

Analysis of 3D Shapes (IN2238)

Frank R. Schmidt
Matthias Vestner

Summer Semester 2016

4. Feature Representation and Linear Assignment Problem

Curves

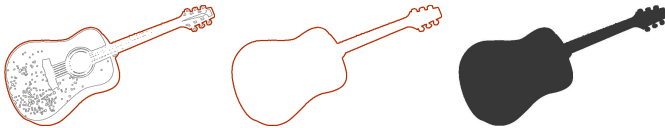
2D Objects



A 2D object is an open set $O \subset \mathbb{R}^2$ such that $B := \partial O$ is a submanifold of dimension 1.

A result from differential geometry is that a 1D manifold is either homeomorphic to \mathbb{S}^1 or to \mathbb{R} . Since we want to represent an object in a compact image domain $\Omega \subset \mathbb{R}^2$, we can assume that B is a collection of closed contours (each homeomorphic to \mathbb{S}^1).

Outer Contour



Assuming that $B = \partial O = \bigcup_{i=1}^k C_i$ is the union of disjoint contours C_i , it is often enough to consider only the **outer contour** of B .

This is equivalent of considering a slightly different object $O' \supset O$ that perceptually is very similar to the original object O .

In conclusion, we assume that $C = \partial O$ is a connected submanifold of dimension 1 that is diffeomorphic to \mathbb{S}^1 . That means we have

$$c: \mathbb{S}^1 \rightarrow \mathbb{R}^2 \quad \|\dot{c}(t)\| \neq 0 \quad (\forall t \in \mathbb{S}^1).$$

Contour Length

Given a curve $c: \mathbb{S}^1 \rightarrow \mathbb{R}^2$, its **length** is

$$\begin{aligned} \text{length}(c) &= \lim_{N \rightarrow \infty} \sum_{k=1}^N \left\| c \left(e^{\frac{2\pi k}{N}i} \right) - c \left(e^{\frac{2\pi(k-1)}{N}i} \right) \right\| \\ &= \lim_{N \rightarrow \infty} \sum_{k=1}^N \left\| \frac{c \left(e^{\frac{2\pi k}{N}i} \right) - c \left(e^{\frac{2\pi(k-1)}{N}i} \right)}{\frac{2\pi}{N}} \right\| \cdot \frac{2\pi}{N} \\ &= \int_{\mathbb{S}^1} \|Dc(t)[t \cdot i]\| dt = \int_{\mathbb{S}^1} \|\dot{c}(t)\| dt \end{aligned}$$

We call c a **uniform parametrization** of $C = \text{Im}(c)$ iff $\|\dot{c}(t)\|$ is constant. If this constant is 1, we call c the **arclength parametrization** of C .

Uniform Parametrization

To every curve $c: \mathbb{S}^1 \rightarrow \mathbb{R}^2$ of C , we can find a different curve that is parametrized uniformly.

To this end let $L := \text{length}(c)$ and

$$\ell: [0, 2\pi] \rightarrow [0, 2\pi] \quad \ell(t) = \frac{2\pi}{L} \cdot \int_0^t \|\dot{c}(e^{\tau i})\| d\tau$$

The curve $\hat{c}: \mathbb{S}^1 \rightarrow \mathbb{R}^2$ with $\hat{c}(e^{t \cdot i}) = c(e^{\ell^{-1}(t) \cdot i})$ satisfies

$$\begin{aligned} \left\| \frac{d}{dt} \hat{c}(e^{t \cdot i}) \right\| &= \left\| Dc(e^{\ell^{-1}(t) \cdot i}) \left[e^{\ell^{-1}(t) \cdot i} \cdot i \right] \cdot \left\| \dot{c}(e^{\ell^{-1}(t) \cdot i}) \right\|^{-1} \right\| \cdot \frac{L}{2\pi} \\ &= \frac{L}{2\pi} \end{aligned}$$

Curvature

For every uniformly parametrized curve $c: \mathbb{S}^1 \rightarrow \mathbb{R}^2$, the expression to compute the curvature can be simplified.

Since we have that $\langle \dot{c}(t), \dot{c}(t) \rangle$ is constant in t , we obtain

$$0 = \frac{d}{dt} \langle \dot{c}(t), \dot{c}(t) \rangle = 2 \langle \ddot{c}(t), \dot{c}(t) \rangle$$

Thus $\dot{c}(t)$ and $\ddot{c}(t)$ are orthogonal to one another and

$$\det(\dot{c}(t), \ddot{c}(t)) = \pm \|\dot{c}(t)\| \cdot \|\ddot{c}(t)\| = \pm \frac{L}{2\pi} \|\ddot{c}(t)\|.$$

Therefore, we have for the curvature $\kappa(c(t))$

$$|\kappa(c(t))| = \left| \frac{\det(\dot{c}(t), \ddot{c}(t))}{\|\dot{c}(t)\|^3} \right| = \|\ddot{c}(t)\| \frac{4\pi^2}{L^2}$$

We already saw that the curvature is invariant with respect to translation and rotation.

Therefore, we can interpret the curvature mapping $\kappa: \mathbb{S}^1 \rightarrow \mathbb{R}$ as a **shape representation**.

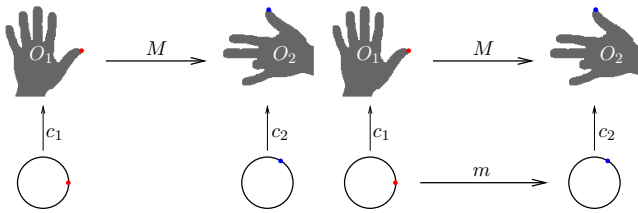
While we excluded the flexibility with respect to translation and rotation, the shape representation via curvature is not unique.

By using an arbitrary self mapping $\varphi: \mathbb{S}^1 \rightarrow \mathbb{S}^1$, we change the curve and the curvature representation

$$\begin{aligned} c: \mathbb{S}^1 &\rightarrow \mathbb{R}^2 & \rightsquigarrow & c \circ \varphi: \mathbb{S}^1 \rightarrow \mathbb{R}^2 \\ \kappa: \mathbb{S}^1 &\rightarrow \mathbb{R} & \rightsquigarrow & \kappa \circ \varphi: \mathbb{S}^1 \rightarrow \mathbb{R} \end{aligned}$$

Shape Matching

Shape Matching



A **shape matching** is a mapping $M: \partial O_1 \rightarrow \partial O_2$ that maps corresponding boundary points onto one another.

It is easier to define a matching between the parametrization domains of both contours, resulting in $m: \mathbb{S}^1 \rightarrow \mathbb{S}^1$.

Feature Representation of 2D Shapes

To perform shape matching, we need a **shape feature** that describes the "shapeness" of a curve rather than the curve itself. In the last decades a lot of descriptive shape feature have been developed.

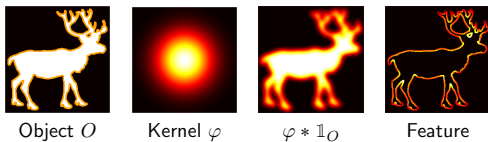
Definition 1. Let \sim be the equivalence relation of objects that defines a shape. If we can find for each curve $c: \mathbb{S}^1 \rightarrow \mathbb{R}^2$ a mapping $f_c: \mathbb{S}^1 \rightarrow \mathbb{R}^k$ such that

$$f_c(t) = f_{c'}(t) \quad \forall c' \sim c \text{ and } \forall t \in \mathbb{S}^1,$$

we call f_c a **shape feature representation** of c and \mathbb{R}^k its **feature space**.

So far, we showed that **curvature** is a **one-dimensional shape feature** with respect to the shape defined by **translation and rotation**.

Integral Invariant



Other shape features like the "integral invariant" will not simply rely on the boundary C of an object O but also on the object itself.

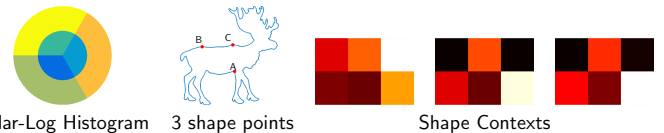
Let $\varphi: \mathbb{R}^2 \rightarrow \mathbb{R}$ be a rotation-invariant kernel with compact support, i.e.,

$$\begin{aligned} \varphi(x) &= \varphi(R \cdot x) & \forall x \in \mathbb{R} \text{ and } R \in SO(2) \\ \varphi(x) &= 0 & \forall x \notin B_\varepsilon(0). \end{aligned}$$

Then, we can define the **integral invariant** via the following convolution

$$f: \mathbb{S}^1 \rightarrow \mathbb{R} \quad t \mapsto \int_O \varphi(c(t) - x) dx = (\varphi * \mathbb{1}_O)(c(t))$$

Shape Context



The shape context can be seen as an extension of the integral invariants. Instead of one, we use multiple kernels $\varphi_i: \mathbb{R}^2 \rightarrow \mathbb{R}$ in a log-polar scale. The resulting feature is a high-dimensional histogram representation.

The resulting feature is only translation invariant. To make it rotational invariant, one might use the tangent space at $p \in C$ as a baseline. To make the computation practically feasible, only those rotations are used that are represented by the histogram kernels.

Comparing Features

Given two curves $c_1, c_2: \mathbb{S}^1 \rightarrow \mathbb{R}^k$ of the same shape together with their shape feature representations $f_1, f_2: \mathbb{S}^1 \rightarrow \mathbb{R}^k$. If the two points $c_1(t_1)$ and $c_2(t_2)$ correspond to one another, we know that $f_1(t_1) = f_2(t_2)$.

Therefore, we can measure the similarity of two arbitrary points $c_1(t_1)$ and $c_2(t_2)$ via $\text{dist}(f_1(t_1), f_2(t_2))$, where the **distance function** $\text{dist}: \mathbb{R}^k \times \mathbb{R}^k \rightarrow \mathbb{R}_0^+$ measures the similarity of two features in \mathbb{R}^k .

Common distance functions are

$$\begin{aligned} \text{dist}(\kappa_1, \kappa_2) &= (\kappa_1 - \kappa_2)^2 & (\text{Curvature}) \\ \text{dist}(I_1, I_2) &= (I_1 - I_2)^2 & (\text{Integral Invariant}) \\ \text{dist}(C^{(1)}, C^{(2)}) &= \sum_{i=1}^k \frac{(C^{(1)} - C^{(2)})^2}{C^{(1)} + C^{(2)}} & (\text{Shape Context}) \end{aligned}$$

Discretization

In order to solve the shape matching problem, we like to work with a **finite** representation. The process of transforming a continuous problem into such a "finite" problem is called **discretization**.

Let us assume that two curves $c_1, c_2: \mathbb{S}^1 \rightarrow \mathbb{R}^2$ are provided in a uniform parametrization. Given the corresponding features $f_1, f_2: \mathbb{S}^1 \rightarrow \mathbb{R}^k$, we choose the following discretization

$$\begin{aligned} F^{(1)} &= \left(f_1 \left(e^{\frac{2\pi}{N}} \cdot i \right) \quad \dots \quad f_1 \left(e^{\frac{2\pi \cdot j}{N}} \cdot i \right) \quad \dots \quad f_1 \left(e^{\frac{2\pi \cdot N}{N}} \cdot i \right) \right) \in \mathbb{R}^{k \times N} \\ F^{(2)} &= \left(f_2 \left(e^{\frac{2\pi}{N}} \cdot i \right) \quad \dots \quad f_2 \left(e^{\frac{2\pi \cdot j}{N}} \cdot i \right) \quad \dots \quad f_2 \left(e^{\frac{2\pi \cdot N}{N}} \cdot i \right) \right) \in \mathbb{R}^{k \times N} \end{aligned}$$

This provides us with a **cost matrix** $D \in \mathbb{R}^{N \times N}$, i.e., $D_{i,j} = \text{dist}(F_i^{(1)}, F_j^{(2)})$, which stores the similarity between the i -th point of the first shape and the j -th point of the second shape.

The goal of shape matching is to find corresponding points between two shapes. This is necessary because the feature representation uses a specific parametrization.

One way of formulating this problem is to look for a **permutation** $\pi: \{1, \dots, N\} \rightarrow \{1, \dots, N\}$ such that

$$E(\pi) = \sum_{i=1}^N D_{i,\pi(i)}$$

is minimized.

In other words, we assign to each shape point of the first shape a unique point of the second shape and the cost that we assign to this **assignment** depends **"linearly"** on this choice.

Therefore, this problem is called **Linear Assignment Problem (LAP)**.

Hungarian Method

The LAP has to optimize a function over the space of all permutation. Since there are $N!$ different permutations, it is not clear whether this problem can be solved in polynomial time.

In 1955 Kuhn presented a method that has a time complexity $\mathcal{O}(N^4)$. 1957, Munkres improved the running time to $\mathcal{O}(N^3)$. Kuhn's original work was based on the work of the Hungarians König and Egerváry. For that reason, the method is sometimes referred as the **Kuhn-Munkres method** or the **Hungarian method**.

The main idea is to change the entries of the non-negative cost matrix D in order to simplify the problem. If there is a permutation π such that $D_{i,\pi(i)} = 0$, we know that we found the global optimum.

An important observation is that by adding a value $a \in \mathbb{R}$ to one row or to one column, we change the value of the minimum by a , but the optimal permutation is still the same.

Trivial Solutions

The following cost matrices are minimized by any permutation. Why?

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

1	1	1	1
0	0	0	0
0	0	0	0
0	0	0	0

1	1	1	1
4	4	4	4
2	2	2	2
3	3	3	3

0	0	3	0
1	1	4	1
0	0	3	0
0	0	3	0

1	1	6	1
4	4	9	4
2	2	7	2
3	3	8	3

1	2	3	4
5	6	7	8
1	2	3	4
5	6	7	8

Example

90	75	75	80
35	85	55	65
125	95	90	105
45	110	95	115

 $\Rightarrow 245 +$

C	C	C	
15	*	0	5
*	50	20	30
35	5	*	15
0	65	50	70

- For each row r : Find the minimum a_r .
- Subtract from each row r its minimum a_r .
- For each "0" in the matrix, replace it by a * iff there is no * in the same column or row.
- Mark each column that contains a *.
- iff every column is marked, the stars form an optimal permutation.
- Otherwise, find the minimal entry $a \geq 0$ of the non-covered entries.

Example

C		C	
15	*	0	/
*	50	20	25
35	5	*	10
0	65	50	65

 $\Rightarrow 255 +$

C			
20	*	5	/
*	45	20	20
35	/	*	5
0	60	50	60

- Subtract a from each (unmarked) row and add it to each marked column.
- Replace one zero of the uncovered entries with /. Call its row r .
- If there is a * at position (c, r) , unmark the column c and mark row r .
- Find the minimal entry $a \geq 0$ of the non-covered entries.
- Subtract a from each unmarked row and add it to each marked column.
- Replace one zero of the uncovered entries with /. Call its row r .
- If there is a * at position (c, r) , unmark the column c and mark row r .
- Find the minimal entry $a \geq 0$ of the non-covered entries.

Example

40	*	5	/
*	25	0	/
55	/	*	5
/	40	30	40

 $\Rightarrow 275 +$

40	*	5	0
0	25	0	*
55	0	*	5
*	40	30	40

- Subtract a from each (unmarked) row and add it to each marked column.
- Replace one zero of the uncovered entries with /. Call its row r .
- If there is a * at position (c, r) , unmark the column c and mark row r .
- Find the minimal entry $a \geq 0$ of the non-covered entries.
- Subtract a from each unmarked row and add it to each marked column.
- Replace one zero of the uncovered entries with /. Call its row r .
- If there is no * in row r , increase the amount of * via back-tracking.
- If the amount of * is maximal, they form the optimal permutation.

Hungarian Method

- Subtract from each row its minimum. $\Rightarrow D_{i,j} \geq 0$.
- Replace each zero with a * as long as there is no * in that row or column.
- Mark each *-column. If N columns are marked go to Step 12.
- Compute the minimum a of the unmarked entries.
- Subtract a from the unmarked entries and add it to the twice marked entries.
- Find an unmarked "0" at position (r, c_0) and replace it with /.
- If there is a * at position (c, r) , unmark column c , mark row r and go to Step 4.
- If there is a * at position (r_0, c_0) , there is a / at position (r_1, c_0) . This back-tracking terminate with a /.
- Exchanging the back-tracked / and * increases the amount of * by 1.
- Unmark all columns and rows and replace every / with a 0.
- If we have N *, go to Step 12. Otherwise go to Step 4.
- The N stars in the matrix define the optimal permutation.

Features

- Belongie et al., *Shape Matching and Object Recognition Using Shape Context*, 2002, IEEE TPAMI (24) 24, 509–521.
- Manay et al., *Integral Invariants and Shape Matching*, 2006, IEEE TPAMI (28) 10, 1602–1618.

Hungarian Method

- Kuhn, *The Hungarian Method for the Assignment Problem*, 1955, Naval Research Logistics Quaterly 2, 83–97.
- Munkres, *Algorithms for the Assignment and Transportation Problems*, 1957, Journal SIAM (5) 1, 32–38.