Shape Matching

Michael Kazhdan

(601.457/657)
Overview

- Intro
- General Approach
- Minimum SSD Descriptor
Goal

Given:
- 3D model database
- query shape

Fine:
- The database models most similar to the query.
Applications

- Entertainment
- Medicine
- Chemistry/Biology
- Archaeology
Applications

- Entertainment
  - Model generation
- Medicine
- Chemistry/Biology
- Archaeology
Applications

- Entertainment
- Medicine
  - Automated diagnosis
- Chemistry/Biology
- Archaeology

Images courtesy of NLM
Applications

- Entertainment
- Medicine
- Chemistry/Biology
  - Docking and binding
- Archaeology
Applications

• Entertainment
• Medicine
• Chemistry/Biology
• Archaeology
  ◦ Reconstruction

Image Courtesy of Stanford
Overview

• Motivation
• General Approach
• Minimum SSD Descriptor
Shape Matching

General approach:
Define a function that takes in two models and returns a measure of their proximity.

\[ D \left( \begin{array}{c} M_1 \\ M_2 \end{array} \right) \leq D \left( \begin{array}{c} M_1 \\ M_3 \end{array} \right) \]

\( M_1 \) is closer to \( M_2 \) than it is to \( M_3 \)
Database Retrieval

- Compute the distance from the query to each database model

\[ D(Q, M_i) \]

3D Query

Database Models
Database Retrieval

- Sort the database models by proximity

\[ D(Q, M_i) \leq D(Q, M_j) \quad \forall i \leq j \]
Database Retrieval

- Return the closest matches

\[ D(Q, \tilde{M}_i) \leq D(Q, \tilde{M}_j) \quad \forall i \leq j \]
Overview

• Motivation

• General Approach
  ◦ Shape Descriptors

• Minimum SSD Descriptor
Shape Matching

**General approach:**
Define a function that takes in two models and returns a measure of their proximity.

\[ D(M_1, M_2) \leq D(M_1, M_3) \]

\( M_1 \) is closer to \( M_2 \) than it is to \( M_3 \).
Shape Descriptors

**Shape Descriptor:**
A structured abstraction of a 3D model that is well suited to the challenges of shape matching.
Matching with Descriptors

Preprocessing

➢ Compute database descriptors

Run-Time
Matching with Descriptors

Preprocessing
➢ Compute database descriptors

Run-Time
➢ Compute query descriptor
Matching with Descriptors

Preprocessing
➢ Compute database descriptors

Run-Time
➢ Compute query descriptor
➢ Compare query descriptor to database descriptors
Matching with Descriptors

Preprocessing
➢ Compute database descriptors

Run-Time
➢ Compute query descriptor
➢ Compare query descriptor to database descriptors
➢ Return best Match(es)
Shape Matching Challenge

Need shape descriptor that is:

- Concise to store
  - Quick to compute
  - Efficient to match
  - Discriminating
Shape Matching Challenge

Need shape descriptor that is:

- Concise to store
- Quick to compute
- Efficient to match
- Discriminating

3D Query ➔ Shape Descriptor ➔ 3D Database ➔ Best Matches
Shape Matching Challenge

Need shape descriptor that is:
- Concise to store
- Quick to compute
- Efficient to match
- Discriminating

3D Query  →  Shape Descriptor  →  3D Database  →  Best Matches
Shape Matching Challenge

Need shape descriptor that is:
- Concise to store
- Quick to compute
- Efficient to match
- Discriminating
Shape Matching Challenge

Need shape descriptor that is:

- Concise to store
- Quick to compute
- Efficient to match
- Discriminating
  - Invariant to transformations
    - Invariant to deformations
    - Insensitive to noise
    - Insensitive to topology

Different Transformations
(translation, scale, rotation, mirror)
Shape Matching Challenge

Need shape descriptor that is:

- Concise to store
- Quick to compute
- Efficient to match
- Discriminating
- Invariant to transformations
  - Invariant to deformations
- Insensitive to noise
- Insensitive to topology

Different Articulated Poses
Shape Matching Challenge

Need shape descriptor that is:

- Concise to store
- Quick to compute
- Efficient to match
- Discriminating
- Invariant to transformations
- Invariant to deformations
- Insensitive to noise
- Insensitive to topology

Scanned Surface

Image courtesy of Ramamoorthi et al.
Shape Matching Challenge

Need shape descriptor that is:

- Concise to store
- Quick to compute
- Efficient to match
- Discriminating
- Invariant to transformations
- Invariant to deformations
- Insensitive to noise
- Insensitive to topology

Images courtesy of Viewpoint & Stanford
Overview

• Applications
• General Approach
• Minimum SSD Descriptor
Shape Matching Approach

Q: How should we measure the similarity between two shapes?
Shape Matching Approach

A: Define shape (dis)similarity as the sum of squared distances from points on one surface to the closest points on the other.

\[ d(A, B) = \int_A \text{dist}(p_A, B)^2 dp_A + \int_B \text{dist}(p_B, A)^2 dp_B \]
Shape Matching Approach

A: Define shape (dis)similarity as the sum of squared distances from points on one surface to the closest points on the other.

\[ d(A, B) = \int_A \text{dist}(p_A, B)^2 dp_A + \int_B \text{dist}(p_B, A)^2 dp_B \]
Shape Matching Approach

A: Define shape (dis)similarity as the sum of squared distances from points on one surface to the closest points on the other.

\[ d(A, B) = \int_A \text{dist}(p_A, B)^2 dp_A + \int_B \text{dist}(p_B, A)^2 dp_B \]
Overview

• Applications

• General Approach

• Minimum SSD Descriptor
  ◦ (Euclidean) Distance Transform
(Euclidean) Distance Transform

The (Euclidean) Distance Transform (DT) of a surface is a function giving the distance of every point in space to the boundary.

\[ DT_S(p) = \min_{q \in S} \|p - q\| \]
(Euclidean) Distance Transform

Grass-Fire Algorithm:

- Think of space as a field of dry grass.
- Set fire to the boundary and measure the amount of time for the fire to reach each point.
(Euclidean) Distance Transform

Grass-Fire Algorithm:

• Think of space as a field of dry grass.

• Set fire to the boundary and measure the amount of time for the fire to reach each point.

The points where the fire gets quenched define the skeleton of the shape.

$S$ $DT_S$
Computing $DT_S$

Brute Force:

Compute the distance to each surface point and store the minimum.

If there are $m$ surface points and we want the values on a grid of resolution $R$, the overall complexity becomes:

- $O(R^2m) \approx O(R^2 \cdot R)$ for a 2D grid
- $O(R^3m) \approx O(R^3 \cdot R^2)$ for a 3D grid
Computing $DTS$

Graphics Hardware (2D):

1. For each surface point $(x, y)$, draw a 3D right-cone with apex at $(x, y, 0)$ and axis aligned with the positive $z$-axis.

2. Render with orthographic projection, looking down the positive the $z$-axis.

3. Read the values of the depth-buffer to get the values of $DTS$. 
Computing $DT_S$

General Problem:

Given a set of points, $P = \{p_1, \ldots, p_n\} \subset \mathbb{R}^2$ and given a point $p \in \mathbb{R}^2$ we would like to compute the distance to the closest point in $P$:

$$d(p, P) = \min_i \|p - p_i\|$$

A good place to start is to consider how we can compute the distance from the point $p$ to the point $p_i$. 
Computing $DTS$

Graphics Hardware (2D):

At $p$, the height of a **right**-cone with apex at $p_i$ is the distance from $p$ to $p_i$.

Given two points $p_i$ and $p_j$ the distance from a point $p$ to the closer of the two points is the minimum of the two heights.
Computing $DT_S$

Graphics Hardware (2D):

At $p$, the height of a **right**-cone with apex at $p_0$ is the distance from $p$ to $p_0$.

Given a collection of points in the $xy$-plane:
Computing $DT_S$

Graphics Hardware (2D):

At $p$, the height of a right-cone with apex at $p_0$ is the distance from $p$ to $p_0$.

Given a collection of points in the $xy$-plane:

- Draw right-cones at each point

![Diagram showing two cones with 45° angles and points in the xy-plane]
Computing $DT_S$

Graphics Hardware (2D):

At $p$, the height of a right-cone with apex at $p_0$ is the distance from $p$ to $p_0$.

Given a collection of points in the $xy$-plane:

- Draw right-cones at each point
- View along the $z$-direction
Computing \( DT_s \)

Graphics Hardware (2D):

At \( p \), the height of a right-cone with apex at \( p_0 \) is the distance from \( p \) to \( p_0 \).

Given a collection of points:

- Draw right-cones at each point
- View along the \( z \)-direction
- Read back the depth-buffer
Computing $\mathcal{D} T_s$

Graphics Hardware (2D):
- Draw right-cones at each point
- View along the $z$-direction
- Read back the depth-buffer
Overview

• Applications

• General Approach

• Minimum SSD Descriptor
  ○ (Euclidean) Distance Transform
Shape Matching Implementation

Preprocessing:

Compute **rasterization** and **squared distance transforms**

- The value of the rasterization at a 3D point (voxel) is:

  \[ R_A(p) = \begin{cases} 
  1 & \text{if } p \in A \\
  0 & \text{otherwise} 
  \end{cases} \]

- The value of the distance transform at a 3D point is:

  \[ DT_A^2(p) = \min_{q \in A} ||p - q||^2 \]
Shape Matching Implementation

Run-Time:
Compute mesh similarity with two dot-products/integrals

\[ d(A, B) = \langle R_A, DT_B^2 \rangle + \langle DT_A^2, R_B \rangle \]
Shape Matching Implementation

Run-Time:
Compute mesh similarity with two dot-products/integrals

The dot product of $R_A$ with $DT_B^2$ is the sum of the product of the two functions:

$$\langle R_A, DT_B^2 \rangle = \int_{\mathbb{R}^3} R_A(p) \cdot DT_B^2(p) dp$$

$$= \int_A DT_B^2(p) dp$$

$$= \int_A \min_{q \in B} \|p - q\|^2 dp$$

because the rasterization $R_A$ is equal to zero off of $A$ and is equal to one on it.
Shape Matching Implementation

- Advantages:
  - Squared EDT is quick to compute
  - Match surfaces without correspondences
  - Can use compression techniques to reduce storage.
  - Can solve for the optimal rigid-body alignment using fast signal processing techniques.
Summary

Minimum sum of squared distances descriptor:

• Advantages:
  ◦ Compact
  ◦ Discriminating
  ◦ Quick to compute
  ◦ Allows for matching over rigid body transformations
Summary

Minimum sum of squared distances descriptor:

• Advantages:
  ◦ Compact
  ◦ Discriminating
  ◦ Quick to compute
  ◦ Allows for matching over rigid body transformations

• Limitations:
  ◦ Difficult to use for partial object matching
Summary

Minimum sum of squared distances descriptor:

• Advantages:
  ◦ Compact
  ◦ Discriminating
  ◦ Quick to compute
  ◦ Allows for matching over rigid body transformations

• Limitations:
  ◦ Difficult to use for partial object matching
  ◦ Difficult to use for articulated figures