Aligning surfaces in 3D space

In the last chapter we learned how to reconstruct meshes from point clouds. Reconstructing shapes with an optical sensor has the drawback that, most of the time, one camera can not capture the entire object at once, since part of the object are occluded from certain viewing angles. This necessitates capturing multiple views of the object from different angles. If we do not have perfect certainty about the relative locations of our cameras to each other, it is difficult to exactly reconstruct the shape (Figure 1). Such uncertainty is more common than you might think – it is hard to capture perfect relative location data, for example, from phones, since geolocation systems do not have fine enough resolution, and phone accelerometers are often not reliable enough.

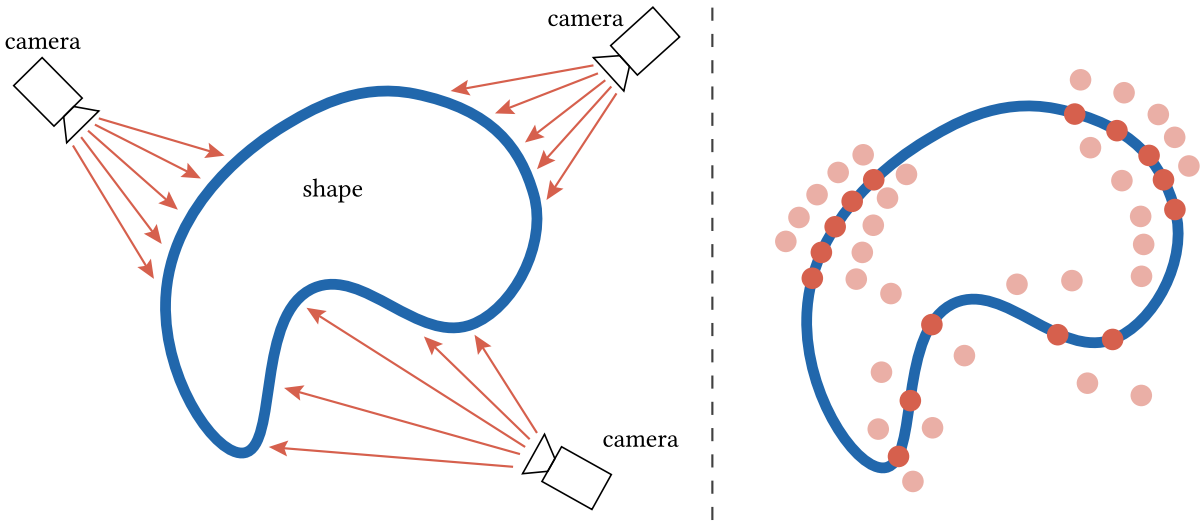

Figure 1: A schematic overview of the laser scanning process *(left)*. We do not always have exact location information for the captured data, leading to uncertainty in the captured points *(right)*.

The solution to this problem is to have scans from many viewing angles so that the scans partially overlap each other, and can then be aligned, forming a coherent shape (Figure 2).

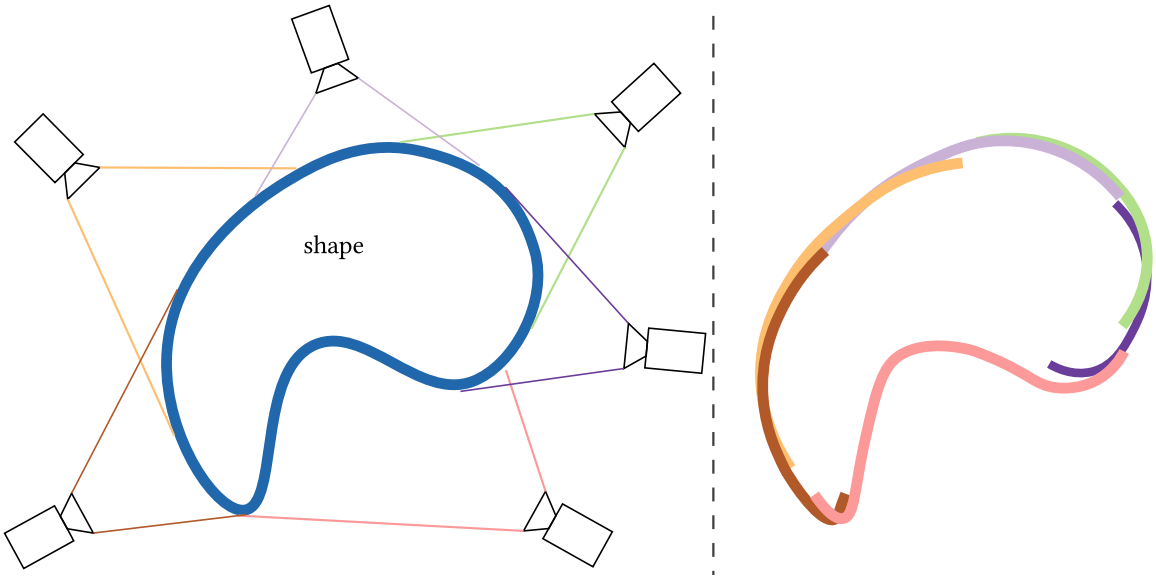

Figure 2: If many cameras capture the shape from many different angles, the overlapping parts of the shape can be aligned to each other to reconstruct the final surface.

Another scenario where the alignment problem is relevant is if we scan something whose base geometry we already know, and we need to align the scanned surface to the base geometry. This is

relevant, for example, in motion capturing, where we scan points of an object we already know. In this chapter we will learn how to align two shapes to each other based on their geometry.

Types of deformations

Before we can compute an alignment, let us define what we are even looking for in our alignment problem. In general, we have some source shape and some target shape, and we want to put them in the same place in space. Let S be our source surface, and T our target surface. We are looking for a function

$$f : S \to \mathbb{R}^d \tag{1}$$

such that $f(S)$ is aligned with T in some way.

We will return to the question of what it means for one shape to be aligned with another, and focus on the function f for now. What kind of transformations do we allow?

Rigid transformations

We make a crucial assumption on the camera:[1] The camera has the same distortion no matter what distance the object is from the camera, and no matter the position of the object in space.[2] This means that f is a *rigid map* (see Figure 3).

Definition 1: Rigid map

A function $f : \mathbb{R}^d \to \mathbb{R}^d$ is a *rigid map* if, for any two points $x, y \in \mathbb{R}^d$,

$$\|f(x) - f(y)\| = \|x - y\| \tag{2}$$

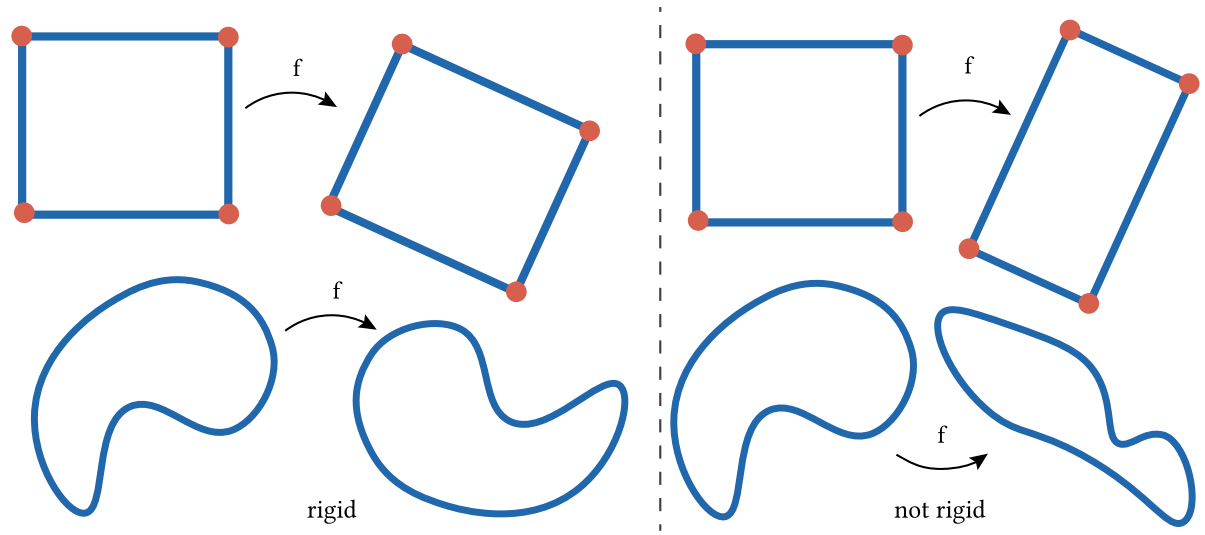

Figure 3: Examples of rigid and nonrigid maps. Rigid maps preserve the pairwise distances between any two points.

Every rigid map can always be written as a combination of rotation and translation:

[1] You decide how realistic this is in practice!

[2] Or, at least, we can remove the effects of distance and position with preprocessing.

Theorem 1: Rigid map

A rigid function $f : \mathbb{R}^d \to \mathbb{R}^d$ can always be written as the combination of a
- orthogonal matrix $R \in O(d)$; and a
- translation vector $v \in \mathbb{R}^d$

such that $f(x) = Rx + v$.

f preserves orientation if and only if $R \in SO(d)$, i.e., R is a rotation matrix.

Let us try to prove one direction of Theorem 1: maps consisting of orthogonal maps and translations are rigid. Let $R \in O(d), v \in \mathbb{R}^d$. We will show that $f(x) = Rx + v$ is a rigid map. To that end, let $x, y \in \mathbb{R}^d$ be arbitrary points. Then:

$$\begin{aligned}\|f(x) - f(y)\|^2 &= \|Rx + v - (Ry + v)\|^2 = \|R(x - y)\|^2 \\ &= (x - y)^T R^T R(x - y)\end{aligned} \tag{3}$$

Since R is an orthogonal matrix, $R^T R$ is the identity, and thus

$$\|f(x) - f(y)\|^2 = (x - y)^T (x - y) = \|x - y\|^2. \tag{4}$$

This proves one direction of the theorem. ■

The other direction is a little bit more tricky, and we will not prove it in this chapter.

Using the language of rigid maps, our problem now has reduced to the following: find a rotation matrix $R \in SO(d)$ and a translation vector $v \in \mathbb{R}^d$ such that $RS + v$ aligns with T.

Measuring distances between surfaces

Now that we have formalized our transformation, how do we formalize alignment? What does it mean for $RS + v$ to align with T? This is actually quite easy to write as a relation of two sets – we want the transformed surface S to be a subset of T:

$$RS + v \subseteq T \tag{5}$$

Achieving (5) is, however, quite difficult. We want to eventually solve an optimization problem, which means that we want to write down a distance that we optimize, and that distance is zero when we have achieved perfect alignment. How does one turn (5) into a distance?

The goal of this section is to compute the distance between two surfaces X and Y, which we will call $d(X, Y)$. How does one compute the distance between two surfaces? If X and Y were just one point, then this would be easy: For $X = \{x\}, Y = \{y\}$,

$$d(X, Y) = \|x - y\|. \tag{6}$$

Of course, X and Y do not consist of just one point.

The Hausdorff distance

Let us look at another slightly more complicated special case. What if X is still only one point, $X = \{x\}$, but Y is now many points? We can then define the distance between X and Y as the distance between x and the closest point to it on Y:

$$d(X, Y) = \inf_{y \in Y} \|x - y\| \tag{7}$$

This still is not the general solution but we want, but the minimization operation from (7) will be a useful tool for later:

Definition 2: Surface projection

The projection of x onto the surface Y is denoted by $P_Y(x)$ and defined as the point $y \in Y$ that minimizes the expression

$$\inf_{y \in Y} \|x - y\|, \tag{8}$$

i.e.,

$$\|x - P_Y(x)\| = \inf_{y \in Y} \|x - y\|. \tag{9}$$

Something to think about 1: Computing the surface projection

Computing the surface projection $P_Y(x)$ is not a trivial task, but if your target Y is a triangle mesh, then the task can be broken down into computing the distance between x and any triangle. This is now a problem solvable with algebra and trigonometry – try it!

We can now further generalize this to surface-to-surface distances. While it would be tempting to define $d(X, Y) = \inf_{x \in X} \|x - P_Y(X)\|$, this would not be a particularly useful definition – even if only one point in X coincided with one point in Y, this would already be 0. Instead, we want the distance of *every* point in X to Y to be 0.

Definition 3: Hausdorff distance

The (directed) Hausdorff distance from X to Y is defined as

$$d_{\vec{H}}(X, Y) = \sup_{x \in X} \|x - P_Y(x)\| \tag{10}$$

This *Hausdorff distance* is now a useful distance measure between two surfaces that is 0 if $X \subseteq Y$, and nonzero otherwise.

Undirected Hausdorff distance

The Hausdorff distance of Definition 3 only works in one direction: It measures whether X is contained in Y, but even if X is very tiny and not at all of the same shape as Y, the directed Hausdorff distance will be zero. This is what we want for the surface alignment problem, since we want to match a small patch to a larger surface. But it is not a good measure of equality of two surfaces.

If one wants a truly symmetric Hausdorff distance that measures whether two surfaces are absolutely identical up to rigid maps (which we do not want for the alignment problem), the directed Hausdorff distance is

$$d_H(X, Y) = \max(d_{\vec{H}}(X, Y), d_{\vec{H}}(Y, X)). \tag{11}$$

Integrated closest point distance

The Hausdorff distance of Definition 3 is a good distance measure to verify whether we have solved the alignment problem, but it is not a particularly good distance to optimize. This is because $d_H(X, Y)$ only measures the difference between the *worst* two points on X and Y - if we optimize it with a gradient-based method, only one point would move.

Instead, we want a measure of distance where all points in X that are not also in Y contribute to the distance measure. We do this by turning the supremum into an integral, thus *diffusing* the distance contribution over the whole surface X.

Definition 4: Integrated closest point distance

The integrated closest point distance from X to Y is defined as

$$d_{\vec{C}}(X,Y)^2 = \int_X \|x - P_Y(x)\|^2 dx \tag{12}$$

Every point $x \in X$ now contributes to the distance $d_{\vec{C}}(X,Y)$ in proportion to how far it is from $P_Y(x)$. This is the distance we want to minimize to solve our alignment problem.

Optimizing distances between surfaces

We now know how to measure distances between two surfaces. So, in order to align a source surface S with a target surface T, we wish to find a rotation matrix R and a translation vector v that minimize the integrated point distance

$$\underset{R,v}{\operatorname{argmin}}\, d_{\vec{C}}(RS+v,T) = \underset{R,v}{\operatorname{argmin}} \int_X \|Rx + v - P_T(Rx+v)\|^2 dx. \tag{13}$$

Iterative closest point

Optimizing (13) is harder than it looks. Even though (13) contains a quadratic objective, we can not just compute the gradient with respect to R and v, since the space of all rotation matrices is not a linear subspace - we need to enforce the rotation constraint somewhat. We will do this with the *iterative closest point* (ICP) algorithm [1,2].

We first tackle the question of how to compute the integral in (13). We know how to compute $P_T(x)$, but it is hard to compute it for every point $x \in X$ required for the integral dx. An easy way around this is to just sample lots of points $x_1, ..., x_n \in X$ and then replacing the integral with a sum over all these points:

$$\underset{R,v}{\operatorname{argmin}} \sum_{i=1}^{n} \|Rx_i + v - P_T(G(x_i))\|^2 dx, \tag{14}$$

where G is our current guess for the best transformed source mesh (which we will try to improve).

With this discretization of the integral in hand, we can formulate the ICP algorithm. It works by keeping around a best guess for the optimal rigid transformation R, v, transforming the source surface S according to this transformation, and then trying to improve the rigid transformation to decrease the energy (14). In pseudocode form:

```
ICP(V_S, F_S, V_T, F_T)
  V_guess = V_S
  F_guess = F_S
  repeat until energy minimized:
    X = sample_points_on(V_guess, F_guess)
    PX = project_points_onto_mesh(X, V_T, F_T)
    R, v = optimize_discretized_distance(X, V_guess, F_guess, V_T, F_T)
    V_guess = R * V_S + v
  return V_guess, F_guess
```

It remains to solve the actual optimization with respect to R and v, `optimize_discretized_distance`, from (14). Let's reformulate this with a fixed target point:

$$\underset{R,v}{\operatorname{argmin}} \sum_{i=1}^{n} \|Rx_i + v - p_i\|^2 dx, \tag{15}$$

where p_i is the point x_i on our best-guess surface of the current iteration.

Let's rewrite everything as large matrices, where

$$X = \begin{pmatrix} x_1^T \\ \cdots \\ x_n^t \end{pmatrix}, \quad P = \begin{pmatrix} p_1^T \\ \cdots \\ p_n^t \end{pmatrix}. \tag{16}$$

Then the optimization problem is

$$\underset{R,v}{\operatorname{argmin}} \left\| RX^T + v\mathbf{1}^T - P^T \right\|_F^2, \tag{17}$$

where $\|\cdot\|_F^2$ is the squared *Frobenius norm*, the sum of all squared elements of a matrix. Since v is not constrained, we find the optimal v by taking a simple gradient:

$$0 = \mathbf{1}^T\mathbf{1}v + RX^T\mathbf{1} - P^T\mathbf{1}. \tag{18}$$

And thus:

$$v = \frac{P^T\mathbf{1} - RX^T\mathbf{1}}{\mathbf{1}^T\mathbf{1}}. \tag{19}$$

This is simply equivalent to averaging $p_i - Rx_i$. Let's define these averages as $\bar{x} = \frac{1}{n}\sum_1^n x_i$, $\bar{p} = \frac{1}{n}\sum_1^n p_i$. Then $v = \bar{p} - R\bar{x}$. Thus we can update our optimization problem (17) to remove the translation term entirely:

$$\underset{R}{\operatorname{argmin}} \left\| RX^T + (\bar{p} - R\bar{x})\mathbf{1}^T - P^T \right\|_F^2, \tag{20}$$

Let's now try to solve for the rotation matrix R. This is much harder. We will start by trying to further simplify (20).

$$\left\| RX^T + (\bar{p} - R\bar{x})\mathbf{1}^T - P^T \right\|_F^2 = \left\| R(X^T - \bar{x}\mathbf{1}^T) - (P^T - \bar{p}\mathbf{1}^T) \right\|^2 = \left\| R\bar{X}^T - \bar{P}^T \right\|^2 \tag{21}$$

for appropriate matrices $\bar{X}$, $\bar{P}$. Since R is orthogonal, $R^TR = \mathrm{Id}$, and we can continue to simplify:

$$\begin{aligned}
\underset{R}{\operatorname{argmin}} \left\| R\bar{X}^T - \bar{P}^T \right\|^2 &= \underset{R}{\operatorname{argmin}} \operatorname{tr}\Big(\big(\bar{X}R^T - \bar{P}\big)\big(R\bar{X}^T - \bar{P}^T\big) \Big) \\
&= \underset{R}{\operatorname{argmin}} \operatorname{tr}\Big(\big(R\bar{X}^T - \bar{P}^T\big)\big(\bar{X}R^T - \bar{P}\big) \Big) \\
&= \underset{R}{\operatorname{argmin}} \operatorname{tr} R\bar{X}^T\bar{X}R^T - \operatorname{tr}\bar{P}^T\bar{X}R^T - \operatorname{tr} R\bar{X}^T\bar{P} + \operatorname{tr}\bar{P}^T\bar{P} \\
&= \underset{R}{\operatorname{argmin}} \operatorname{tr} R\bar{X}^T\bar{X}R^T - \operatorname{tr}\bar{P}^T\bar{X}R^T - \operatorname{tr} R\bar{X}^T\bar{P} + \operatorname{tr}\bar{P}^T\bar{P} \\
&= \underset{R}{\operatorname{argmin}} \Big(-\operatorname{tr} R\bar{X}^T\bar{P} \Big) = \underset{R}{\operatorname{argmin}} \left\| R - \bar{P}^T\bar{X} \right\|^2,
\end{aligned} \tag{22}$$

where trace is the trace operator, the sum of diagonal elements of a matrix.

This problem is called the Procrustes problem (finding the closest orthogonal / rotational approximation to a matrix), and there are various standard ways of solving it.[3] The only thing you need to know for now is that it can be solved using the *singular value decomposition* (SVD). To solve the procrustes problem $\operatorname{argmin}_R \|R - M\|^2$, we employ the SVD to transform M into two orthogonal matrices and a diagonal matrix:

[3] Look it up on Wikipedia for more fascinating details and a history.

$$M = U\Sigma V^T. \tag{23}$$

Then the optimal R must be:

$$R = UV^T. \tag{24}$$

Note that this only guarantees that R is orthogonal, it does not guarantee that R is a rotation matrix – for that we need to ensure that its determinant is 1. If it is not 1, we ca achieve this by flipping the sign of one of the columns of U.

With the Procrustes problem solved, we can now implement the ICP algorithm!

Bibliography

[1] Paul J. Besl and Neil D. McKay. 1992. A Method for Registration of 3-D Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14, 2 (1992).

[2] Sofien Bouaziz. 2015. Realtime Face Tracking and Animation (PhD thesis, EPFL).