1 being aligned more accurately than Q_2. q_2, q_1, q' represent the same position in the coordinate systems of the three point clouds. $q \in Q$, but in general $q' \notin P$ except when Q and P have exactly the same constellation. $p_1, p_2 \in P$ are the correspondence points chosen for q_1 and q_2 by the closest point criterion.

Hypothesis (1) translates into $d(q_1, q') < d(q_2, q') \Rightarrow d(p_1, q') < d(p_2, q')$. If the point clouds consisted only of the line $\bar{q'p_1}$ this would be true from the triangle similarity. Clearly

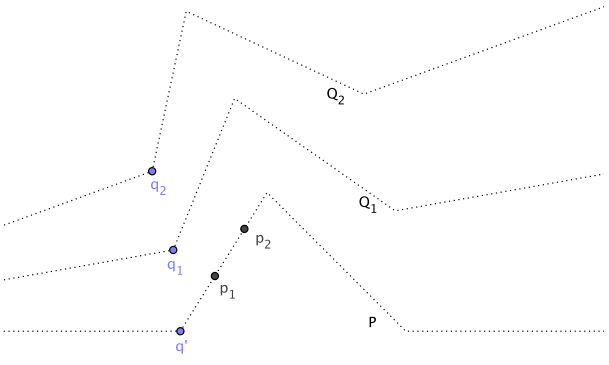

Figure 3.1.: Single point at two ICP iterations

$\lim_{q \to q'} d(p, q) = 0$. But the convergence is generally not monotonous. For example if q were to come in from another angle, p could oscillate between the closest points from the two segments adjacent to q'.

The error minimization step pulls all q_i closer to q_i. Because $\{p_i\}$ and $\{q_i\}$ have different constellations, they cannot be made to coincide, but $d(p_i, q_i)$ will get smaller in the least squares sense. [1] This shows that $d(q_1, p_2) < d(q_2, p_2)$. p_1 is the closest point to q_1 by the closest point criterion, meaning that q_2 cannot be closer. Hence $d(q_1, p_1) < d(q_1, p_2) < d(q_2, p_2)$. This proves the convergence of the ICP algorithm with the closest point criterion.

For hypothesis (2) to be satisfied, it must additionally be true that $d(q', p_1) < d(q', q_2)$. When this is not true, the algorithm can converge towards a local minimum.

Local minima

ICP minimizes an error function $e : \mathbb{R}^6 \to \mathbb{R}$ taking as input a rigid transformation, which can be represented using 6 independent variables. The function e is different at each iteration, and depends on the correspondences.

3.3.2. Generalized ICP

Conceptually, the difference between point-to-point ICP and point-to-plane ICP is that in the point-to-plane variant, the position of a point on a surface has no impact on the error metric and minimization. This is useful because the point clouds represent solid surfaces approximated using a discrete set of points, and the goal of a registration algorithm is to align those surfaces, and not the points that represent it. The registration algorithm should be agnostic to the distribution of points on a surface.

Generalized ICP, first described in [Segal *et al.* , 2009], is a generalization that covers both these variants. Each point is replaced by a gaussian probability distribution that models the uncertainty of its position on the surface and orthogonal to the surface. From this a new error metric formula to minimize is deduced that includes covariance matrices for the two points' distributions.

3.4. Coarse Registration

For coarse registration, no initial alignment of the point clouds is used, and the goal is to obtain an approximative alignment of matching point clouds, which can then be improved upon using fine registration.

3.4.1. Manual registration

Most commonly, coarse registration is done manually. One method is to define at least three pairs of corresponding positions in the two point clouds, another to rotate and translate the point clouds using a 3D interface.

Both methods can be time consuming because one works using a two-dimensional projection of the two point clouds on a computer screen, the view is difficult to recognize when they incorrectly overlap, and one needs to be able to rotate and translate both the camera and the two point clouds. For these reasons automatic solutions can be preferred in practice, especially when the scanning project consists of many point clouds.

[1] If it were to get larger, then $\mathbf{I}$ would be a better transformation estimation than the one found by least squares minimization.

4. Registration of Large Models

The subject area of this paper is the registration of scans taken of a large and structurally complex physical objects. The thesis is set in the context of the full 3D scanning of the Brussels "Hôtel de Ville", a current project by LISA. In particular this paper focusses on the registration of long-range scans of the entire building with short-range, high resolution close-up scans of some features of it.

4.1. "Hôtel de Ville" scanning project

This is a current 3D documentation projection by the the Image research unit of the *Laboratories of Image, Signal processing and Acoustics* (LISA) at ULB. The goal is to create a full 3D model of the "Hôtel de Ville de Bruxelles" [1] building. It is a medieval building built in the beginning of the 15th century, with a Brabantine Gothic-style architecture.

4.2. Relief point cloud

As indicated in the beginning, the physical object is imagined as an ensemble of continuous two-dimensional surfaces embedded in three-dimensional space. The point cloud is taken to be a discrete set of points that lie on those surfaces. Because many different kinds of objects can be scanned and good approaches for processing and registering them vary depending on many factors, it is useful to restrict the scope to a particular kind of point cloud.

Therefore only point clouds that model a single, approximatively planar surface are considered. The defining characteristic is how well it can be represented as a height map on a plane A_P. Scans of engravings on a stone wall, such as the "dessus-de-porte" on figure **??** can for instance be put into this category. This type of surface will be called "relief" in this paper.

4.2.1. def

Let A be a plane and P a point cloud, both in the same coordinate system. For each $\vec{p} \in P$, $d(p)$ is the signed orthogonal distance from $\vec{p}$ to A, and $\vec{p_A}$ the projection of $\vec{p}$ onto A.

4.2.2. Artificial relief

To be able to study registration of point clouds, it is useful to generate artificial points clouds that fit into this category and for which exact values of the underlying surface can be computed.

An algorithm was devised to generate artificial relief surfaces, which look as shown on figure 4.1.

[1] Brussels Town Hall

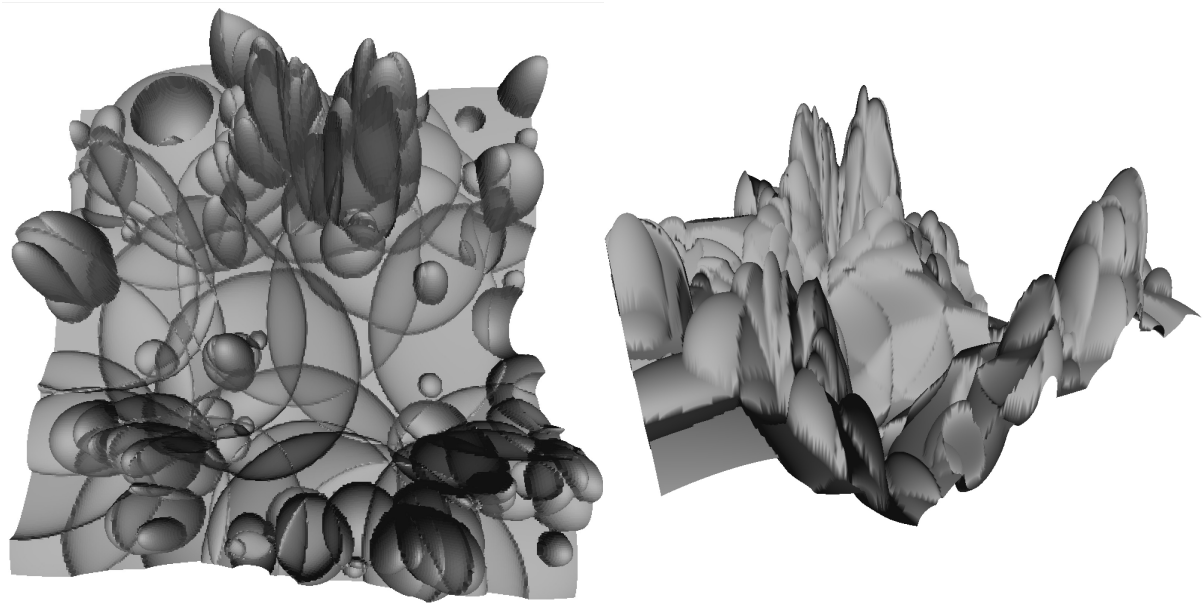

Figure 4.1.: Artificial relief surface, seen from two view points

The entire algorithm is randomized, but can be made deterministic by specifying a seed value for the random number generator. The surface to generate is specified by a *width* w, a *height multiplier* h, and this seed value.

The generated point cloud is a height map on the XY-plane. Let $z = R[x, y]$ be the z component of the single point with given x, y components. It is defined for $-\frac{w}{2} \leq x, y \leq +\frac{w}{2}$. At first, let $R[x, y] = 0$ for all these points. The result is a square surface of side length w.

The algorithm proceeds by pushing randomized "embossings" into the surface. The embossings are the shape of a half-sphere distorted in one direction, and is described using the height map formula

$$B_i[x, y] = \pm h_i \sqrt{1 - \frac{(x_i - x)^2 + (y_i - y)^2}{r_i^2}} \tag{4.1}$$

A plot of its two-dimensional analogue is shown in figure 4.2. $B_i[x, y]$ is set to 0 for coordinates x, y outside its domain, that is, for $x, y : (x_i - x)^2 + (y_i - y)^2 > r^2$. As a consequence a sharp circular corner is formed around the border.

A fixed number n of embossings are generated with different parameters, and are added to R, so that

$$R[x, y] = \sum_{i=1} n B_i[x, y] \tag{4.2}$$

The resulting height map will be split into regions $\{(x, y)\}$ where different subsets of $\{B_i\}$ are active. (B_i is active when (x, y) lies in its domain). R has sharp corners at the border points of all B_i. As soon as more than one embossing is active in one region, the z coordinate becomes a sum of square roots, producing a complicated shape for both the continuous surface areas and the corners. Its partial derivatives can still easily be calculated analytically, which allows for accurate computation of normal vectors.

The radius r_i, height h_i and center (x_i, y_i) are randomly chosen for each embossing B_i. x_i, y_i are chosen with a uniform distribution in $[-\frac{w}{2}, +\frac{w}{2}]$. The parameters for choosing r_i and h_i are set in such a way that the resulting surface will contain both flat regions and "spikes", which occlude parts of the surface when viewed from the side. h_i can be both positive or negative.

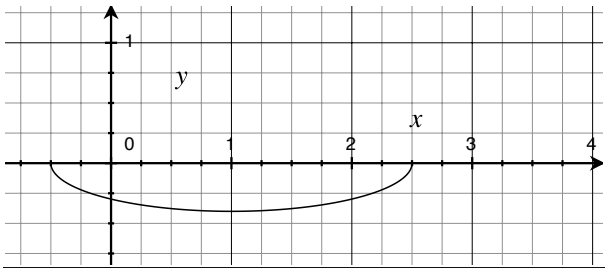

Figure 4.2.: 2D example for relief embossing, with $h_i = -0.4$, $r_i = 1.5$, and $x_i = 1.0$

4.2.3. Point cloud generation

Top-down view

Two ways of generating a point cloud of a relief surface are used. The simplest way is to simply take a set of points $\{x, y, R[x, y]\}$. It results in a *top-down* view of the surface, as seen by an orthogonal projective camera looking in $-z$ direction. From that view point the model has no occluded surfaces. The x, y coordinates are arranged on a square grid, with a given density ρ. Figure 4.3 shows an example of such a point cloud, itself projected with a perspective camera.

Occluded view

However the goal of the artificial relief surface was to simulate the kinds of surfaces that occur on real objects, and to get a point cloud with similar properties to a 3D scan of it. So it is important to be able to generate point clouds of the relief as seen from another camera positions, with the occlusions that occur.

A virtual camera is placed near the surface at a given pose, and a range image is generated using it. With an orthogonal projection camera, the point density on the surfaces will remain constant, and with a perspective projection camera, it will decline with distance to the camera.

For the algorithm that creates this range image, a first attempt was to first generate a top-down view point cloud with high enough density, and then generate project that point cloud into a range image as described in 2.2.7. However, this inevitably leaves points in the occluded areas.

Another attempt was to implement a ray-tracing method that operates on the expression of $R[x, y]$: For each image pixel, calculate the intersections with the view ray and the surface R and take the closest one. However, these intersections cannot easily be calculated analytically. Firstly, the various regions of R with different active embossings must be considered separately. But even on a single such region, multiple intersection points can still occur, and there is no direct closed-form expression for finding them. So a lot of combinatorics and numerical approximation would be required.

Instead, the implemented algorithm generates a mesh of the surface, projects a depth map of it onto image space, and then back-projects the image pixel coordinates into points. Figure 4.4 shows the resulting point cloud from two view points, the second one near the projection camera pose.

A triangle mesh is generated by taking points $\{x, y, R[x, y]\}$ with the (x, y) coordinates forming a square grid of a given density ρ, and adding a diagonal edge into each square, in alternating direction. The number of squares per side must be even for this. As can be seen on the renderings in figure 4.1, this mesh does not handle the sharp corners well, but it is sufficient as errors are rectified in a later step.

For each triangle, its three vertices are projected into image space, using the given projection camera.

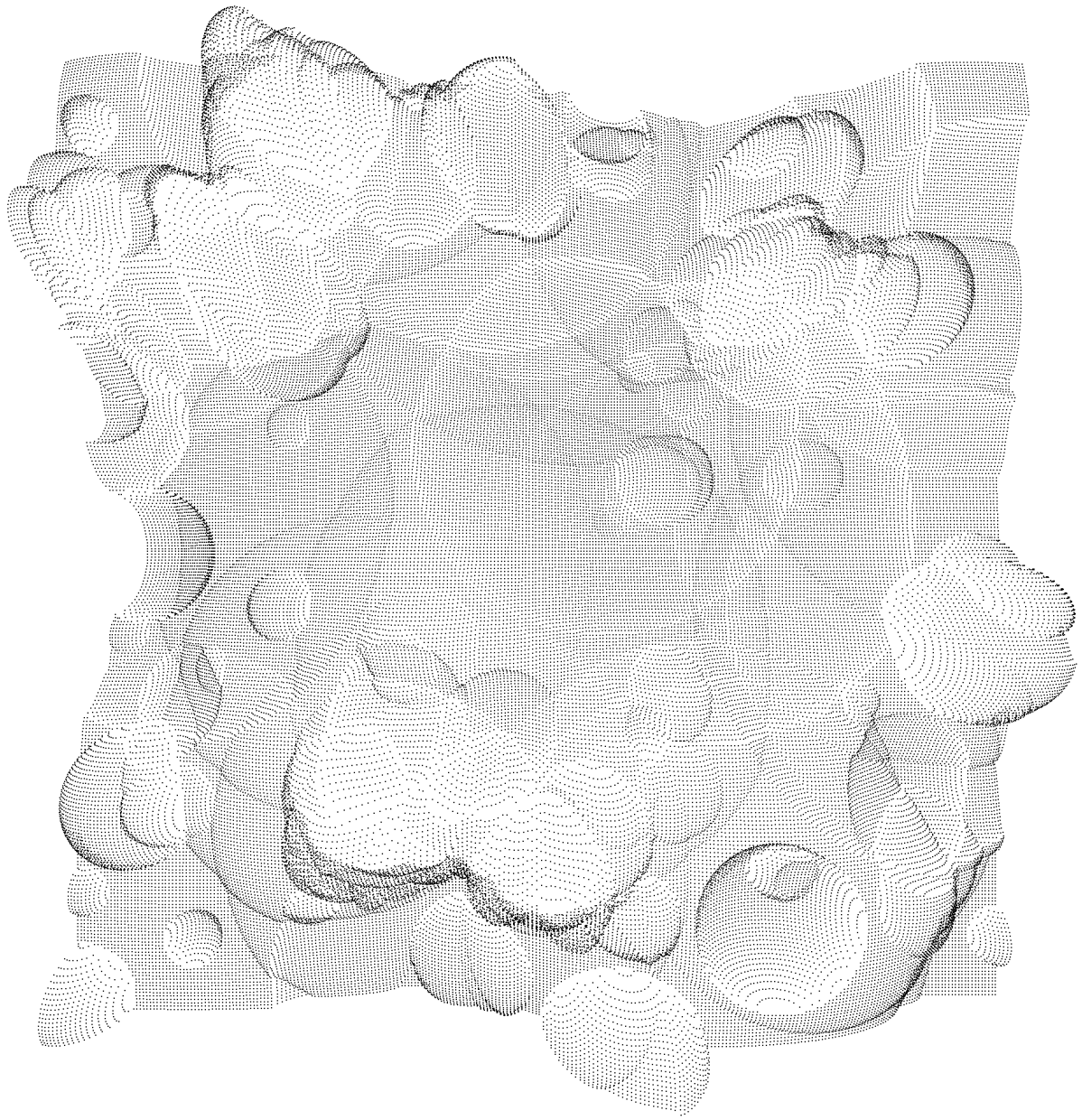

Figure 4.3.: Top-down point cloud of relief

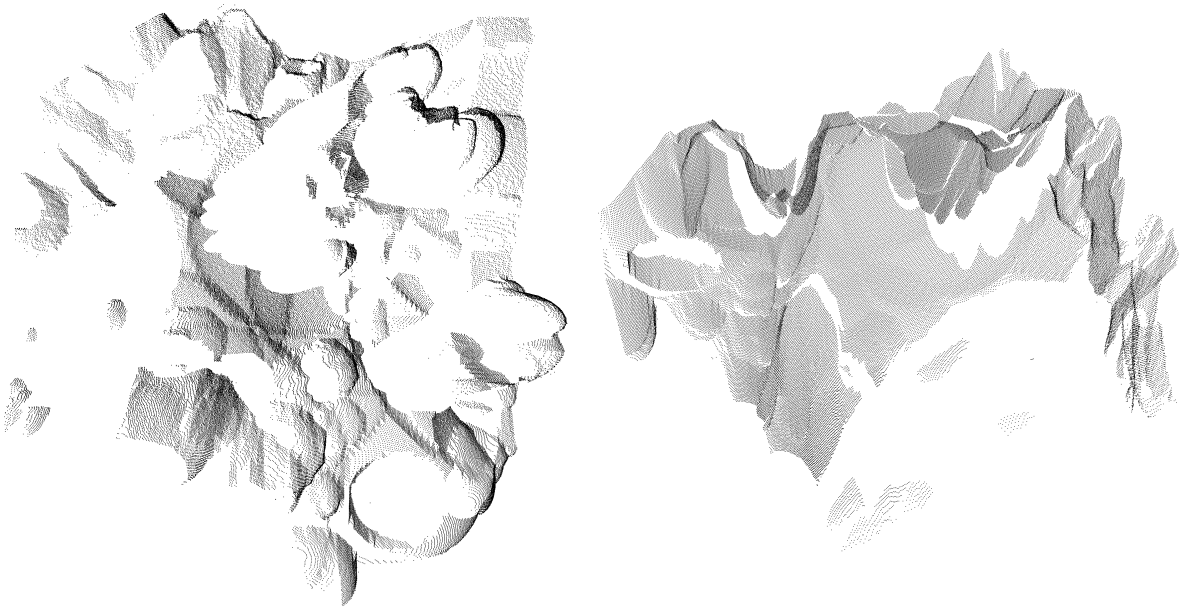

Figure 4.4.: Artificial relief surface point cloud with occlusion

This image is a Z-buffer contains, for each pixel, the inverted projected depth of the point[2].

The width and height of this image space is set higher than that of the range image by a factor of about $k = 10$. Now each triangle is filled using a 2D rasterization algorithm. For each pixel, the inverted projected depths of the three corner points are linearly interpolated by using barycentric coordinates. This results in the inverted projected depth of the corresponding 3D surface point.

The actual occlusion culling is now done using a depth test: A pixel value is overwritten only if the inverse depth is higher than the previous one. This is the case only if the point is closer to the camera.

Each $k \times k$ square on this image corresponds to a pixel of the final range image. The center pixel is taken. Given (x_i, y_i) in image space, and the inverse projected depth of the surface point, the 3D coordinates (x, y, z) of the surface points can now be calculated.

Due to the limited accuracy of the mesh, and floating point precision problems, there is some error in this result. It can be rectified by recalculating $z' = R[x, y]$.

Similarity with actual scans

Figure 4.5 (called R) shows an artificial relief point cloud with occluded view and additionally cropped to a random polygonal region inside the original square. It is superimposed on a top-down point cloud of the same relief.

Figure 4.6 (called D) is the high-resolution scan of the "dessus-de-porte", with colors removed.

The two point clouds are similar these some ways:

- Approximatively planar. For R the plane is the XY-plane, for D it is the stone surface behind the five statues.
- Seen from a side angle and partially occluded.
- Contain both smooth surfaces and sharp corners.

[2] Z coordinate after application of camera projection matrix.

- Points dispersed on a grid, as a result of the scan-lines, or of the image space in the virtual projection camera.

The important difference it that the underlying surface behind R is known. The goal will be to develop a registration method that works for both R and D because of these common characteristics. With R it can be tested both with and without knowledge of the surface.

4.3. Models

The following point clouds are used as input for the experiments.

4.3.1. Stanford Models

The Stanford Models are a collection of point clouds that are frequently used for testing 3D algorithms and applications. The most well known one if the *Stanford Bunny*. Here the *Bunny* and the *Dragon* models are used.

4.3.2. Dessus-de-porte

4.3.3. Relief model

This is an attempt to generate a random artificial "relief" model.

4.4. Evaluation of registration algorithms

The output of any registration algorithm that aligns a loose point clouds Q with a fixed point cloud P is a rigid transformation matrix $\hat{\mathbf{M}}$, or possibly an indication that the algorithm has failed. In the ideal case, which is not reachable in practice, it will be equal to the *true* transformation $\mathbf{M}$.

In order to evaluate the result, it is useful to have a numerical metric $e(\hat{\mathbf{M}})$ that indicates the "accuracy" of $\hat{\mathbf{M}}$, both for the cases when the true transformation is known and when it is unknown. It should be minimal when $\hat{\mathbf{M}} = \mathbf{M}$, and it should indicate a spatial distance measure.

4.4.1. Known true transformation

When $\mathbf{M}$ is known, the accuracy is best measured by how much $\hat{\mathbf{M}}$ deviates from $\mathbf{M}$. The rigid transformation, relative to the true transformation, is given by $\hat{\mathbf{M}}_{\text{rel}} = \mathbf{M}\,\hat{\mathbf{M}}\,\mathbf{M}^{-1}$.

The fixed point cloud P is no longer used in this case, and $e(\hat{\mathbf{M}})$ is defined as a function of $\hat{\mathbf{M}}_{\text{rel}}$.

Using $\hat{\mathbf{M}}_{\text{rel}}$, one can calculate for each point $q \in Q$ the *actual* correspondence point $q' = \hat{\mathbf{M}}_{\text{rel}}^{-1}\,\vec{q}$. It is the position in P that corresponds to $q \in Q$. Unless P and Q have the exact same constellation of points, there is generally no $p \in P$ that coincides with this point q'. Here the knowledge of $\mathbf{M}$ is used to simulate the existence of P with exactly the same constellation.

As a metric for the accuracy of $\hat{\mathbf{M}}$, the average of the unsigned distances between q and q' can be used:

$$e(\hat{\mathbf{M}}) = \frac{1}{n}\sum_{i=1}^{n} \|q - q'\| \tag{4.3}$$

This value will be called the *true error*. It is similar to the ICP point-to-point error metric, just with the real correspondences, and without squaring the terms. Using a point-to-plane or other metric would not be useful because the correspondences are exact.

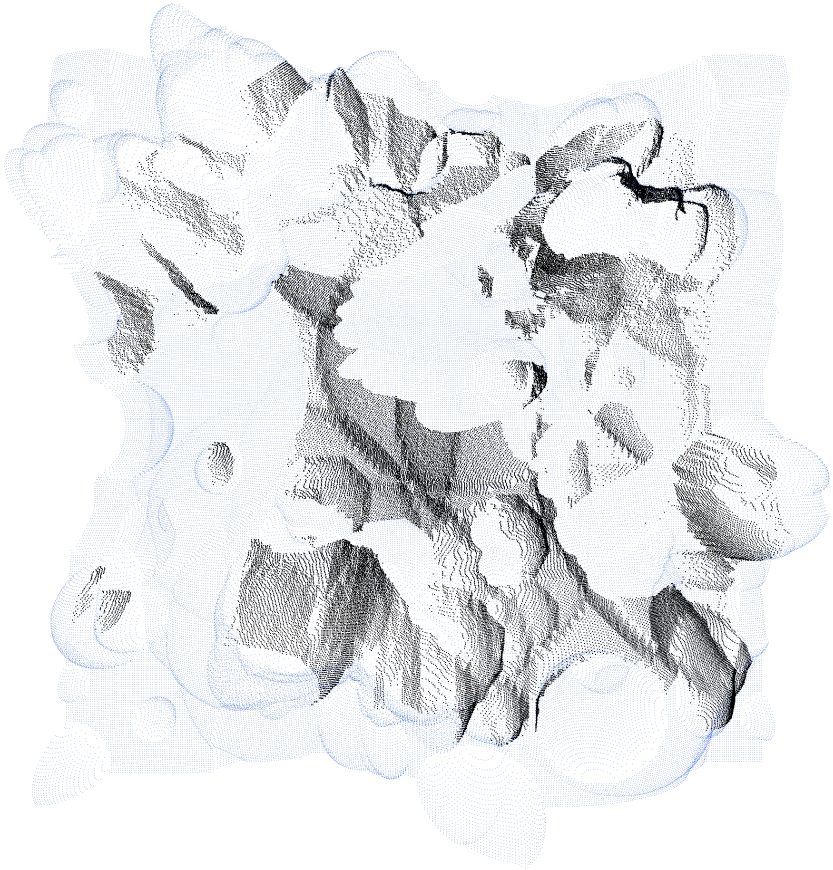

Figure 4.5.: R: Occluded view point cloud, and top-down point cloud of relief

Figure 4.6.: D: “Dessus-de-porte” point cloud

When $\hat{\mathbf{M}} = \mathbf{M}$, the absolute minimum $e(\hat{\mathbf{M}}) = 0$ is reached, as all points $q = q'$ coincide. When $\mathbf{M}$ is only a translation $\vec{t}$, $e(\hat{\mathbf{M}}) = \|\vec{t}\|$. When a small rotation with center $\vec{0}$ (the origin Q) is added, all points q move away from q' in a circular motion. Using trigonometric approximation for small angles, this length of movement is proportional to $\|q, \vec{0}\|$ (and not to the squared distance). Hence taking the average of unsigned distances $q - q'$ can given a useful value.

4.4.2. Unknown true transformation

When $\mathbf{M}$ is unknown, no exact metric for the accuracy can be defined. As with ICP, the correspondences can be approximated using the closest point criterion or a variation of it. However, here the point clouds are not assumed to be in the process of converging to an optimal alignment, but rather the goal is to evaluate the accuracy of a finished alignment, or to tell whether the registration has led to a local minimum.

Because of incorrect correspondences and outliers, a single valued metric such as the average or median of the distances is less useful, and instead the histogram of distances between the corresponding points reveals more information.

The *distance histogram* is considered in two cases: For an *own distance histogram*, P and Q are taken to be perfectly aligned and have exactly the same surfaces, but may have disparities as mentioned in section 3.2. So the closest points are always taken from one position on the surface to another on the same surface.

Then for the *cross distance histogram* P and Q may be imperfectly aligned and/or have slightly different surfaces. So they may come from two different scans.

Correspondences are always taken from a point in Q to the corresponding point in P. So each point in Q becomes a sample used to construct the histogram, and the number of samples should be as large as possible.

4.5. Own distance histogram

When used on two exact same copies of the same point clouds that are perfectly aligned, all measured distances are 0, and so the histogram consists only of one spike.

If P is artificial and its surface is known, Q can be constructed by taking a large number of points on that same surface.

4.5.1. Plane with random dispersion

Figure 4.7 shows an own distance histogram, where P and Q represent a single plane, and the points are randomly dispersed onto it with a density $\rho(P) = 30000$. The number of samples is $N(Q) = 749956$. For each sample q the distance $d(q, P)$ to the closest point in P is measured.

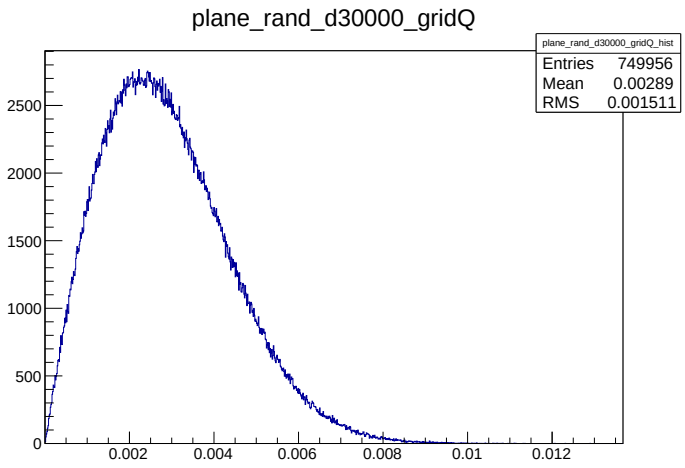

Figure 4.7.: Own distance histogram for plane P with random point distribution

The underlying probability density function f_R of the closest point distance r from any point q is given by the exponential function

$$f_R(r) = 2\pi\rho(P)\, r\, e^{-\pi\rho(P)\, r^2} \tag{4.4}$$

A plot is shown in figure 4.8, for $\rho(P) = 30000$.

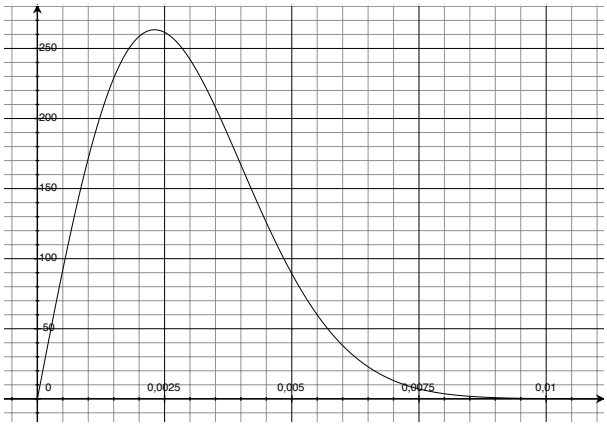

Figure 4.8.: Probability density function of closest point distance, for plane P with randomly dispersed points

Proof. Let P be a bounded continuous surface in $\mathbb{R}^2$ of area $\mathcal{A}(P)$. For the example on the figure 4.7, it is a square with side length 5. Let $\{p_i \in P\}$ be a discrete set of n randomly chosen points on that surface, with an uniform probability distribution. $\rho = \frac{n}{\mathcal{A}(P)}$ is the surface density of this points distribution.

Let $A \subset W$ a another bounded continuous subregion of P, with a variable area $\mathcal{A}(A)$ on the order of magnitude of the distances between adjacent neighboring points p_i. Let $N(A) \in \mathbb{N}$ be the number of points $p_i \in P$ that lie inside A. In the following formulas, the area value $\mathcal{A}(A)$ is also denoted A.

For any single point $p_i \in P$, the probability that it lies in A, and the probability that it does not, are given by

$$P[p_i \in A] = \frac{\mathcal{A}(A)}{\mathcal{A}(P)} = \frac{A\rho}{n} \qquad \text{and} \qquad P[p_i \notin A] = 1 - \frac{A\rho}{n} \tag{4.5}$$

For any two p_i, these events are stochastically independent, and the probability that two events occur is obtained through multiplication. The probability that none of the n points lie in A, and conversely the probability that at least one lies in A, are;

$$P[N(A) = 0] = \left(1 - \frac{A\rho}{n}\right)^n \qquad \text{and} \qquad P[N(A) \geq 1] = 1 - \left(1 - \frac{A\rho}{n}\right)^n \tag{4.6}$$

The value $P[N(A) \geq 1]$ converges for $n \to \infty$. The information about the density of points is captured by ρ, so the probability be be expressed without $\mathcal{A}(P)$ and n. Setting $n' = -\frac{n}{A\,\rho}$ and using the identity $\lim_{x\to\infty}\left(1 + \frac{1}{x}\right)^x = e$:

$$\begin{aligned}
\lim_{n\to\infty} P[N(A) \geq 1] &= \lim_{n\to\infty}\left[1 - \left(1 - \frac{A\rho}{n}\right)^n\right] \\
&= 1 - \lim_{n\to\infty}\left(1 - \frac{A\rho}{n}\right)^n \\
&= 1 - \lim_{n'\to\infty}\left(1 + \frac{1}{n'}\right)^{-A\rho\,n'} \\
&= 1 - \left[\lim_{n'\to\infty}\left(1 + \frac{1}{n'}\right)^{n'}\right]^{-A\rho} \\
&= 1 - e^{-A\rho}
\end{aligned} \tag{4.7}$$

For the final expression $A \in \mathbb{R}$ denotes only the area, as no further information on the region's shape is needed.

The goal what to find the underlying curve of the histogram in 4.7. In order to obtain a smooth function, the histogram taken with a sparse set of points $q \in Q$ is replaced by a probability density function of the closest point distance from any $q \in Q$ to its nearest neighbor $p_i \in P$. The surfaces P and Q are the same.

Let $q \in P$ be any point lying on the plane P. (Not necessarily one of the discrete set of points p_i.) Let $D_{q,r} \in P$ be the disk centered at q with radius r. Its area is $\mathcal{A}(D_r) = \pi r^2$. By definition, for any point $p \in D_{q,r}$, $\|p - q\| \leq r$. Starting from $r = 0$, the radius of the disk is increased until $N(D_{q,r}) = 1$. $r = r_{\text{closest}}$ is then the closest point distance to q.

One has $r \geq r_{\text{closest}}$ if and only if $N(D_{q,r}) \geq 1$: ($\Rightarrow$) As r gets larger, the disk can be made to contain additional points, but no points get removed. It contains one point when $r = r_{\text{closest}}$ (or possibly multiple equidistant points). ($\Leftarrow$) For the disk to contain one point, it must at least have a radius large enough to contain the closest point to q.

Let $R : q \mapsto r_{\text{closest}}$ be the random variable expressing the distance from any $q \in P$ to its closest neighbor. Now $P[R \leq r] = P[N(D_r) \geq 1]$. The probability density function $f_R(r)$ is obtained by differentiating:

$$\begin{aligned}
f_R(r) &= \frac{\mathrm{d}}{\mathrm{d}r} P[R \leq r] = \frac{\mathrm{d}}{\mathrm{d}r} P[N(D_r) \geq 1] \\
&= \frac{\mathrm{d}}{\mathrm{d}r}\left(1 - A\rho\, e^{-A\rho}\right) \\
&= -\frac{\mathrm{d}}{\mathrm{d}r} e^{-A\rho} \\
&= -\frac{\mathrm{d}}{\mathrm{d}r} e^{-\pi\rho\, r^2} \\
&= 2\pi\rho\, r\, e^{-\pi\rho\, r^2}
\end{aligned} \tag{4.8}$$

□

4.5.2. Dispersion of sample points

The histogram shown, in figure 4.10 is the same as 4.7, but with the sample points $q_i \in Q$ being randomly dispersed on the plane (and a different number of samples). This has to the problem that the local surface density of any small region of $A \subset Q$ has a higher variance and can deviate more from ρ, leading to a more jagged histogram. The probability $P[N(A) = m]$ is given by

$$P[N(A) = m] = \binom{n}{m} \left(\frac{A\rho}{n}\right)^m \left(1 - \frac{A\rho}{n}\right)^{n-m} \tag{4.9}$$

Proof. Each of the n points $p_i \in P$ may lie inside or outside of A. The probabilities of these events are given in equation 4.5, and they are stochastically independent. For there to be exactly m points inside A, the event $p_i \in A$ must occur m times, and the event $p_i \notin A$ must occur $n - m$ times. By commutativity of the product, the probability for each one of those sequences of events to occur is always

$$\left(\frac{A\rho}{n}\right)^m \left(1 - \frac{A\rho}{n}\right)^{n-m} \tag{4.10}$$

Since there are $\binom{n}{m}$ such sequences, and they are mutually exclusive, the probability for any one of them to occur is the expression given in 4.9. □

It is not a single spike at the expected value $\bar{N}(A) = A\,\rho$, and the most likely outcome can be different from the expected value. Figure 4.9 shows $P[N(A) = m]$ and $\bar{N}(A)$, interpolated to use real values for $N(A)$.

This effect is greatly reduced when the sample points on Q are instead dispersed on a regular square grid. The deviation of $N(A)$ from $\bar{N}(A)$ then occurs only due to aliasing near the border of the region A, so the variance is near-zero. It gets lower with a higher density of the samples $q \in Q$ and consequently lower side length of the square grid. This was used to obtain the histogram 4.8.

So the histogram becomes less jagged and converges towards the underlying probability density distribution, when ρ gets larger and its variance gets smaller.

4.5.3. Plane with square grid dispersion

Points on a planar surface will not be randomly dispersed, but rather be constrained in an approximately square grid. This depends on the parameters of the scanner, and the planar surface's orientation relative to the scanner. For example, figure 4.11 shows a closeup view of the points on the Bunny model's surfaces.

Figure 4.12 shows the distance histogram of two perfectly aligned planar surfaces P and Q, for $\rho(P) =$ 20000 and 30000, respectively. The number of samples is $N(Q) = 300000$.

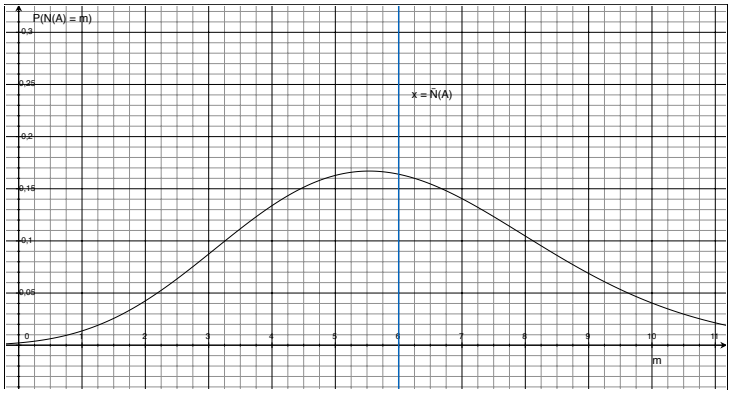

Figure 4.9.: $P[N(A) = m]$ for randomly dispersed points on a plane, interpolated to real values

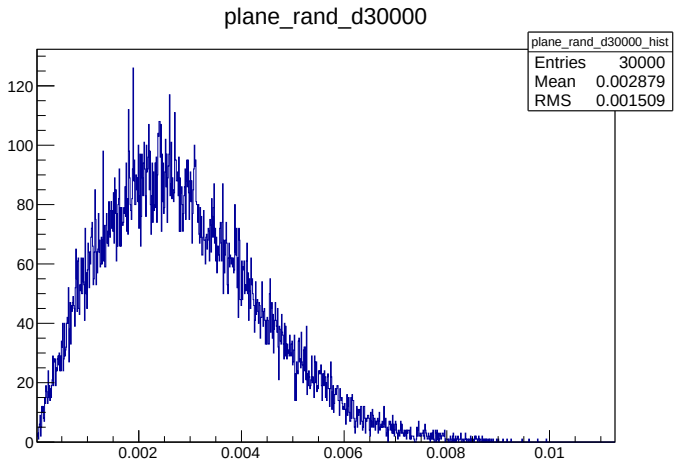

Figure 4.10.: Same as figure 4.7 but with randomly dispersed samples $q \in Q$

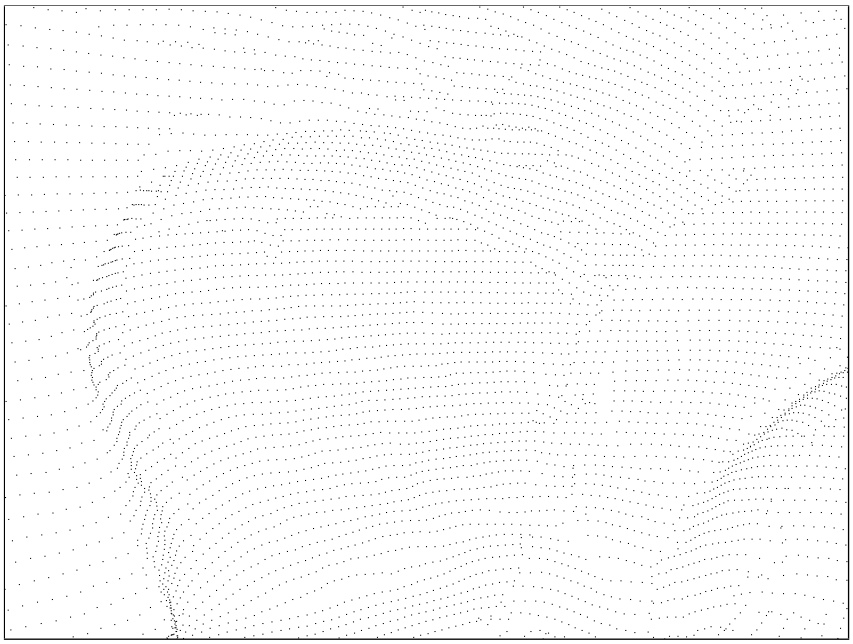

Figure 4.11.: Closeup of the surface distribution of points on the Bunny point cloud

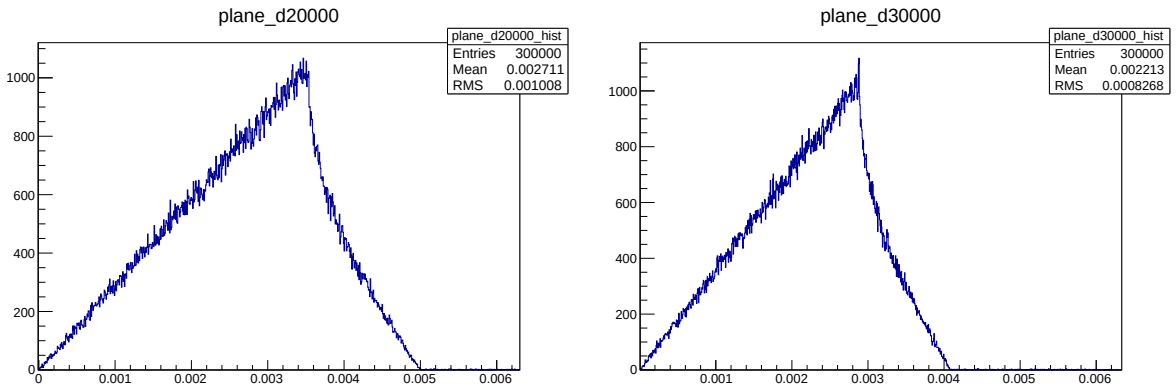

Figure 4.12.: Own distance histogram for plane P with square grid point dispersion

Under the assumption of two perfectly aligned planar surfaces P and Q, where points on P are arranged on a square grid with surface density $\rho(P)$, the probability density function f_R becomes

$$f_R(r) = \frac{1}{2l} \times \begin{cases} \frac{\pi}{4}\, r & 0 \le r \le \frac{l}{2} \\ \left(\frac{\pi}{4} - \arctan \sqrt{\left(\frac{2r}{l}\right)^2 - 1} \right) r & \frac{l}{2} < r < \frac{l}{2}\sqrt{2} \end{cases} \tag{4.11}$$

with $l = \frac{1}{\sqrt{\rho(P)}}$. Its plot is shown in figure 4.13 for $l = 2$.

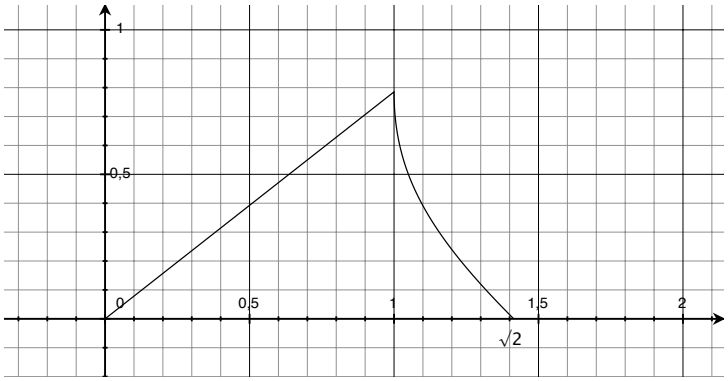

Figure 4.13.: Probability density function of closest point distance, for plane P with square grid point arrangement

Proof. The points are arranged on a square grid, with side length l. For the purpose of the following explanation, let $l = 2$.

Figure 4.14 shows four points $p_1, p_2, p_3, p_4 \in P$ with two-dimensional coordinates, and a point q that lies on the same plane as those four points. The background color indicates for each position the distance to the closest point of P. It remains constant on each of the white contour lines. $d(q, P)$ is maximal when q is in the center of one of the squares, as it does on the figure. So $d_{\max} = \sqrt{2}$.

The histogram corresponds to the probability distribution of $d(q, P)$. The entire plane can be tiled with the triangle Δ which is drawn in the figure, with the probability distribution being the same in each of these tiles, as the Δ would only get flipped along one of its sides. So it is sufficient to only look at the field within Δ. Its side lengths are $1, 1, \sqrt{2}$, and its angle at the point $p_1 = (0, 0)$ is $\frac{\pi}{8}$. For each point inside Δ, p_1 is its closest point in P.

Each value for $d \in [0, d_{\max}]$ occurs in Δ, exactly at the points that lie on the arc formed by the intersection of Δ and the circle C_d of radius d. Let $a(d)$ be the length of this arc in function of d. On the figure, k is the intersection point of $\overline{p_1 q}$ with C_1. When $d \leq 1$, the arc ranges from the abscissa axis to the diagonal, which is one eight of the circle, and so $a_1(d) = \frac{\pi}{4}\, d$.

When $1 < d < d_{\max}$, it is additionally cut off by the line $x = 1$. By solving $(x, y) \in C_d \wedge x = 1$, its intersection point with that line is $(1, \sqrt{d^2 - 1})$. Instead of starting from the abscissa, the remaining arc now starts from $\varphi = \arctan \sqrt{d^2 - 1}$, and so $a_2(d) = (\frac{\pi}{4} - \varphi)\, d$. The function a is

$$a(d) = \begin{cases} \frac{\pi}{4}\, d & 0 \leq d \leq 1 \\ \left(\frac{\pi}{4} - \arctan \sqrt{d^2 - 1}\right)\, d & 1 < d < \sqrt{2} \end{cases} \tag{4.12}$$

under the assumption that $l = 2$.

Removing this assumption, l is now determined as a function of the surface points density ρ. The density is constant throughout the surface, so ρ expresses the number of points that lie inside any region of P, divided by the area of that region. The area of a square region formed by four points is l^2. After a small translation of the point in any direction so that they don't lie on the region's borders, there is 1 point in each l^2 region. So $\rho = \frac{1}{l^2}$ and $l = \frac{1}{\sqrt{\rho}}$. In this general case, the function a becomes

$$a(d) = \frac{l}{2} \times \begin{cases} \frac{\pi}{4}\, d & 0 \leq d \leq \frac{l}{2} \\ \left(\frac{\pi}{4} - \arctan \sqrt{\left(\frac{2d}{l}\right)^2 - 1}\right)\, d & \frac{l}{2} < d < \frac{l}{2}\sqrt{2} \end{cases} \tag{4.13}$$

Since the arcs completely completely cover Δ and none of them overlap, $\int a(d)\, \mathrm{d}d = \mathcal{A}(\Delta) = l^2$. Normalizing the integral to 1, the probability density function f_R becomes

$$f_R(d) = \frac{a(d)}{l^2} \tag{4.14}$$

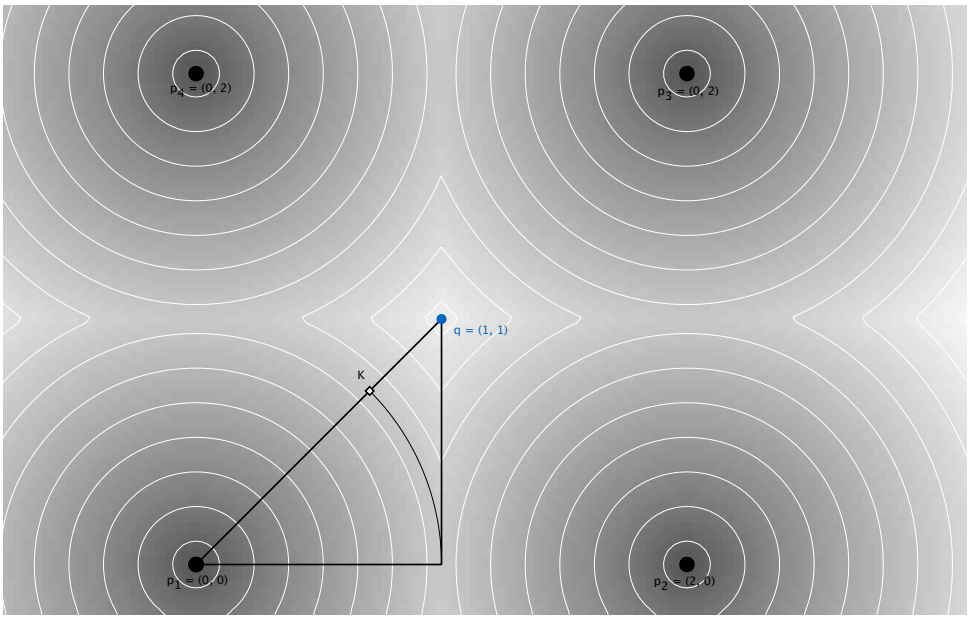

Figure 4.14.: Closest points distance field with square grid distribution of P, $l = 2$

□

4.5.4. Relief with square grid distribution

Figure 4.15 shows the distance histogram on one same relief point cloud, with four different height values h. The points are again arranged on a square grid when viewed on a top-down projection. There is no occlusion, and P and Q have the same bounds.

When h is low, the point cloud is almost equal to a plane with square grid distribution and the histogram takes on the same shape as in figure 4.12. When h gets larger, more correspondence pairs with higher distances occur because of the point's displacement in the Y axis.

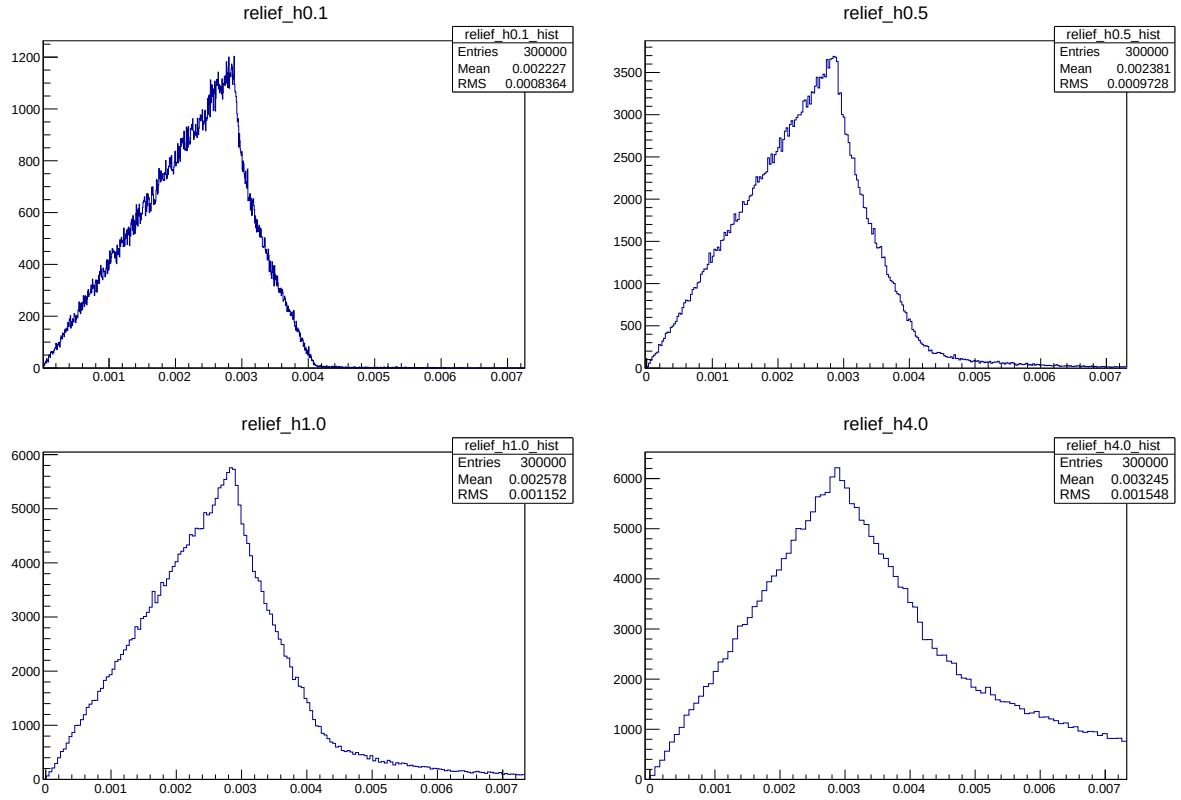

Figure 4.15.: Own distance histogram for relief P with square grid distribution, and different heights

4.5.5. Occlusion and different bounds

Up until now P and Q were considered to be perfectly aligned, constitute the same surfaces, and both have points dispersed on the whole surface at an uniform distribution.

4.6. Cross distance histogram

4.7. High-low density registration

When attempting to register a short-range scan of a relatively small object with the same object in a long-range scan, the short-range point cloud will have a much higher resolution. But fine registration algorithms generally make the assumption that the two point clouds have similar resolutions. The issue

of registering point clouds with different resolutions seems to be largely ignored in the literature about point cloud registration algorithms.

4.7.1. First experiment

A first observation is that in general for ICP, lowering the resolution of the loose point cloud does not much reduce the accuracy of the registrations. This is shown in experiment **??** (see appendix), in which the Stanford Bunny model is fine registered with a lower density copy of itself. Let P be the fixed point cloud and Q the (downsampled) loose point cloud.

The most basic variant of ICP is used: All points are selected, correspondences from are taken Q to P by the closest point criterion, no correspondences are rejected, weights are uniform, and the point-to-point error metric is used. The copies are made in such a way that they never have two points in common: P is constructed by taking randomly chosen 50% of the points from the original model, and Q is constructed from the remaining 50%. After this Q is randomly downsampled by 60 different amounts.

The experiment is done in three instances. For the first one (figure A.1), P and Q start out perfectly aligned, and for the two other ones (figures A.2 and (figure A.3)), they start out with a small (or larger) random initial transformation. 40 iterations of the registration algorithm are run and the final errors are recorded.

The plots show the error measured using the mean unsigned distances of the true correspondences, as defined in section 4.4.1. It is zero if and only if P and Q are perfectly aligned. The X axis indicates the ratio of the number of points $\frac{\|Q\|}{\|P\|}$.

Analysis

Two things can be observed: The final error does not depend much on the downsampling level, and the error always converges to about 0.001, even when P and Q were perfectly aligned to start with.

To define a rigid transformation, three pairs of corresponding points are sufficient as long as the three points do not lie on the same plane. (see section 2.4.3) So even when Q is reduced to three points the point-to-point error metric can be minimized. RANSAC-based approaches to registration, such as 4PCS are based on this.

Figure A.4 shows how the true error evolves during the 40 executions of ICP on the first experiment without initial displacement. P and Q were generated to have no common points. When choosing the points closest to the true corresponding point instead, the error does not cancel out completely, and thus the correct alignment is no longer the global minimum of the error metric.

Figure **??** shows the state after the registration. Q is rendered in blue color and P in red. From each point $q_i \in Q$ a line segment towards the true corresponding point q_i' is shown, which is not a point $p_i \in P$. On this part Q has from the correct alignment deviated towards the right.

When $\forall q_i, p_i = q_i'$, the correct alignment would be found. When P and Q are already aligned, $q_i = q_i'$. Figure 4.17 shows the histogram of $\|q_i' - p_i\| = \|q_i - p_i\|$ for the case when 50% (or 80%) of the model points are taken for P and the remaining for Q, and no additional downsampling is applied.

In both cases, $\|q - p\| \approx 0.001$ is a mode in the distribution. Smaller values are infrequent. Additional spikes occur for some values above 0.001.

The reason is that for the original Bunny point clouds, the points are evenly distributed on the surface on an approximatively square grid, with a mean distance of about 0.001 between adjacent points, as seen in the close-up view in figure 4.11. Figure **??** shows a histogram made by taking from each point p on the Bunny point cloud B, the closest point $p' \in B$ with $p' \neq p$. In the closest point histograms from Q to P, any point that is in Q is missing in P and hence the closest point is often the one at a distance

of 0.001. Some instances appear where this point is not in P either, so the closest point is further. This explains the spike at 0.002. The other spikes occur when the closest point is in a diagonal direction on the grid. This "grid" is approximate and the surface is embedded in a non planar way in 3D space, so most samples do not fall exactly in one of these spikes.

For the true correspondences, the histogram would be a single spike at $\|p_i - q'_i\| = 0$.

4.7.2. Experiments on relief

4.7.3. Limits of accuracy

4.7.4. Metric for resolution

Informally, the resolution of a point cloud indicates how many samples (i.e. points) of the represented object is contains. For a range image taken by a 3D scanner, the resolution is defined with its width and height, which correspond to the number of measurements taken per scan line and the number of scan lines, for its whole field of view.

The assumption was made that the points in the point cloud lie on a set ensemble of two-dimensional surfaces, which are embedded in three-dimensional space. A points density measure in terms of number of points per unit of volume is thus not meaningful. Instead the *density* of the distribution of points on a surface area if used.

4.8. Separation of large scans

4.9. Variable density

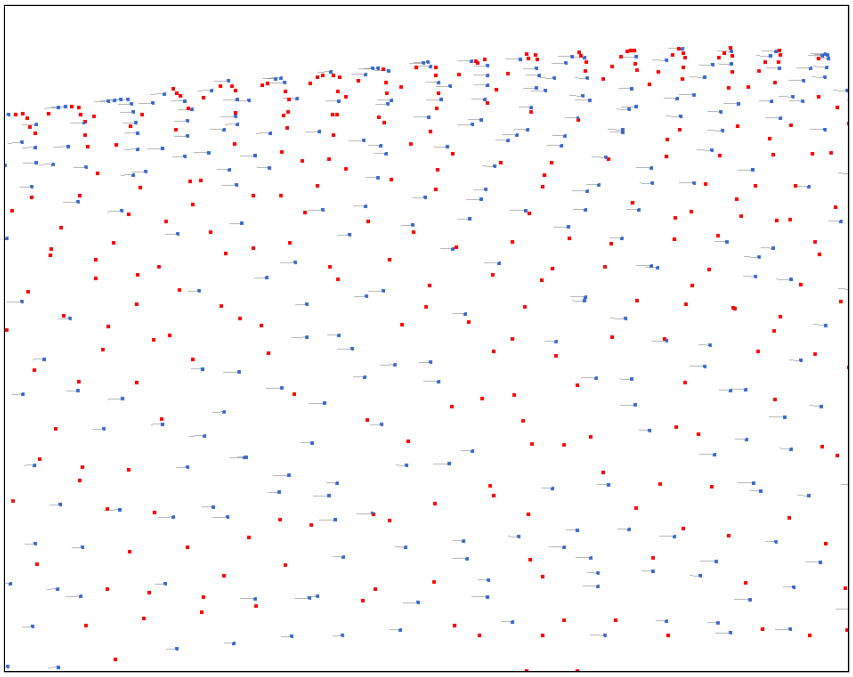

Figure 4.16.: Bunny model registered to itself, true correspondences shown

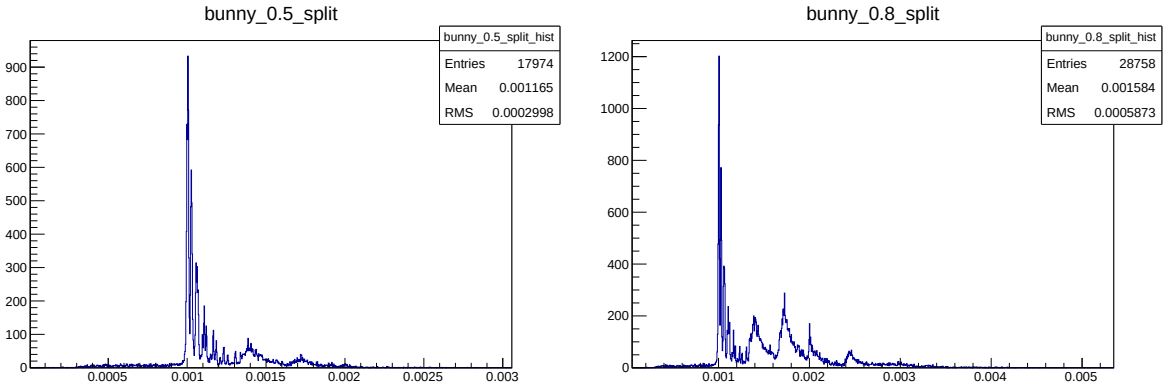

Figure 4.17.: Histograms of $\|q_i - p_i\|$ for 50% and 80% split

5. Implementation

5.1. Architecture

5.2. Usage of C++

5.3. Optimization

5.4. Visualization

5.5. User interface

6. Conclusion

A. Experimental Results

A.1. Data on models used

A.2. Different resolutions, Bunny model

Method	ICP. Select all points, closest point criterion, equal weights, no rejection, point-to-point error metric.
Model	Stanford Bunny model.
Fixed	50% of model points, randomly chosen.
Loose	Starting from the other 50%, randomly downsampled by given amount. 60 steps.
Displacement	See captions on the figures.
Y Axis	True error, after 40 iterations.
X Axis	Number of points in Loose divided by number of points in Fixed.

A.2.1. Final true error v. downsampling level

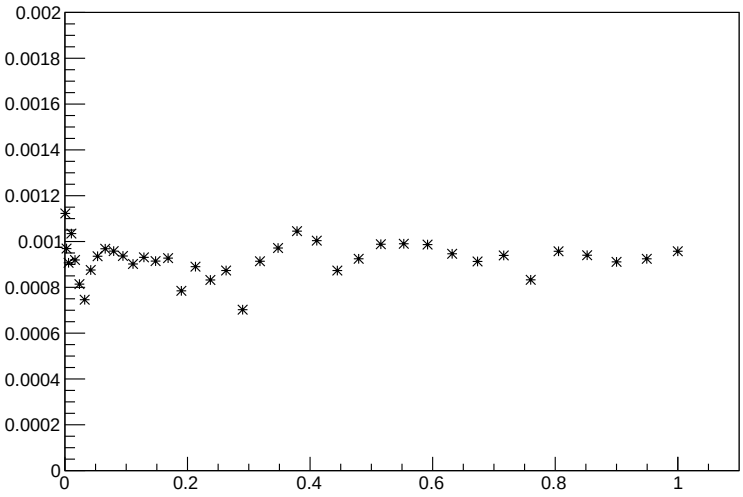

Figure A.1.: no displacement

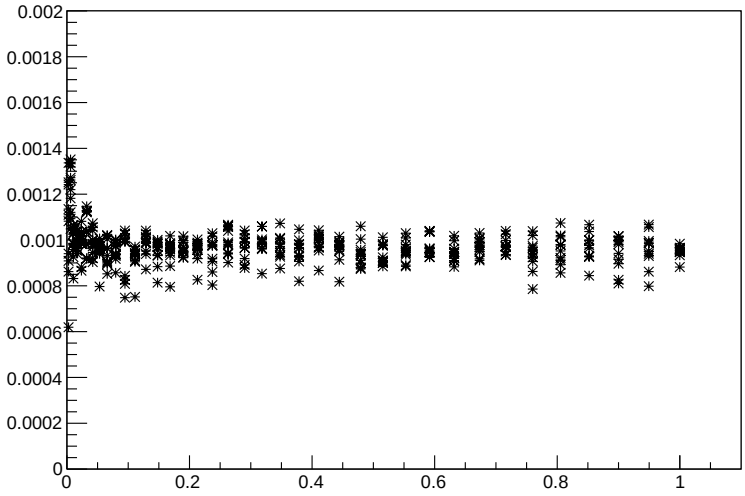

Figure A.2.: random translation of 0.01 and rotation of 3°, chosen 10 times

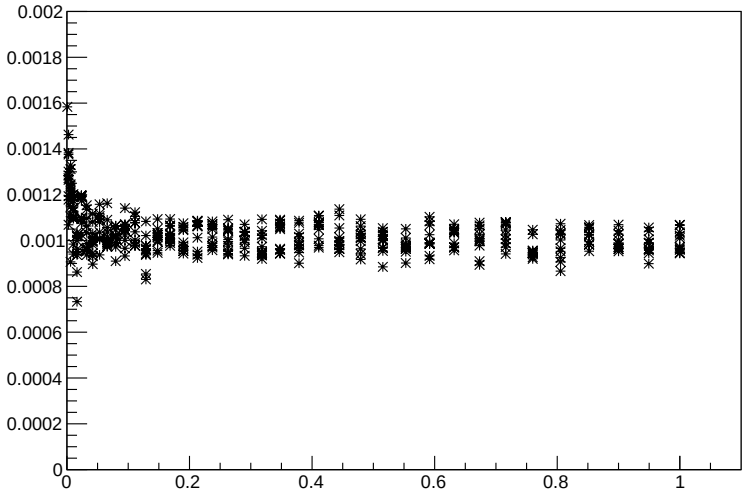

Figure A.3.: random translation of 0.01 and rotation of 15°, chosen 10 times

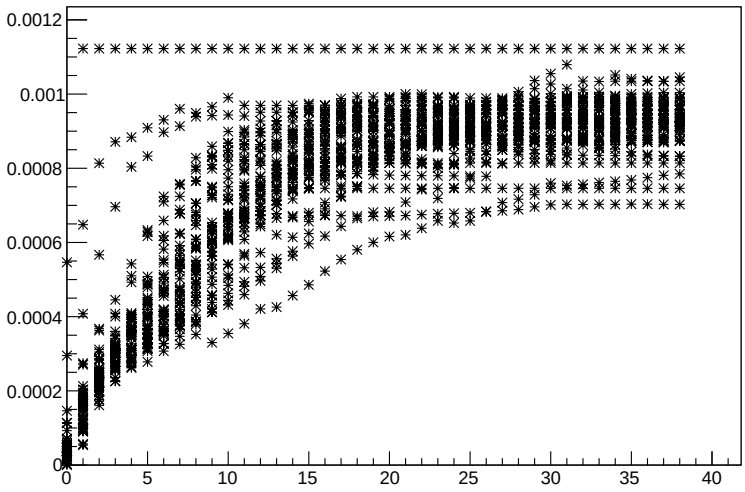

Figure A.4.: Evolutions of true error for experiment **??**

Bibliography

[Che, 1991] 1991 (April). *Object modeling by registration of multiple range images.* Vol. 3. IEEE International Conference on Robotics and Automation.

[Besl & McKay, 1992] Besl, Paul J, & McKay, Neil D. 1992 (February). A Method for Registration of 3-D Shapes. *Pages 239–256 of: IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14. IEEE.

[Bouaziz *et al.* , 2013] Bouaziz, Sofien, Tagliasacchi, Andrea, & Pauly, Mark. 2013. Sparse Iterative Closest Point. *Computer Graphics Forum (Symposium on Geometry Processing)*, **32**(5), 1–11.

[Kyöstilä *et al.* , 2013] Kyöstilä, Tomi, C., Daniel Herrera, Kannala, Juho, & Heikkilä, Hanne. 2013. Merging Overlapping Depth Maps into a Nonredundant Point Cloud. *Image Analysis, 18th Scandinavian Conference, SCIA 2013*, **7944**, 567–578.

[Lorusso *et al.* , 1995] Lorusso, A., Eggert, D.W., & FIsher, R.B. 1995. A Comparison of Four Algorithms for Estimating 3-D Rigid Transformations.

[Rusinkiewicz & Levoy, 2001] Rusinkiewicz, Szymon, & Levoy, Marc. 2001. Efficient Variants of the ICP Algorithm. vol. 3-D Digital Imaging and Modeling. Stanford University.

[Salvi *et al.* , 2005] Salvi, Joaquim, Matabosch, Carles, Fofi, David, & Forest, Josep. 2005. A review of recent range image registration methods with accuracy evaluation. *Image and Vision Computing*, August.

[Segal *et al.* , 2009] Segal, Aleksandr V., Haehnel, Dirk, & Thrun, Sebastian. 2009. Generalized-ICP. *Proceedings of Robotics: Science and Systems.*