UNIVERSITÉ LIBRE DE BRUXELLES

Faculté des Sciences
Département d'Informatique

Registration and fusion of large scale three-dimensional models

Tim Lenertz

Promoteur:
Arnaud Schenkel

Mémoire présenté en vue de
l'obtention du grade de
Master en sciences informatiques

Année académique 2014 - 2015

Contents

1. Introduction

In the context of 3D documentation projects, objects or environments are often digitized in the form of a *point cloud*. This data recorded by 3D scanners consists of a set of point coordinates lying on the object surfaces. No information about the connectivity of the surface points is retained.

To get a full 3D model of an object, scans from different view points are recorded. Before merging the point clouds, they need to be put into exactly the same coordinate system. This is called *registration*. Algorithms exists which automate this step, the most well-known of which is ICP.

For large scale 3D documentation projects, additional difficulties appear, like for example the registration of low resolution point clouds of an entire building, with high resolution point clouds focussed on a stone sculpture on the building's facade. To address this problem, an attempt is made in this thesis to develop an registration algorithm which looks at the arrangement of points in the lower resolution point cloud, to infer additional information about the object's geometry.

In chapter 2, some preliminary theory about 3D scanning and the mathematical notions on which registration algorithms are based, will be presented. Chapter 3 is a review of literature about existing registration algorithms, and in particular variants of ICP. Chapter 4 consists of two main parts: First some experiments with ICP registration are run to investigate its stability when the two point clouds are of different resolutions. Secondly, an registration error metric is developed which, as described, is based on the arrangement of points. Finally, chapter 5 describes the C++ implementation of a software framework developed for this project.

2. Theory

In this chapter some preliminary theory on 3D scanning, mathematics and computer science is presented. These notions will be applied in the following chapters.

2.1. 3D Scanning

The general goal of a 3D scanning project is to create a digital representation 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 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 artifacts, etc. The object is idealised mathematically as a single closed two-dimensional surface embedded in three-dimensional space.

To scan an object, typically 3D scanners and photogrammetry are used. Laser scanners emit light and detect the reflections from the object, in order to record a set of three-dimensional coordinates of points lying on the object's surfaces. For smaller objects, contact scanners which physically probe the object also exist. Photogrammetry consists of taking multiple photographic pictures of the object, and algorithmically recovering depth information by comparing photos from different camera poses. The scanning system can be set up such that the object and/or the scanners and cameras are rotated until the entire object is covered.

For larger of more complex objects, or for 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 [Kersten *et al.* , 2006], archeological sites [Keiner, 2011] [Grussenmeyer *et al.* , 2012] or rock formations. It may be desirable to record specific parts of the scene at a higher resolution. Data is collected using a combination of laser scanning and photogrammetry, where the scanner is kept at a fixed position relative to the object. The result is a *range image* or *depth map*, 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 consists in filtering the raw data and combining range images and photographs from different view points, in order to obtain a full 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 of a *mesh* of vertices forming triangular faces. In some applications, such as in architecture or in retro-engineering, the goal is to obtain a CAD model, in which shapes like planes or cylinders, or more complex objects, are identified.

In other contexts, specifically in medical imaging, the goal is not to scan surfaces, but the whole insides 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*.

Other applications of laser scanning include for example meteorology, where backscatter in the at-

mosphere is measured, forestry, where airborne laser scanning is used to obtain a terrain height-map with multiple layers representing the ground and treetops, the scanning of astronomical objects, robotics [Biber, 2003] or autonomous vehicles, and more.

This paper focusses on the problem of registering different scans of parts of the same object, especially in the case where the scans have different resolutions. The scanned 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. Point cloud

In the context of this paper, 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 cartesian spatial coordinates defined in a coordinate system specific for this scan. Each point can be attributed with additional information, such as an RGB color, a temperature, or the normal vector of the surface at that point. Laser scanners usually record an intensity value that records the strength of the reflected light beam at that point, and possibly a confidence value indicating the correctness of the measurement.

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. The error can be defined as the point's offset from the actual surface. Points that do not 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 to be 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 color are defined, and points get classified as inliers or outliers. Then this surfaces model is represented by means of a sparse set of points, which introduces additional considerations such as the dispersion, 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, which are possibly in motion during the scan, there is not enough information available in the point cloud to represent the surfaces of the individual leaves. When instead considering the wall to be one plane, the vegetation gets represented as an error value in subsequent processing.

2.1.2. Range image

Additional information can be contained in the ordering of the points in the recorded point cloud data. 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 the scanner was pointing at. The range image can be described as the function $r : \mathbb{N}^2 \to P \cup \{\epsilon\}$. 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.

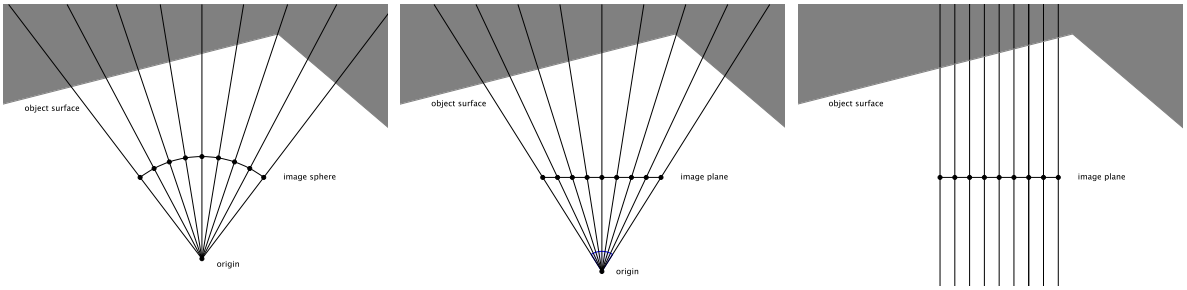
However it is such that a small difference in the pixel coordinates corresponds to a small difference in azimuth and elevation, which can for example be used to find neighbouring points efficiently. Creating a picture where each pixel (x, y) gets the color attributed to the point $r(x, y)$ (or a background color when $r(x, y) = \epsilon$), gives a photographic view of the scan. An example is shown on figures 2.1 (point color) and 2.2 (distances from point to camera), taken from the "Hôtel de Ville" scans. For this scanner the mapping to pixel coordinates is not a linear mapping of the spherical coordinates, as can be seen by looking at the curvatures of straight edges.

So the range image is a point cloud augmented by two pieces of information: The mapping of pixel coordinates to points r is given for the points in the point cloud. And, the point cloud is set in a coordinate system where the scanner is at origin $(0, 0, 0)$, and it oriented facing the direction $(0, 0, -1)^\intercal$. This knowledge will allow to make predictions on the dispersion and density of surface points.

Camera parameters

As indicated before, when working with point clouds generated from actual 3D scans, the scanner's exact mapping of 3D point coordinates to 2D pixel coordinates is not known. Supposing that the scanner is a laser scanner placed at a fixed position, it corresponds to the mapping of pixel coordinates to azimuth and elevation of the point's spherical coordinates. When generating artificial point clouds for testing purposes, a model of a scanner (called "camera" in the rest of this paper) is made for which those characteristics are well defined.

The following figure shows the 2D equivalents of three possible types of cameras:



(a) Range camera (b) Perspective camera (c) Parallel camera

The *range camera* does a linear mapping from the angle of the ray to the pixel on the image. The image, when put in three-dimensional space such that the rays pass through its pixels, has the shape of a rectangular sphere section. For the three-dimensional equivalent, additional complexities occur because of the different possible ways to define spherical coordinates. The field of view can cover the entire range of directions. This model corresponds most closely to a real laser scanner.

For the *perspective camera*, the image is a plane instead of a sphere. This corresponds to the way that photographic cameras take pictures. The resulting image contains some distortion relative to the one from the range camera, as the angles between rays get smaller towards the extremities of the field of view. On *parallel camera*, all the rays are parallel. For both the perspective and parallel camera, the mapping from 3D point coordinates to pixel coordinates on the image plane is a linear function and can be described as a multiplication with a 4×4 matrix in homogeneous coordinates. This is further described in the next sections. For the range camera, trigonometric functions are necessary to describe the mapping.

For the parallel camera, all rays have the direction of the normal vector $\vec{n}$ of the image plane. When considering only a small subregion of the field of view, the rays can also be considered to be parallel by

Figure 2.1.: Color view of range image

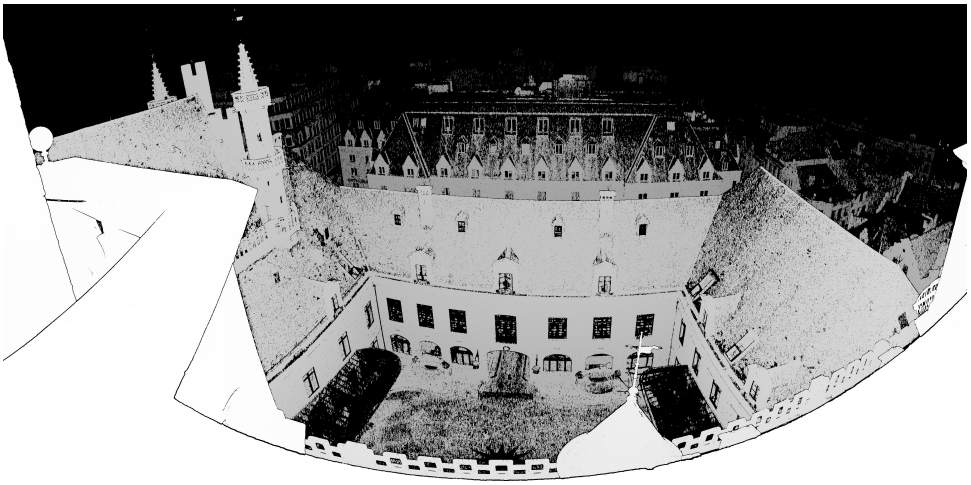


Figure 2.2.: Depth view of range image

approximation for the range and perspective cameras, as well as for real 3D scanners. Thus for any small part cropped out of a range image point cloud, the camera can be modeled as a parallel camera. These rays will also by approximation be equidistant from each other, and arranged on a square or rectangular grid on an image plane.

This way the point dispersion on small, locally planar surfaces can be modeled using a parallel camera.

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 measure 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* until the reflected beam is received by the optical sensor is measured. The distance of the object can then be determined 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 that are 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 a 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 by 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 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 Light Detection and Ranging (Lidar) scanners, especially in applications other than the scanning of solid object surfaces, as mentioned before. This is by analogy to Radio Detection and Ranging (Radar) and Sound Navigation and Ranging (Sonar), which operate in a similar way using radio waves and sound waves to detect objects, respectively.

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 camera is 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 point of view of the laser and the point of view of the camera. Also the pose of the camera relative to the laser is fixed. These three pieces of information are sufficient to calculate the three-dimensional location of the laser dot on the object surface.

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 chapter 5 data structures used to lay the point set out in memory are considered.

2.2.1. Basic operations

Fusion

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

When two point clouds represent different views of the same object and the goal is to get a point cloud the covers a greater part of the object, they must first be *registered* and put into the same coordinate system.

Fusing 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].

Transformation

To apply an affine transformation to the object represented by the point cloud, the same transformation matrix is simply applied to each point. The transformation will be relative to the origin point of the point cloud. When the points are attributed with normal vectors, the linear part of the transformation (i.e. not the translation) also needs to be applied to them.

For point cloud registration, a rigid transformation will be applied to one *loose* point cloud to put it into the coordinate system of the *fixed* point cloud.

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.

When done on a range image, the coordinate system must be maintained since the camera will still be at the origin, which can be far off the cropped area.

Closest point finding

Finding for any *position* (x, y, z), the point $p \in P$ closest to it. This can implemented efficiently by using a tree structure representation of the point cloud, as will be described in a later chapter.

Usually the Euclidian distance between points is used, though for some applications other distance metrics are useful as well.

Nearest neighbor finding

Finding for any *point* $p' \in P$ in the point cloud, the point $p \in P$ other than itself that is closest. For many applications it is useful to find the k nearest neighbors, either for one point, or for a whole set of points.

Also here, different data structures allow for implementing this in an efficient manner. Different optimizations are possible here than for the closest point problem, because p' must be a point of the point cloud, and not just any position in space.

2.2.2. 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 would be obtained from scanning the real object with a scanner having the same pose and coordinate mapping parameters. However the point cloud contains only a sparse set of point representing the surfaces, without 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, it 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 2.4 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.

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.

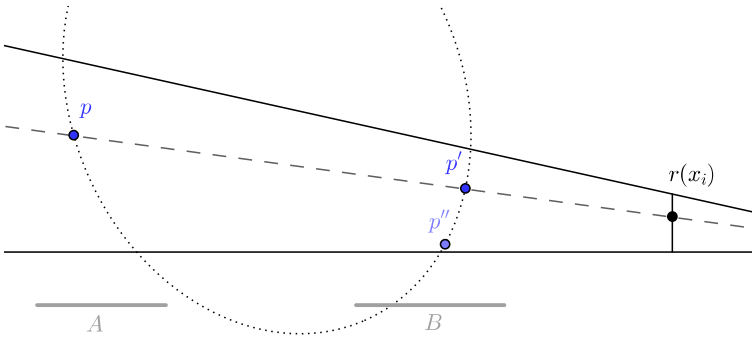


Figure 2.4.: Illustration of point cloud to range image projection in 2D

2.2.3. 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 the next chapter 3.

2.2.4. Other operations

Computation of normal vector

Computing a estimation of the normal vector of the surface at a given point $p \in P$. Different approaches exist, and because the surface is unknown there is not necessarily a best method or a clear way to evaluate its accuracy. One basic approach is to compute the k nearest neighbors for some value k, and then take the normal vector of a plane fitted to those $k + 1$ points.

Other per-point values such as a local density or curvature measure can be useful, and a version for those two will be defined chapter 4.

Image-to-cloud registration

Wrapping a photographic image on a point cloud, for example with the goal of attributing colors to points from a raw scan. This involves photogrammetric and image analysis techniques for finding correspondences between the point cloud and the image.

Meshing

Adding connectivity to the points, by constructing a triangular mesh that joins together neighboring points on the same surfaces.

2.3. 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 possibly additional attributes. The positions in one point cloud are all set in the same coordinate system. When the points are attributed 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 to any vector-type attributes of it.

2.3.1. Classification

The transformations used here are classified as follows:

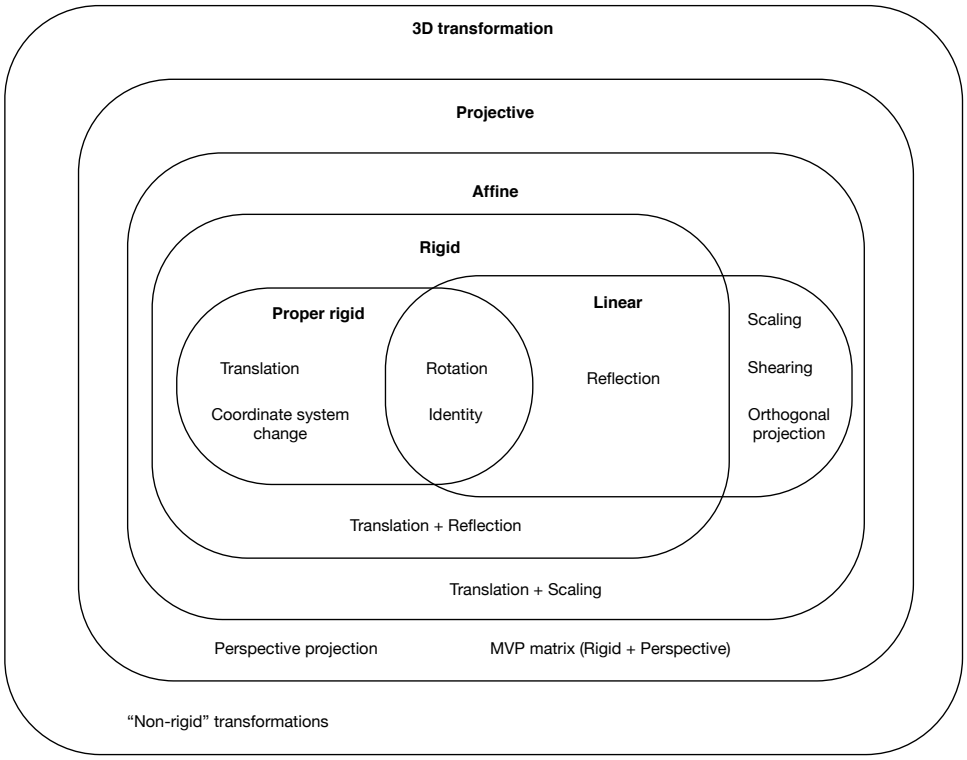


Figure 2.5.: Venn diagram of three-dimensional transformations

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 is an orthogonal matrix. It preserves the distance between every pair of points. *Proper rigid* transformations exclude reflection, 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 for all possible orders of application the results 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 any 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.

2.3.2. 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 $\mathbf{R}_\phi^x \mathbf{R}_\theta^y \mathbf{R}_\psi^z$, 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.

Rotation around an axis

A rotation can also be represented as axis of rotation $\vec{\omega}$, and an angle θ around that axis. Using *Rodrigues' rotation formula* the image $\vec{p'}$ of point $\vec{p}$ can be calculated as

$$\vec{p'} = (\cos\theta)\vec{p} + (\sin\theta)(\vec{\omega} \times \vec{p}) + (1 - \cos\theta)(\vec{\omega}\vec{p})\vec{\omega} \tag{2.5}$$

This translates directly to the rotation on a plane, by setting $\vec{u}$ to be the normal vector of that plane.

Rotation between two vectors

There are infinitely many possible rotations that map a vector $\vec{v}$ into another vector $\vec{u}$. $\vec{v}, \vec{u}$ are two linearly independent unit vectors. One preferred rotation can be chosen which fixes the plane spanned by $\vec{v}$ and $\vec{u}$.

Its axis of rotation is calculated using the cross product $\vec{\omega} = \vec{v} \times \vec{u}$, and the angle using the dot product $\theta = \arccos(\vec{v}\vec{u})$.

Quaternions

The quaternions $\mathbb{H}$ are a four-dimensional extension of the complex numbers, using four imaginary units i, j, k. The multiplication rules of the imaginary units, and the real unit 1 are defined by the fundamental formula

$$i^2 + j^2 + k^2 + ijk = -1 \tag{2.6}$$

Unlike with the real or complex numbers, quaternion multiplication is not commutative.

Unit quaternions are quaternions of norm one. They can be used to represent three-dimensional rotations and map directly to the angle-axis representation, using

$$\dot{q} = \left(\cos\tfrac{\theta}{2}, \omega_x \sin\tfrac{\theta}{2}, \omega_y \sin\tfrac{\theta}{2}, \omega_z \sin\tfrac{\theta}{2}\right) \tag{2.7}$$

A composition of rotations corresponds to the product of the quaternions, and the inverse of a quaternion corresponds to the inverse rotation. Representing rotations using quaternions is a good compromise between Euler angles and rotation matrices: There is no gimbal lock, and the representation is more compact as rotation matrices. It can also be more numerically stable as less operations are involved in

computing the composition or inverse of a rotation. Applying the rotation to a point using its quaternion representation amounts to two quaternion multiplications, or alternately the quaternion can be converted to a rotation matrix.

2.3.3. Interpolation of rotation

Interpolating a translation can be done simply by linear interpolation of its three components: $\vec{t}(t) = (1-t)\,\vec{t}_{\min} + t\,\vec{t}_{\max}$. By the distributivity of matrix by vector multiplication, this is invariant of the coordinate system. But interpolating a rotation or a complete rigid transformation is more complicated. A linear interpolation of the Euler angles, the quaternion components, or the transformation matrix components generally does not product a smooth animation because of the gimbal lock phenomenon, or because the transformation does not remain rigid.

Informally, an interpolation should such that an object whose pose changes in time between the two ends of the interpolation, moves smoothly at about a constant speed, the same way as one would manually place a real object from one pose into another.

Quaternions allow for defining a smooth interpolation between two rotations, called the *spherical linear interpolation*, or *slerp*. It is such that movements along unit-radius great circle arcs retain a constant speed. Using quaternions, it is defined as

$$s(\dot{q}_0, \dot{q}_1, t) = \dot{q}_0(\dot{q}_0^{-1}\dot{q}_1)^t \tag{2.8}$$

for $0 < t < 1$.

Quaternion exponentiation is defined using $\dot{q}_t = e^{\dot{q}\,\log t}$, where the quaternion exponential function can be written using the power series

$$e^{\dot{q}} = \sum_{n=0}^{\infty} \frac{\dot{q}^n}{n!} \tag{2.9}$$

2.3.4. 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 $\vec{p}$.

$$\vec{p''} = \mathbf{T_2}\,(\mathbf{T_1}\,\vec{p} + \vec{t_1}) + \vec{t_2} = (\mathbf{T_2}\,\mathbf{T_1})\,\vec{p} + (\mathbf{T_1}\vec{t_1} + \vec{t_2}) \tag{2.10}$$

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:

$$\mathbf{T}\,(\vec{p} + \vec{t}) = \mathbf{T}\,\vec{p} + \mathbf{T}\,\vec{t} \neq \mathbf{T}\,\vec{p} + \vec{t} \tag{2.11}$$

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:

$$\mathbf{R_2}\,(\mathbf{R_1}\,\vec{p} + \vec{t_1}) + \vec{t_2} = (\mathbf{R_2}\,\mathbf{R_1})\,\vec{p} + (\mathbf{R_2}\,\vec{t_1} + \vec{t_2}) \tag{2.12}$$

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.

2.3.5. 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 $\vec{v}$ in homogeneous coordinates corresponds to a vector $\vec{v}$ in cartesian coordinates iff

$$\vec{\hat{v}} = [x, y, z, w]^\intercal \qquad \text{and} \qquad \vec{c} = \left[\frac{x}{w}, \frac{y}{w}, \frac{z}{w}\right]^\intercal \tag{2.13}$$

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

$$\hat{\mathbf{T}} = \begin{bmatrix} l_{1,1} & l_{1,2} & l_{1,3} & t_1 \\ l_{2,1} & l_{2,2} & l_{2,3} & t_2 \\ l_{3,1} & l_{3,2} & l_{3,3} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.14}$$

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 represent 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.

[1] Using $\mathbf{L}$ here instead of $\mathbf{T}$.

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

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \times \begin{bmatrix} l_{1,1} & l_{1,2} & l_{1,3} & t_1 \\ l_{2,1} & l_{2,2} & l_{2,3} & t_2 \\ l_{3,1} & l_{3,2} & l_{3,3} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} l_{1,1}\,x + l_{1,2}\,y + l_{1,3}\,z + t_1 \\ l_{2,1}\,x + l_{2,2}\,y + l_{2,3}\,z + t_2 \\ l_{3,1}\,x + l_{3,2}\,y + l_{3,3}\,z + t_3 \\ 1 \end{bmatrix} \hat{=} \begin{bmatrix} l_{1,1}\,x + l_{1,2}\,y + l_{1,3}\,z + t_1 \\ l_{2,1}\,x + l_{2,2}\,y + l_{2,3}\,z + t_2 \\ l_{3,1}\,x + l_{3,2}\,y + l_{3,3}\,z + t_3 \end{bmatrix} \tag{2.15}$$

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.

2.3.6. 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, as will be described in chapter 5.

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

$$O_i = \mathbf{M^{-1}}\, O_{i-1} \qquad \vec{i_i} = \mathbf{M^{-1}}\, \vec{i_{i-1}} \qquad \vec{j_i} = \mathbf{M^{-1}}\, \vec{j_{i-1}} \qquad \vec{k_i} = \mathbf{M^{-1}}\, \vec{k_{i-1}} \tag{2.16}$$

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

Transformation matrices that express position in one coordinate system in another coordinate system are computed by traversing the tree, and calculating a conjunction of transform-to-parent and transform-from-parent matrices.

2.4. 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:

$$\hat{\beta} = \arg\min_{\beta} S \qquad \text{where} \qquad S = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{2.17}$$

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 are 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. [Eberly, 1999]

2.4.1. 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:

$$\frac{\partial S}{\partial \beta_j} = -2 \sum_{i=1}^{n} \frac{\partial f_\beta(x)}{\partial \beta_j} = 0 \tag{2.18}$$

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

$$f_\beta(x) = \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_m x_m \tag{2.19}$$

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

$$\mathbf{X}\,\hat{\beta} = \vec{y} \tag{2.20}$$

where $\mathbf{X}$ is a $n \times m$ 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

$$S = \sum_{i=1}^{n} \|y_i - \sum_{j=1}^{m} X_{i,j}\beta_j\|^2 = \|y - \mathbf{X}\,\vec{\beta}\|^2 \tag{2.21}$$

From 2.19, the partial derivative of a residual becomes

$$\frac{\partial (y_i - f_\beta(x))}{\partial \beta_j} = -X_{i,j} \tag{2.22}$$

Putting 2.22 into 2.18 and rearranging the expression, one obtains the *normal equations*:

$$\sum_{i=1}^{n}\sum_{k=1}^{m} X_{ij}\, X_{ik}\, \hat{\beta}_k = \sum_{i=1}^{n} X_{ij}\, y_i \qquad \text{for} \qquad j = 1, 2, \ldots, m \tag{2.23}$$

Written in matrix form:

$$(\mathbf{X}^\intercal\, \mathbf{X})\, \hat{\beta} = \mathbf{X}^\intercal\, \vec{y} \tag{2.24}$$

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

$$\hat{\beta} = (\mathbf{X}^\intercal\, \mathbf{X})^{-1}\, \mathbf{X}^\intercal\, \vec{y} \tag{2.25}$$

2.4.2. 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.

2.4.3. 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* $w_i \in \mathbb{R}$ 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

$$S = \sum_{i=1}^{n} w_i\, (y_i - \hat{y}_i)^2 \tag{2.26}$$

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 $\lambda w_i \in \mathbb{N}$. This justifies the previous statement about data points duplication.

2.5. 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. [Horn, 1987] 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

$$S = \sum_{i=1}^{n} w_i\, \|\vec{p_i} - (\mathbf{R}\, \vec{q_i} + \vec{t})\|^2 \tag{2.27}$$

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:

$$\vec{c_p} = \frac{1}{N}\sum_{i=1}^{n}\vec{p_i} \qquad \text{and} \qquad \vec{c_q} = \frac{1}{N}\sum_{i=1}^{n}\vec{q_i} \tag{2.28}$$

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), then rotating around the centroid will move each point by about the same amount, again evening out the error.

A rotation matrix $\mathbf{R}$ expresses 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

$$\vec{p_i'} = \vec{p_i} - \vec{c_p} \qquad \text{and} \qquad \vec{q_i'} = \vec{q_i} - \vec{c_q} \tag{2.29}$$

and the minimization problem becomes

$$S' = \sum_{i=1}^{n} w_i \left\| \vec{p_i'} - \mathbf{R}\,\vec{q_i'} \right\|^2 \tag{2.30}$$

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. [Horn, 1987] describes a solution whereby $\mathbf{R}$ is represented as a unit quaternion. This was used in the original description of the ICP algorithm (see 3.3.1) 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, 2.30 becomes

$$S' = \sum_{i=1}^{n} w_i \left\| \vec{p_i'} \right\|^2 + w_i \left\| \vec{q_i'} \right\|^2 - 2\, w_i\, \vec{q_i'}^{\mathsf{T}}\, \mathbf{R}\, \vec{p_i'} \tag{2.31}$$

The first and second terms do not depend on $\mathbf{R}$, hence the problem is reduced to maximizing $\sum_i w_i\, \vec{q_i'}^{\mathsf{T}}\, \mathbf{R}\, \vec{p_i'}$.

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

$$\mathbf{W}\,\mathbf{Q}^{\mathsf{T}} = \begin{bmatrix} - & w_1\, \vec{q_1'}^{\mathsf{T}} & - \\ - & w_2\, \vec{q_2'}^{\mathsf{T}} & - \\ & \vdots & \\ - & w_n\, \vec{q_n'}^{\mathsf{T}} & - \end{bmatrix} \qquad \text{and} \qquad \mathbf{R}\,\mathbf{P} = \begin{bmatrix} | & | & & | \\ \mathbf{R}\,\vec{p_1'} & \mathbf{R}\,\vec{p_2'} & \ldots & \mathbf{R}\,\vec{p_n'} \\ | & | & & | \end{bmatrix} \tag{2.32}$$

It can be seen that

$$\sum_{i=1}^{n} w_i\, \vec{q_i'}^{\mathsf{T}}\, \mathbf{R}\, \vec{p_i'} = \mathrm{tr}(\mathbf{W}\,\mathbf{Q}^{\mathsf{T}}\,\mathbf{R}\,\mathbf{P}) \tag{2.33}$$

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

$$\mathrm{tr}((\mathbf{W}\,\mathbf{Q}^{\mathsf{T}})\,(\mathbf{R}\,\mathbf{P})) = \mathrm{tr}((\mathbf{R}\,\mathbf{P})\,(\mathbf{W}\,\mathbf{Q}^{\mathsf{T}})) = \mathrm{tr}(\mathbf{K}\,\mathbf{R}) \tag{2.34}$$

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

$$\mathbf{K} = \mathbf{P}\,\mathbf{W}\,\mathbf{Q}^\intercal = \sum_{i=1}^{n} w_i\,\vec{p_i'}\,\vec{q_i'}^\intercal \tag{2.35}$$

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

$$\mathbf{K} = \mathbf{U}\,\mathbf{\Sigma}\,\mathbf{V}^\intercal \tag{2.36}$$

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

$$\mathrm{tr}(\mathbf{K}\,\mathbf{R}) = \mathrm{tr}(\mathbf{\Sigma}\,\mathbf{V}^\intercal\,\mathbf{R}\,\mathbf{U}) \tag{2.37}$$

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 $\mathbf{N} = \mathbf{I}$.

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

$$\mathbf{N} = \mathbf{V}^\intercal\,\mathbf{R}\,\mathbf{U} = \mathbf{I} \Rightarrow \mathbf{V} = \mathbf{R}\,\mathbf{U} \Rightarrow \mathbf{R} = \mathbf{V}\,\mathbf{U}^\intercal \tag{2.38}$$

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

$$\vec{t} = \vec{c_p} - \mathbf{R}\,\vec{c_q} \tag{2.39}$$

2.6. Procrustes analysis

This problem of aligning two 3-dimensional point sets of n points can be considered to be a special case of *Procrustes analysis*, where it is generalized in respect to the number n of points and the number of dimensions.

In statistical shape analysis, the *shape* of a geometrical object is understood to be to remainder when information about its position, orientation and scale are removed. The process of finding an optimal transformation that aligns the objects is called Procrustes superimposition, and a metric that compares the shapes of two aligned objects is the Procrustes distance. In *full* Procrustes superimposition the scale of an object is also variable, whereas for the *partial* variant it is part of the shape and only the rotation and translation are variable.

[2] Also called *correlation* matrix.

As shown before, finding the optimal translation is trivial and amounts to calculating the difference vector of the means. When applicable, the scale can be found in a similar way. The *orthogonal Procrustes problem* is the method of finding the orientation, for which singular value decomposition is a possibility in the three-dimensional case. In the two-dimensional case, a simpler closed form least squares solution exists.

Generalized procrustes analysis is the problem of optimally aligning more than two object simultaneously, and can be done in such a way that the error is evenly distributed between the objects. In the context of sets of n three-dimensional points, it can be solved as follows: Each of the k point sets is encoded as using a $3 \times n$ *model matrix* $\mathbf{X}_j$ $(1 \leq j \leq k)$. Its n columns contain the 3 coordinates of each point in the j-th point set. The points in each of these matrices are supposed to represent approximatively the same *shapes*. Let $\mathbf{X'}_j$ denote a version of $\mathbf{X}_j$ on which a rigid transformation has been applied. The *goal* is given by the matrix

$$\mathbf{K} = \frac{1}{k} \sum_{i=1}^{k} \mathbf{X'}_j \tag{2.40}$$

When all $\mathbf{X'}_j$ are such that the point sets are perfectly aligned, $\mathbf{K}$ represents the shape the best possible way, since the errors in the representations $\mathbf{X'}_j$ are averaged out.

The optimal matrices $\mathbf{X'}_j$ and their rigid transformations can be computed iteratively by numerical approximation: [Toldo *et al.* , 2010] First $\mathbf{K}$ is initialized so some initial value, for example one of the model matrices. Then for each model matrix $\mathbf{X}_j$, the matrix $\mathbf{X'}_j$ is computed which optimally aligns it with the current $\mathbf{K}$. Now $\mathbf{K}$ is updated using formula 2.40, and the process is repeated until $\mathbf{K}$ stabilizes.

The transformed matrices $\mathbf{X'}_j$ are calculated as follows: The rigid transformation is represented by a rotation matrix $\mathbf{R}$, a translation vector $\vec{t}$ and possibly a scaling factor c. $\mathbf{X'}_j$ is defined as

$$\mathbf{X'}_j = c\mathbf{X}_j\mathbf{R} + \vec{1}\vec{t}^{\intercal} \tag{2.41}$$

Here $\vec{j}$ is a $n \times 1$ unit vector, and so $\vec{1}\vec{t}^{\intercal}$ becomes a $n \times 3$ matrix where each column is a copy of $\vec{t}$. With $\mathbf{X}_j$ $\mathbf{R}$, the rotation is applied to each column of $\mathbf{X}_j$.

A closed form solution exists to find the $\mathbf{R}$, $\vec{t}$ and c which minimize

$$e = \|\mathbf{X'}_j - \mathbf{K}\|_F \tag{2.42}$$

The corresponding algorithms are described in [Schönemann & Caroll, 1970] and [Schönemann, 1966]. The notation $\|\mathbf{X'}\|_F$ denotes the *Frobenius norm* of the matrix, defined as the sum of squares of *all the components* of the matrix. This is still a valid error measure because it converges to zero when $\mathbf{X'}_j$ and $\mathbf{K}$ become aligned. However, unlike a sum of squares of vector norms, it introduces some bias towards the X, Y and Z axis.

2.7. RANSAC

The random sample consensus (RANSAC) algorithm, first described by [Fischler & Bolles, 1980], is another, probabilistic approach to find parameters β of a model $\hat{y}_i = f_\beta(x_i)$ which best fits a set of *data points* $\{x_i, y_i\}$. The algorithm *randomly* choses a minimal number of data points and constructs a model from it, which is then refined using the remaining data points. The process is repeated multiple times until a model has been found that fits the data points well enough.

The RANSAC algorithm proceeds as described here, exemplified by the problem of finding parameters $\beta = (a, b)$ of model for a line $f_b(x) = ax + b$ that best fits a set of data points $P = \{(x_i, y_i)\}$.

1. A minimal subset $S \subset P$ of data points is randomly chosen. Minimal means it contains the least number of data points needed to construct an estimative model $\hat{\beta}$. In this example it would be two data points, since a line can be defined using two points. S is called the *sample*.

2. A subset $S^* \subset P$ of data points is taken which fit the model well enough, according to some error metric $E(\beta, p)$. Formally, $S^* = \{p \in P : E(\hat{\beta}, p) < \epsilon\}$. This set S^* is called the *consensus set*. For this example, $E(\beta, p)$ could be the distance of p to the line described by β, and ϵ some maximal allowed distance.

3. If $|S^*| < t$ for a predefined threshold t, there are not enough data points that fit the model. It is rejected, and the algorithm repeats from step 1. t could be chosen in function of the total number of data points.

   To make sure that the algorithm halts, the number of times that this situation occurs are counted, and if it happens more than $n_{\max}$ times, the algorithm stops and reports an error.

4. Otherwise, if $|S^*| \geq t$ the model $\hat{\beta}$ is accepted. When possible, a better estimation for β is then computed from all the points in the consensus set. For the line-fitting example, this could be a linear least squares solution.

Unlike the least squares method, RANSAC operates using *minimal*, randomly chosen samples from the data points. This makes it much less sensitive to outliers: Firstly, the probability of choosing from a large set of data points, a small set of outliers is low. Secondly, if S contains at least one outlier, $\hat{\beta}$ will be far off the optimal solution, and as a result the consensus set will be rejected.

Application to point set alignment

RANSAC can also be applied for three-dimensional point set alignment. Given two point clouds P and Q, the data points for RANSAC would be all the possible correspondence pairs (p_i, p_j).

For the sample S, 3 correspondence pairs are chosen. The estimative model is the rigid transformation $\mathbf{M}_\beta$ which best aligns those 3 point pairs, with the scaling factor removed. This can be computed using a closed form solution.

The consensus set S^* is the set of correspondences (p, q) such that $d(p, \mathbf{M}_\beta q) < \epsilon$. At least a subset of it can be found efficiently using a closest point or k-nearest-neighbor algorithm. When accepted, the estimated transformation $\mathbf{M}_\beta$ can be improved using singular value decomposition as described before.

2.8. Histogram comparison

Let R be a random variable, and $S = \{s_i\}$ a discrete set of samples of R. A histogram is a way to graphically represent S in such a way that the underlying probability distribution becomes visible. The domain of R is divided into bins of equal width w. For each bin b_j, the number of samples that fall into it are counted. That is, $b_j = N(s \in S : j\,w \leq s < (j+1)\,w)$. Each sample falls into exactly one bin, and $\sum_j b_j = N(S)$. The values b_j are called the *observed frequencies* o_i.

The bins are then represented using a bar chart. The abscissa axis covers the domain of R. Each bar has width w and height b_j. The lower w is, the higher the bars get. The total area of the bars is proportional to $N(S)$. Because of the law of large numbers, when $N(S) \to \infty$ and $w \to 0$, and the heights are normalized so that the total area becomes 1, the histogram becomes a plot of the probability density function of R. There is no optimum for w or for the number of bins, a good choice depends on the probability distribution, and on what the histogram will be used for.

Goodness of fit tests are used to determine, based on the histogram, how well the samples S match the theoretical probability distribution of R. Alternately they are used to test if two sets of samples S_1 and S_2 are likely to have come from the same (unknown) probability distribution of R.

The *null hypothesis* states that the samples are taken independently, that is, two samples are not statistically related. When this is true an *expected frequency* e_i for any bin of the histogram can be calculated from the theoretical probability distribution. When tests are used to compare two histograms, e_i is set to the corresponding observed frequency from the other histogram, after both have been normalized.

Chi-Squared

The *chi-squared distribution*, denoted χ^2, is the probability distribution of the sum of squares of k independent random variables that each have the standard normal distribution. Its probability density function diverges to infinity at $x = 0$ and converges to 0 as $x \to \infty$. The standard normal distribution is the Gaussian distribution with mean $\mu = 0$ and variance $\sigma^2 = 1$.

A *chi-squared test* is any test where the sampling distribution of the test statistic is a chi-squared distribution when the null hypothesis is true. For the most commonly used *Pearson's chi-square test* the following value is calculated:

$$d_{\text{chi}} = \chi^2 = \sum_{i=1}^{k} \frac{(o_i - e_i)^2}{e_i} \tag{2.43}$$

Here k is the number of histogram bins. So the test statistic is the squared difference between the observed and expected frequency, divided by the expected frequency, for each bin. Under the null hypothesis, the differences $o_i - e_i$ will have a normal distribution and the sum of their squares, normalized to e_i, a chi-squared distribution.

The goodness of fit is then determined by comparing this value to the chi-squared distribution by calculating a *p-value*.

Correlation

Another approach is to measure the correlation between the observed frequencies and the expected frequencies. The *Pearson's correlation coefficient* of two random variables X and Y is defined as

$$\rho_{X,Y} = \frac{E[(X - \mu_X)\,(Y - \mu_Y)]}{\sigma_X\,\sigma_Y} \tag{2.44}$$

When X and Y are independent, $\rho_{X,Y} = 0$. When there is a perfect linear relationship between the variables, i.e. $Y = a + b\,X$, it is 1 or -1, depending on whether b is positive or negative.

When working with discrete samples, in this case observed and expected frequencies on histogram bins, both the mean and variance are estimated from the samples, and the correlation coefficient is calculated as

$$d_{\text{cor}} = \rho = \frac{\sum_i (o_i - \bar{o})\,(e_i = \bar{e})}{\sqrt{\sum_i (o_i - \bar{o})^2\ \sum_i (e_i - \bar{e})^2}} \tag{2.45}$$

3. State of the Art

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 camera 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. Unless P and Q were algorithmically generated from the same 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 lattice gets and the density gets lower.

When large objects are scanned, 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 scanned 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 or 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 with the current alignment.

3.3.1. Iterative Closest Point

The most well-known fine registration algorithm is 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 [Toldo *et al.* , 2010].

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 *down-sampling 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 detailed in section 2.5. 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 2.5. 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.

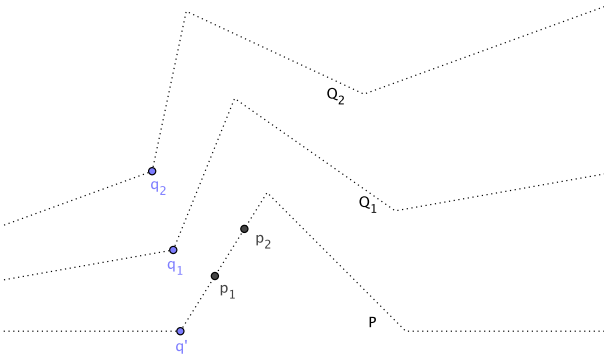


Figure 3.1.: Single point at two ICP iterations

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

$\lim_{q \to q'} d(p, q) = 0$. 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 p_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.

In [Liu, 2008] additional geometric properties of the closest point criterion are identified, which justify its use for point set registration and other application, and show that the correspondences are of "high relative" quality.

Local minima

ICP minimizes an error function $e_{\mathbf{M_c}} : \mathbf{M} \mapsto \mathbb{R}$ taking as input a rigid transformation. The function $e_{\mathbf{M_c}}$ is different at each iteration, and depends on current correspondences. $\mathbf{M_c}$ denotes the transformation estimation for this iteration, based on which the correspondences were taken.

One can define a "global" error function $e(\mathbf{M}) = e_{\mathbf{M}}(\mathbf{M})$. It is non-continuous because different correspondence pairs get chosen as $\mathbf{M}$ changes, and it can not directly be minimized. The global minimum of e represents the sought after optimal transformation. Since the correspondences change only at discrete locations, e is equal to one of the "local" error functions $e_{\mathbf{M_c}}$ in the neighborhood of an input $\mathbf{M}$.

Each local error function $e_{\mathbf{M_c}}$ (for fixed $\mathbf{M_c}$) generally has one global minimum and no local minima, unless there is some symmetry in the point constellation. At each iteration $\mathbf{M}$ is moved to this minimum. For each $e_{\mathbf{M_c}}$ the global minima will be different because the correspondences are not correct, and each of those global minima forms a local minimum in the global error function e.

To summarize:

Error function	global error function e	local error function $e_{\mathbf{M_c}}$
Correspondences	depend on input	fixed, transformation $\mathbf{M_c}$
Global minimum	optimal transformation	transformation estimation
Local minima	global minimum of $e_{\mathbf{M_c}}$, incorrect convergence	generally none
Minimization	no direct solution	least squares or other
Continuous	no	yes
Computation	expensive, need to compute correspondences	cheap

Figure 3.2 is an example of a global error function e. The Y axis indicates the value $e(\mathbf{M})$, the X axis is an interpolation between two randomly chosen points in the 6-dimensional space of rigid transformations. Visualization of error metrics will be described in more detail in section 4.1.3 of the next chapter. The point clouds P and Q used here are two different, coarsely aligned scans of the same object. It can be seen that that the optimal transformation is likely in the region $x \approx 0$. But local minima appear farther off, as well as song smaller ones due to discontinuities.

It is possible for ICP to converge towards one of the local minima at $e(\mathbf{M}_{\text{lmin}})$. All the local error functions $e_{\mathbf{M}\text{near lmin}}$ in which take effect in the neighborhood of $\mathbf{M}_{\text{lmin}}$, will still their global minimum near $\mathbf{M}_{\text{lmin}}$.

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

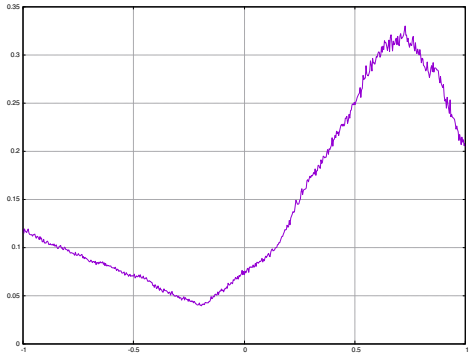


Figure 3.2.: Example of global error function $e(\mathbf{M})$ with local minima

The Go-ICP algorithm, described in [Yang *et al.* , 2013] is a globally optimal method based on ICP, which always converges towards the global minimum of the error metric e. It uses the mentioned properties of the error metrics, and implements a branch-and-bound scheme which searches the space of rigid transformations.

Multi-view ICP

It is also possible to extend ICP to multi-view instead of pairwise registration. [Toldo *et al.* , 2010] presents an algorithm within the ICP framework, where the error metric and correspondence finding steps are modified in order to align k point clouds simultaneously.

For the correspondences, instead of pairs of closest points, k-tuples of *mutually nearest neighbors* are searched. Each ordered pair (p, q) of two points within a tuple must be such that p is the closest point to q.

For the error minimization, *generalized Procrustes analysis* (GPA) is applied, as described before in section 2.6, in order to find the rigid transformations $\mathbf{M}_i$. At each iteration of ICP, the GPA target $\mathbf{K}$ is initialized to the centroids of the correspondence tuples, and then the Procrustes algorithm is run.

Iterative Dual Correspondences

[Lu & Milios, 1997] describes the *iterative dual correspondences* algorithm, an variant of ICP in two-dimensions in which two sets of correspondences are computed at each iteration: One using the closest point criterion, and the second using the "matching range point rule": In polar coordinates, the difference between radii is minimized and the difference in angles constrained to a small interval.

For the error metric the closest point correspondences are used for the translation estimation, while the rotational estimation is computed using the matching range point rule correspondences.

This method was shown to be more robust than standard ICP when the error in the initial pose is large. [Magnusson, 2013]

Probabilistic Iterative Correspondence

This method is described in [Montesano *et al.* , 2005]. It incorporates information from scanner errors and the uncertainty of the initial pose estimate, by modeling the points as random variables taken from a Gaussian distribution. Generalized ICP, described in [Segal *et al.* , 2009], is a similar approach. It is described in more detail in the next section.

Sparse ICP

The least *squares* error metric used by ICP poses the problem that outliers and wrong correspondences, which have greater distances than good correspondences, get an over-proportionally larger weight in the error metric because their squares are taken. This makes ICP sensitive to noise in the input point clouds.

Sparse-ICP, described in [Bouaziz *et al.* , 2013], is a variant which instead minimizes an error metric $\sum_i \|p_i - \mathbf{M}\, q_i\|^p$ with $p < 1$. Geometrically this still measures the Euclidian distance, but that number is put to a power other than 2, giving correspondences with lower distance a higher weight.

The error metric is *sparse* in the sense that the correspondences get implicitly classified into outliers and inliers, because $\|d\|^p \approx 0$ for inliers and $\|d\|^p \approx 1$ for outliers.

The error minimization is done with the *alternating direction method of multipliers*, which essentially translates it into two least squares problems.

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 field 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. The other components of the ICP framework are not changed, and in particular, points correspondences are still established using the closest point criterion.

Let $p \in P$ and $q \in Q$ be the points from the two point clouds to register, and let (p_i, q_i) represent pairs deemed to be corresponding points by the closest point criterion or other. The model assumes the existence of underlying sets of unknown points $\{p'_i\}$ and $\{q'_i\}$ which are such that the correspondences are perfectly correct. That is, : $p'_i = \mathbf{M}\, q'_i$. The points p_i and q_i on which the registration is performed are supposed to be random variables generated from p'_i and q'_i, taken from a normal probability distribution.

$$p_i \sim \mathcal{N}(p'_i, \mathbf{C}_i^P) \qquad \text{and} \qquad q_i \sim \mathcal{N}(q'_i, \mathbf{C}_i^Q) \tag{3.1}$$

The expression $\mathcal{N}(p'_i, \mathbf{C}_i^P)$ represents a three-dimensional multivariate normal distribution, with p'_i as mean and with covariance matrix $\mathbf{C}_i^P$. The mean p'_i is unknown, and $\mathbf{C}_i^P$ (and $\mathbf{C}_i^Q$) are attributes associated to each point beforehand.

This models both the error in both points p'_i and q'_i which makes them deviate from the underlying surface, and the imprecision of the correspondences due to the different point dispersions on P and Q.

Let $d_{i,\mathbf{M}}$ be the distance between p_i and q_i with estimated transformation $\mathbf{M}$.

$$d_{i,\mathbf{M}} = p'_i - \mathbf{M}\, q'_i \tag{3.2}$$

Because p_i and q_i are assumed to have been taken from two independent normal distributions, their distance $d(p_i, q_i)$ is also a random variable with a normal distribution that can be expressed in terms of C_i^P, C_i^Q and the unknown true transformation M:

$$d_{i,\mathbf{M}} \sim \mathcal{N}(0, \mathbf{C}_i^P + \mathbf{M}\, \mathbf{C}_i^Q\, \mathbf{M}^\intercal) \tag{3.3}$$

Using maximal likelihood estimation, $\hat{\mathbf{M}}$ can be estimated:

$$\begin{aligned}\hat{\mathbf{M}} &= \arg\max_{\mathbf{M}} \prod_i p(d_{i,\mathbf{M}}) \\ &= \arg\max_{\mathbf{M}} \sum_i \log(p(d_{i,\mathbf{M}})) \\ &= \arg\min_{\mathbf{M}} \left[d_{i,\mathbf{M}}^{\intercal} \left(\mathbf{C}_i^P + \mathbf{M}\,\mathbf{C}_i^Q\,\mathbf{M}^{\intercal}\right)^{-1} d_{i,\mathbf{M}} \right]\end{aligned} \tag{3.4}$$

This is the error metric which is minimized by the generalized ICP algorithm.

Covariance matrices

When $\mathbf{C}_i^P = \mathbf{I}$ and $\mathbf{C}_i^Q = 0$, the error metric simplifies to the point-to-point error metric. The three diagonal components of $\mathbf{C}_i^P$ represent the variances of the position of p_i relative to p_i', along the X-, Y- or Z-axis. These axis in world space are meaningless in the context of the local neighborhood of points, and instead the coordinate system is set such that one axis is the normal vector of the point. These normal vectors need to have been computed beforehand. So

$$\mathbf{C}_i^P = \mathbf{R}_i \begin{pmatrix} x & 0 & 0 \\ 0 & y & 0 \\ 0 & 0 & z \end{pmatrix} \mathbf{R}_i^{\intercal} \tag{3.5}$$

Where $\mathbf{R}_i$ is a rotation matrix which rotates the surface normal vector $\vec{n}$ of point p_i to the X axis. Now x is set to a low value because a deviation along the axis of the normal vector corresponds to a deviation from the underlying surface and can be considered a scanner error. y and z represent a deviation along the local tangent plane of the surface, and is set to a larger value because a greater deviation is expected due to different point dispersions. The remaining components are set to 0 since there is no covariance exists on the differences in two directions. When $x = 0$ and $y, z \to \infty$, the formula 3.4 becomes the point-to-plane error metric. $\mathbf{C}_i^Q$ is constructed in an equivalent way.

3.3.3. Normal Distributions Transform

The normal distribution transform (ICP) algorithm is an approach which converts the point clouds into a piecewise continuous representation, called the *normal distributions transform*, and then registers then using continuous optimization. In it first described by [Biber, 2003] in the context of robotics and for two-dimensional point clouds representing a map of a mobile robots's environment. Starting with an initial alignment taken from movement sensors on the robot, the algorithm performs fine registration of the map with a model point cloud in an efficient way.

Both two-dimensionals point cloud P and Q are split into a grid of regular square cells. For each cell a gaussian is calculated which best represents the distribution of points in that cell. Let n be the number of points in one cell, and let $\{p_i\}$ be the set of those n points.

The gaussian is a two-dimensional multivariate normal distribution. Its mean $\vec{\mu}$ and its 2×2 covariance matrix $\mathbf{C}$ are calculated from the point in the cell:

$$\vec{\mu} = \frac{1}{n} \sum_{i=1}^{n} \vec{p_i} \qquad \text{and} \qquad \mathbf{C} = \frac{1}{n-1} \sum_{i=1}^{n} (\vec{p_i} - \vec{\mu})(\vec{p_i} - \vec{\mu})^{\intercal} \tag{3.6}$$

Let $p(\vec{x})$ be the probability density function of this gaussian. Now the surface integral of p over an area $\mathcal{A}$, multiplied by n, gives an expected value of the number of points in that area. $p(\vec{x})$ represents the

marginal probability that a point is at position $\vec{x}$. Because of this $\sum_i p(\vec{x_i})$ is maximal when all x_i are points from P.

The normal distribution transform consists the two-dimensional array of there gaussians, for each cell of the grid. Computationally 6 real values are stored per cell ($\vec{mu}$ and $\mathbf{C}$) and the registration will be performed only using those values, without looking at the underlying points. For the fixed point cloud P, it is sufficient to store only the normal distribution transform.

The algorithm proceeds in the following steps:

1. The ICP for the fixed point cloud P is created.
2. The transformation parameters $T = (t_x, t_y, \phi)$, which are going align Q to P, are initialized to some first estimation.
3. The loose point cloud Q is transformed according to T.
4. For each point $q_i \in Q$, the cell c_j from the NDT of P in which it falls is determined, which is trivial because it is a regular square grid.
5. The estimation of T is improved by maximizing the *score* $\sum_{i=1}^{N(Q)} p_{c_j}(\vec{q_i})$. Here p_{c_i} is the probability density function of the gaussian in the cell c_i from the NDT of P. As stated before it is maximal when $\{q_i\}$ and $\{p_i\}$ coincide. The maximization of the (continuous) score function is done using the multi-dimensional Newton's method.
6. Repeat from step 2, until T stabilizes.

[Dold *et al.* , 2007] describes a simple way to extend the ICP algorithm to three dimensions, but still for two-dimensional transformation parameters: The point clouds is split into multiple 2D cross-sections, for example on different height layers. Then the sums of the score functions of the different layers are maximized. A real three-dimensional form of ICP is presented in [Magnusson *et al.* , 2007] and further elaborated in [Magnusson, 2013]. Here the cells are 3D subspaces, and a 3D rigid transformation is computed. The complexities of 3D rotation complicate this problem. In addition to this, a fixed discretisation, where the space is divided into a regular grid of same sized cubes is usually not appropriate: Since the point cloud is an embedding of two-dimensional surfaces in three-dimensional space, the three-dimensional density is not uniform and sparsely distributed, and so smaller cube sizes are required in some parts only. The articles suggest instead an Octree discretisation where different sized overlapping cubes forming an Octree are used, or alternatively an iterative discretisation. Here cube sizes are reduced at each iteration as the algorithm progresses and the transformation estimation gets closer to convergence.

[Magnusson, 2013] also addresses irregularities due to the discontinuities at he cell boundaries using an interpolation method, and the inclusion of color data from the point attributes into the ICP score maximization.

3.3.4. 4-Points Congruent Sets

The 4-points congruent sets (4PCS) algorithm, a completely different approach for fine point cloud registration, is presented in [Aiger *et al.* , 2008]. An improved version called Super-4PCS is described in [Mellado *et al.* , 2014]. It is based on a RANSAC scheme, and is essentially an improvement to the basic method described before in section 2.7 which reduces its complexity and convergence speed.

1. Instead of choosing three point pairs as sample, 4PCS starts by randomly choosing a quadruplet[2] of approximatively coplanar points from the fixed point cloud P. This sample S will be called the *base*.

2. Now the set U of quadruplets of points from Q, which are *congruent* to the base S is computed. Congruent means that S and $u_i \in U$ are (approximatively) related by a rigid transformation $\mathbf{M}_i$.

3. For each u_i, a consensus set $S_i^* \in Q$ is computed which consists of the points $q \in Q$ that

$$\min_{p \in P} \|p - \mathbf{M}_i q\|$$

   is below some threshold distance. These are the points $q \in Q$ that get moved close to the point cloud P by the transformation $\mathbf{M_i}$.

   When $\mathbf{M_i}$ is accurate, S_i^* will include all the points from Q that have a close corresponding point in P. If the planes spanned by S and u_i have approximatively the same normal vectors in P and in Q a good estimate for $\mathbf{M_i}$ is found.

4. The consensus set S^* is then formed by taking the S_i^* which contains the largest number of points. Its corresponding $\mathbf{M_i}$ becomes a candidate for the final transformation estimation. It is the best transformation that can be found using S as *base*.

5. The algorithm is repeated several times, each time with a different, randomly chosen base. At the end the best candidate for the transformation estimation, for which $\|S^*\|$ was the largest, is retained.

Because of the -based approach of choosing inlier points, 4PCS is robust to noise. It also does not need any initial alignment of P and Q.

Search of congruent quadruplets

The fundamental operation of the 4PCS operation is the second step, in which the set of all quadruples that are congruent to S is computed. A naive implementation would consider all $|Q|^4$ sets of four points from Q, and then check if they are coplanar, and if they are congruent to S.

Because the 4 points in $S = (p_1, p_2, p_3, p_4)$ are coplanar, they necessarily contain two pairs of points that form intersecting lines. Without loss of generality, let p_1p_2 and p_3p_4 be those lines, and i_p their intersection point.

It can be shown that the numbers

$$r_1 = \frac{\|p_1 - i_p\|}{\|p_1 - p_2\|} \qquad \text{and} \qquad r_2 = \frac{\|p_3 - i_p\|}{\|p_3 - p_4\|} \tag{3.7}$$

are invariant with respect to an affine transformation of the four points, and that they always change when a non-affine transformation is applied to them. As a consequence, r_1 and r_2 must have the same values for the searched congruent quadruplets $u_i = (q_1, q_2, q_3, q_4)$ taken from Q.

The algorithm now iterates through Q and considers all ordered pairs q_i, q_j of points from Q, and makes the hypothesis that they belong into one of the searched quadruplets u_i and that they form one of its two intersecting lines. If that is true there are two possible positions for the position of their intersection point:

$$i_{q,1} = q_i + r_1(q_j - q_i) \qquad \text{or} \qquad i_{q,2} = q_i + r_2(q_j - q_i) \tag{3.8}$$

[2] A quadruplet of points is called a "4-points" in the article.

When during the iterations two pairs of points $(q_i, q_j), (q'_i, q'_j)$ are such that $i_{q,1} = i'_{q,2}$, it is likely that $u_i = (q_i, q_j, q'_i, q'_j)$ is one of the congruent quadruplets.

This is a necessary, but not a sufficient condition: The values r_1 and r_2 are invariant for *affine* transformations, but for S and u_i to be congruent quadruplets they must be related by a *rigid* transformation. Therefore a filtering stage is added to remove the wrong quadruplets.

Choice of base

To choose a base S, the algorithm first randomly takes three points of P. They are always coplanar. Points with greater distances are preferred. Then the fourth point is chosen such that the quadruple is coplanar.

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.

Coarse registration algorithms typically do not look at the point cloud in detail, but rather at some larger scale descriptor of its shape. They are used to roughly align entire scans that have outliers, relatively small overlap and many complex features that would need more prior filtering to work with fine registration algorithms.

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.

In the context of robotics, or when mobile scanners are being used, an estimation for the scanner poses, and with it of the point cloud alignment is often recorded using odometric sensors.

3.4.2. Feature correspondence

Instead of manually specifying corresponding positions, the process can be automated by placing visual markers on the scene prior to scanning. These can be detected automatically and used to get a rough estimation of the scanner pose. [Matiukas & Miniotas, 2011] describes one procedure where two-dimensional visual markers are used.

It is also possible to use image analysis to find corresponding features on two photographs that have been registered to the scans, and register the point clouds based on those. [Tournas & Tsakiri, 2009] Here image feature detection can be used, in particular invariant (to rotation and scaling) descriptors such as SIFT or SURF. An extensive survey of feature detectors is given in [Tuytelaars & Mikolajczyk, 2007] and [Saxena & Singh, 2014].

It is also possible to identify and put into correspondence features such as lines [Lichtenstein, 2011], planes [Dold & Brenner, 2006] or NURBS curves [Koch, 2008].

3.4.3. Extended Gaussian Image

The extended gaussian image (EGI) is one descriptor of the entire point cloud that can be used to estimate a rough rotational alignment. Planar surfaces are identified in the point clouds P and Q and their normal vectors are recorded. [Dold, 2005] From this a kind of spherical histogram of the normal vectors is constructed. When P and Q represent the same object with enough overlap, these will have a similar shape, but be rotated differently.

A normal vector can be described using two spherical coordinates θ, ϕ. The EGI is defined as the projection of the normal vectors onto the unit sphere. [Horn, 1984] Information about the positions of the planes whose normal vectors were taken is lost.

Because of the topology of the sphere surface, there is some difficulty in representing the EGI digitally. Unlike a planar surface like a rectangular image, it cannot easily be divided into "pixel" cells of equal area. The surface should ideally be divided into cells in such a way that (1) all cells have approximatively the same area and shapes, (2) they must be small enough, and (3) rotations will make the cells (approximatively) coincide.

A division along longitude and latitude axis does not meet the first and third constraints. Platonic solids have equal area cells but are limited to 20 triangular faces. A good compromise is the so-called *geodesic division*.

The EGI is represented by storing for each of these cells the number of normal vectors that fall into it. To find an approximate rotational alignment, the EGIs of P and Q are normalized, and then the EGI of Q is rotated for all spherical angles θ, ϕ and a measure of how well the EGIs match is taken. This may for instance be the sum of squared or absolute differences of the values of coinciding cells, or their correlation.

An extension which is more stable to low overlap and which includes the estimation of a translation is described in [Makadia *et al.* , 2006].

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. In particular the focus is on the registration of long-range scans of the entire building with short-range, high resolution close-up scans of some features of it.

In this chapter, the ICP algorithm's sensitivity to resolution difference between the scans is analyzed. Then a new approach is developed which attempts to make use of the way points are arranged on scanned object surfaces, in order to improve the registration quality.

4.1. Evaluation of registration accuracy

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.

4.1.1. Known true transformation

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

Using $\hat{\mathbf{M}}_{\text{rel}}$, one can calculate for each point $q \in Q$ the *true* 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 point $p \in P$ that coincides with this q'. 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' is used:

$$e(\hat{\mathbf{M}}) = \frac{1}{n}\sum_{i=1}^{n} \|q_i - q_i'\| \tag{4.1}$$

This metric will be called the *true error*. It is similar to the ICP point-to-point (mean square error) 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.

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 give a useful value.

4.1.2. Unknown true transformation

When the true transformation $\mathbf{M}$ is unknown, no exact metric for the accuracy of the registration can be defined. It is not known for which input value any such error metric e should reach its minimum. Deciding whether the registration is accurate enough ultimately needs some heuristics, such as a human checking whether the point clouds visually appear well enough aligned for their intended purpose.

The point-to-point or other error metrics used by ICP depend on the estimated point correspondences, and attain local minima for values of $\mathbf{M}$ where the estimated correspondences are incorrect, but the average distance is low. Also the correspondence selection needs to be adjusted in function of the occlusions, different bounds and densities of P and Q. An error metric calculated from one-on-one point correspondences is necessarily limited by which points are available in P and Q. Techniques such as point-to-plane ICP and generalized ICP try to infer information about the local shape of the surface around a point.

Point-to-point ICP uses a mean squared error, which allows for least squares error minimization. When the goal is just to evaluate the accuracy of a supposed registration, the *mean absolute error* can be more useful. It is the average of the unsigned distances between each point q and its estimated corresponding point in p.

$$e(\hat{\mathbf{M}}) = \frac{1}{n} \sum_{i=1}^{n} \|q_i - p_i\| \tag{4.2}$$

This way the error metric approximates the one defined above for the true correspondences.

4.1.3. Visualization of error metric

The rigid transformation matrix $\mathbf{M}$ has 6 degrees of freedom: three axis of translation and three angles of rotation. Both the translation and the rotation cannot be represented in a continuous way using less than 3 real numbers. A natural way to map 6 real numbers to a rigid transformation is to use a fixed orthonormal coordinate system, and use three values as components for the translation vector, and three values as Euler angles for the rotation.

So an error metric $e(\hat{\mathbf{M}}_{\text{rel}})$ is a function from $\mathbb{R}^6$ to $\mathbb{R}$. For a good error metric it can be expected that $e(\mathbf{M}) \approx 0$, and the area around $\mathbf{M}$ is of interest to evaluate the registration accuracy. It cannot be visualized directly as a six-dimensional density plot, so a way to approximate it is to use two- or one-dimensional cross-sections instead.

An easy way to generate a one-dimensional cross section plot is to fix 5 components and record the values of $e(m)$ as one component varies. But this creates a bias towards the axis of the used coordinate system which is typically irrelevant to the point cloud. A more useful way is to take a random rigid transformation $\mathbf{M}'$, and gradually interpolate from $\mathbf{M}$ to $\mathbf{M}'$.

As shown in section 2.3.3 a linear interpolation between two rigid transformations can be constructed by linearly interpolating the translation vector components, and calculating the *slerp* for the rotational interpolation. Here an interpolation function $i(t, \mathbf{M}')$ is constructed based on this, with $t \in [-1, +1]$. It is such that $i(-1, \mathbf{M}') = \mathbf{M}'^{-1}$, $i(0, \mathbf{M}') = \mathbf{M}$ and $i(+1, \mathbf{M}') = \mathbf{M}'$.

$\mathbf{M}'$ is decomposed into the translational part and the rotational part, both are interpolated as described, and then again composed into a rigid transformation. The negative part needs to be handled separately because the slerp is only defined for $t \in [0, 1]$.

For choosing a random rigid transformation, a angular magnitude θ and translational magnitude m are first fixed. Then two random unit vectors $\vec{t}$ and $\vec{\omega}$ are chosen, according to a uniform probability distribution. The resulting rigid transformation is the composition of the translation $m\vec{t}$, and the rotation

around axis $\vec{\omega}$ by angle θ. Both m and θ can always be taken as positive values, since rotations in the other direction takes place when $\vec{\omega}$ points in the opposite direction. When $\theta = r\,m$ (in radiants), it will move on a sphere of radius r centered at the point cloud origin, by the same amount as it moves on the translation axis. So it makes sense to fix r in function of the size and shape of the model.

Now a visualization of an error metric is created by first choosing n random rigid transformations $\mathbf{M}_i$, and then plotting $e(i(t, \mathbf{M}_1)), \cdots, e(i(t, \mathbf{M}_n))$ against $t \in [-1, +1]$, on the same graph. A similar approach for visualizing error metrics is for instance used by [Bouaziz *et al.* , 2013]. An example of a graph generated this way is shown on figure 4.1. It shows the *mean absolute error* error metric measured on a point cloud, with correspondences established using the closest point criterion.

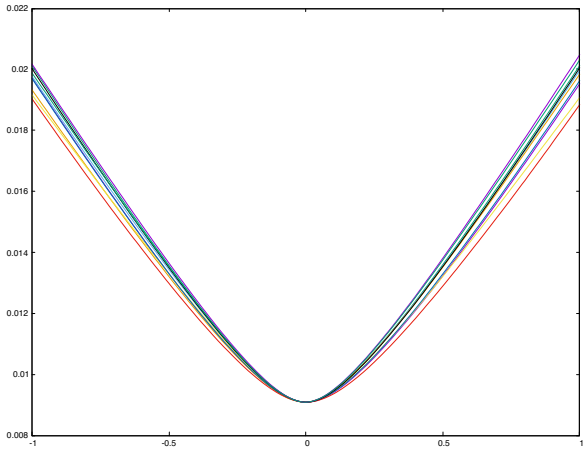


Figure 4.1.: Multi-axis visualization of mean absolute error metric

Because multiple interpolation axis (or planes) cannot easily be visually overlapped, two-dimensional plots of the error function are less useful, unless the two chosen axis have a special significance. For example if they span the plane of an approximatively planar model.

4.2. Models

This section describes the point clouds that will be used as input data to test the algorithms.

4.2.1. "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]. It is a medieval building built in the beginning of the 15th century, with a Brabantine Gothic-style architecture.

The data set consists of colorized range image point clouds, including both long range and short range scans. Figure 4.2 shows one of the long-range scans depicting the front facade. It has a complex shape, and contains stone sculptures and other artwork of which close range scans have been made. The figure shows a reprojection of a colorized point cloud, with the virtual camera position set so as to produce a frontal view of the facade. The scan was taken with the scanner placed on the ground, at the left side near the front gate. As a result, points at higher altitudes, such as on the tower, are less densely distributed. Figure 2.1 from chapter 2 also shows one of the long-range scans, taken from the roof of the building.

4.2.2. Dessus-de-porte

For usage in testing high-to-low resolution registration, scans from the "dessus-de-porte", the stone artwork located on top of the front gate of the building, are used. It can be seen from a distance on the long range scan 4.2. A much higher resolution, close range scan has also been taken of this part only, and is shown in figure 4.3.

As a first step of preprocessing, the same part has been cropped out of both scans, resulting in the two point clouds shown figure 4.4:

The colors have been removed on these depictions. The point clouds still have the color information, though it is not be used for registration here. The point clouds also are still *range images*, and information about the relative scanner pose is retained. It can be seen that the low resolution scan has a much lower point density. Its average distance between neighboring points, on flat surfaces facing the scanner, if about 10 times higher than that of the high resolution scan. The high resolution scan has been randomly downsampled as a side-effect of the visualization software. A close-up view of it is seen on figure 4.19, later in this chapter. It can be seen that different sides of the sculptures are occluded in the two scans, because they have been recorded from different view points.

These point clouds have been manually coarsely aligned, and the surface normal vectors have been computed using an external software. A goal will be to finely register them.

4.2.3. Relief point cloud

Many different kinds of objects can be scanned and good approaches for processing and registering them vary depend on many factors. To be able to study registration of point clouds, it is useful to generate artificial points clouds for which exact values of the underlying surface can be computed.

For this purpose an artificial point cloud class called "relief" point cloud is described, which has similar properties than the dessus-de-porte. An algorithm was devised to generate relief surfaces, which look as shown on figure 4.5.

[1] Brussels Town Hall

Figure 4.2.: Depiction of long range scan of Hôtel de Ville facade.

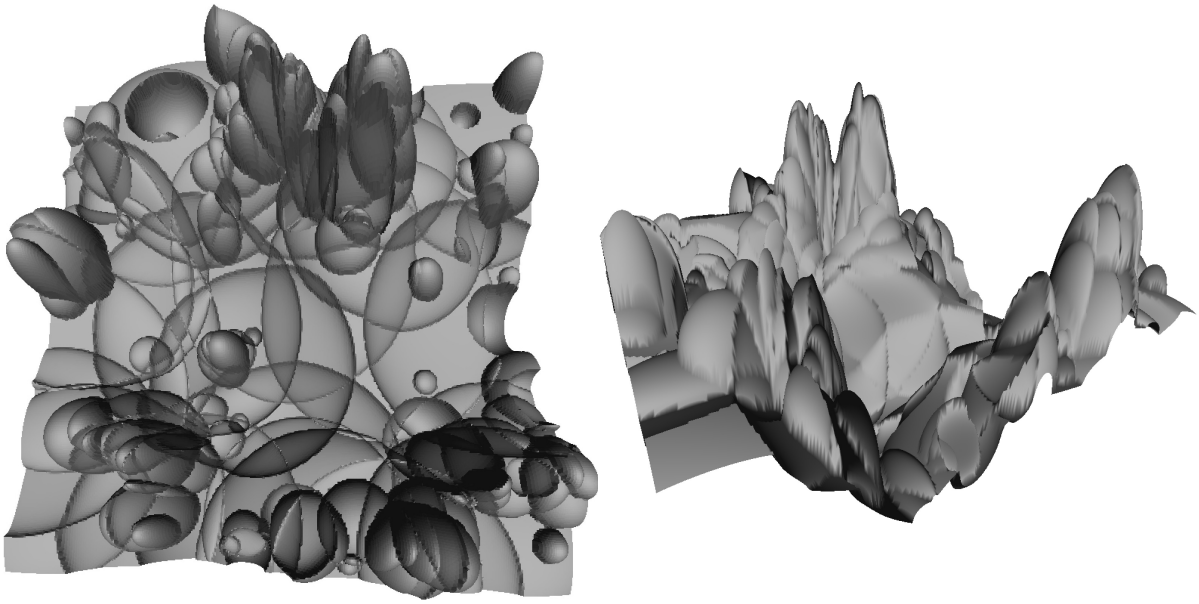
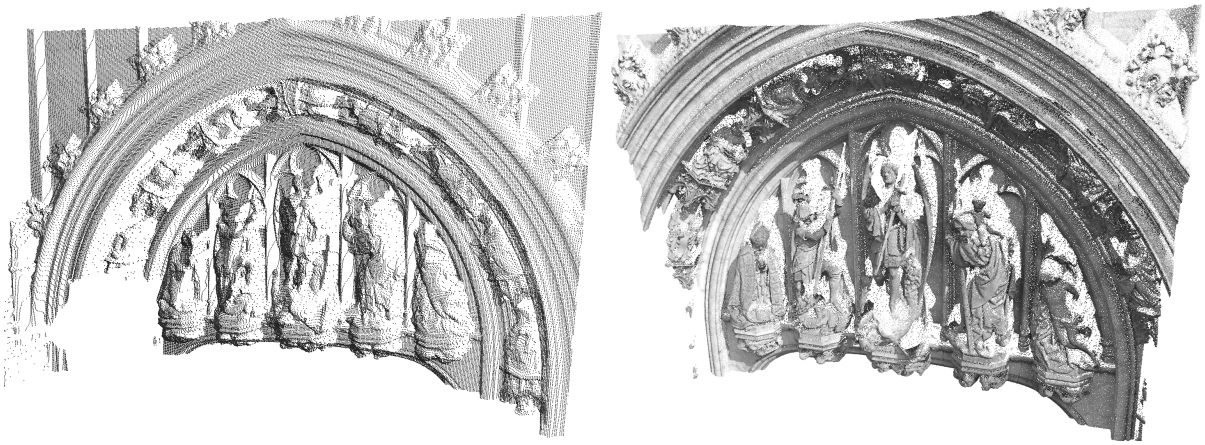


Figure 4.5.: Artificial relief surface, seen from two view points

Figure 4.3.: Depiction of short range scan of dessus-de-porte facade.



(a) Low-resolution (b) High-resolution

Figure 4.4.: Cropped scans of high and low resolution view of dessus-de-porte

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

$$B_i[x, y] = \pm\, h_i \sqrt{1 - \frac{(x_i - x)^2 + (y_i - y)^2}{r_i^2}} \tag{4.3}$$

A plot of its two-dimensional analogue is shown in figure 4.6. $B_i[x, y]$ 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 borders.

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

$$R[x, y] = \sum_{i=1} nB_i[x, y] \tag{4.4}$$

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.

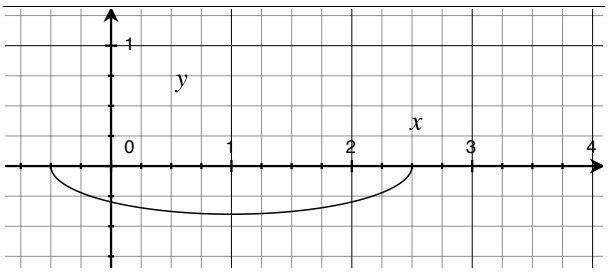


Figure 4.6.: 2D example for relief embossing, with $h_i = -0.4$, $r_i = 1.5$, and $x_i = 1.0$

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.

Top-down view point cloud

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. Figure 4.7 shows an example of such a point cloud, itself projected with a perspective camera.

Occluded view point cloud

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 project that point cloud into a range image as described in 2.2.3. 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

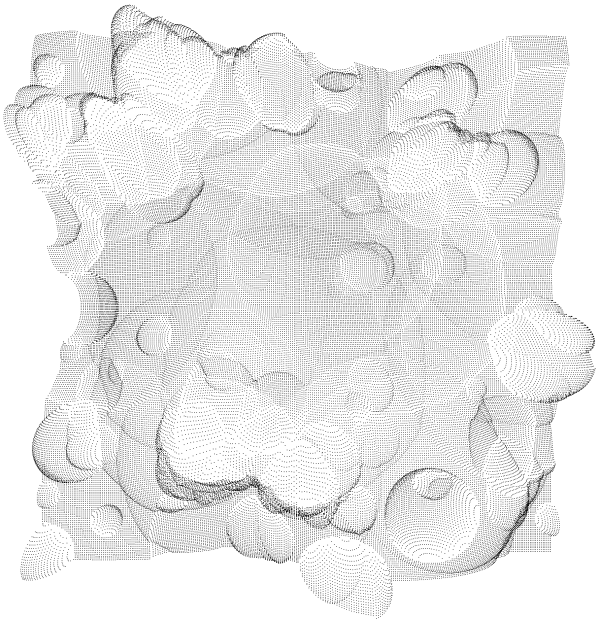


Figure 4.7.: Top-down point cloud of relief

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 4.8 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 4.5, 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. This image is a Z-buffer that contains, for each pixel, the inverted projected depth of the point[2].

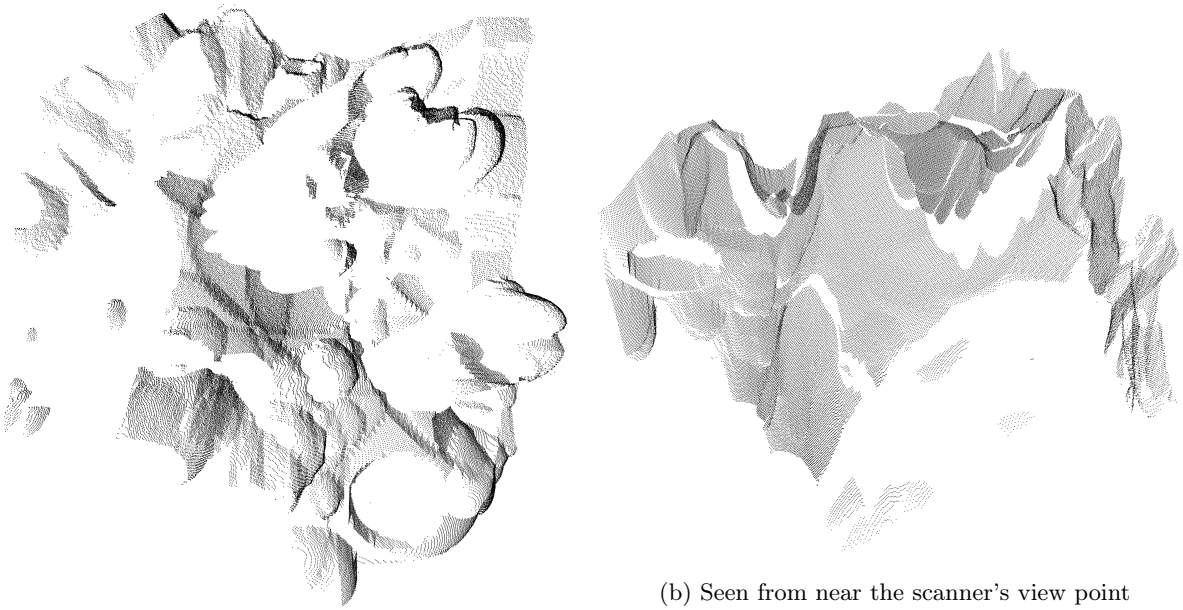
The width and height of this image space is set higher than that of the range image by a factor of about $k = 10$. Now each triangle is filled using a 2D rasterization algorithm. For each pixel, the inverted projected depths of the three corner points are linearly interpolated by using barycentric coordinates. 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. Its 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 $z' = R[x, y]$.

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



(a) Seen from above

(b) Seen from near the scanner's view point

Figure 4.8.: Artificial relief surface point cloud with occlusion

Similarity with dessus-de-porte

Figure 4.9 (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.

It is similar to the dessus-de-porte point cloud (called D) in these 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.
- Some near-planar surfaces, such as non-embossed areas in R, and the background stone surface in D.
- Points dispersed on a regular lattice, as a result of the scan-lines, or of the image space in the virtual projection camera.

The important difference is 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. Using R it can be tested both with and without knowledge of the surface.

4.2.4. Other models

For the tests of the ICP algorithm, the well-known *Stanford Bunny* model will also be used. This is a standard 3D test model used in computer graphics, available for free download along with other similar models. It depicts a ceramic figurine of a rabbit and was generated in 1994 at Stanford University.

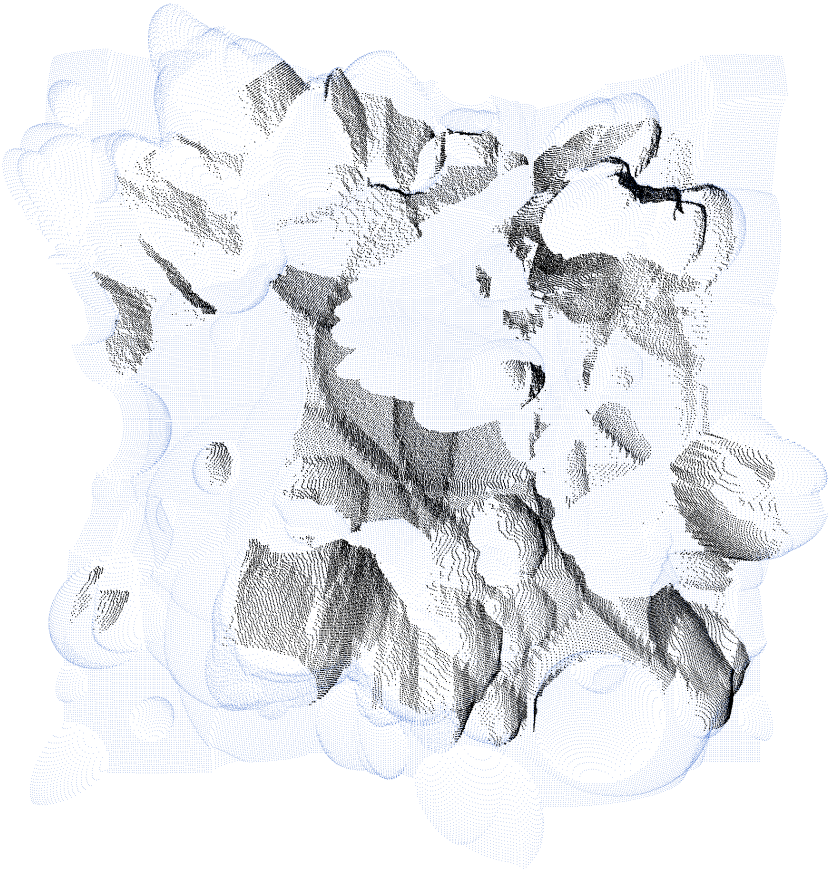


Figure 4.9.: R: Occluded view point cloud, and top-down point cloud of relief

Different versions of it exists, the one used is a point cloud that has already been registered from different view point scans, and covers the entire surface. It consists of 35947 points.

During this chapter, artificial point clouds consisting of a single plane, or of a sphere, are also used.

4.3. ICP registration experiments

Some experiments were run with the ICP algorithm with different artificial point clouds as input in order to determine how a accurately they can be registered. The true transformation $\mathbf{M}$ is known, but ICP only uses the estimated point-to-point error metric using the closest point criterion. After running ICP on the point clouds, it is tested how closely the final estimated transformation is to $\mathbf{M}$, that is, how low the *true error* metric of the estimated transformation is.

Because the aim is not to study the convergence evolution of ICP, but rather the accuracy of the final registration, in most of the test runs the point clouds will be perfectly aligned initially. After registration with ICP they tend to diverge from this true transformation by a small amount. Two cases are studied: (1) The two point clouds have different resolutions, and (2) They have different camera view poses resulting in low overlap.

For these experiments only the most basic variant of ICP has been used, with a point-to-point error metric and the closest point criterion. The software framework implemented to conduct the experiments will be described in chapter 5.

4.3.1. Different resolutions

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.

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 A.1.1 (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 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 *true error*, as defined in section 4.1.1. 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 2.5) 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 4.10 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 deviated towards the right from the correct alignment.

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 4.11 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.

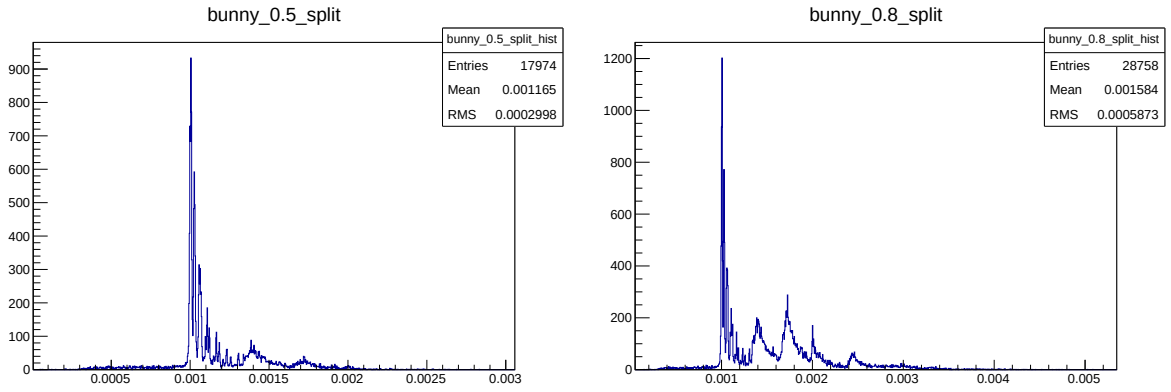


Figure 4.11.: Histograms of $\|q_i - p_i\|$ for 50% and 80% split

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 4.21. Figure 4.12 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 $\|p_i - q_i'\| = 0$.

Sphere experiment

The same experiment was run on artificially generated sphere point clouds, where the points are randomly dispersed on the sphere surface at a given density. Both the number of points on the fixed and on the loose point clouds are varied. Because of the random dispersion they never coincide.

In appendix A.1.3 the results are shown. The X-axis on the plot shows the *maximal* number of points, of the loose point clouds and of the fixed point cloud. Initially the point clouds are perfectly aligned, and they deviate from that alignment during icp registration.

It can be seen that the mean error, and its variance are slightly larger when fewer points are involved,

because the point dispersion is less dense on the surfaces, and so the nearest neighbor distances get greater. However, because the points here are dispersed randomly, there is already a high variance in the nearest neighbor distances, as will be shown later.

Conclusion

It can be concluded that the accuracy of registration that one can hope to attain with the ICP point-to-point error metric is limited by the density with which points are dispersed on the object surfaces. For this point cloud the final *true error* after ICP registration tends to be off in average by the side length of the square grid which the points form.

4.3.2. Different view points

Another experiment was run to test ICP's sensitivity to occlusion caused by different camera angles. A relief model as described in 4.2.3 is used. The fixed and loose point clouds are both generated by generating a projected range image of it, with a camera placed at a higher altitude, looking down on the relief, and placed at an angle θ relative to the Y axis. For both the loose and the fixed point cloud these angles θ_{fixed} and θ_{loose} are independently varied from 0 to 2π.

When $\theta_{\text{fixed}} = \theta_{\text{loose}}$ both point clouds feature the relief looked at from the same view point, thus there are no occluded parts from one point clouds relative to the other. As the difference gets greater the overlapping area gets smaller.

An example is shown on figures 4.13. On the first figure both point clouds are projected from the same view point. They are cropped to have different bounds, like point clouds from real scans would have. On the second figure the view point of the red point cloud is different.

Initially the fixed and loose point clouds are perfectly aligned, and then ICP is run. The final true error is recorded. The result is shown in appendix A.1.4. It can be seen that as the overlap gets smaller, ICP tends to converge towards an alignment that lays further off the true tranformation, even when the point clouds were perfectly aligned to start with.

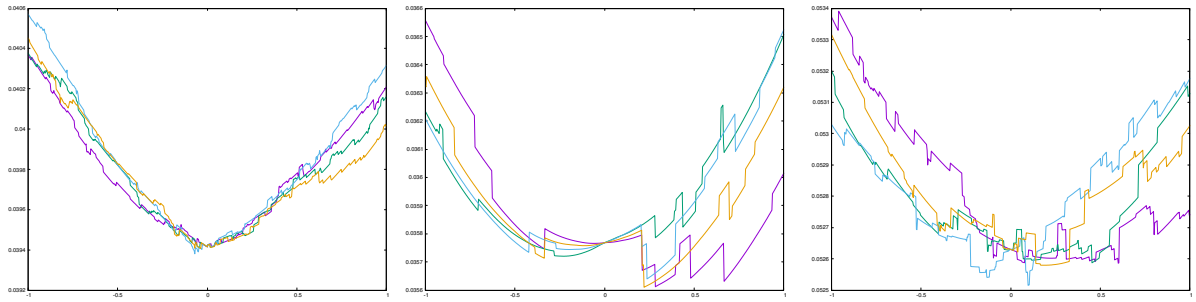
Thus the global minimum of the error metric minimized by point-to-point ICP diverges from the true transformation.

4.3.3. Instability of error metric

The experiment show that the point-to-point error metric used by ICP is unstable, in the sense that its accuracy deteriorates when one point cloud has lower resolution, and when they have low overlap due to different camera view poses.

Figure 4.14 is a visualization of the *mean absolute error* with closest-point correspondences, taken on a relief model. As explained before the curves show the value of $e(\hat{\mathbf{M}})$ for randomly chosen one-dimensional cross-sections of the rigid transformation space, such that $\hat{\mathbf{M}}$ is the true transformation $\mathbf{M}$ at $x = 0$. Therefore all the curves on the plots intersect at $x = 0$. An accurate error metric would have its global minimum at $x = 0$ and no local minima.

On these cross sections $\hat{\mathbf{M}}$ deviates from $\mathbf{M}$ by a maximum of 0.003 translation length and 0.3° rotation angle. The relief model has a width of 5 by comparison.



(a) Same (b) Different resolution (c) Different view-point

Figure 4.14.: Mean absolute error metric visualization with closest-point criterion

In the first plot the fixed and loose point cloud have approximately the same resolution and total overlap, whereas in the other two cases their resolution or view-point has been altered like in the experiments. It can be seen that in those two cases, $\mathbf{M}$ is no longer the global minimum, and so ICP converges to another transformation as it did in the experiment runs.

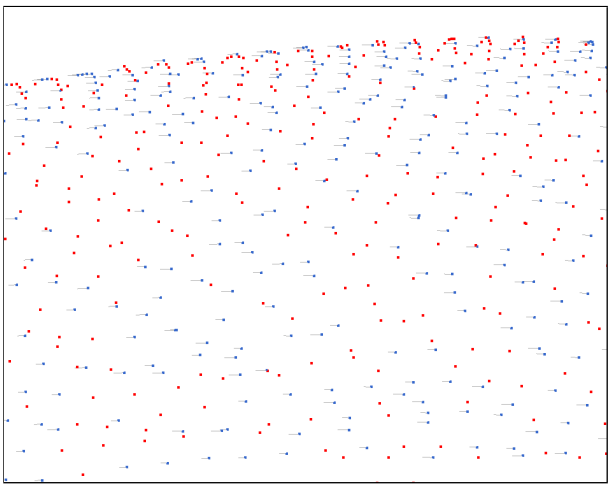


Figure 4.10.: Bunny model registered to itself, true correspondences shown

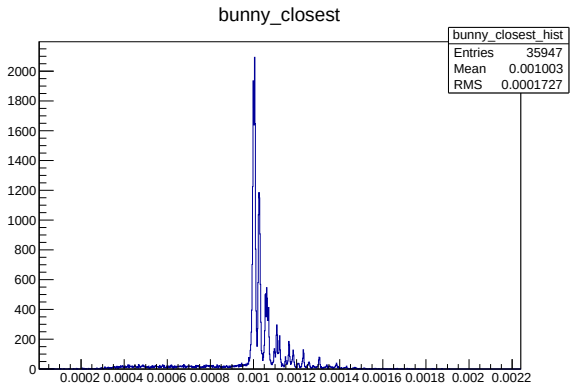
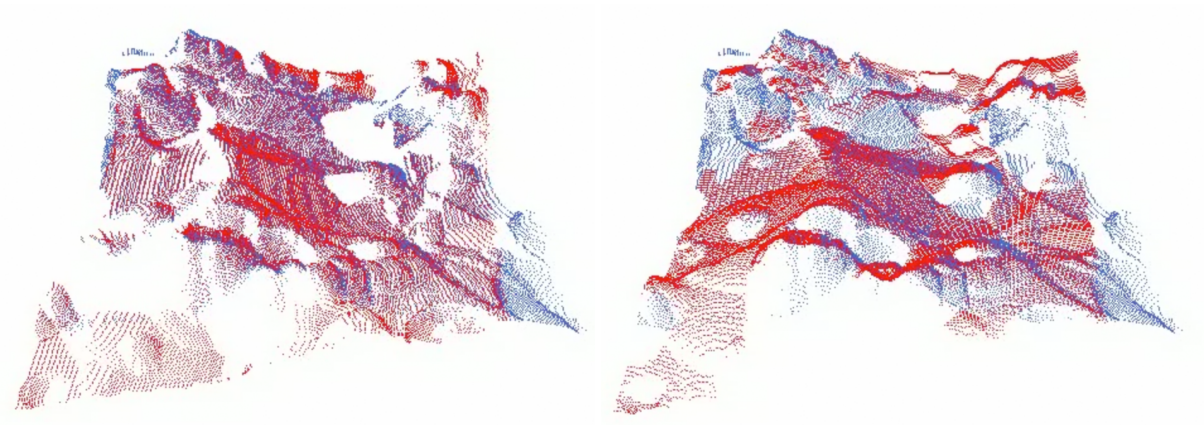


Figure 4.12.: Nearest neighbor distance histogram on Bunny point cloud



(a) Same view point (b) Different view points

Figure 4.13.: Aligned relief point clouds, projected from different angles.

4.4. Developing an error metric

Any error metric that does not use the true transformation is an *estimation*, and its global minimum generally does not exactly coincide with the one from the *true error*. But when the true transformation $\mathbf{M}$ is known, it can be directly compared with the *true error*. An approach to develop a good estimative error metric is the following:

1. First the scope is limited to point clouds having some *characteristics* that the error metric will make use of.
2. An artificial 3D model which also has these characteristics is defined, along with an algorithm to generate artificial point clouds from it. On these point clouds, the true transformation is known, and parameters such as scanner noise and point errors can be simulated and controlled.
3. An estimative error metric is developed measures the alignment of two different point clouds generated from the model, without taking into account the true transformation $\mathbf{M}$, and based on the *characteristics*. To evaluate its accuracy it is compared with the *true error* metric.
4. When the estimative error metric works well enough, it is now applied to a real scan. It the artificial point clouds model the real scan well enough, it can be expected to still yield good results. Its accuracy is compared to the *mean absolute error*, and evaluated manually by visual inspection.

In this chapter, an attempt will be made to define a fine registration error metric, based on the histogram formed by the distances of point pairs chosen using the closest point criterion. For this the characteristic dispersion of points on the surfaces will be taken into account.

4.4.1. High- to low-resolution registration

The set goal was to improve registration when one point cloud has a lower resolution than the other. Naturally the first step is to develop an error metric which indicates when the point clouds are accurately registered. Then the second step is to develop an algorithm to find a transformation that minimizes this error metric.

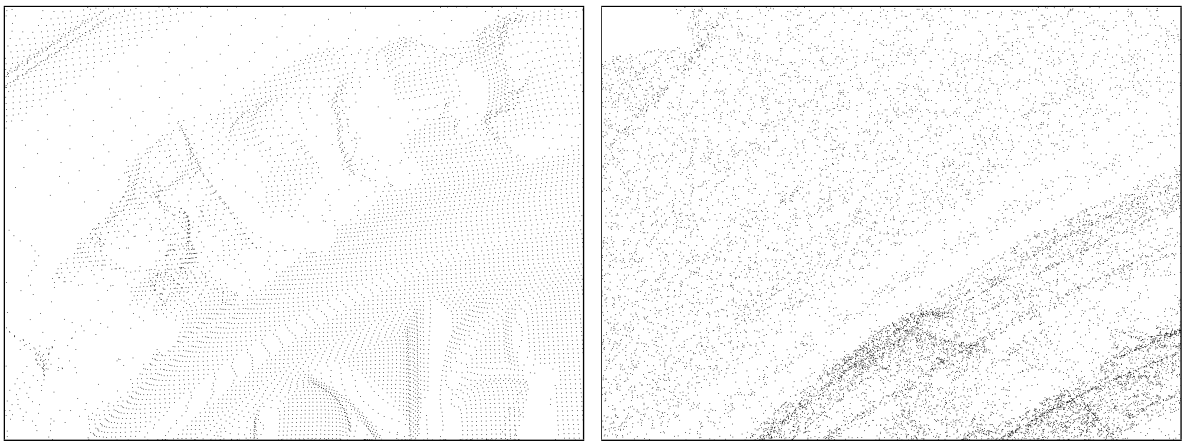
This can be considered to be an ill-posed problem when the true transformation $\mathbf{M}$ is not known: Informally, an error metric attempts to extract information about $\mathbf{M}$, only from the two input point clouds P and Q. When the number of points in P is reduced, there is less information available. Now the goal is to find an algorithm such that $\text{register}(P, Q) = \mathbf{M}$, based only on the hypothesis that $\mathbf{M}$ exists. P and Q are data whereas the hypothetical $\mathbf{M}$ is a physical quantity. In other words it is not well defined what output the algorithm should produce. Point clouds taken from real 3D scans are very unstable data sets to work with because, as indicated in chapter 2, both the modeling of the object as a continuous surface and its representation using a set of points are approximations and imply that a degree of uncertainty is associated with every point.

ICP with the closest-point criterion essentially relies on the expectation that for all the $q \in Q$, the divergences of a correspondence point p to the true corresponding point q' on the other surface P, will cancel each other out because to the large number of points q. If the number of points gets lower, this will no longer be the case.

Intuitively, in order to obtain a better error metric, more information about the surfaces needs to be extracted. Standard ICP's view of the point cloud P is limited to one corresponding point for each $q \in Q$. As shown in the previous chapter, many variants have been developed that take into account more information, such as the normal vectors, covariance matrices of the error, point attributes, etc.

Approach

Since the scope here is limited to range images, that have near-planar surfaces, the point dispersion pattern that the scanner's rays will produce on the surfaces can be considered to encode information about those surfaces. The following two point clouds are scans from the same object (dessus-de-porte).



(a) Points arranged on regular lattice

(b) Points dispersed randomly

The first one shows the points' arrangement as produced by the scanner, whereas in the second one they are randomly dispersed, at about the same density.[3] To a human looking at these two screenshots, is it immediately apparent that the first one contains more information about the object's geometry. To the point-to-point error metric, on the other hand, these two point clouds would "look" the same since it does not consider the positions of points in P relative to one another.

The points become arranged so that they form a parallelogram lattice[4] on planar surfaces of the object. When two point clouds P and Q of the same object, where Q one if of higher resolution, are perfectly aligned and superimposed, points $q \in Q$ will fall between adjacent lattice points, and inside the parallelograms that they form.

The algorithm will record all of the distances from q to the closest point $p \in P$ in a histogram. Only points p that lie on locally planar surfaces are considered. At the same time, using the normal vectors of the points p relative to the camera view ray directions, it is possible to predict properties of parallelogram lattice on which p lies. This includes the probability distribution of the distance from any position on the same plane to the closest point of the lattice. This probability density function will be compared to the recorded histogram.

If P and Q are well aligned they should form the same distribution. If not, the distances on the histogram will be diverge more because an additional displacement in the third dimension relative to the surface is added. An error metric is extracted from the comparison of the histogram.

Unlike the point-to-point error metric, this looks at the *distribution* of the distances and not just at the *average*. The expected distribution is based on the lattice formed by the points $p \in P$ relative to one another.

[3] It was generated from the higher resolution scan, followed by random downsampling

[4] will also be called a "parallelogram grid" in this paper

4.5. Analysis of point clouds

In this section the point clouds are looked at in more detail on a local per-point scale, including the dispersion pattern of points on the surfaces. Some local measures will be defined that assign to each point in the point cloud a value in relation with its surrounding points. A way to predict the dispersion of points will be developed.

4.5.1. Local density

The *local surface density* $\rho(p)$ of a point cloud P around a point $p \in P$ indicates how densely points are dispersed on the surface around p, expressed in number of points per surface area. Because the shape of the underlying surface is unknown, a precise measure cannot be defined, and instead an approximation is used.

Taking the k nearest neighbors around a point p that is in a region where the surface is approximately planar results in a set of points located approximately on a disk around p. In this case, an estimate for the density is $\rho(p) = \frac{k}{\pi\, r_{\max}^2}$, where $r_{\max}$ is the maximal distance of one of the neighbors to p.

Even when the surface is not locally planar in that region, some level of accuracy is retained because Euclidian distances in three-dimensional space are measured, which are approximatively equal to distances measured along the surface on which the disk would be wrapped.

Using $r_{\max}$ makes the measure more sensitive to outliers, and can overestimate the density: $r_{\max}$ by definition is the smallest least radius such that k points are inside the disk. An alternative is to use the median r_{med} of the radii, and set $\rho(p) = \frac{k}{2\,\pi\, r_{\text{med}}^2}$. For the median value, half of the k points have a smaller radius and are thus inside the disk.

When the density is supposed to be constant for each point in the point cloud, it will also be denoted as ρ or $\rho(P)$.

Square grid density

For artificially generated point clouds, per-point densities $\rho(p)$ are set to their theoretical values. For a planar surface where points are arranged on a square grid with side length l, this density is $\rho(p) = \frac{1}{l^2}$, because 1 point can be counted per square. This will be extended to parallelogram grids in the next section.

4.5.2. Local curvature

In the rest of this chapter, properties of the dispersion of points on planar surfaces of the model will be used. It is therefore important to distinguish between approximatively planar regions of the surface, and more sharp edges.

Unlike the density, this is a measure of the surface and not of the point dispersion on it. This implies that the metric should be invariant of the point dispersion. In addition to this, it is dependent on a scale parameter: for example a point cloud representing a wall of a building would be planar on a scale of a few centimeters, but not on a millimeter scale where the texture of the wall is considered.

A measure of *local curvature* $c(p, r)$ around a point $p \in P$ with a radius $r \in \mathbb{R}$ will be defined. A tangent plane is attached to the point p, with the same normal vector $\vec{n}$ as the point. This normal vector is assumed to have been calculated beforehand, for example by least-squares of RANSAC plane fitting. Then it is measured how well the neighboring points fit on the plane.

Using the local density $\rho(p)$, the expected number of points located in a radius r on the surface is $\rho(p)\,\pi\,r^2$. Using a kNN algorithm the $k = \lceil \rho(p)\,\pi\,r^2 \rceil$ nearest neighbors $N_k \subset P$ are searched. When k is below a predefined threshold it is increased to a minimal value. This is for example the case on oblique surfaces where the density is lower and the nearest neighbor distances get larger.

If the surface is locally planar around and p in the given radius, the points N_k will be located in a disk around p, with a radius of approximatively r. Because the circumference of a circle is proportional to its radius, N_k contains more points at higher radii, and the probability density of their distance to p increases linearly. But these points at a higher distance that fit on the plane should not overcompensate for nearer points that don't. Weights are attributed to the points to cancel out that effect: The weight of the closest point p_1 is set to 1, and those of the remaining points p_i to $\frac{\|p_1 - p\|}{\|p_i - p\|}$. Then the weights are normalized to sum up to 1.

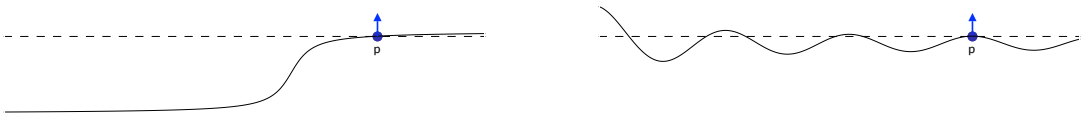
To measure how well a point $p_i \in N_k$ fits the plane, two values are useful: Its distance d_i to the plane, and the absolute angle $|\alpha_i|$ between its normal vector and that of the plane. Both can be calculated using the dot products:

$$d_i = \vec{n}\,(\vec{p_i} - \vec{p}) \qquad \text{and} \qquad \cos\alpha_i = \vec{n}\,\vec{n_i} \tag{4.5}$$

The local curvature metric is calculated as weighted average of those values for the k neighboring points:

$$c(p, r) = \frac{1}{k} \sum_{i=1}^{k} w_i\ (A\,|\alpha_i| + D\,d_i) \tag{4.6}$$

where the coefficients A and D are set so as to attribute different weights to the two measures. To avoid evaluating arccos for each point, the angle can be approximated using $\alpha_i' = \frac{\pi}{2}(1 - \vec{n}\,\vec{n_i})$. These two figures show two surfaces (in two dimensions) where one metric it high and the other low.



(a) Large $\sum d_i$, small $\sum |\alpha_i|$

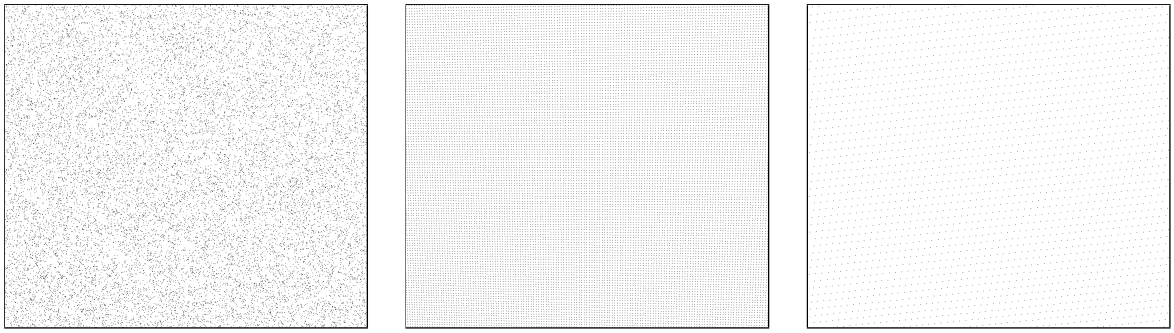
(b) Large $\sum |\alpha_i|$, small $\sum d_i$

For the purposes that the curvature measure is used here, A should be set higher, because surfaces such as the left-side can still be considered to be locally planar.

In figure 4.17 the "dessus-de-porte" scan is shown with points colorized according to this curvature measure. The color scale is shown on the figure. The blue and green parts indicate lower curvature, and thus more planar surface. It can be seen that the straight wall in the background is identified as being more planar, whilst rounded surfaces of the stone figures get a higher curvature value. Here $A = 10$ and $D = 3$, and $r = 0.01$ is used. The scale ranges from $c(p, r) = 0.000$ to 0.003.

4.5.3. Point dispersion

Point dispersion refers to how points in a point clouds are arranged on a surface. When the surface is locally planar, regions of the surface can be approximated by a plane. The following three point dispersions on the plane will be analyzed:



(a) Random dispersion (b) Square grid (c) Parallelogram grid

Figure 4.18.: Different point dispersions on planar surface

The *random dispersion* is the most general case, where the x and y coordinates of each points are independent, random variables, with a uniform distribution. It can be seen visually that the local density in that case is not constant.

Point clouds recorded by a laser scanner will produce a dispersion that is more akin to the *square grid dispersion*. If the scanner processes in sequential scan-lines, and uses a regular graduation of azimuth and elevation coordinates, a planar surface placed perpendicular to the scanner ray will *locally* get points approximatively arranged in squares. The dispersion will be considered to be invariant to a two-dimensional rotation of the plane itself.

4.5.4. Parallelogram grid

When the surface is placed at an oblique angle to the scanner line, the points on the plane will instead be dispersed on a *parallelogram grid*. Figures 4.19, 4.20 show two close-up views from the "Hôtel de Ville" scans, featuring the parallelogram grid point dispersion on approximatively planar surfaces. Figure 4.21 is taken from the Stanford Bunny point cloud, which was also recorded using a laser scanner. A square grid or rectangular grid are special case of this.

The diagram on figure 4.22 shows the parallel projection of a square from camera image space onto a plane in three-dimensional space. $\vec{n}$ is the normal vector of the plane, p_l the width and height (side length) of the square, and x, y the corresponding side lengths of the projected square. It can be seen that the projected square takes on the shape of a parallelogram on the plane. This models the projection of scanner rays on a surface. The coordinate system is such that the scanner is placed at origin. Since only a small region of a locally planar surface is considered (compared to the field of view of the scanner), adjacent rays in azimuth and elevation direction are modeled as parallel. When the square is extended to form a square grid in the XY-plane, a parallelogram grid is formed on the plane in which all parallelograms have the same two side lengths and inner angles.

It can be seen on figure 4.23 that several different parallelogram grids are possible for the same dispersion of points on the plane. The two grids in these examples have no sides in common. The projection of the camera square grid on the plane results in one of the possible parallel grids. When the plane is placed at a more oblique angle from the camera, it tends to be a grid with longer side lengths, such as the second on on the figure. It is not necessarily such that it includes the shortest possible parallelogram edge.

The "parallelogram grid" is described mathematically as a lattice in $\mathbb{R}^2$. The projections of $\vec{x} = (p_l, 0)^\intercal$ and $\vec{y} = (0, p_l)^\intercal$ from the camera image constitute one of the infinitely many possible bases for this lattice. [Galbraith, 2012]

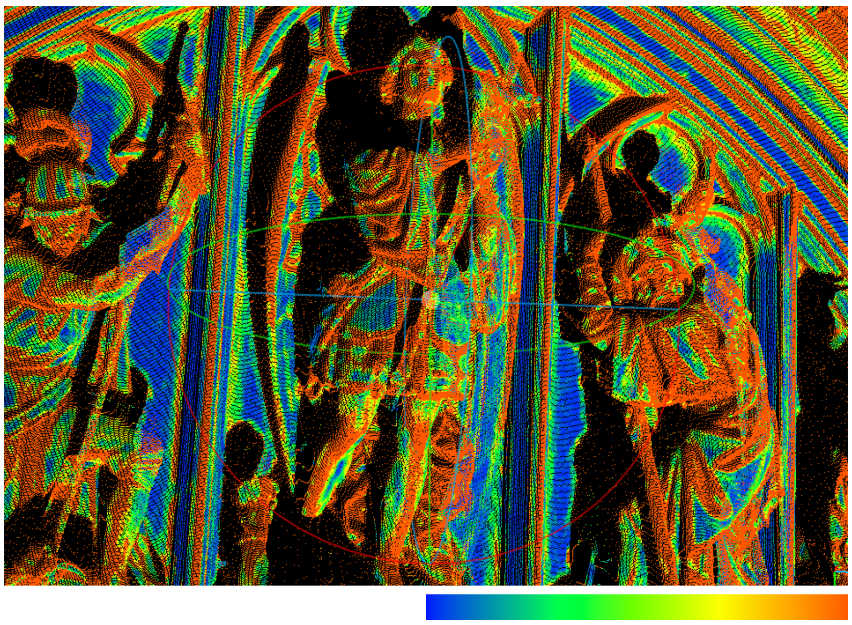


Figure 4.17.: Example point cloud colorized according to curvature measure

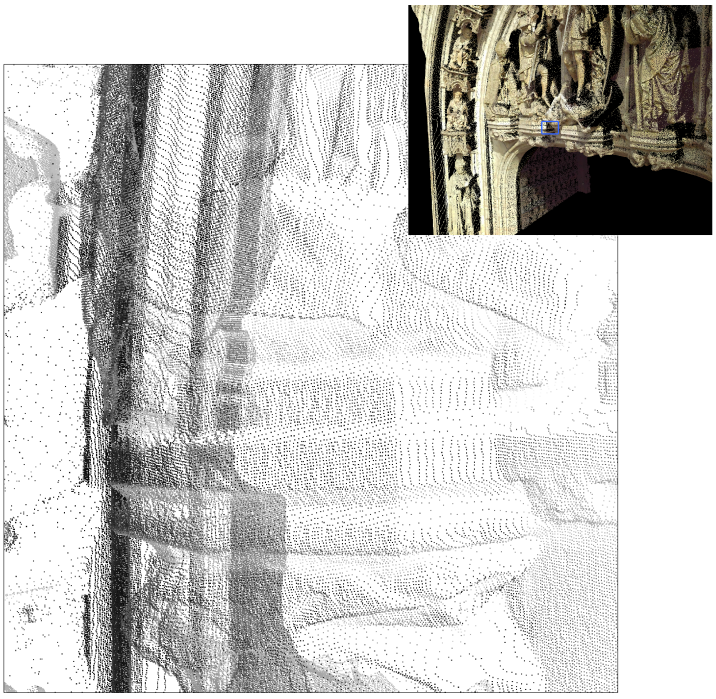


Figure 4.19.: Closeup of the surface distribution of points on front wall of building

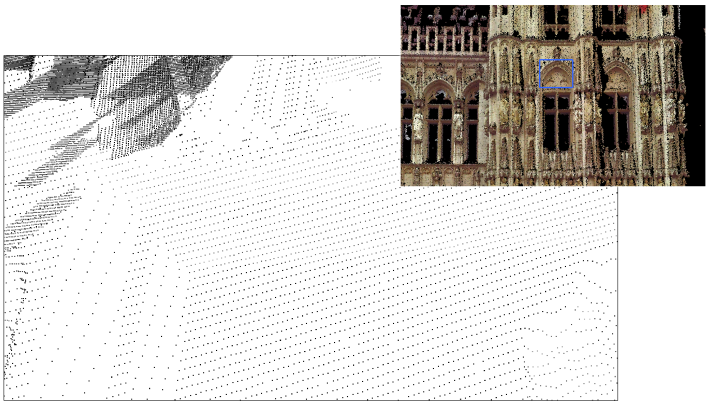


Figure 4.20.: Closeup of the surface distribution of points on detail of “dessus-de-porte”

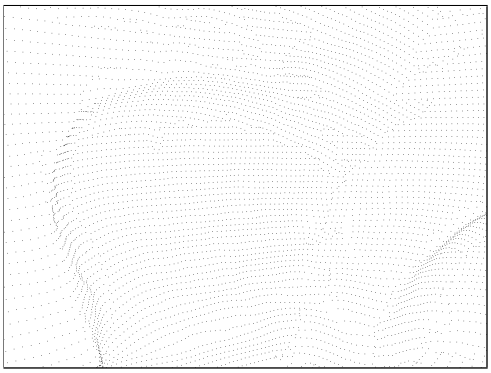


Figure 4.21.: Closeup of the surface distribution of points on Bunny point cloud

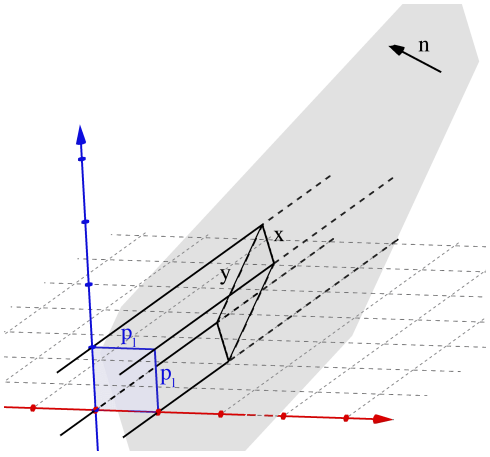


Figure 4.22.: Parallel projection of square onto plane in space

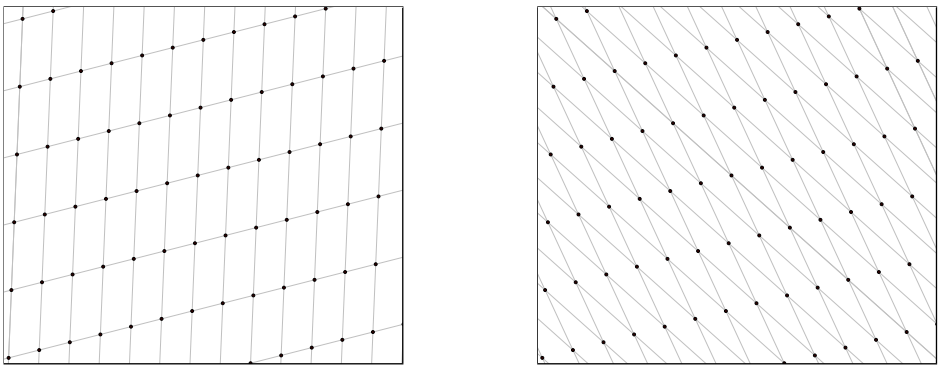


Figure 4.23.: Different parallelogram grids for same point dispersion

4.5.5. Measures on grid

The point density ρ of the points in the parallelogram grid dispersion on the object surface, can be calculated as

$$\rho = \frac{|n_z|}{p_l^2} \tag{4.7}$$

The shortest possible parallelogram side length $l_{\min}$, for any parallelogram grid put on the points, can be estimated with the following formula, in function of p_l and the normal vector $\vec{n}$ of a point $p \in P$. It is an estimate of the distance from p to its closest neighbor. The expression gives correct values when P is locally planar around p, and this plane is not too oblique.

$$l_{\min} \approx p_l \sqrt{1 + \frac{\min\left\{n_x^2,\, n_y^2,\, 1 + 2\,n_x\,n_y,\, 1 - 2\,n_x\,n_y\right\}}{n_z^2}} \tag{4.8}$$

A threshold condition for how oblique the surface may be is given by this formula

$$|n_z| > \cos\alpha \bigvee \left[\left|\frac{n_x}{n_y}\right| < \tan\beta \bigwedge \left|\frac{n_y}{n_x}\right| < \tan\beta \bigwedge \tan\left(\tfrac{\pi}{4} - \beta\right) < \left|\frac{n_x}{n_y}\right| < \tan\left(\tfrac{\pi}{4} + \beta\right)\right] \tag{4.9}$$

where the angles α and β are adjusted to some constant. The lower these angles, the more restrictive the condition gets. $\alpha = 50°$ and $\beta = 10°$ were used for the experiments.

To check whether P is locally planar around p, a curvature measure as described in 4.5.2 can be used. This is important since formula 4.8 depends only on the normal vector, but on curved surfaces, the

normal vectors gradually change direction. So on curved surfaces they reach any intermediate value, even when the nearest neighbor distances don't correspond to the predicted value from the formula.

The deduction and proofs for these formulea are given in appendix B.1.

Analysis

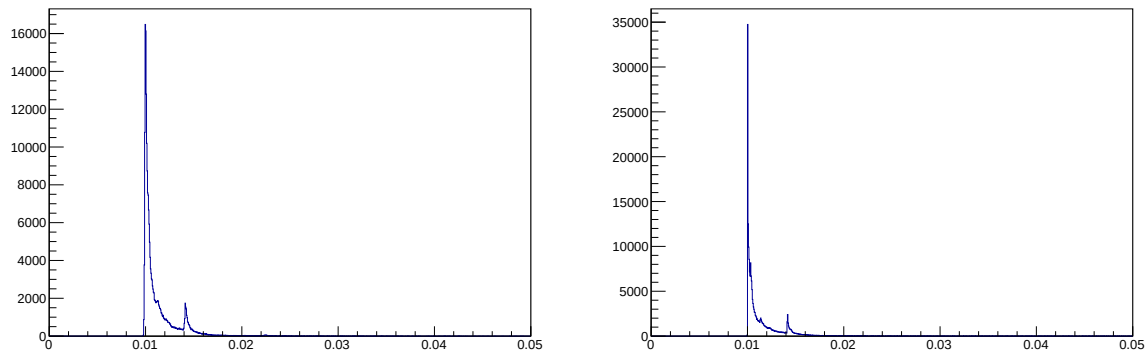
Figure 4.24 shows the value $l_{\min}$, in function of the normal vector $\vec{n}$ of the plane. For the color scale a logarithmic scale is used. Blue and green correspond to lower values. The x and y axis of the plot correspond to n_x and n_y, the third component is set to $n_z = \sqrt{1 - n_x^2 - n_y^2}$. The plot is symmetric around the X and Y axis. The origin point $(0, 0)$ corresponds to $\vec{n} = (0, 0, 1)^\intercal$, that is, the plane is perpendicular to the camera ray and has a square grid dispersion. Then $l_{\min} = 1$, the side length of the square. For the points on either axis, the plane is turned on one direction only, resulting in a rectangular grid, where the shortest side remains $l_{\min} = 1$. Around the border of the plot, the plane is nearly parallel to the camera rays. When $n_x \approx n_y$, the parallelograms take on a rhombus shape, similar to the second grid shown on figure 4.23. The shortest edge becomes the projection of the squares' diagonal with length $\sqrt{2}$, whereas the projection of its sides results in longer edges.

Verification

The first (left-side) histogram on figure 4.25 was generated by recording for each point $p \in P$ on a point cloud, the distance d to its closest neighbor. (That is, the point $p' \in P$ such that $p' \neq p$ and $\|p - p'\|$ is minimal.) The point cloud P used is a "relief" model as described before in section 4.2.3, projected without occlusion using a parallel projection camera with $p_l = 0.01$.

Two clusters form around 0.01 for parts of the surface that are near-parallel to the camera rays, and around 0.0115, where the surface is more oblique in both directions. Since the surface is not everywhere locally planar the histogram does not form sharp spikes.

For the second histogram the values $l_{\min}$ are instead recorded using the normal vectors of the points $p \in P$, and with fixed $p_l = 0.01$. This histogram is generated solely by evaluating the formula 4.8, without looking at the coordinates of the points, or doing a closest neighbor search in P.



(a) Closest neighbor distances on P

(b) Values $l_{\min}(\vec{n})$ on P

Figure 4.25.: Comparison of measured closest neighbor distances and $l_{\min}$ values on relief point cloud.

A similar distribution can be observed on both histograms. The spike at 0.01 occurs because the camera is placed such that the majority of the surface is approximatively perpendicular to the rays. This confirms the correctness of the formula for $l_{\min}$ for the parallelogram dispersion generated by parallel

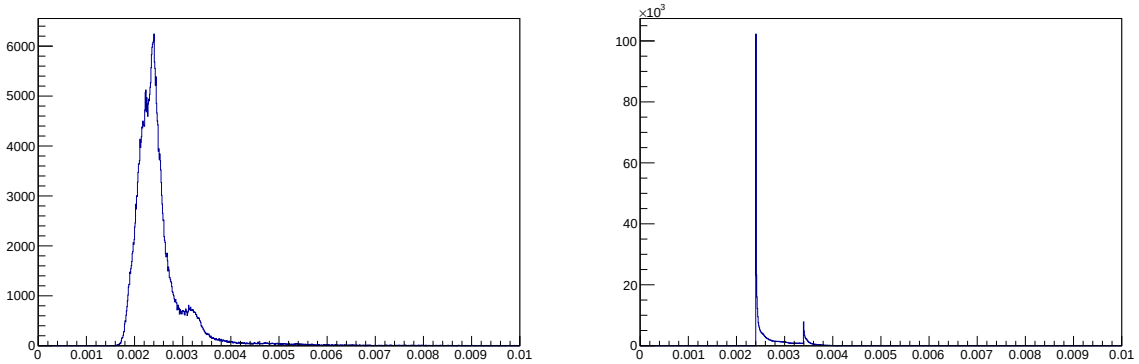
projection. The similarity of these two histograms also allows for estimation of p_l, even when it is not a mode.

Usage on real point cloud scan

Figure 4.26 shows the same two histograms, using the "dessus-de-porte" scan for P instead of an artificial relief point cloud. The points whose normal vectors fail the threshold condition 4.9 or whose curvature is too high are removed prior to taking the histogram.

Some similarity can still be seen, notably the two spikes near $d = 0.0024$ and $d = 0.0034$. Many factors cause the discrepancy between the two histograms:

- There is some error in the points' positions, which becomes significant in the order of magnitude of nearest neighbor distances. They do not lie exactly on the object surfaces, and do not form a perfect lattice on the surfaces.
- The scanner does not form exactly the parallelogram lattices as predicted.
- The scanner rays are not exactly parallel throughout the point cloud.
- The surface is nowhere completely planar.
- As explained, the formula 4.8 is only correct in a subset of the cases, even when the condition 4.9 is applied.
- The normal vectors are not completely accurate.



(a) Closest neighbor distances on P

(b) Values $l_{\min}(\vec{n})$ on P

Figure 4.26.: Comparison of measured closest neighbor distances and $l_{\min}$ values on "dessus-de-porte" scan.

4.5.6. Estimation of projection parameters

If the point cloud P was generated using a parallel projection camera where the graduation in the x and y directions on the image planes is the same, a value p_l exists for the point cloud. This remains true by approximation for small extracts of large 3D scans. In the following the assumption is made that a constant value p_l exists for the entire point cloud.

As shown before, using the properties of the parallelogram grid dispersion and formula 4.8, p_l can be estimated by analyzing the points of the point clouds and their normal vectors. The formula is such that

$l_{\min}$ is proportional to p_l, and the proportionality constant $m(\vec{n}) = \min\{\cdots\}$ is a function of $\vec{n}$. In this formula the coordinate system is such that the camera is at origin, so that a normal vector $\vec{n} = (0, 0, 1)^\intercal$ points towards it. When P is set in a different coordinate system but the camera pose in it is known, the necessary transformation must be applied. As stated in section 2.1.2 it is supposed to be the case for range images point clouds.

Assuming that the parallelogram grid covers the entire surface and that the surface is locally planar, $l_{\min}(p)$ can be computed at each point by measuring the distance to the closest neighbor in the point cloud.

A way to estimate p_l is to record $\frac{l_{\min}(p)}{m(\vec{n})}$ for each point and take the median value. The histogram is such that there is one large spike at the correct value p_l, and some outliers far off. For point clouds with a high number of points, this estimation remains stable when samples are recorded only for a random subset of the points.

4.6. Own distance histogram

Let P and Q be two perfectly aligned point clouds with identical surfaces, but with different point dispersions. They can differ as described in 3.2. P is called the *model*, and the points $q \in Q$ the *samples*. For each sample $q \in Q$, the closest point $p \in P$ is chosen, and the distance $\|p - q\|$ is recorded. The histogram of these distances will be called the own distance histogram (OH). The points p are chosen by the closest point criterion, without additional constraints at first.

In this section the shape of this histogram will be analyzed, and in the next section it will be compared to the cross distance histogram (CH), for which P and Q are no longer identical or perfectly aligned.

By replacing the samples $q \in Q$ with a random variable $\mathbf{Q}$ with a given probability distribution, the OH can be idealized into a probability density function of the closest point distance, free of noise resulting from the sparse set of samples.

In the following, Q will be taken to have a much higher point density than P. When the number of samples becomes infinite, the histogram converges towards the ideal probability density function. The results will later be applied to devise a method for registering high resolution with low resolution point clouds.

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.

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

4.6.1. Plane with random dispersion

Figure 4.27 shows an OH, where P and Q represent a single plane, and the points $p \in P$ are randomly dispersed onto it with an uniform density $\rho(P) = 30000$. For each sample q the distance $d(q, P)$ to the closest point in P is measured. Randomly dispersed means that the x and the y coordinate of each point are chosen randomly and independently, with a uniform distribution in a fixed interval.

Since P and Q are perfectly aligned, in this situation all distances are measured on the two-dimensional plane.

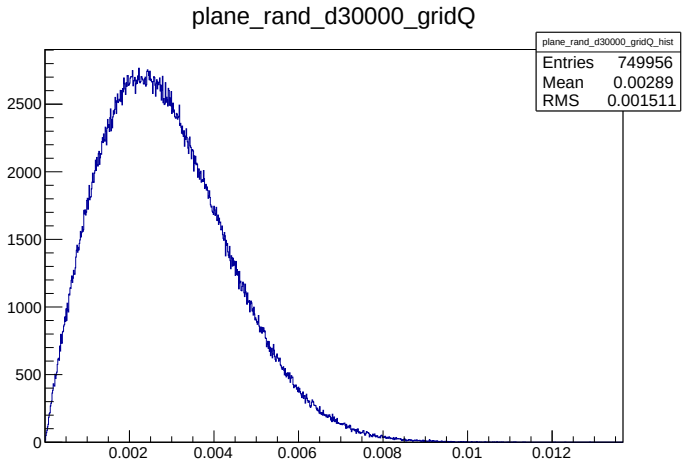


Figure 4.27.: Own distance histogram for plane P with random point distribution

Unless two points coincide, the probability at $d = 0$ is zero. It then increases, because the probability

that a disk with radius d around a sample point q contains at least one $p \in P$ gets larger proportionally to the disk's area. But that disk must increase in radius to contain more than one point, otherwise the closest distance is the distance to the first, closer point inside the disk, and not its radius. This intuitively explains the global shape of the histogram.

The underlying probability density function f_R of the random closest point distance R from any point q is the exponential function

$$f_R(r) = 2\pi\rho(P)\, r\, e^{-\pi\rho(P)\, r^2} \tag{4.10}$$

This formula is proven in appendix B.2. A plot is shown in figure 4.28, for $\rho(P) = 30000$. It can be seen that the probability density function takes on the same shape as the histogram. By solving $\frac{\mathrm{d}}{\mathrm{d}r} f_R(r) = 0$, one obtains that the probability density function reaches its maximum at

$$r_{\text{mode}} = \frac{1}{\sqrt{2\pi}\sqrt{\rho(P)}} \tag{4.11}$$

The mean $\bar{r}$ value for the closest point distance is obtained using the integral

$$\bar{r} = \int_0^\infty r\, f_R(r)\, \mathrm{d}r = \frac{1}{2\sqrt{\rho(P)}} \tag{4.12}$$

This random points dispersion represents the most general case, where no information about the dispersion of sample points on the surfaces is known.

4.6.2. Dispersion of sample points

In order to obtain a histogram that closely resembles the theoretical probability density function, the dispersion of the sample points $q \in Q$ should be such that its density is much higher than that of the model points, and has a low variance. If the density is not high enough, the range of possible placements of sample points relative to surrounding model points is sampled too sparsely, leading to a low resolution histogram. If the variance is too high, some placements will be oversampled in comparison to others, and the histogram will contain more noise.

Figure 4.29 shows the same histogram, but with the sample points $q_i \in Q$ also being randomly dispersed on the plane (and a different number of samples). It is shown in appendix B.3 that the local density is not constant.

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 4.28.

When working with artificially generated, perfectly aligned point clouds with a square grid point dispersion, the side lengths l_P and l_Q for the model and the sample point clouds must be chosen such that $\frac{l_P}{l_Q} \notin \mathbb{Q}$, otherwise same placements of sample points relative to model points will repeat, and a larger number of sample points doesn't increase histogram quality. For example if $l_P = 0.1$ and $l_Q = 0.02$, each square of the model will have 25 sample points placed within it at the same relative positions. Taking into account that floating point numbers with limited precision are used, it means that when considering l_P and l_Q to be rational numbers, they should be chosen such that their least common multiple should is as high as possible.

4.6.3. Plane with square grid dispersion

Figure 4.30 shows the distance histogram of two perfectly aligned planar surfaces P and Q, where the points on P are dispersed on a square grid. $\rho(P) = 20000$ and 30000, respectively. The number of

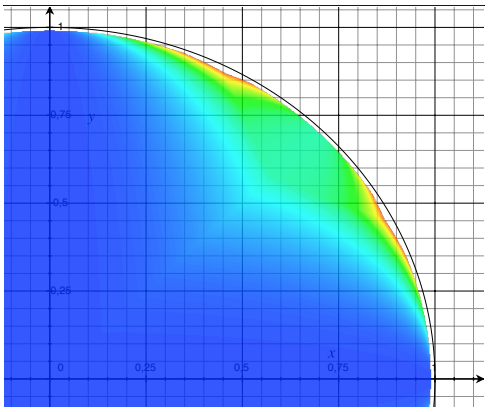


Figure 4.24.: Plot of $l_{\min}$ for $\vec{n} = (x, y, \sqrt{1 - x^2 - y^2})^\intercal$ and $p_l = 1$

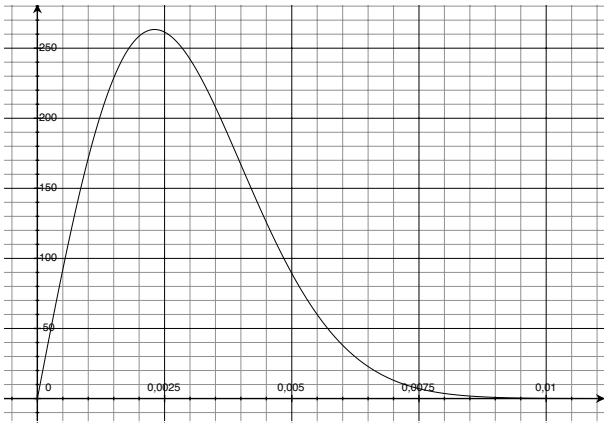


Figure 4.28.: Probability density function of closest point distance, for plane P with randomly dispersed points

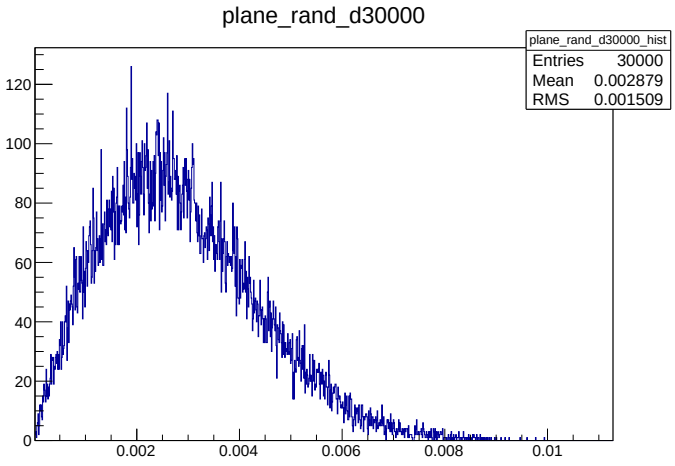


Figure 4.29.: Same as figure 4.27 but with randomly dispersed samples $q \in Q$

samples is $N(Q) = 300000$.

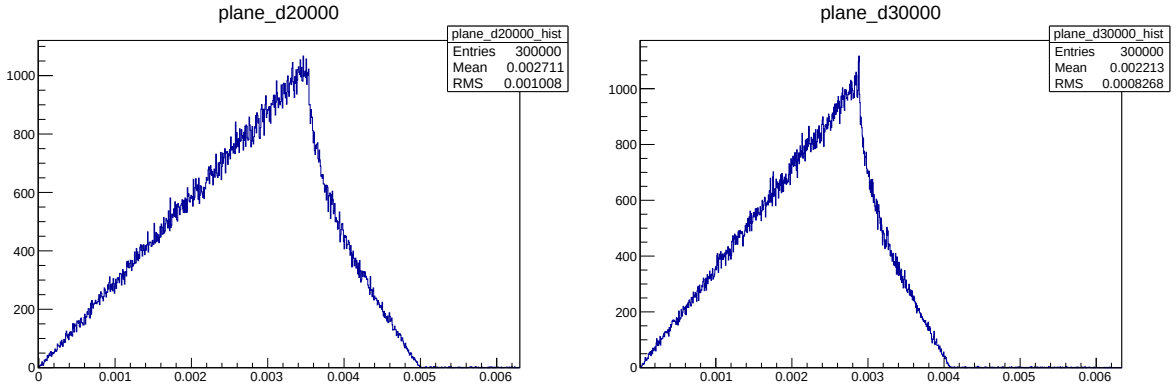


Figure 4.30.: Own distance histogram for plane P with square grid point dispersion

The probability density function f_R is

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

with $l = \frac{1}{\sqrt{\rho(P)}}$. This is proven in appendix B.4. The plot is shown in figure 4.31 for $l = 2$.

The probability rises linearily from O to to its mode at $r = \frac{l}{2}$. Within that range, the sample point falls within the non-overlapping disks of radius $\frac{l}{2}$ around the model points.

4.6.4. Plane with parallelogram grid dispersion

The square grid dispersion in a special case of the parallelogram grid dispersion. Figure 4.32 shows an example of an own closest point histogram obtained when the model point cloud has a parallelogram grid point dispersion. As seen on figure 4.22, it is the result of a parallel projection of a square grid on the camera image frame onto a plane in space with a normal vector $\vec{n}$, relative to the camera's coordinate system. For this example, $\vec{n} \approx (-0.253023, 0.787174, -0.562438)^\intercal$, and the square grid on the image plane has a side length of $p_l = 0.067$. s Figure 4.33 is a close-up view of the plane, showing the parallelogram grid of P, and the higher density square grid sample point cloud Q in blue.

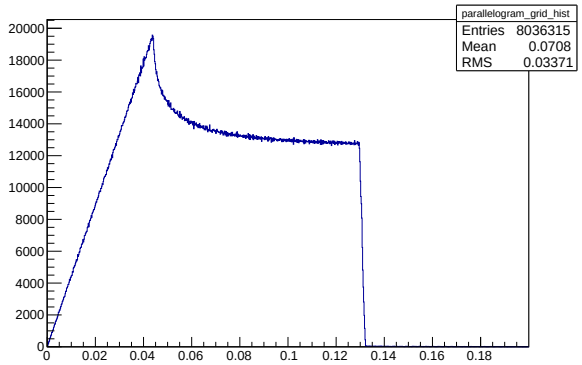


Figure 4.32.: for plane P with parallelogram grid point dispersion

It can be seen that the histogram's underlying probability density function is again a piecewise function, and that its first segment is still a linear increase from the zero point to the mode of the histogram.

Using a similar argument as for the square grid dispersion, one can show that the linear increase happens from 0 to $\frac{1}{2}l_{\min}$. Applying the formula 4.8 developed in the previous section, a value of approximatively 0.0441 is found for this example. It corresponds to the mode seen on the histogram.

This segment of linear increase will be called the region of interest (ROI) of the histogram. It will be used to construct an error metric.

4.6.5. Adjusted own distance histogram

It is possible to calculate the range of the region of interest (ROI) of the histogram, using only the plane's normal vector $\vec{n}$ and the side length p_l of the parallel projection camera. Here an attempt is made to use this result to produce an adjusted own distance histogram (aOH), which still has the initial linear increase, for point clouds that are not planes.

When the surface is no longer a plane, $\vec{n}$ will no longer be constant, but can have a different value for each point. However for smooth surfaces on the model, the normal vector will remain approximatively constant on local regions of neighboring points, as can be seen on the close-up views in the previous sections. On these planar regions the parallelogram grid point dispersion appears.

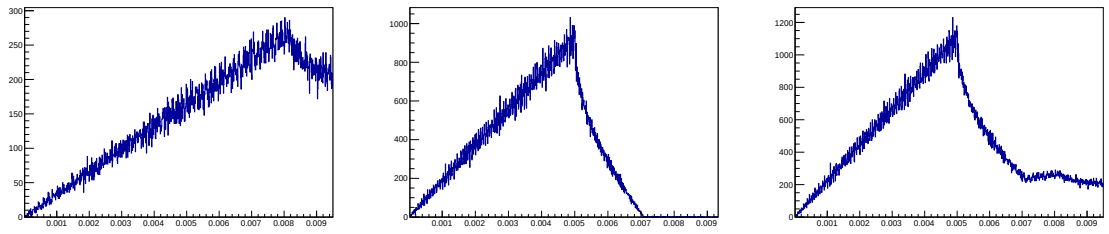
Under the assumption that most of the point clouds consists of such planar regions, its own distance histogram will be a superposition of the kinds of parallelogram grid histograms seen before, with an initial linear increasing segment.

Multi-planes point cloud

In order to test the shape of this superposition histogram, an artificially generated "multi-planes" point cloud is used. It consists of multiple planes places randomly in space at different orientations, projected using a parallel projection camera. An example with two planes is shown in figure 4.35, shown with decreased point density.

The two planes are bounded to the shapes of disks. This does not have an effect of the histograms, and just makes the point cloud easier to visualize. If the bound was a square, it would change depending on the rotation of the plane on its own axis.

Both planes have the parallelogram grid point dispersion. The example is chosen so that one plane is approximatively facing the camera while the other is more oblique, and has a lower ρ and higher $l_{\min}$ as a result. Figure 4.34 shows the own distance histograms for the two individual planes, and the superposition histogram for the entire multi-planes point cloud.



(a) Oblique plane (b) Parallel plane (c) Superposition

Figure 4.34.: Own distance histogram for multi-plane point cloud P (individual planes, and superposition)

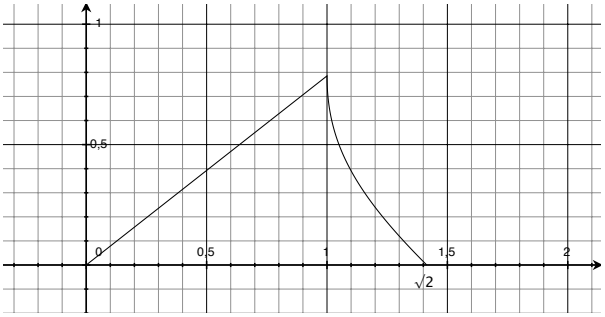


Figure 4.31.: Probability density function of closest point distance, for plane P with square grid point arrangement

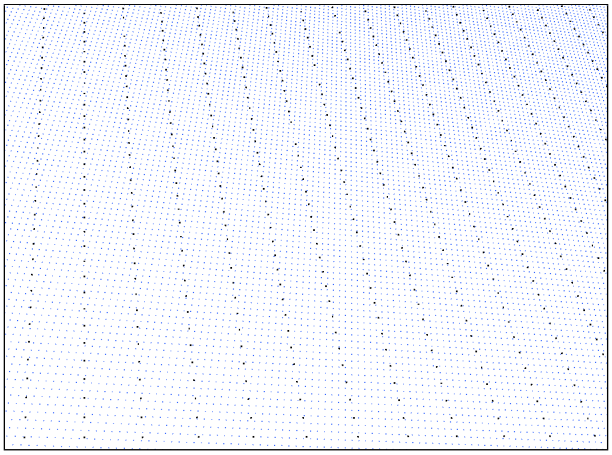


Figure 4.33.: Close-up view of parallelogram grid in P (black), and sample point cloud Q (blue)

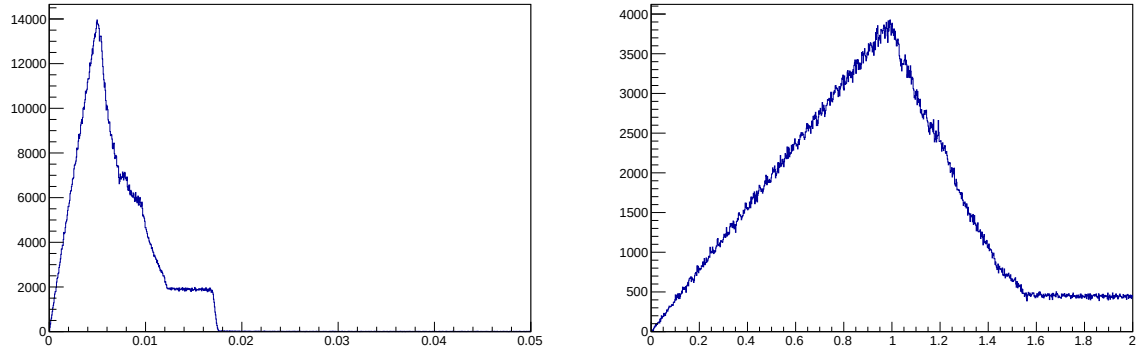
It can be seen that the linear ROI of the superposition histogram ranges up to about 0.005, which is the minimum of the ranges on the two individual histograms. Thus the information from the remaining linear range of the oblique plane is lost.

Adjusted histogram

The adjusted own distance histogram (aOH) is created as follows: For each $q \in Q$, the closest point $p \in P$ is taken, for which $d = \|p - q\|$ is minimal. Then instead of d, the value $d' = \frac{2\,d}{l_{\min}(\vec{n})}$ is recorded in the histogram. $l_{\min}$ is calculated from the normal vector $\vec{n}$ of p using the approximation formula 4.8. Samples are only added to the histogram when the threshold condition 4.9 is met.

All the histograms created from subsets of the point cloud containing only regions of the surface with an approximatively constant normal vector $\vec{n}$, would have a linear increase until $d = \frac{1}{2}\,l_{\min}(\vec{n})$. The adjusted own distance histogram is a superposition of these, distorted such that its ROI ranges from $d' = 0$ to $d' = 1$.

The following figure 4.36 shows this aodh, in comparison to the non-adjusted OH, for the same point cloud P. P is a multi-planes as before, but with more planes. Figure 4.37 shows the same histograms where P and Q is a relief point cloud. It appears more irregular than the OH, but the ROI contains a higher number of samples.



(a) Non-adjusted histogram

(b) Adjusted histogram

Figure 4.36.: OH, and aOH of P

4.6.6. Occlusion and different bounds

Point clouds obtained from real scans will not cover the entire surface due to occlusion, and any two point clouds P and Q will have different bounds. Thus there can be sample points q on surface positions that are not covered by P, and vice versa.

It is assumed that ideally the parallelogram grid on P is the same everywhere, and that the point dispersion on locally planar surface parts depends only on the local normal vector. Therefore when part of the sample point cloud Q are removed, it only decreases the quality of the histogram as there are less samples, but does not alter its shape.

However when parts of the model point cloud P are removed, for sample point $q \in Q$ that fall within these regions, the closest point $p \in P$ will be much farther away. As a result much greater distances are recorded, than for the samples falling within the parallelogram grid of P.

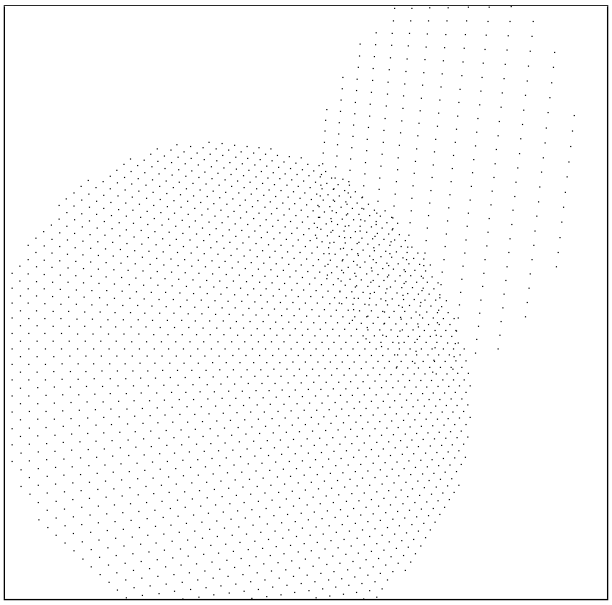
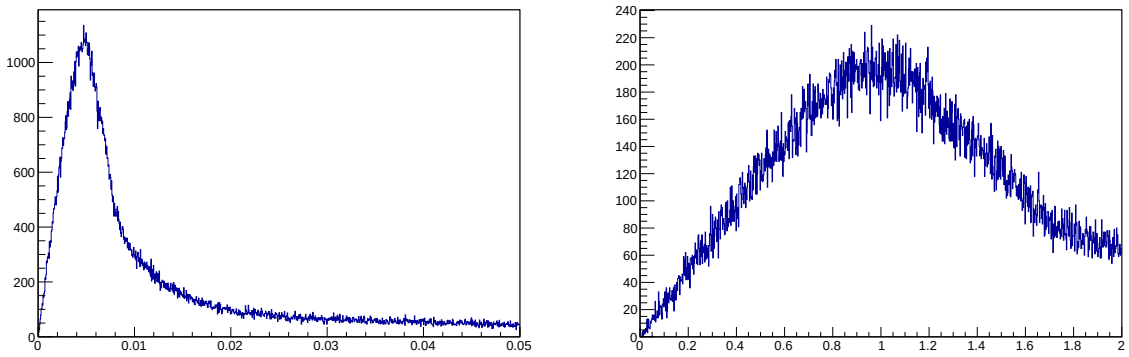


Figure 4.35.: Example of multi-plane point cloud



(a) Non-adjusted histogram

(b) Adjusted histogram

Figure 4.37.: OH, and aOH of relief point cloud

These distance values add more samples on the right side of the histogram, and do not affect the shape of the ROI of the histogram.

In the histograms on figure 4.37, P is a sideways view-point of the relief model, and it can be seen that the OH contains samples at greater distances, at a low but near constant rate. When zoomed out, this extends to a much longer range.

4.6.7. Sample for real point cloud

When P is not an artificially generated point cloud but is taken from a 3D scan, the sample Q cannot easily be constructed as the surfaces are unknown.

Using the scan as sample and a dowmsampled version of it as model does is not possible, because downsampling removes the properties of the point dispersion. The histogram of the nearest neighbor distances, that is, the distance from each point in P to the closest point in P except for itself, does not produce an OH.

One approach to construct a higher resolution sample Q from the point scan P is the following:

1. P is supposed to be a range image. Its camera position is at the origin of its coordinate system. As shown in section 4.5.6, p_l and $l_{\min}$ are be estimated from the point cloud.

2. All points from P which no not satisfy the threshold condition for obliqueness 4.9, and the points for which the local surface curvature is too large, are removed.

3. Q is initialized to an empty point cloud.

4. For the aOH, points with $d > \frac{1}{2} l_{\min}$ will fall outside its ROI, so Q does not need to include those.

   A copy P' of P is constructed, and each of its points p is randomly displaced along its tangent plane. The tangent plane is calculated using each point's normal vector. The random displacement along the polar coordinates r and θ on this plane, centered on the point p, is done according to an uniform distribution with $r \in [0, \frac{k}{2} l_{\min}]$ and $\theta \in [0, 2\pi]$, where $k \geq 1$ (and not smaller than 1). It can be set to a greater value to compensate for when the scanner rays are actually not parallel.

5. P' is fused into Q. Then step 4 is repeated, for a total of N times.

The resulting sample point cloud Q now has N times the number of points than P, its still lie on the object surfaces (assuming P is locally planar in the direct neighborhood of its points), and they are uniformly distributed around the points.

Results

The aOH obtained from the point cloud as P, and a sample point cloud Q constructed from it with this method, is shown in 4.38. It still shows the linear increase in the ROI. However this result is not surprising, since Q was specifically constructed to have an uniform distribution of the displacement radii r on the tangent planes.

The idea is that if another, higher resolution scan of the same object is used as Q, the linear increase should also be present, but only when the point clouds are perfectly aligned. Measuring how linear the ROI is should give an indication of how well P and Q are aligned.

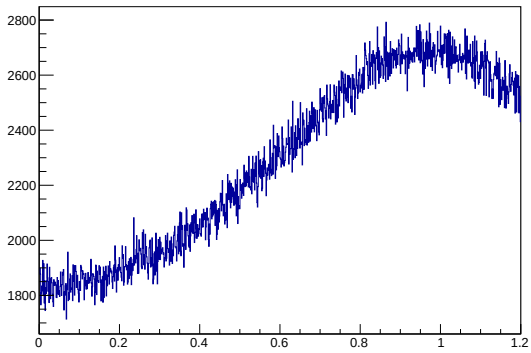


Figure 4.38.: AOH for P from constructed sample point cloud P

4.7. Cross distance histogram

Up until now P and Q were considered to represent the same surface and be perfectly aligned. For the cross distance histogram (CH) this last constraint is removed, so there is a rigid transformation $\mathbf{M}$ which aligns P to Q.

4.7.1. Adjusted histogram

As before the aCH is constructed from the distances of the sample points $q \in Q$ to their closest point $p \in P$.

To reduce the number of wrong correspondences, they are only accepted when the normal vectors of p and of q are *compatible*, that is, the angle $\arccos \vec{n}(p)\,\vec{n}(q)$ is below a fixed threshold value. If the normals point in different directions, the points do not represent the same locations on the surface. Also, correspondences where $d(p,q)$ is above a fixed maximal distance are rejected.

Because the surfaces are no longer perfectly aligned, the distribution of the distances $d = d(p,q)$ will be different. Whenever q is at a distance from the surface of P on which p lies, q no longer falls into the two-dimensional parallelogram grid, and d will be larger. Hence the more $\mathbf{M}$ deviates from $\mathbf{I}$, the more the distribution of d will deviate from the one from the aOH, and it will no longer have the characteristic linear increase from $d = 0$ to 1.

4.7.2. Rejection histogram

In addition to this aCH the adjusted rejection cross distance histogram (aRCH) will be considered. Here the sample points $q \in Q$ are projected onto the surface of P, and then the distance from this *projected point* q_{proj} to p is measured and recorded into the histogram. The projected point is calculated using the normal vector of p, denoted $\vec{n}_p$:

$$\vec{q}_{\text{proj}} = \vec{q} - [\vec{n}_p(\vec{q} - \vec{p})]\,\vec{n}_p \tag{4.14}$$

And the distance $d_{\text{proj}} = \|\vec{q}_{proj} - \vec{p}\|$ is recorded. The vector $\vec{q}_{proj} - \vec{p}$ is also called the *rejection*.

p is still computed as the closest point to q and not to q_{proj}, as otherwise the closest point algorithm would have to be run twice. This should not have a noticeable impact on the results.

Assuming that $\vec{n}_p$ is accurate, $\vec{q}_{\text{proj}}$ is placed on the surface, and as a result the histogram should take on the shape of the aOH. However the sample points $vecq_{\text{proj}}$ are no longer as evenly distributed. When

Q is rotated relative to P, they will tend to cluster together more in areas further from the center of rotation. As shown before in 4.6.2, more variance in the density of the sample points leads to a more jagged histogram.

Also, unlike the aCH, this aRCH is, ideally, based solely on the dispersion patterns of the points of the surface, and does not take the distances of the surfaces of P and Q into account.

4.7.3. Measure of alignment

In order to find out how the aCH and aRCH change when a small rigid transformation is applied to P, several small translations and rotations were tested. The model is same artificial relief model as in the end of the previous section. For P it is projected from a sideways angle. Results are shown and commented in appendix A.2.

The goal was to construct a numerical error metric which indicates how well $\mathbf{M}$ aligns P and Q. For this the adjusted cross distance histogram is compared to the hypothetical linear increase from $d = 0$ to 1, using one of the measures described in section 2.8.

Figure 4.39 shows a observed histogram, along with the expected linear increase. The samples with $d > 1$ are discarded. The bins of the histogram are set to have equal width w. On the figure w is exaggerated, and neither the observed values nor the expected values (i.e. the blue line) are drawn with the correct bin size.

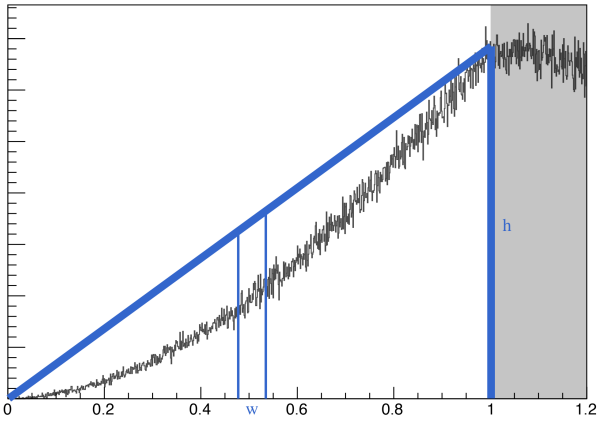


Figure 4.39.: Representation of observed and expected histogram

The "height" h of the histogram at $d = 1$ is unknown, and depends on the bin width. Because the range is 1, the area under the histogram curve is equal to the total number of included samples n. If the shape is precisely the linear increase, it should be equal to the area of the triangle formed by the blue lines, which is $A = \frac{1}{2} \times 1 \times h = \frac{h}{2}$. So $h = 2 \times n$.

The number o_i of observed values which fall into one bin of width w, centered at d_i, is the number of samples d such that $d_i - \frac{w}{2} < d < d_i + \frac{w}{2}$. The corresponding expected value is then calculated as the area of the rectangle of with w and height $\frac{d_i}{1} \times h$, so $e_i = w \times d_i \times 2 \times n$.

Both values are normalized by dividing them by n. The expected and observed values used to compare histograms are

$$o_i = \frac{1}{n} \times N\left(d_i : \frac{w}{2} < d < d_i + \frac{w}{2}\right) \qquad \text{and} \qquad e_i = 2\,w\,d_i \tag{4.15}$$

Using this, the *chi-squared* error metric is calculated. Let e_χ be this chi-squared measure when applied on the aCH, and $e_{r,\chi}$ its equivalent on the aRCH.

When the observed histogram increases linearly in the ROI, e_χ and $e_{r,\chi}$ should be zero, otherwise, it should be larger. So this new error metric should have its minimal value when $\mathbf{M}$ perfectly aligns the point clouds.

These error metrics, was compared with the mean absolute error taken with the closest point criterion. The same relief point cloud is used for both, and the error metric is visualized as described before. The results are shown and commented in appendix A.4.

4.7.4. Results

Two observations could be made:

1. e_χ does work as an error metric, and attains a local minimum at the true transformation at small scales. However, it is inferior in quality to the mean absolute error.

2. For point clouds where one has a much lower resolution and with rotational transformations only, $e_{r,\chi}$ reaches a local minimum at the true transformation, at a very small scale where the mean absolute error no longer gives useful results.

The first observation at least shows that the registration error can be measured using an approach as described in this chapter. However, it could not be applied to real scans, because there is too much error in the points' positions, compared to the predicted parallelogram lattice on planar surfaces. Also, e_χ only starts to behave like an error metric when the alignment is already very close to the optimum, at a scale where a registration of real point cloud scans would already be considered to be good enough.

For the second observation, it is not clear from the experiments whether it is an anomaly resulting from the implementation of the experiments, or an actual property of the aRCH. For this histogram, when P and Q are well aligned, more sample points $q \in Q$ get projected onto the correct locally planar surfaces of P, where they form a more uniformly distributed sample for measuring the closest point distances in the parallelogram lattices. This probably cannot be applied to real point scans either.

4.7.5. Conclusions

The goal of improving improving fine registration of point clouds with different resolutions could not be reached, but it could be shown that an approach based on the comparison of probability distributions originating from the point dispersion patterns, is possible.

The problem of minimizing e_χ can be addressed by randomly sampling the neighboring transformations space at each iteration if ICP. Alternately, it might be possible to set the weights of the correspondences to values that would "pull" the ROI histogram into the intended shape.

In practice, such registration problems can usually already be solved using one of the variants of ICP or another fine registration algorithm. It has been shown in section 4.3 that ICP is relatively insensitive to resolution differences. For example, in figure 4.40 the high and low resolution versions of the were registered with a standard point-to-point ICP. The only additional constraint was that only 20000 random points were chosen for each iteration, and points located too far away from the centroid were ignored.

The premise was that information about the true error $e(\mathbf{M}')$ is "encoded" within the coarsely aligned point clouds P and Q, and can be extracted by analyzing their point dispersion patterns, or other attributes. Based on this it may be possible to find apply *data mining* or similar artificial intelligence techniques on the problem. By using a large testing set, automatically find a function f that best fits

Figure 4.40.: Example of good registration of

$f(P, Q, \mathbf{M}', x_1, x_2, \cdots, x_m) \approx e(\mathbf{M}')$, where x_i are various features extracted from P and Q. These would for instance include metrics of the closest point distance histograms.

5. Implementation

As part of this thesis, a relatively large software framework for working with point clouds was developed. Its main purpose was to have a toolkit based on which to process point clouds and experiment with different algorithms. It is not a finished product, as parts of it were changed or added depending on the requirements, but has developed into a sufficiently stable tool.

This chapter presents some of the work done, including software design choices, problems encountered, data structures and algorithms, and optimizations for handling large data sizes. It might be useful as a source of advice for similar software development projects.

5.1. General architecture

The software framework is called `pcf` (Point Cloud Fusion), and consists of three object-oriented libraries written in C++:

core Main part containing implementation of the relevant objects, including point clouds with different data structures and point attributes, images, registration algorithms and geometrical data types.

viewer Visualization of the objects using OpenGL.

experiment System for running large number of ICP registrations with different parameters, and for generating visualizations of error metric functions, as shown in the previous chapter. For the ICP registrations, results are stored in an *SQLite* database, from where different kinds of plots can be extracted.

5.1.1. Programming language

Point cloud algorithms work with large quantities of data. For example, the scan of the front facade of the Hôtel de Ville shown on figure 4.2 in the previous chapter is a 202.2 MB file containing about 7 million points. For any algorithms working on a per-point basis, minuscule execution time differences per point quickly multiply when applied on large point clouds. So it is important for code to be compiled into machine code instructions as efficiently as possible.

This makes C++ a logical choice for the programming language. Its template meta-programming and function inlining facilities allow for object oriented code that expresses the operation to be performed in a semantically logical way to be reduced, at compile time, to the shortest possible set of instructions performing the same operations. For example the following code could sort a set of points `points` by increasing distance to a point `q`:

```
sort(points.begin(), points.end(), bind(distance, q, _1))
```

This would instantiate a template function for `sort`, with a specific implementation of the sorting algorithm that uses the given iterators of the `points` set, the distance function`distance(q, p)`, and the swapping function for two points `swap(p1, p2)`. Both of these can then be inlined, removing the overhead of one of more function calls per point. Then the compiler could further optimize this specific

instantiation of `sort`, for example by comparing the distances $|x^2 + y^2 + z^2|$ instead of $\sqrt{x^2 + y^2 + z^2}$ because this would produce the same results, and saves one square root evaluation per point.[1]

In particular this is used in linear algebra libraries, where techniques such as *expression templates* essentially allow for algebraic expressions involving scalars, vectors and matrices and others to be simplified at compile-time. For instance $k(\mathbf{A}+\mathbf{B})$, where $\mathbf{A}, \mathbf{B}$ are two large matrices, is computed more efficiently than $k\mathbf{A} + k\mathbf{B}$. For linear algebra, the *Eigen* library is used in the project. It implements those optimizations, as well as SIMD vectorization. An operation on a 4-vector (for example a three-dimensional spatial vector in homogeneous coordinates), can be translated into a single SIMD instruction. It also provides support for 3D geometry, such as affine transformations, quaternions and planes.

Because point clouds processing often times involves executing the same operations for a large number of points, the algorithms are typically good candidates for parallelization. *OpenMP* is used throughout the implementation to parallelize loops. However it is not part of standard C++ and not supported by all compilers.

5.1.2. Interactive console

Most of `pcf` is implemented in the form of a library, as the goal was to have a toolkit providing the relevant functionality, and not just a single application. For applications such as running the ICP experiments, a corresponding executable program was written that uses the library.

But the compile-link-execute cycle makes this impracticable for interactively working with the point cloud objects, and in particular for visualizing the scene. Developing a textual or graphical user interface for the library would have been a large and useless workload, since eventually a user interface for every functionality of the library would have to be added.

A better approach would be to have an interactive console, similar to the ones that exist for scripting languages such as *Python*, *JavaScript* or *Perl*. An attempt was made to add a *Python* binding to the C++ library using *SWIG*, but it proved to create too make complications, in particular because of the intensive use of template meta-programming.

Instead the *ROOT* framework is used, which provides the console and just-in-time compiler *Cling* for C++. Despite some problems, it provides a practical way for using the API of `pcf` interactively. More details are described later in this chapter.

5.2. Core

As indicated, this main part of the implementation consists of template classes representing point clouds with different data structures, several algorithms to process points, a generic implementation of the ICP algorithm, a tree structure of space objects with poses defined relative to their parent object, virtual cameras, and other components.

5.2.1. Point data type

Two types of *point* are provided. The most basic `point_xyz` consists only the cartesian coordinates, and a flag indicating whether the point is *valid*. It is represented using 4 floating point values, and when the point is valid, the fourth component is set to 1.0.

[1] This particular optimization is probably not done by compilers because in some edge cases it can produce different results due to floating point imprecision.

This way a 4-vector of the point's position in homogeneous coordinates is formed, which can be processed efficiently using SIMD instructions. With only three components point sets could be packed more densely in arrays, but because their addresses and sizes are not aligned on a power-of-two boundary, efficient processing using SIMD vectorization would not be possible.

The *valid* flag is for example used in range images, where points in memory are arranged in a two-dimensional array, and some represent background pixels in image space. It also allows for removing points out of a point cloud without rearranging the other points or implementing a separate mask array.

Its subclass `point_full` additionally contains an RGB color, a normal vector, a scalar weight and an integer index. It is padded to a size of 64 bytes the nearest power of two. The index is used in some situations, for instance when corresponding points are ordered differently in memory for two point clouds, but should retain a reference to one another.

Different point types could also be useful, but are not implemented. Most parts of the framework are generalized to work with different point types.

5.2.2. Point cloud data structures

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.

In the library different data structures are implemented as subclasses of a common abstract `point_cloud` base class. The base class allocates and owns memory for the points, provides iterators to access them, including one which implicitly transforms their coordinates into another coordinate system. It also provides functions that do not modify the points, such as calculating the bounding box.

The subclasses handle the actual ordering of the points in memory, and implement algorithms such as closest point or k-nearest neighbor finding with it. Four subclasses are implemented: *unorganized*, *tree*, *grid* and *range*. Tough the subclasses may allocate additional memory to store information about the data structure, the points are always stored in an array ordered a certain way. This way any function which is agnostic about the point ordering can still access the point cloud the same way.

To simplify implementation, `point_cloud` and its subclasses are themselves immutable objects: They can only be created once, but not reassigned, since this would require rebuilding the data structure. The points can still be modified. According to the programming by contract principle, the user of the classes needs to make sure not to modify the coordinates of points when they would make the points ordering invalid.

Putting a point cloud from one data structure into another is done via copy or move construction. In the latter case, the same memory is reused, reducing the total memory usage when working with large point clouds.

Unorganized point cloud

This class does not impose any ordering on the points, and is used for modifying it. For example, functions are provided to add noise, randomly displace the points, apply a transformation to the points, apply downsampling, and others.

Tree point cloud

Here the point clouds gets arranged into a space subdivision tree, such as a KdTree or an Octree. It is a generic implementation supporting different kinds of trees via a template argument. Points in memory are ordered such that points falling into same same tree node are stored in the same memory segment.

The tree point cloud allows for an efficient implementation of the closest point finding algorithm. To find the point closest to any position $\vec{x}$, it first searches only inside the leaf of the tree that $\vec{x}$ lies inside. If it doesn't find the closest point there, it also searches in the neighboring nodes. KdTrees make this most efficient, because there are only 2 child nodes per parent, and the tree depth is balanced. There can be edge cases where the closest point actually lies in a node that is several layers in the tree away from the one containing $\vec{x}$. Therefore the algorithm can be accelerated by specifying a *accept distance* $a > 0$ and a *reject distance* $r < \infty$. When any point closer than a is found it is accepted as closest point, and when it is determined that no point can lie closer than r, the algorithm fails.

There are other approaches for optimizing closest point finding on tree structures, such as the one presented in [Kim & Lee, 2009] for Octrees.

Grid point cloud

For the grid point cloud the points instead divided into a regular grid of equal-sized, axis-aligned cubic cells. Using this an efficient k-nearest neighbor algorithm was implemented, based on the approach described in [Sankaranarayanan *et al.* , 2007]. Its execution time gets shorter the smaller the cubic cells are set. But this also entails a cubical increase in memory usage, because a separate list of the memory offsets for each cell is kept.

Range point cloud

Represents a range image. Points are ordered row-by-row like the image pixels. Invalid points are put in for background pixels of the range image. The points are always set in the coordinate system where the camera is at origin.

A subclass of it the *camera range point cloud*, also includes the virtual camera object by which the points were projected. When artificial range image are generated, a point cloud of that class is returned. It provides for example a function to back-project any position in space to the corresponding position in image space.

Const-correctness

Properly implementing const-correctness on the point cloud classes can be a difficult problem. The point cloud are container classes which *own* their points, and thus *const* access to the point cloud should only allow *const* to its points. This would for example require two instances of the `closest_point` algorithm: A *const* version which also returns a reference to a *const* point, and a corresponding *non-const* version with exactly the same code.

The underlying problem is C++'s lack of support for templatizing member functions for *const*-ness of `this`. Proposals exist to add this into future versions of the C++ standard. Some solution possibilities are (1) to implement the algorithm in a non-member function which takes the *const* or *non-const* class type as template argument, and then have the two member functions call this algorithm, (2) to implement only one version of the member function, and have the other one forward the call to it using `const_cast`, or finally (3) to outsource algorithms into non-owning helper classes.

This last approach just moves the problem somewhere else, as there would have to be for example a *const* and a *non-const* `point_cloud_segment` class.

5.2.3. Space object

The point clouds, as well as other classes such as geometrical objects or cameras, are subclasses of `space_object`, which contains a pose, and references a parent `space_object`. Each space object thus defines a coordinate system.

Points in a point cloud contain the coordinates in its own coordinate system. The unorganized point cloud provides a function to *apply* the pose: It sets the space object pose to identity, but changes the points' coordinates to the equivalent positions. This can not be done in organized (i.e. tree, grid, range) point clouds, because altering the point positions would make the data structure invalid.

The point cloud base class also provides a *transform iterator*, which iterates through the points, but instead of returning the actual point, return a transformed version of it. This way it is possible to use a construct like

```
for(auto it = P.begin_relative_to(Q); it != P.end_relative_to(); ++it) {
  const point_full p = *P;
  const point_full& q = Q.closest_point(p);
  cout << "closest point distance = " << distance(p, q) << endl;
}
```

This iterates through the points of P using a transform iteration which transforms them into the coordinate system of Q. Then the closest point distance to a point in Q is measured. `begin_relative_to` traverses the space object tree to calculate the appropriate rigid transformation. The point cloud functions like `closest_point` expect a point or position in the same coordinate system as the point cloud as input.

When transforming a point, both its position coordinates, and its normal vector coordinates need to be changed. Only the linear part of the transformation is applied to the normal vector. The point classes are not subclasses of `space_object`. A good alternate approach may be to introduce a wrapper class which associates a space object pose to a point, so that points are automatically put in the right coordinate system as needed, simplifying usage. This is not implemented, and it would have to be designed in such a way that any additional overhead gets resolved at compile time.

5.2.4. Point algorithms

Several algorithm operating on point, or on point clouds are implemented, for example the curvature measure developed in chapter 4, or the creation of a sample point cloud Q by adding points on tangent planes around the existing points of P.

The algorithms are separated into two categories: Some which operate on any set of points, and some which operate on a point cloud, and make use of the point cloud's member functions such as its `closest_point` implementation. Similar to the Standard Template Library (STL) `<algorithm>` header, they are implemented as standalone template functions and not as member functions of the point class.

The reason is that *point* algorithms should be able to work with any list of points, not only points in a `point_cloud`. This includes for example points yielded by the transform iterator. Like the stl algorithms, they take two generic iterators as input which delimit the range. *Point cloud* algorithms take a generic point cloud subclass as input.

5.2.5. Iterative Correspondences Registration

A generic implementation the ICP framework algorithm was created in form of a template class taking as template arguments the loose and fixed point clouds types, a *correspondences* class, an *error metric* class, and a *transformation estimation* class. Only point-to-point ICP is implemented.

The *correspondences* classes output the found correspondence pairs to a generic *receiver* callback object given as function argument. When parallelization is used, it also creates new *receiver* instances, sends correspondence pairs to them on different threads, and then joins them together.

The *iterative correspondences registration* class itself uses a *receiver* class that forwards the incoming correspondence pairs to both a *error metric* and a *transformation estimation* instance.

It can run the ICP registration for a fixed number of iterations, or until the error metric has reaches a predefined minimal value. Because it takes only *const* references to the loose and fixed point clouds, it does not modify the loose point cloud's pose. Instead, the accumulated transformation is stored which can then be applied to the point cloud by the class user.

Having the class modify the point cloud would for example make it impossible to run several registration attempts on it simultaneously, or to use registration on a *const* point cloud.

5.2.6. Other components

Import and export of point clouds from and to the PLY file format is implemented. This is a de factor standard file format for point cloud data and is also supported by most other point cloud processing software. The format is versatile, and can also be used to store meshes, triangular faces, various kinds of point attributes, and even data other than 3D models.

The framework implements a custom memory allocator for the point cloud class. When the requested size is higher than a fixed threshold (set for example at 1 GB), it does not allocate member using the standard `malloc` call, but instead creates a temporary memory-mapped file on the disk, and allocates a virtual memory region which maps to it. This operating system-specific feature is implemented for both Windows and Darwin. Whether it provides an advantage depends on the operating system, but it typically alters the virtual memory pagination strategy such that it no longer tries to load as many pages as possible into physical memory. On Darwin, this would cause the system to stall as pages get swapped out when allocating large memory chunks, leading eventually to a segmentation fault. With memory-mapped files this issue was resolved.

5.3. Viewer

The viewer is also implemented in form of a library, and can be controlled interactively from the *Cling* console. It provides a *scene* with *scene objects*, which can be moved in space either programmatically or using the keyboard and mouse on the visualized. *Scene objects* are tied to *space object* like point clouds, and hook functions are installed so that when the *space object*'s absolute pose is changed (which can also happen when its parent's pose changes), the *scene object* gets also updated.

Scene objects implement rendering of a visualization of the object in an active OpenGL context. They are implemented for point clouds, some geometric objects, and correspondences.

5.3.1. Point cloud visualization

Copying the entire set of points into GPU and rendering it using `glDrawArrays` would cease to work for large point clouds. A system which dynamically reduces the number of points sent to OpenGL as

needed.

An algorithm is implemented which dynamically extracts a visible subset of the point cloud in function of the camera pose and field of view, using both viewing frustum culling and live downsampling. The *scene point cloud* internally stores a copy of the point cloud in an as an Octree *tree point cloud*. The storage order points in the Octree leafs is randomly shuffled.

Viewing frustum culling

Each node of the Octree corresponds to a cubic region in space, an can be completely outside, completely inside, or partially inside the camera viewing frustum. The algorithm takes only the points lying in nodes that are completely inside the viewing frustum. For nodes that are partially inside, it recursively traverses its child nodes, and takes points from child nodes that are completely inside, and also leafs that are partially inside. This is necessary because leafs are not further subdivided.

Real-time down-sampling

If the number of points in this subset P' if greater than a predefined rendered capacity c, it is down-sampled in real-time. A simple random down-sampling is applied, in which each point is chosen with a probability of $\frac{c}{\|P'\|}$. Simply taking the c first (or last) points in the array P' would include more points from the first included node, and none from the later ones. But the chosen points should be evenly distributed among the nodes.

An algorithm which runs the random number generator could be inefficient, depending on the pseudo-random number algorithm used. Also it would not guarantee that exactly c points are selected.

If $\|P'\|$ is dividable by c, then a good deterministic approach is to select every k-th point, where $k = \frac{\|P'\|}{c}$. If this not the case, a method akin to *Bresenham's line algorithm* is used to select point indices between 1 and $\|P'\|$. This is efficient, deterministic, and choses points from the array at an approximatively uniform distribution.

The reason that points in the leaf nodes were shuffled is to remove and spatial bias resulting from the way points were ordered previously in the original point cloud, from which this *scene point cloud* was constructed. For example when Bresenham's line algorithm selects two neighboring points in the array, their spatial positions should not be closer than for any other two points.

Buffer swapping

A dual-buffered method is used to apply this dynamic down-sampling in the renderer. Two Graphics Processing Unit (GPU) buffers of capacity c are allocated. The first one contains the array of points currently visible onscreen, and is passed to the OpenGL pipeline at each frame using a `glDrawArrays` call.

The dynamic down-sampling algorithm is run in a separate thread, whenever the camera is moved. It outputs the selected point into the second buffer. When it has finished, the two buffers are *swapped*, meaning they change roles, without any copying taking place. This way blocking is minimized.

5.4. Console

As described, the *ROOT* framework with the *Cling* just-in-time C++ compiler was used to have an interactive console to access the library.

ROOT Data Analysis Framework is a C++ framework providing functionality to handle and analyze large amounts of scientific data. It contains components for recording, analyzing and visualizing two-dimensional and three-dimensional physical fields, function plots, histograms, images, and more. The framework is available for free download under the *LGPL* license. It is developed by *CERN*, and was initially created in the context of a particle physics experiment in 1997.

The framework is built so that C++ can be used interactively is a console interface similar to *iPython* for example. The core components for this is the included *Cling* just-in-time compiler, which a fork of the open source *LLVM/Clang* C++ compiler. *Cling* provides a console, which adds some functionality to C++ such as the ability to dynamically instantiate objects in a new *console* scope. *ROOT* also provides a large set of graphical user interface components, which all respond dynamically to input from the console. For example the histograms in chapter 4 were created using this.

5.4.1. Usage with framework

Importantly, *Cling* is able to load external C++ headers, and link to prebuilt dynamic libraries. Being based on *Clang*, it has full support for C++. Linking to the `pcf` framework from *Cling* works well, even if this was done in an afterthought. For example, the following code can be entered line by line in the console:

```
using namespace pcf;

// Create two point cloud object of the Bunny model
unorganized_point_cloud_full P = import_point_cloud("bunny.ply");
kdtree_point_cloud_full Q = P; // store Q in KdTree for efficient closest point finding

// P = loose point cloud.
// Make point colors white, randomly displace points, and randomly down-sample to 30%
set_unique_color(P.begin(), P.end(), rgb_color::white);
P.randomly_displace_points(std::uniform_real_distribution<float>(-0.002, +0.002));
P.downsample_random(0.3);

// Q = fixed point cloud
// Make point colors red, and apply some translation and rotation
set_unique_color(Q.begin(), Q.end(), rgb_color::red);
Q.move_x(0.01);
Q.rotate_y_axis(angle::degrees(0.1));

// Register P and Q using ICP, selecting all points from P, and run for 10 iterations
auto reg = make_iterative_closest_point_registration(Q, P, accept_point_filter());
reg.maximal_iterations = 10;
reg.run();

// Create OpenGL visualizer, and display both P and Q
viewer_window vw;
vw->add(P, Q);

// Now apply the estimated transformation to P
// The visualizer responds interactively and shows P with its new pose
P.set_relative_pose(reg.accumulated_transformation());
```

5.4.2. Problems

However, some problems were encountered when using `pcf` with *Cling*, including:

- Each time the console is started, it needs to link the dynamic library and parse headers. The whole process usually takes about 4 seconds.
- Due to bugs in the current version of *Cling*, entering syntax errors into the console sometimes messes up the internal state of *Cling*, so that subsequent commands no longer work properly. The console needs to be relaunched.
- Some C++ constructs are unsupported by *Cling* and produce error messages.
- Objects dynamically created on the global *console* scope cannot be deleted. The *ROOT* framework was instead designed to allocate all objects on the heap and use smart pointers.
- The *Cling* compiler does not support OpenMP.
- Most importantly, the current version of *Cling* compiled code without optimizations. Because `pcf` is heavily template-based, most of its functions are just-in-time (JIT) compiled by *Cling* and not precompiled in the dynamic library. This makes most algorithms much less efficient, especially when working with large data sets. To solve this, some template functions and classes are explicitly instantiated in `pcf`, so that frequently used versions of it are compiled with optimizations and included into the dynamic library when building it.

6. Conclusion

In this paper the problem of registration of 3D scans was studies from three points of view. First, a survey of previously developed point cloud registration algorithms was presented, especially variants of ICP. Some techniques as well as the mathematical theory behind it were laid out in more detail.

Secondly, the stability of standard ICP registration was tested experimentally with regard to different resolution point clouds, and different camera view points. For this ICP registrations were run on a large number of different point clouds and results were collected and analyzed.

Thirdly, an attempt was made to develop an error metric specifically for the case where the point clouds have different resolutions. It takes the dispersion of points on surfaces into account as additional information about the surface geometry. Moreover, it is not based on the assumption that closest point correspondences are a sufficient approximation to true correspondences. Instead, it tests in how far the distribution of closest point distances is as it should be when the surfaces are perfectly aligned.

For artificially generated point clouds, positive results were obtained, but the error metric could not be applied to real scans.

A. Experimental Results

A.1. ICP registration

A.1.1. Resolutions and ICP result, 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. Lower value means Loose has lover resolution.

A.1.1.1. No displacement

The two point clouds are perfectly aligned to start with.

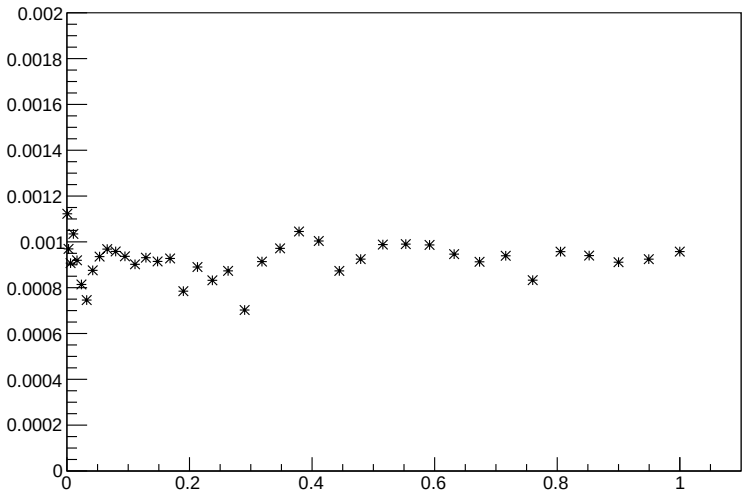


Figure A.1.: no displacement

A.1.1.2. Small displacement

Small displacement. Translation by magnitude of 0.01 in random direction, rotation by 3° or 15° on random axis direction. The Bunny modes has a width, height and depth of about 0.15.

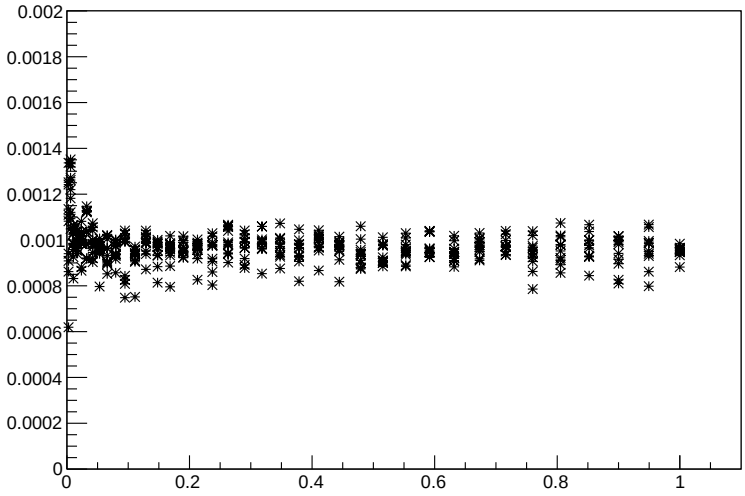


Figure A.2.: random translation of 0.01 and rotation of 3°, chosen 10 times

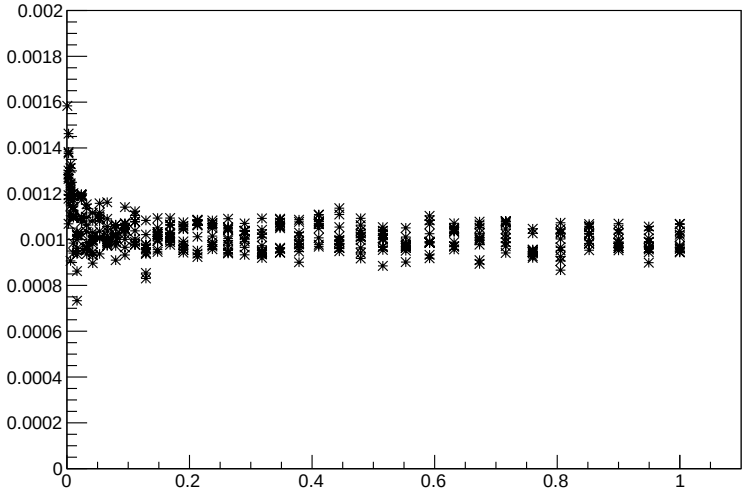


Figure A.3.: random translation of 0.01 and rotation of 15°, chosen 10 times

A.1.2. Evolution of ICP registration, Bunny model

This plot shows the evolution of the true error ICP registration, for the same input point clouds. Initially the point clouds are perfectly aligned. The Loose point clouds is randomly generated as before, results from all 60 registration runs are superimposed.

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	No displacement.
Y Axis	True error at iteration i.
X Axis	Iteration step i.

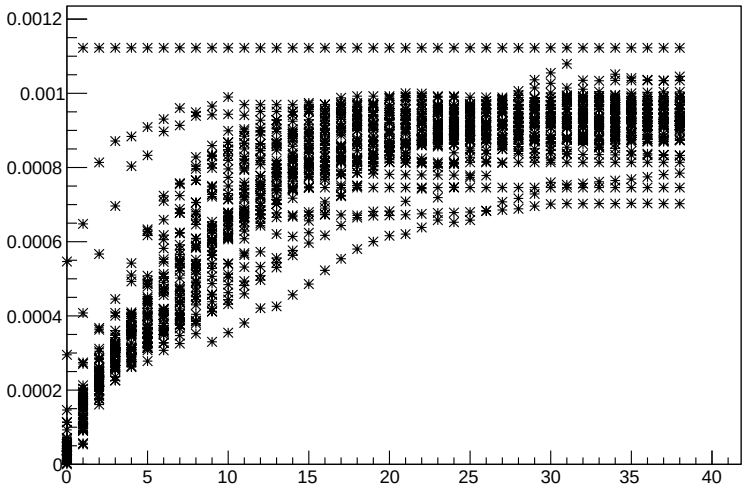
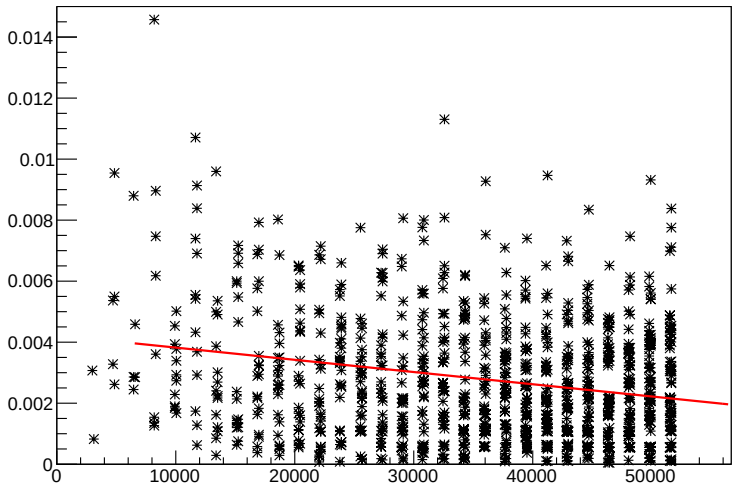


Figure A.4.: Evolutions of true error for experiment A.1.1

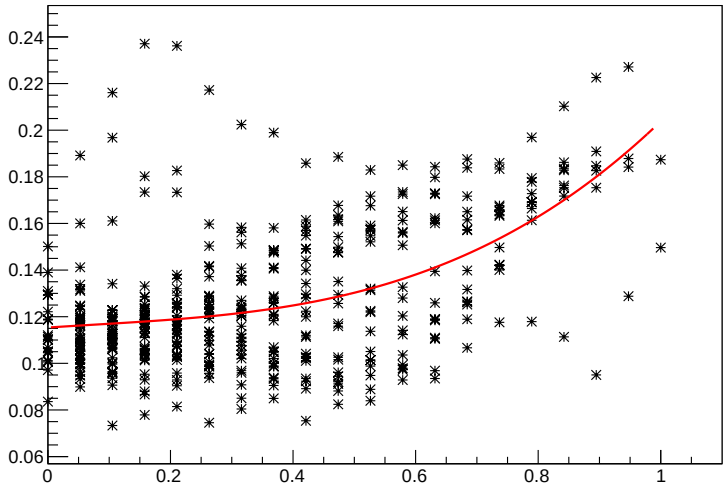
A.1.3. Resolutions and ICP result, sphere model

Method	ICP. Select all points, closest point criterion, equal weights, no rejection, point-to-point error metric.
Model	Artificially generated sphere point cloud with random point dispersion.
Fixed	Randomly chosen number of points from 0 to 50000. 30 steps.
Loose	Randomly chosen number of points from 0 to 50000. 30 steps.
Displacement	No displacement.
Y Axis	True error, after 40 iterations.
X Axis	Maximum of number of points in Loose and number of points in Fixed.



A.1.4. View-point and ICP result, relief model

Method	ICP. Select all points, closest point criterion, equal weights, no rejection, point-to-point error metric.
Model	Relief point cloud, projected with occlusion from camera placed at angle looking down on relief.
Fixed	Point cloud of relief with camera angle varying in $[0, 2\pi]$ around relief. 20 steps.
Loose	Point cloud of relief with camera angle varying in $[0, 2\pi]$ around relief. 20 steps.
Displacement	No displacement.
Y Axis	True error, after 40 iterations.
X Axis	Angle between Fixed and Loose camera positions.



A.2. Relief small transformation aCH

The following figures depict aCH of a relief point cloud P like the one on figure 4.7, on which a small transformation $\mathbf{M}$ was applied. For the point cloud P, the average nearest neighbor distance on surfaces perpendicular to the camera ray is $p_l = 0.01$. The plane of the relief is along the X and Y axis.

A.2.1. Translations

Translations are applied on two axis, by the amounts { 0, 0.002, 0.004, 0.006, 0.008, 0.010 }.

A.2.1.1. X and Y axis

Here $\mathbf{M}$ is a translation in X and Y axis, by these amounts. No translation in Z axis and no rotation is applied.

The top-left cell shows the own distance histogram, where $\mathbf{M} = \mathbf{I}$. Only translations in the positive directions are made, translations in the negative directions would yield similar results.

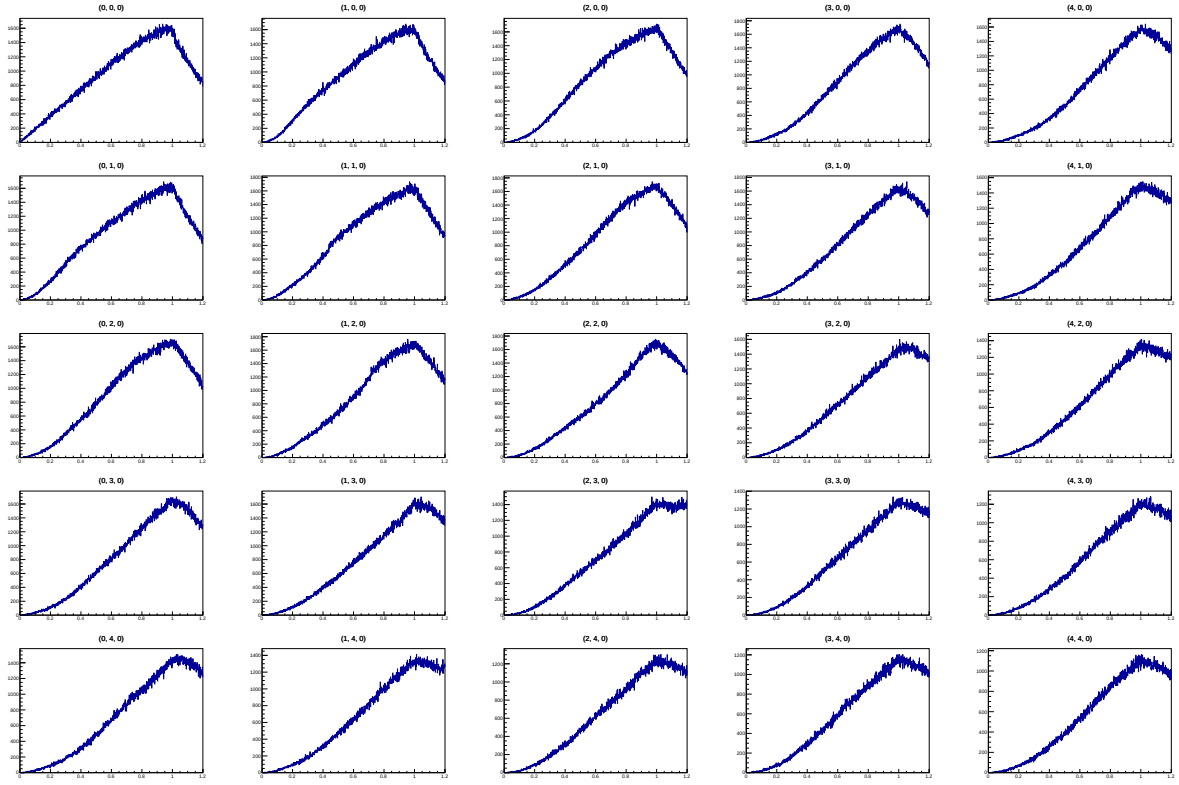


Figure A.5.: Horizontal: X translation, vertical: Y translation

Observations Translation on the X and on the Y axis have similar results, which is to be expected since the plane of the relief spans in those directions. If P was a perfect plane, the histograms would retain exactly the same shape. Because this is not the case, distances between points on more oblique parts of the surface are increased, resulting in a slightly more convex curvature.

A.2.1.2. X and Z axis

Y axis translation fixed to 0. Translations by same amounts.

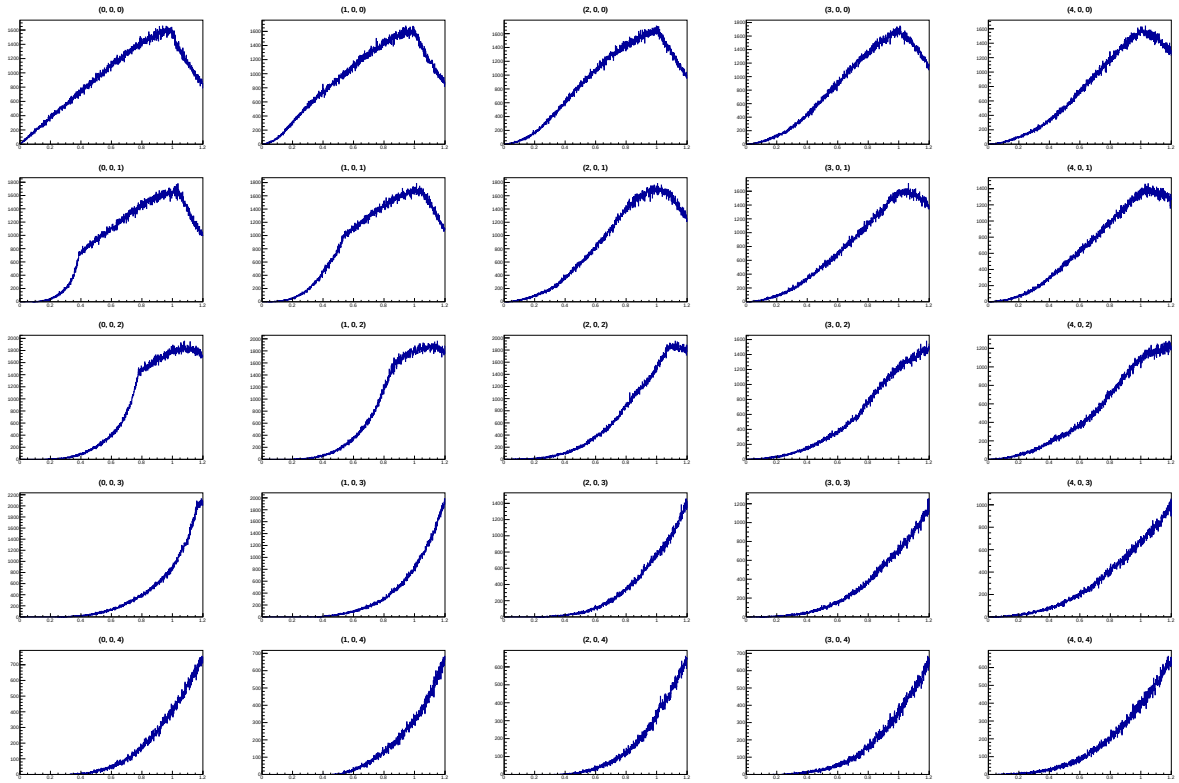


Figure A.6.: Horizontal: X translation, vertical: Z translation

Observations Translation on the X axis is the same as before. But the translation on the Z axis pulls the surfaces, and the histogram gets an offset to the right side. Some oblique surfaces of the model remain close, explaining that parts of the linear increase can remain unchanged.

A.2.2. Rotations

Here instead of a translation, a rotation gets applied on the same point cloud. The point of rotation is the center point of the relief plane. The amounts of rotation are { 0, 1°, 2°, 3°, 4°, 5° }.

A.2.2.1. Rotation around X and Y axis

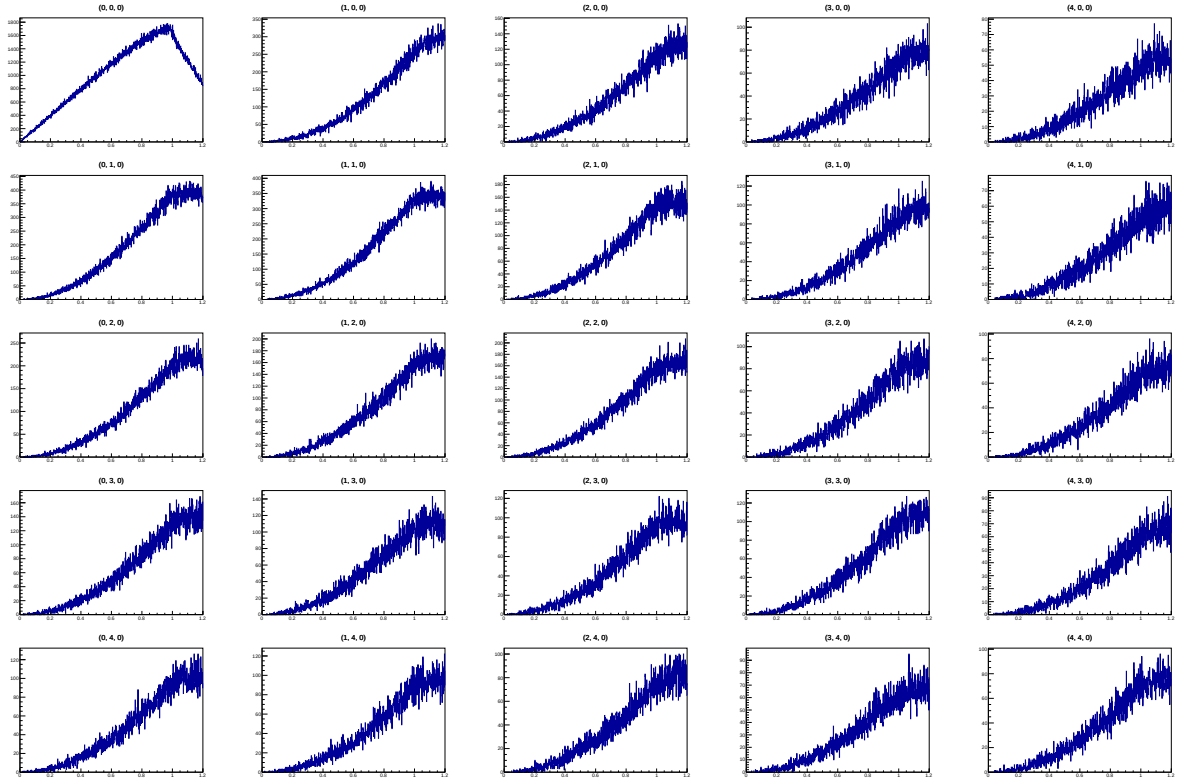


Figure A.7.: Horizontal: X axis rotation, vertical: Y axis rotation

Observations When the point cloud is rotated along the center point, distances of points change more, the further the points are from the center. Because less distances are recorded within the range $[0, \frac{1}{2}l_{\min}]$, the adjusted histogram becomes more jagged.

A.2.2.2. Rotation around X and Z axis

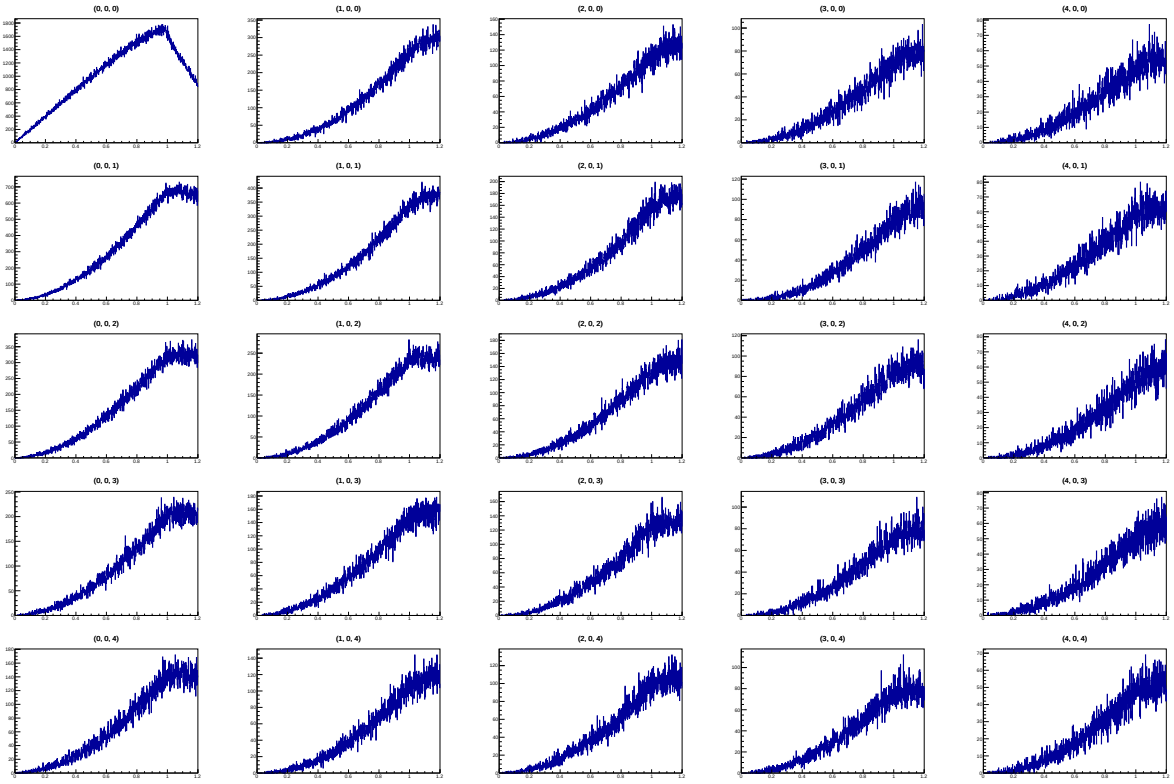


Figure A.8.: Horizontal: X axis rotation, vertical: Z axis rotation

Observations The results are similar as before. The increase in jaggedness is weaker for rotations on the Z axis, because a larger part of the surfaces do not get pulled off each other but slide on each other.

A.3. Relief small transformation aRCH

Here the same translations are rotations are applied, and the aRCH instead of the aCH is shown.

A.3.1. Translations

A.3.1.1. Translation in X and Y axis

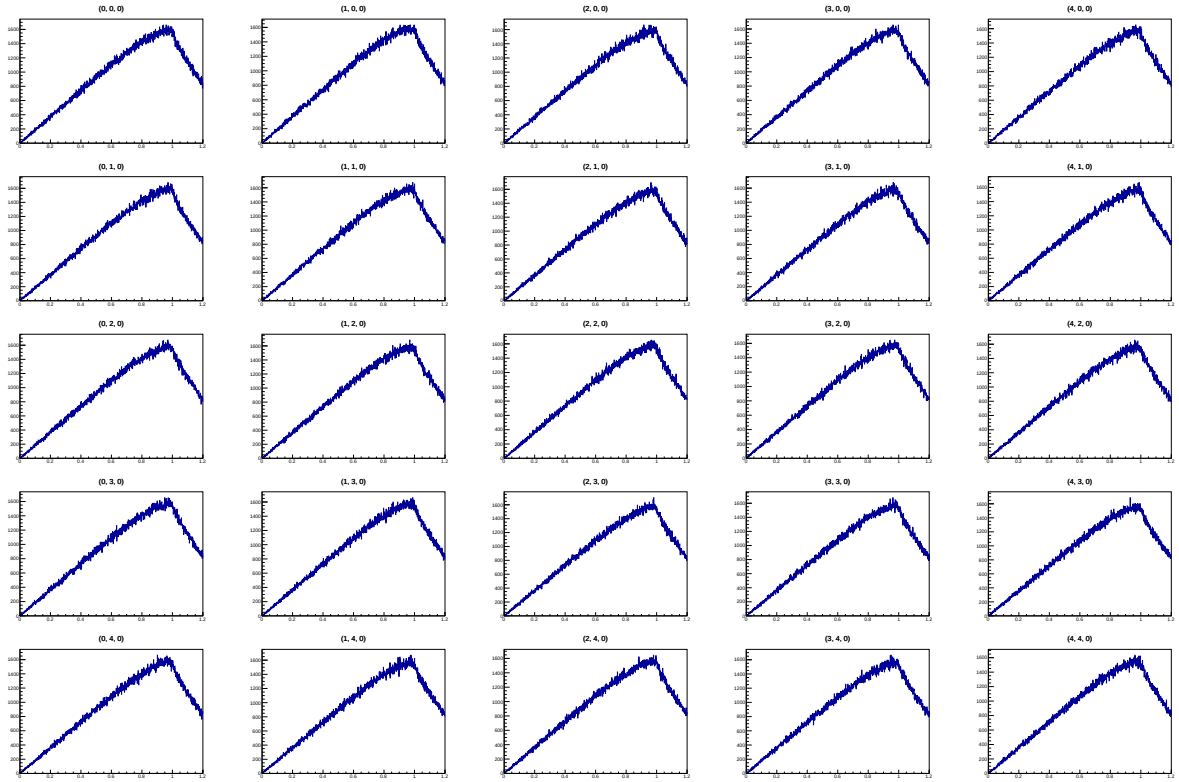


Figure A.9.: Horizontal: X translation, vertical: Y translation

Observations The histograms remain unchanged. In comparison to A.2.1.1, the effect of the oblique surfaces get cancelled out because points are projected back onto the surface after the translation.

A.3.1.2. Translation in X and Z axis

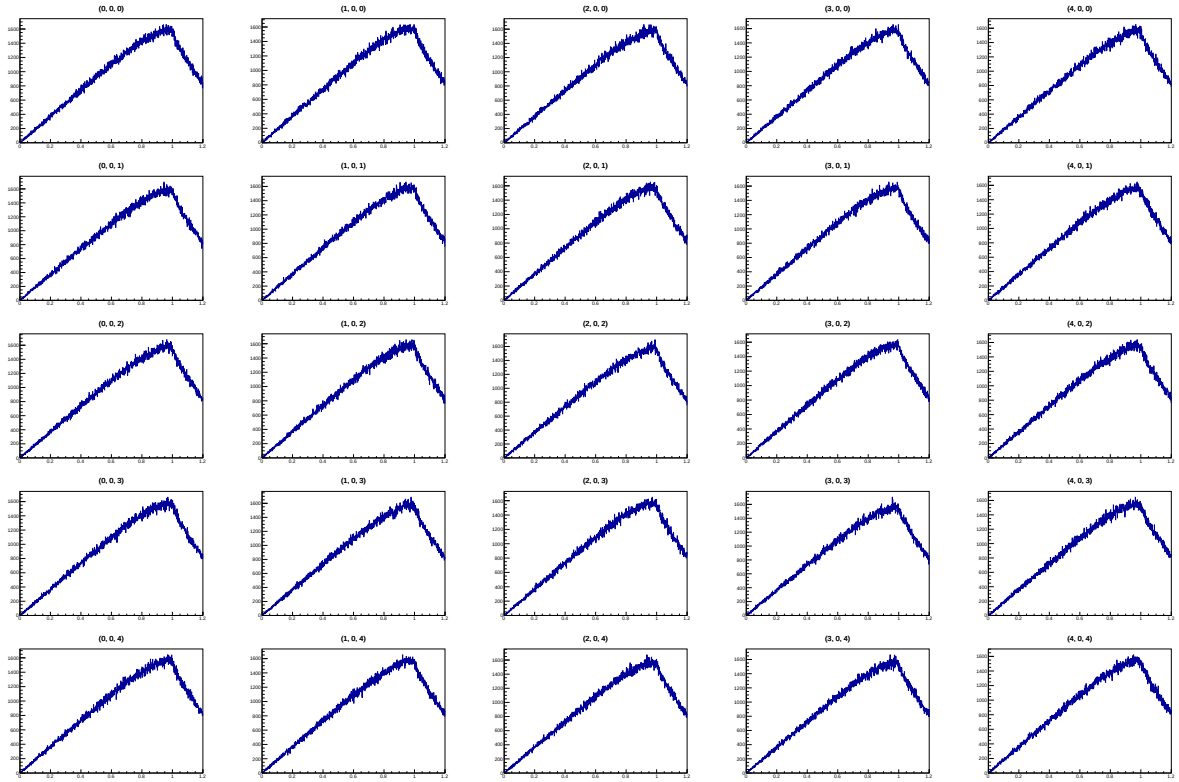


Figure A.10.: Horizontal: X translation, vertical: Z translation

Observations The histograms also remain unchanged.

A.3.2. Rotations

A.3.2.1. Rotation around X and Y axis

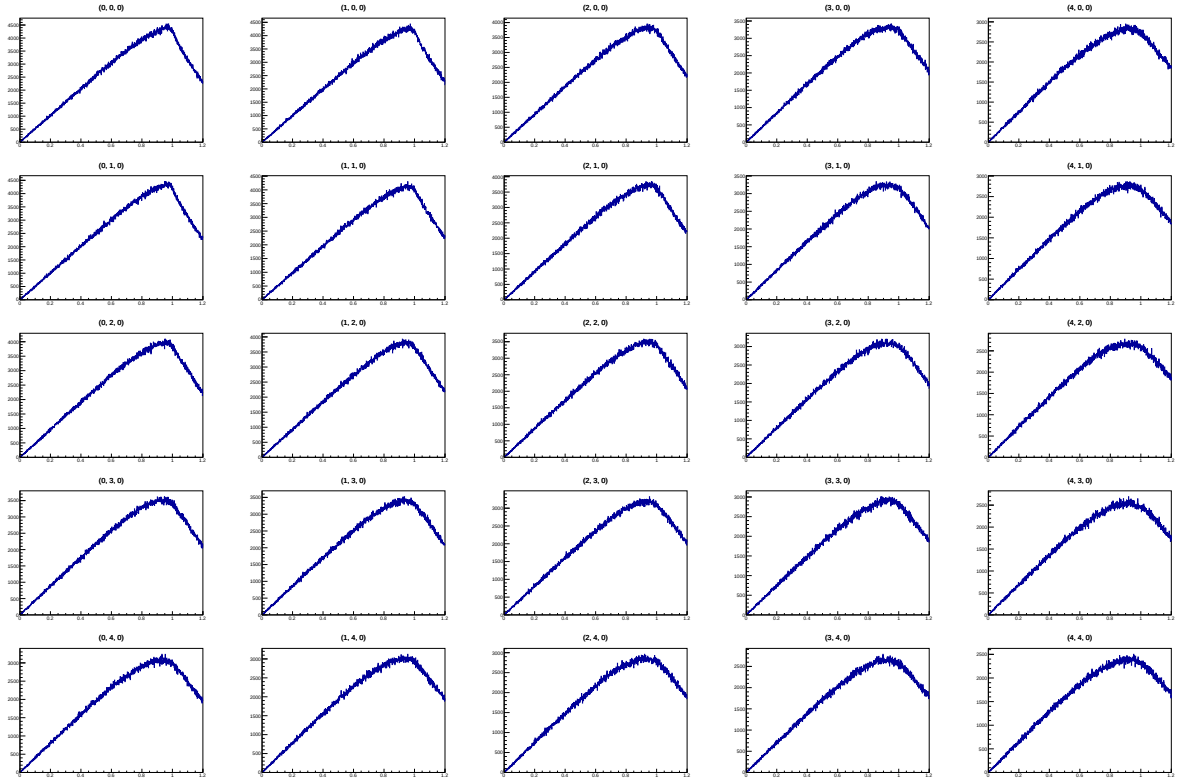


Figure A.11.: Horizontal: X axis rotation, vertical: Y axis rotation

Observations It can be observed that the histogram becomes slightly less linear and more jagged with larger rotation angles.

A.3.2.2. Rotation around X and Z axis

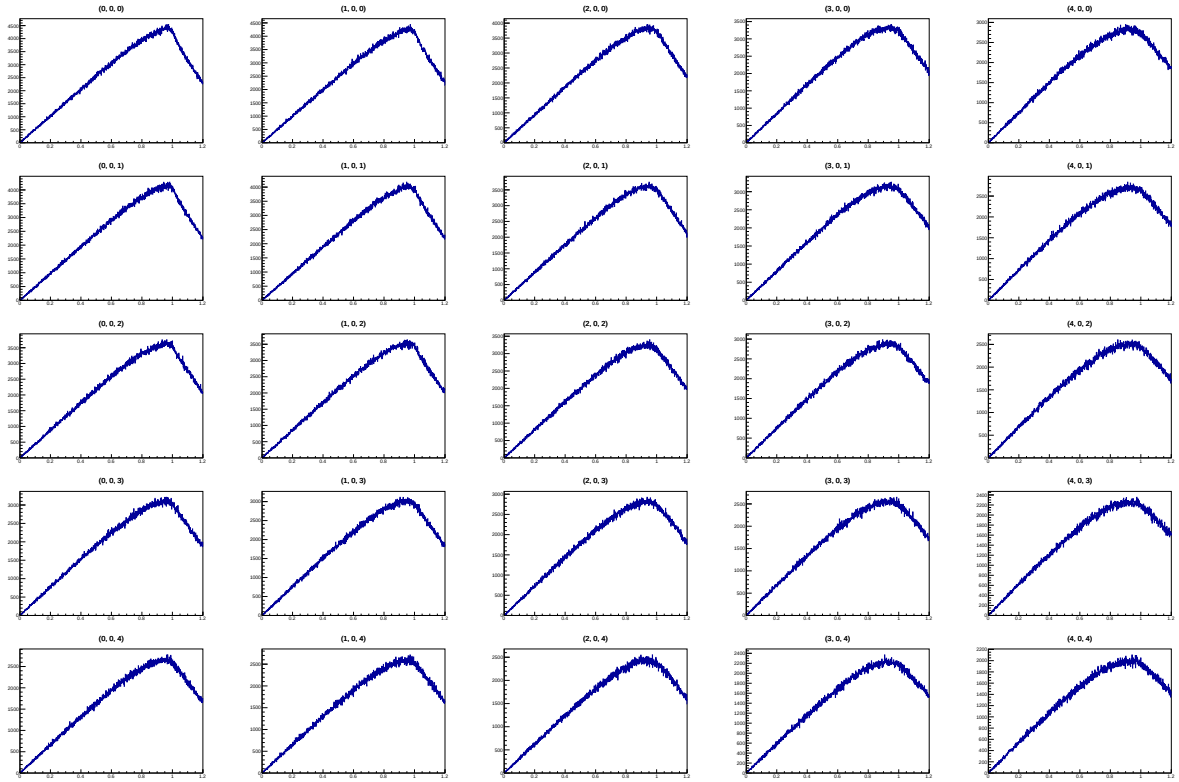


Figure A.12.: Horizontal: X axis rotation, vertical: Z axis rotation

Observations Same as for the rotation on X and Y axis.

A.4. ACH histogram comparison error metric

For all of these tests, a relief point cloud is used. The plots in the first row show the histogram comparison error metric, while second row shows the mean absolute error with the closest point criterion.

The three columns differ by the range of the transformations applied. The maximal rotation and translations are the following:

scale	translation	rotation
L	0.1	2.29°
M	0.01	0.23°
S	0.001	0.023°

So the plot for M can seen as a zoomed in version of the tenth of the plot for L, around $x = 0$. Same for S and M. However, on each plot, the curves are different random cross sections. The rotation angles are chosen so that a rotation on the X or Y axis results in an average displacement of true corresponding points by about the same about as the listed translation.

In each of the tests, a loose point cloud Q is registered to a fixed point cloud P of the same model, but with different resolution or view-point. Q is the sample point cloud for the histograms and has a higher resolution. A graphic is shown with each test that shows a close-up view of both point dispersions, along with a scale indicating the L, M and S translation lengths.

A.4.1. Same

Here Q only has a slightly higher resolution to P. The close-up view A.13 shows the density of P (black) and Q (blue), along with the maximal extent of the translations for the three columns.

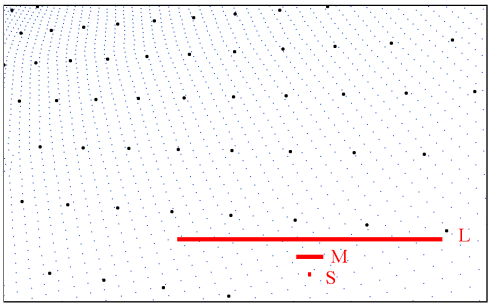
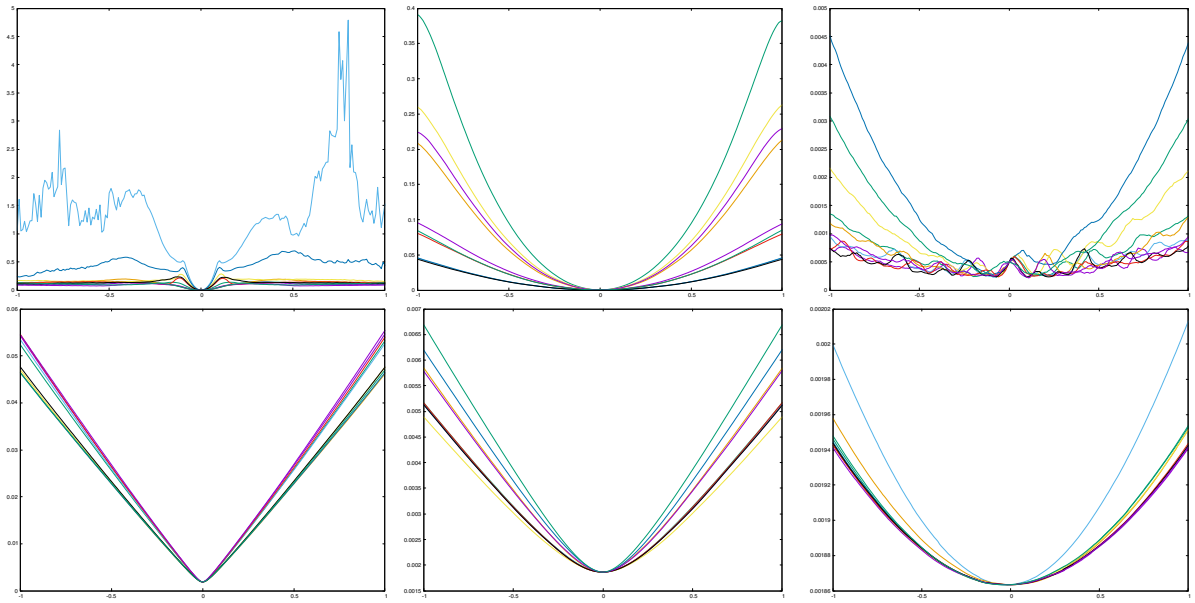


Figure A.13.: Close-up view with scale

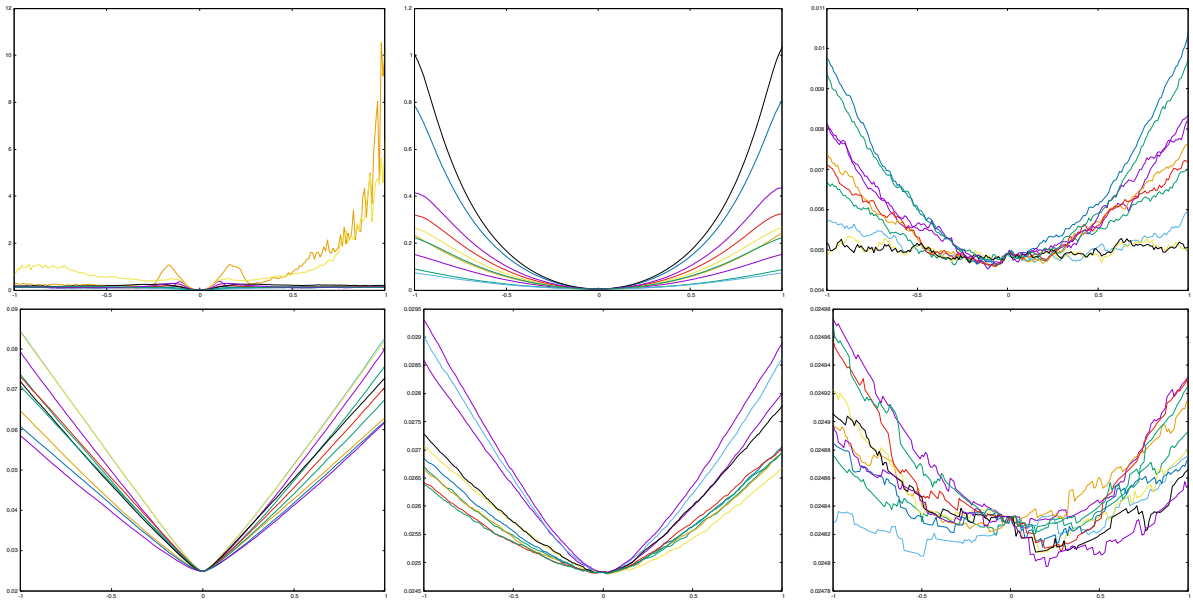


(a) L (b) M (c) S

Observations Firstly, it can be seen that the histogram comparison error metric e_χ is indeed an error metric for the registration. But no improvement compared to the mean absolute error can be seen. For large transformations, e_χ diverges, which is expected because the points q are far away from the parallelogram lattice of P, making the histogram comparison results meaningless. For the scale M, the qualities of the error metrics are about the same. On scale S, the metric e_χ degenerates, but the mean absolute error remains more stable.

A.4.2. Different view-points

Here the resolutions are the same, but P and Q are projections from different view-points, and the overlap is lower.



(a) L (b) M (c) S

Observations The results are similar than before. But on the S scale, both e_χ and the mean absolute error degenerate, as there is a lower number of point correspondences. On this example, e_χ appears to remain slightly more stable.

A.4.3. Different resolutions

Here P has a much lower resolution than Q. Again the scale is seen on the close-up view on figure A.16.

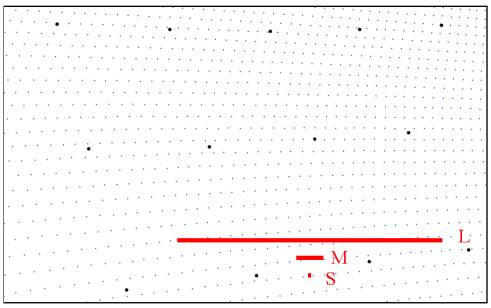
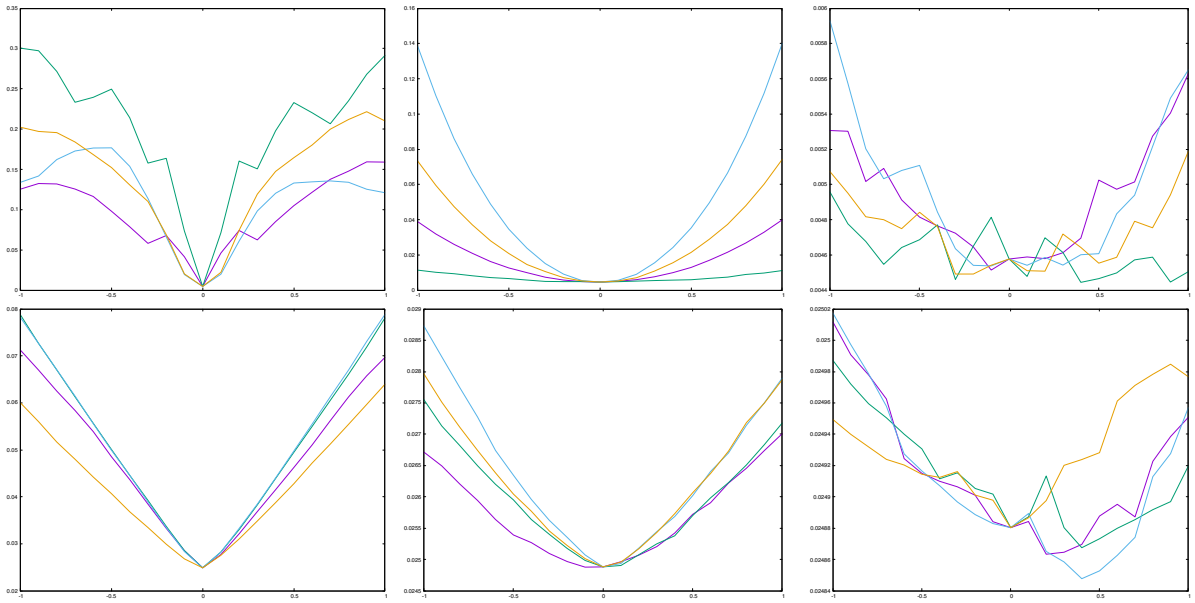


Figure A.16.: Close-up view with scale

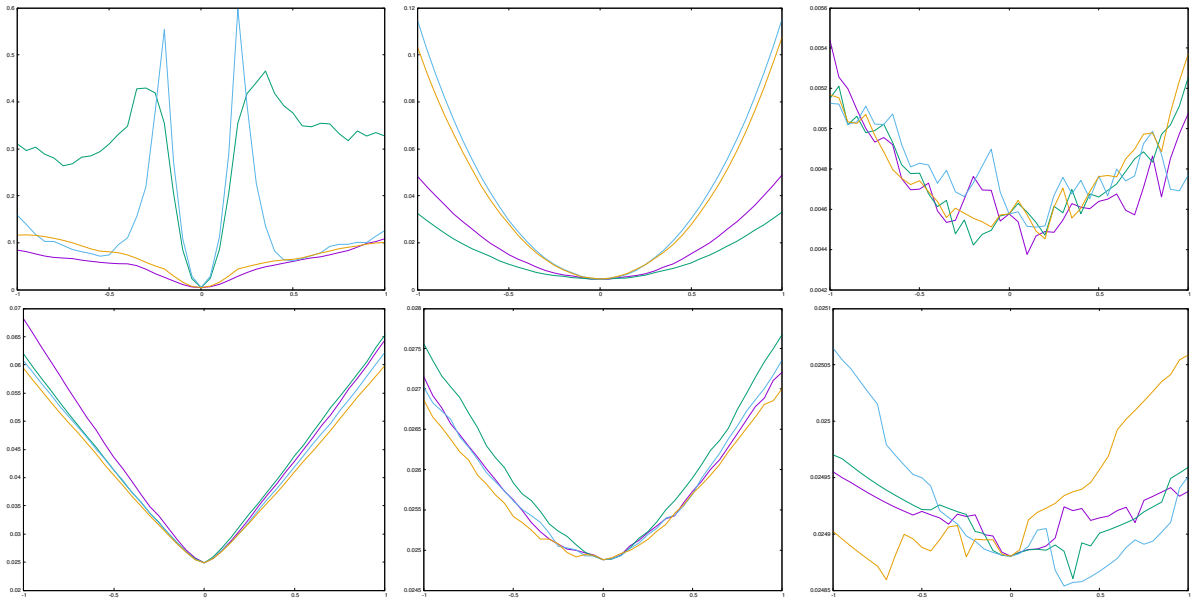


(a) L (b) M (c) S

Observations To save time, these plots were recorded less precisely.[1] The results remain similar to before, no improvement of e_χ compared to the mean absolute error metric can be seen.

A.4.3.1. Translation only

Here only the translational part of the transformation is applied.



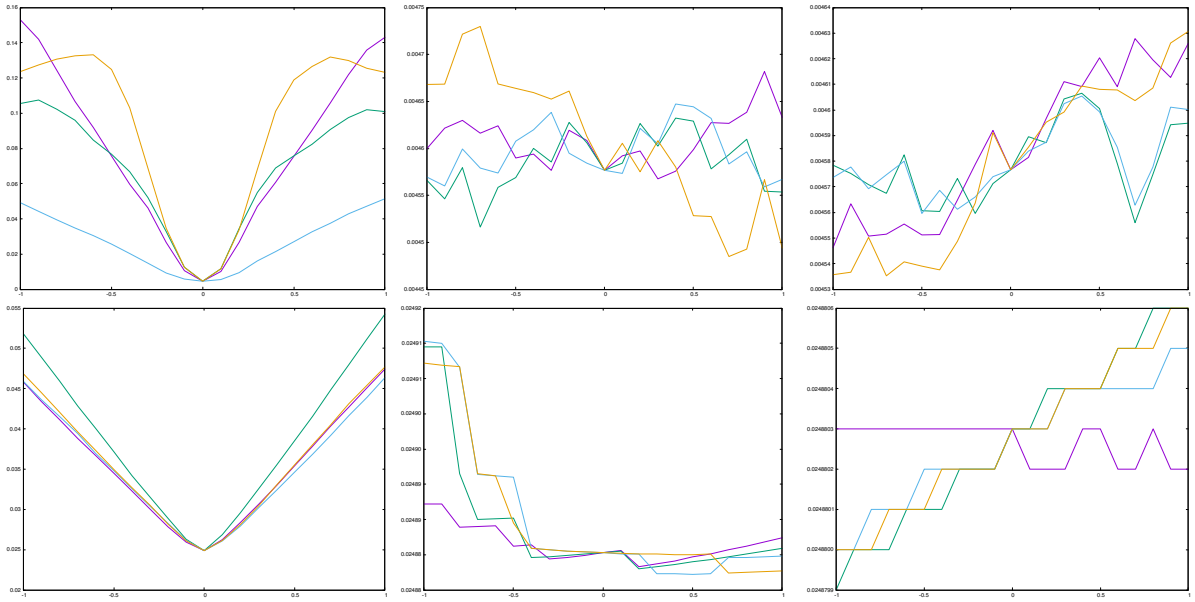
(a) L (b) M (c) S

[1] This means that the number of intermediary transformations $\mathbf{M}'$ at which the two error metrics are evaluated it lower, and there are fewer cross-section curves. The point clouds and their resolutions are the same. The plots are drawn as piecewise linear functions.

Observations Again, no improvement of e_χ compared to mean absolute error.

A.4.3.2. Rotation only

Now only the rotational part is applied.



(a) L (b) M (c) S

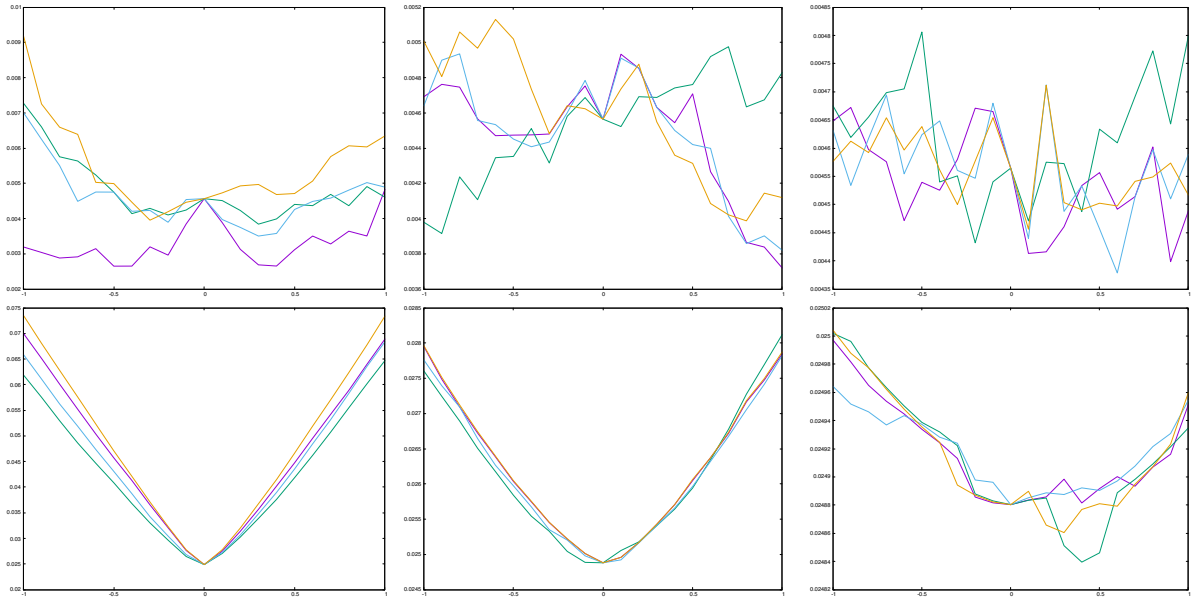
Observations Again, no improvement of e_χ compared to mean absolute error.

A.4.4. Different resolutions, rejection

Here, the same experiments are before are repeated, but the aRCH is recorded, producing the $e_{r,\chi}$ error metric.

A.4.4.1. Translation only

Here only the translational part of the transformation is applied.

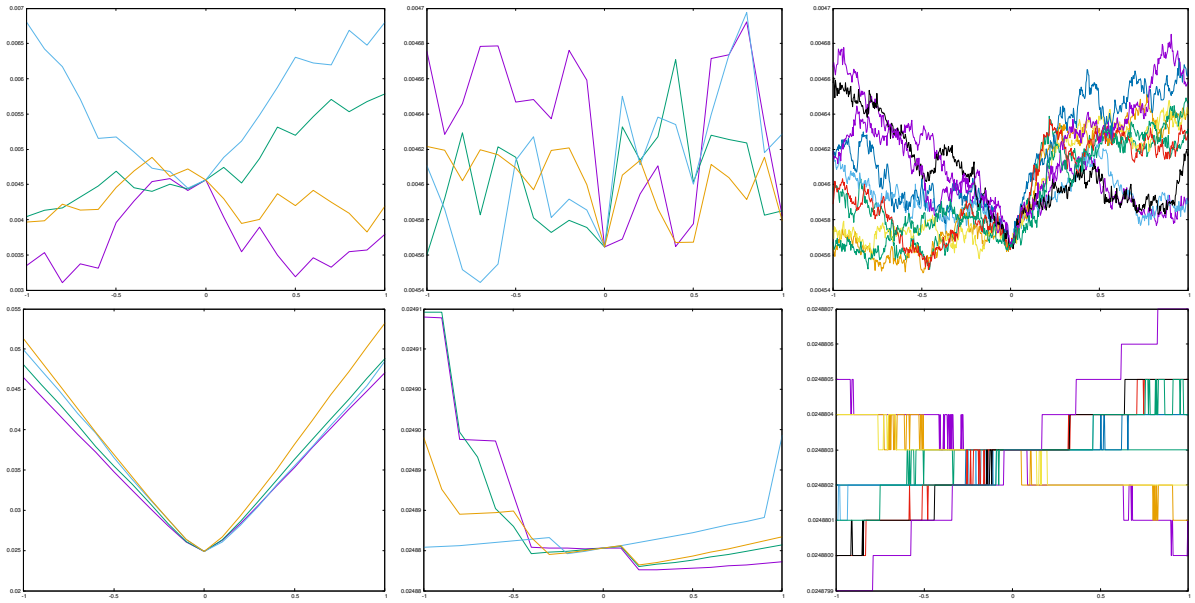


(a) L (b) M (c) S

Observations Still, no improvement of $e_{r,\chi}$ compared to mean absolute error.

A.4.4.2. Rotation only

Now only the rotational part is applied.



(a) L (b) M (c) S

Observations At the L and M scales, no results can be observed. The two metrics at the S scale were recorded with more detail, because some result can be seen here: $e_{r,\chi}$ appears to converge towards a local minimum near the true transformation $\mathbf{M}$.

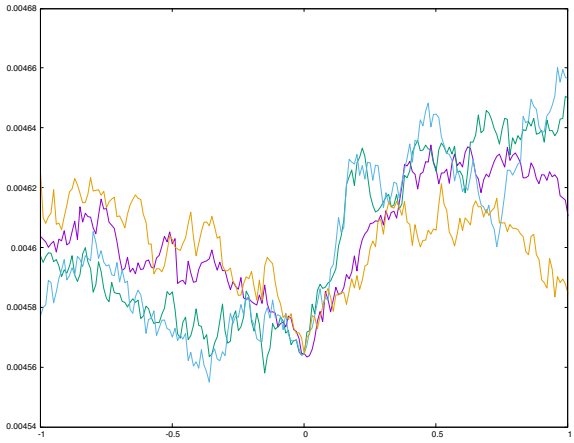
Because the curves represent values of the error metrics in cross sections of the rigid transformation space, they necessarily intersect at $x = 0$ which represents the true transformation. What is interesting about $e_{r,\chi}$ is that the curves have an inflection point at $x = 0$. The mean absolute error does not show this property.

A.4.4.3. Rotation only, different centers

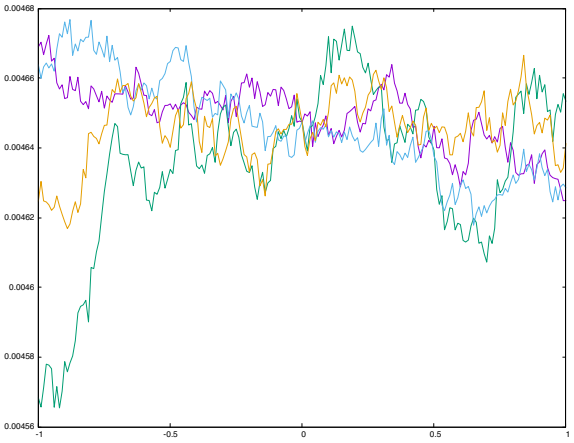
To verify this, the following two plots were recorded the same way. The first one is recorded the same way as this last plot of $e_{r,\chi}$ at scale S, and again shows the local minimum at the true transformation $\mathbf{M}$.

On the second plot, the cross sections in the rigid transformation space were instead chosen to intersect at another random transformation located near the true transformation $\mathbf{M}$. Here the curves still intersect, but no local minimum can be observed.

This result may have been by chance. However, it shows that the inflection points of the curves are not a side effect of the interpolation in the transformations space at $t = 0$. It is were so, it would also be visible on the second plot.



(a) Centered on true transformation



(b) Centered on random transformation

B. Proofs

B.1. Measures on parallelogram grid dispersion

The parallelogram dispersion of points on object surfaces occurs when sample points which are arranged in a square grid on the image plane are projected onto the object surface. The rays are perpendicular to the image plane, as a parallel projection camera is supposed for the point cloud.

This is shown on the figure B.2. Here the image space spans the XY plane. 16 pixels from the image space (in blue) are shown as the intersections of the square grid. PO is the origin point with coordinates $(0, 0)$, and $PX = (p_l, 0)$, $PY = (0, p_l)$ are the first points in X and Y direction. Their equivalents on the object surface (in light gray) are O, X, Y. $\vec{n}$ is the normal vector of this surface. The parallelogram grid on it is the projection of the square grid from the image space.

The points on the object surface form a lattice, for which the grid represents one possible basis. Others would be possible, for example one where the parallelograms have an edge joining X and Y. In general, a lattice can be defined as the set of points in $\mathbb{R}^2$

$$\{a\vec{v} + b\vec{u} : a, b \in \mathbb{Z}\} \tag{B.1}$$

Where $\vec{v}, \vec{u} \in \mathbb{R}^2$ are two vectors forming one of infinitely possible bases for this lattice. On this example $\vec{v} = OX$ and $\vec{u} = OY$ is one basis.

$l_{\min}$ is defined to be the shortest possible Euclidian distance between any two of these lattice points. On the image space, $d(PO, PX) = d(PO, PY) = p_l$ is the shortest distance, whereas $d(PX, PY) = p_l\sqrt{2} > p_l$. The example on the figure is set so that $d(X, Y) < d(O, X) < d(O, Y)$, and $d(X, Y)$ is the shortest distance. By making the surface even more oblique, it is also possible that two points that are even more far off in image space become the closest points on the projection. In general, the problem of finding $l_{\min}$ on a lattice is known as the *shortest vector problem*. For this two-dimensional case, a simple algorithm exists for finding the minimal basis consisting of the shortest and second-shortest vector. [Galbraith, 2012]

Because the lattice is invariant to a translation of the object surface plane P in space, let P pass through the origin. For any vector $\vec{p} = (p_x, p_y)^\intercal$ on the image plane, the three-dimensional coordinates of the projected vector on the object surfaces become

$$\vec{p'} = \left(p_x, p_y, -\frac{n_x\, p_x + n_y\, p_y}{n_z}\right)^\intercal \tag{B.2}$$

Its norm is

$$\|\vec{p'}\| = \sqrt{p_x^2 + p_y^2 + \frac{(n_x p_x + n_y p_y)^2}{n_z^2}} \tag{B.3}$$

The square grid points on image space have the coordinates

$$\{(x\, p_l, y\, p_l) : x, y \in \mathbb{Z}\} \tag{B.4}$$

For any segment joining two points on the grid, an identical segment exists where the first point is the origin O. So to find the shortest segment it is sufficient to compare the lengths $\|\vec{p'}\|$ where $\vec{p}$ is the

vector from the origin to another point on the image space lattice. So the shortest length would be the result of

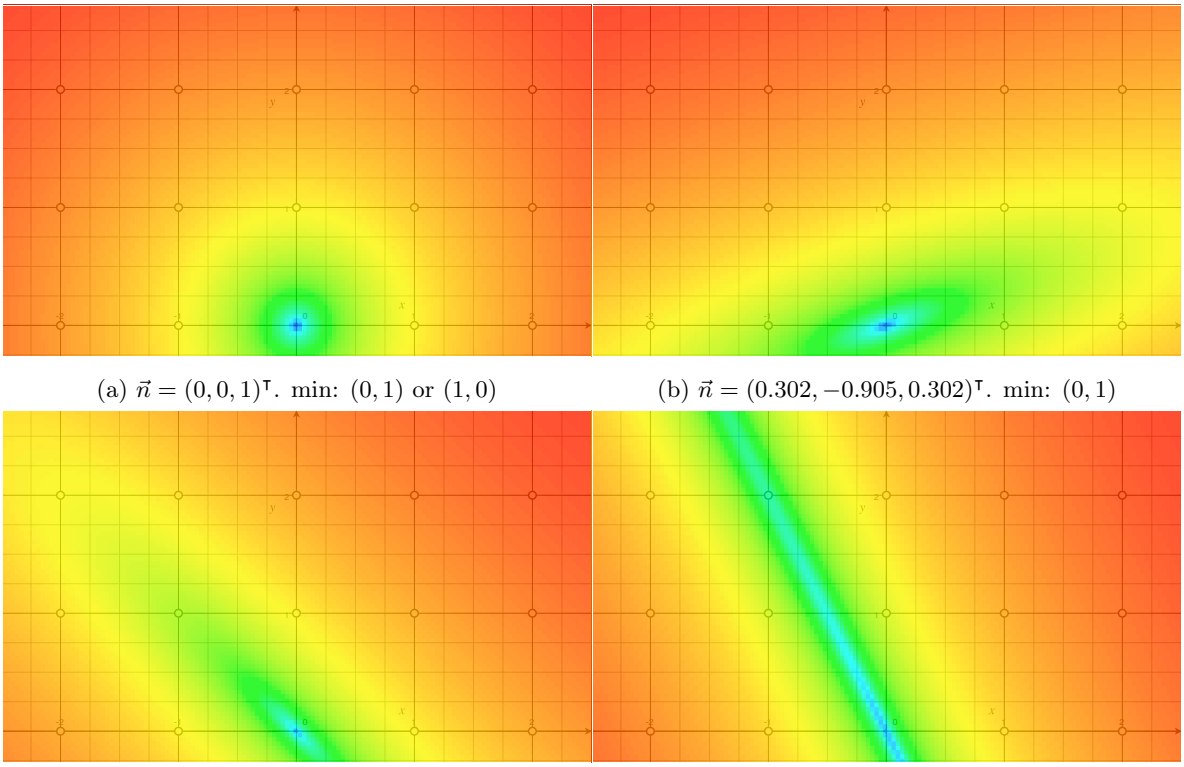
$$l_{\min} = p_l \min_{\substack{x,y\in\mathbb{Z} \\ (x,y)\neq(0,0)}} d(x,y) \quad \text{where} \quad d(x,y) = \|\vec{p'}\| \quad \text{and} \quad \vec{p} = (x,y)^\intercal \tag{B.5}$$

This function d has the property that $d(-x,-y) = d(x,y)$, and so it is sufficient to only test the values where $y \geq 0$.

B.1.1. Approximation

In most cases, when the surface is not too oblique, the minimum for (x,y) is found in $\{(0,1),(0,1),(1,1),(-1,1)\}$. This can be seen be visualizing the function $d(x,y)$ for a given $\vec{n}$. Figure B.1 shows some examples. The color gradient shows the value of the function on a logarithmic scale, where blue represents the lowest values. The goal is to find the points where x and y are integers for which the function is minimal.

By taking the minimum of $\|\vec{p'}\|$ for those four values of (x,y), the formula in 4.8 is obtained.



(a) $\vec{n} = (0,0,1)^\intercal$. min: $(0,1)$ or $(1,0)$

(b) $\vec{n} = (0.302,-0.905,0.302)^\intercal$. min: $(0,1)$

(c) $\vec{n} = (0.688,0.688,0.299)^\intercal$. min: $(-1,1)$

(d) $\vec{n} = (0.894,0.447,0.022)^\intercal$. min: $(-1,2)$

Figure B.1.: Density plot of $d(x,y)$ for different surface normal vectors $\vec{n}$

B.1.2. Threshold for obliqueness

This is an attempt to define a limit, based on the normal vector $\vec{n}$ or the object surface, for when the approximation is more likely to be accurate. That is, when $\arg\min d(x,y)$ is likely to be in the set $M = \{(0,1),(0,1),(1,1),(-1,1)\}$.

The angle α between $\vec{n}$ and a ray from the camera with directing vector $\vec{v} = (0,0,-1)^\intercal$ is obtained from their dot product: $\alpha = \arccos n_z$.

When it is 0 the function plot becomes more radial, that is, its contour lines $d(x, y) = c$ for constant c become more circular. As the angle approaches its limit $\frac{\pi}{2}$ they becomes ellipses distorted towards an angle β. When α is low the shape does not stretch out much to one specific direction, and the approximation is likely to be correct. This is seen in examples (a) and (b).

This direction β in which the surface is inclined, for which $\tan\beta = \frac{n_y}{n_x}$. When it is near a multiple of $\frac{\pi}{4}$, then again it is likely that one of the points in M gets the minimal value as the shape stretches in its direction, as seen on example (c). When β it lies between those regions, and α is high, then a point outside of M can get the minimal value, like on example (d).

This first condition is true when $|n_z| > \cos\alpha_{\max}$, taking into account that the normal vector may point in the opposite direction. The second condition is true when $|\frac{n_x}{n_y}| < \tan\beta_{\max}$ or $|\frac{n_y}{n_x}| < \tan\beta_{\max}$ or $\tan\left(\frac{\pi}{4} - \beta_{\max}\right) < |\frac{n_x}{n_y}| < \tan\left(\frac{\pi}{4} + \beta_{\max}\right)$. Using these expressions the trigonometric functions only need to be evaluated once for the constants in an implementation.

So the threshold condition on the normal vector can be written (using α and β to denote the threshold values)

$$|n_z| > \cos\alpha \bigvee \left[\left|\frac{n_x}{n_y}\right| < \tan\beta \bigwedge \left|\frac{n_y}{n_x}\right| < \tan\beta \bigwedge \tan\left(\tfrac{\pi}{4} - \beta\right) < \left|\frac{n_x}{n_y}\right| < \tan\left(\tfrac{\pi}{4} + \beta\right) \right] \tag{B.6}$$

B.1.3. Density

Evaluating $\|\vec{p'}\|$ for $(x, y) = (0, 1), (1, 0), (1, 1)$, one gets the side lengths a, b of one possible parallelogram, and one if its diagonal. A diagonal splits a parallelogram into two halves into equal area. Using Heron's formula $A = \sqrt{s\,(s-a)\,(s-b)\,(s-c)}$ where $s = \frac{a+b+c}{2}$, one gets the area of the triangle with side lengths a, b, c, and thus the parallelogram has area $2\,A$.

The points can be translated on the parallelogram grid such that each parallelogram contains one point and no points lie on the edges. This is not possible for a triangle grid. Counting one point per parallelogram area, the density is $\rho = \frac{1}{2\,A}$.

Evaluating and simplifying this expression, one obtains $\rho = \frac{|n_z|}{p_l^2}$.

B.2. Closest point histogram with random point dispersion on plane

Point clouds P and Q are two perfectly aligned planes, the points $p \in P$ are randomly dispersed on P. Q contains a large number of sample points. The probability density function for the distance of a random point $q \in Q$ to its closest neighbor in P is given by

$$f_R(r) = 2\pi\rho(P)\, r\, e^{-\pi\rho(P)\, r^2} \tag{B.7}$$

Proof. Let P be a bounded continuous surface in $\mathbb{R}^2$ of area $\mathcal{A}(P)$. For the example on the figure 4.27, 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

$$P[p_i \in A] = \frac{\mathcal{A}(A)}{\mathcal{A}(P)} = \frac{A\rho}{n} \qquad \text{and} \qquad P[p_i \notin A] = 1 - \frac{A\rho}{n} \tag{B.8}$$

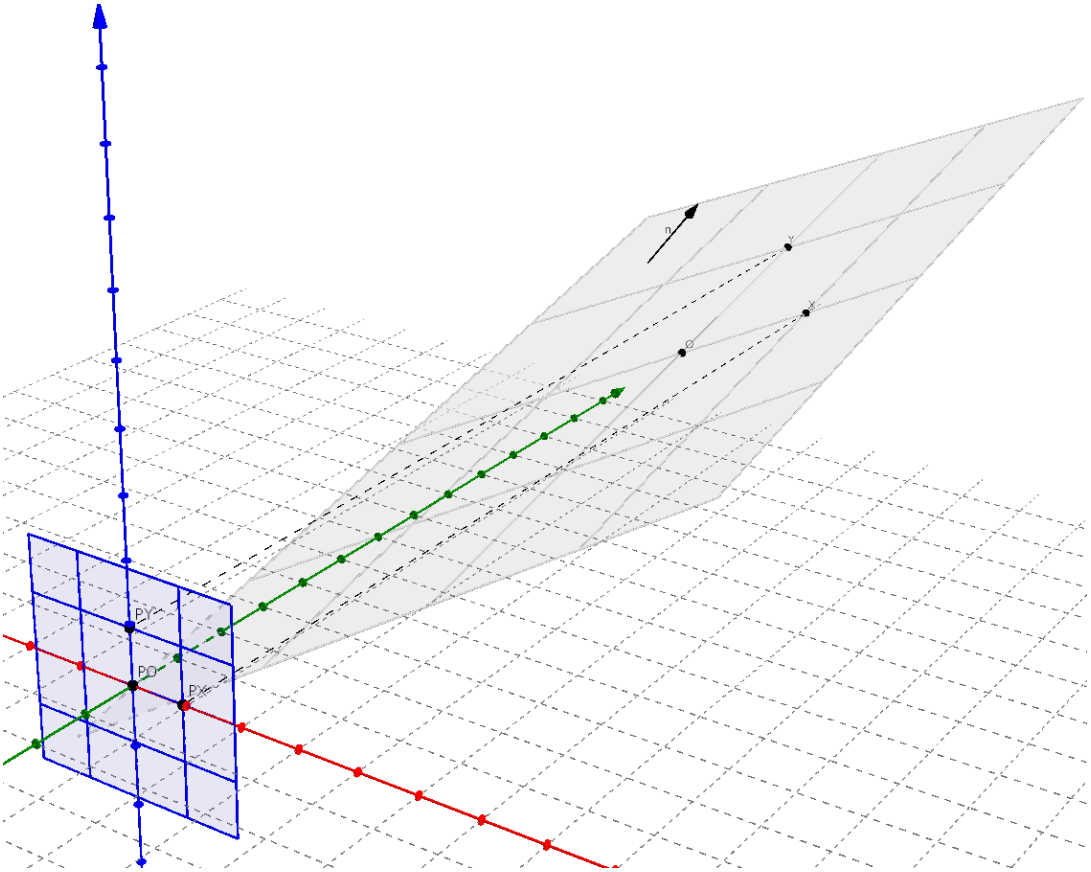


Figure B.2.: Depiction of projection of image space points to parallelogram lattice on object surface

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;

$$P[N(A) = 0] = \left(1 - \frac{A\rho}{n}\right)^n \qquad \text{and} \qquad P[N(A) \geq 1] = 1 - \left(1 - \frac{A\rho}{n}\right)^n \tag{B.9}$$

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 $\lim_{x\to\infty}\left(1 + \frac{1}{x}\right)^x = e$:

$$\begin{aligned}
\lim_{n\to\infty} P[N(A) \geq 1] &= \lim_{n\to\infty}\left[1 - \left(1 - \frac{A\rho}{n}\right)^n\right] \\
&= 1 - \lim_{n\to\infty}\left(1 - \frac{A\rho}{n}\right)^n \\
&= 1 - \lim_{n'\to\infty}\left(1 + \frac{1}{n'}\right)^{-A\rho\,n'} \\
&= 1 - \left[\lim_{n'\to\infty}\left(1 + \frac{1}{n'}\right)^{n'}\right]^{-A\rho} \\
&= 1 - e^{-A\rho}
\end{aligned} \tag{B.10}$$

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 4.27. 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:

$$\begin{aligned} f_R(r) &= \frac{\mathrm{d}}{\mathrm{d}r} P[R \leq r] = \frac{\mathrm{d}}{\mathrm{d}r} P[N(D_r) \geq 1] \\ &= \frac{\mathrm{d}}{\mathrm{d}r} \left(1 - A\rho\, e^{-A\rho}\right) \\ &= -\frac{\mathrm{d}}{\mathrm{d}r} e^{-A\rho} \\ &= -\frac{\mathrm{d}}{\mathrm{d}r} e^{-\pi\rho\, r^2} \\ &= 2\pi\rho\, r\, e^{-\pi\rho\, r^2} \end{aligned} \tag{B.11}$$

□

B.3. Variance of density with random point dispersion

The density of points dispersed on a plane is defined as $\rho = \frac{N(A)}{A}$, where $N(A)$ is the number of points in an area A.

The probability $P[N(A) = m]$ is given by

$$P[N(A) = m] = \binom{n}{m} \left(\frac{A\rho}{n}\right)^m \left(1 - \frac{A\rho}{n}\right)^{n-m} \tag{B.12}$$

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 B.8, 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

$$\left(\frac{A\rho}{n}\right)^m \left(1 - \frac{A\rho}{n}\right)^{n-m} \tag{B.13}$$

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 B.12. □

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 B.3 shows $P[N(A) = m]$ and $\bar{N}(A)$, interpolated to use real values for $N(A)$.

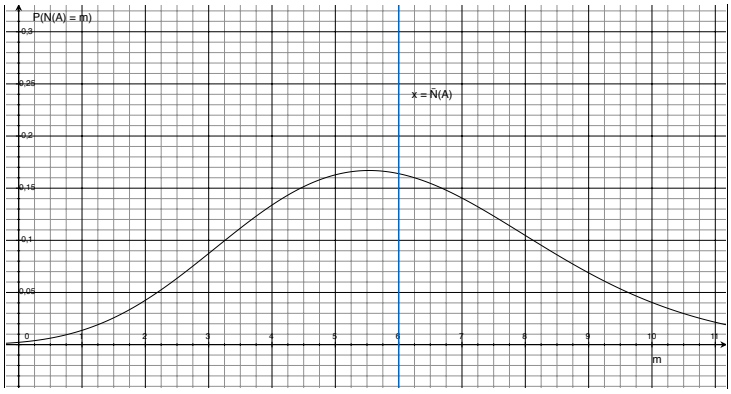


Figure B.3.: $P[N(A) = m]$ for randomly dispersed points on a plane, interpolated to real values

B.4. Closest point histogram with square grid point dispersion on plane

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 is

$$f_R(r) = \frac{1}{2l} \times \begin{cases} \frac{\pi}{4}\, r & 0 \le r \le \frac{l}{2} \\ \left(\frac{\pi}{4} - \arctan\sqrt{\left(\frac{2r}{l}\right)^2 - 1}\right) r & \frac{l}{2} < r < \frac{l}{2}\sqrt{2} \end{cases} \tag{B.14}$$

with $l = \frac{1}{\sqrt{\rho(P)}}$.

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

Figure B.4 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 $d \le 1$, 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

$$a(d) = \begin{cases} \frac{\pi}{4}\, d & 0 \le d \le 1 \\ \left(\frac{\pi}{4} - \arctan\sqrt{d^2 - 1}\right)\, d & 1 < d < \sqrt{2} \end{cases} \tag{B.15}$$

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

$$a(d) = \frac{l}{2} \times \begin{cases} \frac{\pi}{4}\, d & 0 \leq d \leq \frac{l}{2} \\ \left(\frac{\pi}{4} - \arctan \sqrt{\left(\frac{2d}{l}\right)^2 - 1} \right) d & \frac{l}{2} < d < \frac{l}{2}\sqrt{2} \end{cases} \tag{B.16}$$

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

$$f_R(d) = \frac{a(d)}{l^2} \tag{B.17}$$

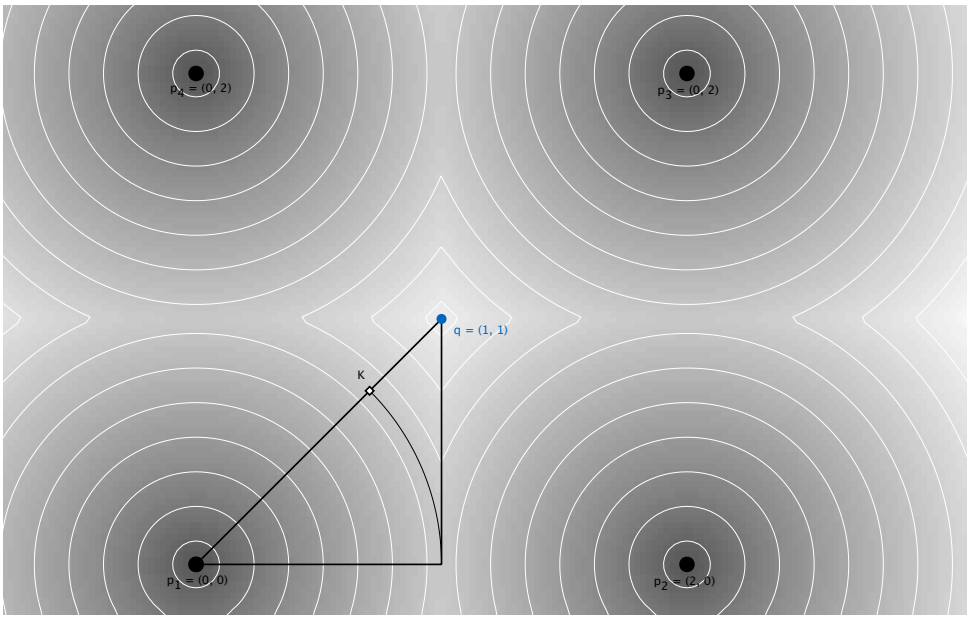


Figure B.4.: Closest points distance field with square grid distribution of P, $l = 2$

□

Glossary

4PCS 4-points congruent sets. 36, 37, 50

aCH adjusted cross distance histogram. 75, 76, 93, 97, 101

aOH adjusted own distance histogram. 70, 72–75

aRCH adjusted rejection cross distance histogram. 75–77, 97, 106

CH cross distance histogram. 66, 75

EGI extended gaussian image. 38, 39

GPU Graphics Processing Unit. 85

ICP iterative closest point. 5, 29–34, 40, 50, 52, 53, 55, 77, 79, 80, 83, 84, 88–92

ICP normal distribution transform. 35, 36

JIT just-in-time. 87

Lidar Light Detection and Ranging. 10

OH own distance histogram. 66, 72–74

Radar Radio Detection and Ranging. 10

RANSAC random sample consensus. 24, 25, 36, 50, 57

ROI region of interest. 70, 72, 74, 77

Sonar Sound Navigation and Ranging. 10

STL Standard Template Library. 83

Bibliography

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

[Aiger *et al.* , 2008] Aiger, Dror, Mitra, Niloy J, & Cohen-Or, Daniel. 2008. 4-Points Congruent Sets for Robust Pairwise Surface Registration. *ACM Transactions on Graphics*, **27**(3).

[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.

[Biber, 2003] Biber, Peter. 2003. The Normal Distributions Transform: A New Approach to Laser Scan Matching.

[Bouaziz *et al.* , 2013] Bouaziz, Sofien, Tagliasacchi, Andrea, & Pauly, Mark. 2013. Sparse Iterative Closest Point. *Eurographics Symposium on Geometry Processing*, **32**(5).

[Dold, 2005] Dold, Christoph. 2005. Extended Gaussian Images for the Registration of Terrestial Scan Data. Institute of Cartography and Geoinformatics, University of Hannover, Germany.

[Dold & Brenner, 2006] Dold, Christoph, & Brenner, Claus. 2006. Registration of Terrestrial Laser Scanning Data using Planar Patches and Image Data. *Pages 25–27 of: International Archives of Photogrammetry and Remote Sensing.* Leibniz University of Hannover.

[Dold *et al.* , 2007] Dold, Christoph, Ripperda, Nora, & Brenner, Claus. 2007. Vergleich verschiedener Methoden zur automatischen Registrierung von terrestrischen Laserscandaten. *Beiträge der Oldenburger 3D-Tage*, 196.

[Eberly, 1999] Eberly, David. 1999 (July). Least Squares Fitting of Data. Magic Software, Inc.

[Fischler & Bolles, 1980] Fischler, Martin A, & Bolles, Robert C. 1980 (March). *Random Sample Consensus: A paradigm for model fitting with applications to image analysis and automated cartography.* Tech. rept. Artificial Intelligence Center, SRI International, Menlo Park, California.

[Galbraith, 2012] Galbraith, Steven. 2012. *Mathematics of Public Key Cryptography.* Cambridge University Press.

[Grussenmeyer *et al.* , 2012] Grussenmeyer, P., Alby, E., Landes, T., Koehl, M., Guillemin, S., Hullo, J.-F., Assali, P., & Smigiel, E. 2012. Recording Approach Of Heritage Sites Based On Merging Point Clouds From High Resolution Photogrammetry And Terrestrial Laser Scanning. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, **XXXIX-B5**, 553–558.

[Horn, 1984] Horn, Berthold K.P. 1984. Extended Gaussian Images. *Proceesings of the IEEE*, **72**(12), 1671–1686.

[Horn, 1987] Horn, Berthold K.P. 1987 (April). Closed-form solution of absolute orientation using unit quaternions. *Page 629 of: Journal of the Optical Society of America A*, vol. 4. Department of Electrical Engineering, University of Hawaii at Manoa.

[Keiner, 2011] Keiner, Christopher. 2011. *Bildregistrierung von 3D-Laserscans und Bildern für Wadi Sura.* M.Phil. thesis, Fachbereich Mathematik und Informatik der Freien Universität Berlin, April.

[Kersten *et al.* , 2006] Kersten, Thomas, Przybilla, Heinz-Jürgen, & Lindstaedt, Maren. 2006. Integration, Fusion und Kombination von terrestrischen Laserscannerdaten und digitalen Bildern. November.

[Kim & Lee, 2009] Kim, Jaewoong, & Lee, Sukhan. 2009. Fast Neighbor Cells Finding Method for Multiple Octree Representation. *2009 IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA)*, December, 540–545.

[Koch, 2008] Koch, K. R. 2008. Fitting Free-Form Surfaces to Laserscan Data by NURBS. *Allgemeine Vermessungs-Nachrichten*, **4**(August), 134–140.

[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.

[Lichtenstein, 2011] Lichtenstein, Maria. 2011. *Strukturbasierte Registrierung von Punktwolken unter Verwendung von Bild- und Laserscannerdaten.* Ph.D. thesis, Rheinisch-Westfälische Technische Hochschule Aachen.

[Liu, 2008] Liu, Yonghuai. 2008. Constraints for closest point finding. *Pattern Recognition Letters*, **29**, 841–851.

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

[Lu & Milios, 1997] Lu, Feng, & Milios, Evangelos. 1997. Robot Pose Estimation in Unknown Environments by Matching 2D Range Scans. *Journal of Intelligent and Robotic Systems*, **18**(3), 249–275.

[Magnusson, 2013] Magnusson, Martin. 2013. *The Three-Dimensional Normal-Distributions Transform - an Efficient Representation for Registration, Surface Analysis, and Loop Detection.* Ph.D. thesis, School of Science and Technology - Örebro university.

[Magnusson *et al.* , 2007] Magnusson, Martin, Lilienthal, Achim, & Duckett, Tom. 2007. Scan Registration for Autonomous Mining Vehicles Using 3D-NDT. *Journal of Field Robotics*, **24**(10), 804–827.

[Makadia *et al.* , 2006] Makadia, Ameesh, IV, Alexander Patterson, & Daniilidis, Kostas. 2006. Fully Automatic Registration of 3D Point Clouds. *Pages 1297–1304 of: CVPR '06 Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1. University of Pennsylvania.

[Matiukas & Miniotas, 2011] Matiukas, V., & Miniotas, D. 2011. Point Cloud Merging for Complete 3D Surface Reconstruction. ISSN 1392–1215 ELECTRONICS AND ELECTRICAL ENGINEERING, no. 7. Department of Electronic Systems, Vilnius Gediminas Technical University.

[Mellado *et al.* , 2014] Mellado, Nicolas, Aiger, Dror, & Mitra, Niloy J. 2014. SUPER 4PCS - Fast Global Pointcloud Registration via Smart Indexing. *Eurographics Symposium on Geometry Processing*, **33**(5).

[Montesano *et al.* , 2005] Montesano, Luis, Minguez, Javier, & Montano, Luis. 2005. Probabilistic Scan Matching for Motion Estimation in Unstructured Environments. *Pro- ceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS).*

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

[Sankaranarayanan *et al.* , 2007] Sankaranarayanan, Jagan, Samet, Hanan, & Varshney, Amitabh. 2007. A fast all nearest neighbor algorithm for applications involving large point-clouds. *Computers Graphics*, **31**, 157–174.

[Saxena & Singh, 2014] Saxena, Siddharth, & Singh, Rajeev Kumar. 2014. A Survey of Recent and Classical Image Registration Methods. *International Journal of Signal Processing, Image Processing and Pattern Recognition*, **7**(4), 167–176.

[Schönemann, 1966] Schönemann, Peter H. 1966. A generalized solution of the orthogonal procrustes problem. *Phychometrika*, **31**(1), 1–10.

[Schönemann & Caroll, 1970] Schönemann, Peter H, & Caroll, Robert M. 1970. Fitting one matrix to another using choice of a central dilation and a rigid motion. *Phychometrika*, **35**(2), 245–255.

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

[Toldo *et al.* , 2010] Toldo, Roberto, Beinat, Alberto, & Crosilla, Fabio. 2010. Global registration of multiple point clouds embedding the Generalized Procrustes Analysis into an ICP framework. 3DPVT 2010 Conference.

[Tournas & Tsakiri, 2009] Tournas, E., & Tsakiri, M. 2009. Automatic 3D Point Cloud Registration for Cultural Heritage Documentation. *Laser scanning 2009, IAPRS*, **XXXVIII**(September), 189–194.

[Tuytelaars & Mikolajczyk, 2007] Tuytelaars, Tinne, & Mikolajczyk, Krystian. 2007. Local Invariant Feature Detectors: A Survey. *Foundations and Trends in Computer Graphics and Vision*, **3**(3).

[Yang *et al.* , 2013] Yang, Jiaolong, Li, Hongdong, & Jia, Yunde. 2013. Go-ICP: Solving 3D Registration Efficiently and Globally Optimally. *IEEE International Conference on Computer Vision (ICCV)*, 1457–1464.