A Physically-based Method for 2D and 3D Similarity and Affine Invariant Alignments

Jim X. Chen and Harry Wechsler
Department of Computer Science, George Mason University
{jchen, wechsler}@cs.gmu.edu

Abstract

We present a physically-based alignment method for two point sets, which are invariant to either similarity or affine transformations. Here we assume a known correspondence between the point sets. We then employ the method to align 2D face images, find a mean-face, which is an average of all the aligned faces, and transform all face images to mean-face images for statistical face recognition. After that, we employ the same method to align 3D MRI slices with a reference model to reconstruct a patient’s knee surface model for surgery assistance. Our method is relatively easy to implement, has no analytical deductions, and allows interactive visualizations of the transformation process.

1 Background

When comparing images to other images or models, one would like to cancel camera transformations. Werman and Weinshall developed 2D metrics to measure the similarity and affine distances between two point sets [20]. Their transformation matrices and other similar analytical methods are widely employed in 2D image alignments and recognitions [3][5][14][18]. The analytical methods are concise and direct solutions to some 2D alignment problems. However, there is no known closed form for the alignment of two 3D point sets. For knee surgery assistance [21], we need to align two 3D point sets when we try to reconstruct a patient’s 3D knee model from the MRI slices.

Here we introduce a physically-based iterative method for the alignments of both 2D and 3D point sets with weighted extremes, which is an alternative method different from the analytical methods. Compared to the analytical methods, our method avoids complex analytical deductions needed for different weighted extremes, solves similar problems both in 2D and 3D, and is relatively easier to implement.

2 Similarity Invariant Alignment

Given two point sets, \( P = \{(x_i, y_i, z_i)\}_{i=1}^{n} \) and \( Q = \{(u_i, v_i, w_i)\}_{i=1}^{n} \), where \( P \) and \( Q \) are two \( 3 \times n \) matrices, our goal is to find a similarity transformation that will align the first point set with the second point set, so that the Euclidean distance between points will be at minimum. That is, after translation, rotation, and uniform scale along x, y, and z axes, we have

\[
P' = sR + T,
\]
so that \( \sum k_i d_i \) is at minimum, where \( k_i \) is the extreme emphasis and \( d_i = \| P[i] - Q[i] \| \), namely \( d_i = \sqrt{(x_i' - u_i)^2 + (y_i' - v_i)^2 + (z_i' - w_i)^2} \).

We employ a physical-based iterative method to achieve the goal, as shown in Figure 1. First, we assume that the points in \( P \) are unit mass particles subject to two constraints: 1) \( P[i] \) is connect to \( Q[i] \) by a spring with a force subject to Hooke’s law. That is,

\[
F[i] = k_i (Q[i] - P[i]),
\]

where \( Q[i] \) is fixed and \( k_i \) is the spring stiffness; 2) \( P \) is deformable in the sense that it can be uniformly scaled along \( x, y, \) and \( z \) axes. In other words, the size of \( P \) as an object can be changed, but the shape remains. Then, we allow \( P \) to move and deform (through translation, rotation, and uniform scale) until it achieves complete equilibrium, so that

\[
F = \sum_{i=1}^{n} F[i] = 0,
\]

\[
M_c = \sum_{i=1}^{n} r_i \times F[i] = 0,
\]

and \( \sum \| F[i] \| \) is at minimum,

where \( r_i \) is the vector from the center of mass \( C \) to \( P[i] \). The result, \( P' \), is the similarity invariant alignment we are looking for.

Fig. 1: Transformation according to a physically-based model

From the total force, we can calculate the translation of \( P \) over time according to Newton’s second law:
\[ a_c = \frac{\mathbf{F}}{m_c} = \frac{\mathbf{F}}{n}, \quad V_c = V_{old} + \int_{t_0}^{t_1} a_c \, dt, \quad \text{and} \quad \mathbf{P} = \mathbf{P}_{old} + \int_{t_0}^{t_1} V_c \, dt \quad (6) \]

Similarly, from the total moment in Equation 4, we can calculate the rotation of \( \mathbf{P} \) over time from

\[ \mathbf{H}_c = \int_{t_0}^{t_1} \mathbf{M}_c \, dt = \sum_i r_i \times (\mathbf{\omega} \times \mathbf{r}_i)m_i, \quad (7) \]

where \( \mathbf{\omega} = (\omega_x, \omega_y, \omega_z) \).

For scale, we can calculate

\[ F_s = \sum_i (\|\mathbf{P}[i] - \mathbf{C}\| - \|\mathbf{Q}[i] - \mathbf{C}\|)\|\mathbf{F}[i]\|, \quad (8) \]

where positive \( F_s \) is the shrinking force and negative \( F_s \) is the expanding force of object \( \mathbf{P} \).

Therefore, according to Equation 6 to Equation 8, for each time frame, we can find the corresponding translation, rotation, and scale to calculate new \( \mathbf{P} \), which will be better aligned with \( \mathbf{Q} \). After a period of time, the point set \( \mathbf{P} \) will be in alignment with point set \( \mathbf{Q} \).

Instead of strictly follow the physics equations, we further simplify and use an iterative method to find the solution. For example, in order to satisfy Equation 3, we translate \( \mathbf{P} \) along \( x \) axis an initial distance \( tx \). After the translation, if \( \|\mathbf{F}\| \) is getting smaller, we are in the right direction and the iteration continue, otherwise we reverse the direction and reduce the translation distance (i.e., \( tx = -tx/2 \) in programming code), and continue to converge to the final solution. Similar iterations are carried out for translations along other axes, rotations, and scales along \( x, y, \) and \( z \) axes at the center of mass, respectively. The algorithm is as follows:

\begin{verbatim}
Alignment (P, Q) {
    while (tx+ty+tz+wx+wy+wz+s > threshold) {
        translatex (P, tx);
        Fnew = CalculateCurrentForce();
        // if force not reduced, reverse direction of convergence
        if (Fnew > F) tx = - tx/2; F = Fnew;
        translatey (P, ty); Fnew = CalculateCurrentForce();
        if (Fnew > F) ty = - ty/2; F = Fnew;
        translatez (P, tz); Fnew = CalculateCurrentForce();
        if (Fnew > F) tz = - tz/2; F = Fnew;
        translatexyz(P, -C); // rotate and scale along the centroid
        rotatex(P, wx); Fnew = CalculateCurrentForce();
        if (Fnew > F) wx = - wx/2; F = Fnew;
        rotatey(P, wy); Fnew = CalculateCurrentForce();
        if (Fnew > F) wy = - wy/2; F = Fnew;
        rotatez(P, wz); Fnew = CalculateCurrentForce();
        if (Fnew > F) wz = - wz/2; F = Fnew;
    }
}
\end{verbatim}
scalexyz(P, s); Fnew = CalculateCurrentForce();
if (Fnew > F) s = - s/2; F = Fnew;
translatexyz(P, C);
}
output(P)

The algorithm can be improved in a number of ways in the implementation, but the idea is the same. The time complexity of the algorithm depends mainly on the number of points in P and the threshold. It is an O(n) algorithm. Given initial value of $q = t_x+t_y+t_z+w_x+w_y+w_z+s$, where $q<<n$, it takes about $\log_2 q$ iterations to get to a threshold of one. In practice, the method is fast enough to achieve real-time animation of face model alignment.

3 Finding 2D Shape-Free Face Images

In face image categorization and recognition, we need to cancel camera transformations and normalize the orientation and shape of the images [3][4][14][15]. Our goal is to model grey-level appearance independently of shape. Given the $N$ original face images which may have different shapes and orientations, first we register the landmark points (or simply points) so that each image corresponds to a 2D point set, which is also called the shape of the face. Then, we transform the point sets to align with an predefined shape model according to similarity invariant alignment. After that, we average all the point sets to find a mean shape. Finally, we warp each face image according to its shape and the mean shape to find its corresponding mean-shape face image, or shape-free image. The resulting face images have the same shape, but different textures. The process is shown in Figure 2.

![Fig. 2: Process of finding shape-free face images](image)

3.1 Registration and Alignment

First, we automatically superimpose a point set geometry on a face image, as shown in Figure 3. The landmark points, assistant lines, and the Bezier curve from point 19 to 27 are all displayed with the image. Then, we use hand-tracing to adjust those points that are not at the corresponding face image landmarks. The assistant lines and Bezier curve will interactively move according to the modified control points to assist finding the boundary of the face outline. The corresponding angles and relative distances will remain fixed to constrain the hand-tracing. This process is repeated for all $N$ faces so we have all the face shapes, which is registered as point sets $P[i]$, where $1 \leq i \leq 20$. 

...
Then, alignment function is called to match with a given normalized shape, which can be an arbitrary shape in the face image set or the same as in Figure 3. The 2D alignment algorithm is a simplification of the algorithm given in Section 2 without translation and scale along $z$ axis, and rotation along $x$ and $y$ axes. The result after alignment is saved to a file faceShape.j.txt, where $1 \leq j \leq N$ for all face images.

![Geometry of the landmark points](image)

**Fig. 3: Geometry of the landmark points**

### 3.2 Mean-Face Shape and Triangulation

After the alignment, we average all the faces’ corresponding landmark points to find a mean-face shape: $x_i = \frac{1}{N} \sum_{j=1}^{N} x_{ji}$ and $y_i = \frac{1}{N} \sum_{j=1}^{N} y_{ji}$. The mean-face shape is then triangulated according to the minimum spanning algorithm, which repeats connecting the current shortest points pairs that has no cross intersection with other edges until there is no connection possible. The result is shown in Figure 4.

![Triangulation according to the growing minimum spans](image)

**Fig. 4: Triangulation according to the growing minimum spans**
3.3 Warping to Shape-Free Face Images

First, we apply the triangulation topology of the mean-face shape to all the face shapes. Then, according to the corresponding triangles, we warp the original face images into their corresponding mean face images, or shape-free images [1][15][19].

We employ a scan-line algorithm to scan convert and fill the triangles in the output image. Given a pixel address \((x, y)\) in the shape-free image, the inverse mapping to the corresponding point \((u, v)\) in the original face image is shown in Figure 5 (Equation \((g)\) and \((h)\) are used to find the corresponding point.) Here \(u\) and \(v\) are floating-point numbers. The objective is to obtain the intensity at \((u, v)\), which will be used as the intensity at \((x, y)\). This is also called the resampling of the point \((u, v)\). There are many different ways to calculate the pixel intensity of \((x, y)\) from an interpolation of intensity at point \((u, v)\) according to its surrounding pixel intensities [19]. Here bicubic interpolations are employed to find the corresponding intensities.

\[
\begin{align*}
x_{12} &= x_1 + (x_2 - x_1) \frac{y - y_1}{y_2 - y_1} \quad (a) \\
x_{13} &= x_1 + (x_3 - x_1) \frac{y - y_1}{y_3 - y_1} \quad (b) \\
u_{12} &= u_1 + (u_2 - u_1) \frac{y - y_1}{y_2 - y_1} \quad (c) \\
v_{12} &= v_1 + (v_2 - v_1) \frac{y - y_1}{y_2 - y_1} \quad (d) \\
u_{13} &= u_1 + (u_3 - u_1) \frac{y - y_1}{y_3 - y_1} \quad (e) \\
v_{13} &= v_1 + (v_3 - v_1) \frac{y - y_1}{y_3 - y_1} \quad (f) \\
u &= u_{12} + (u_{13} - u_{12}) \frac{x - x_{12}}{x_{13} - x_{12}} \quad (g) \\
v &= v_{12} + (v_{13} - v_{12}) \frac{x - x_{12}}{x_{13} - x_{12}} \quad (h)
\end{align*}
\]

![Fig. 5: Mapping from output triangle to input triangle](image)

3.4 Results

Figure 6 shows how shape-free images are generated: first, the control points are collected. Then, the control points of face shapes are aligned, and mean-face shape is calculated. After that, triangulation is applied and each face is transformed into a shape-free image by mapping the corresponding triangle textures (warping).

![Fig. 6: Collecting control points, triangulation after alignment, and warping](image)
Figure 7 shows the results of 20 shape-free images. The first (upper-left corner) face is used as the template for alignment.

**Fig. 7: Results of 20 shape-free images**

### 4 Generating 3D Patient-Specific Knee Model

Longevity of a knee joint is dependent upon keeping inside certain physiological limits for those stresses (pressures) that occur on both the convex and concave contact surfaces during normal activities. These limits include total load per unit area for the joint and a specific pattern for distributing the load on the curved surfaces. Alignment is critical for satisfying these conditions.

Knee osteotomy is a kind of orthopedic surgery to re-align the lower limb by opening or cutting a bone wedge from the leg. It is a better alternative than other types of knee replacement surgeries, especially for young people. However, an osteotomy requires understanding of the imbalance of pressures at the knee joint, predicting abnormal walking gait cycle, and cutting the bone wedge precisely. Therefore, knee osteotomy is extremely difficult and prone to further damage despite the fact that it is simply a bone cut. As a result, many surgeons favor other kinds of knee replacement surgeries. Our objective is to use computer graphics to assist the knee surgery study and operation.

Current static examinations of the knee such as X-rays and MRI’s will be replaced by interactive visualization, surgery study, planning, exercise, and results predictions. In recent years, some computer-based surgical simulation systems are developed to help surgeons carry out knee surgeries [2][8]. However, the knee models are not patient-specific data [9]. Here we present a method of reconstructing a 3D patient-specific knee model from a set of MRI slices. After receiving a set of MRI slices of the knee, we employ image segmentation, alignment, surface reconstruction, as well as deformation or warping to reconstruct the patient’s 3D knee surface model in computer.
4.1 Image Segmentation

The purpose of image segmentation is to find the contour line of certain 2D object within a 2D image (or contour surface of certain 3D object within a 3D image). Manual segmentation (hand tracing), which many exiting knee reconstruction methods employ [9][11], is time consuming. On the other hand, the clinical knee MRI slices usually have high noise/signal ratio and low resolution. Therefore, fully automatic segmentation is not possible and certain human interventions have to be employed to set the initial conditions, guide the segmentation process, or correct errors.

Our solution to this problem uses a graphical user interface to highlight a few control points of one MRI slice (Figure 8_a), employs a deformable template (snake) method [12] to find the bone contour points, and exploits the previous contour points to serve as starting control points for the current slice, and so forth for the rest of the slices. A contour editor is constructed to modify the error output from the algorithm (Figure 8_b). The modified points will serve as input control points back to the algorithm, which will generate modified contour interactively. After this, an MRI model is constructed (Figure 8_c).

4.2 Alignment

The purpose of image alignment is to find a 3D similarity invariant transformation so that the points in one image can be related to their corresponding points in another image or a surface model [1][3][7][18][20]. Surface reconstruction rebuilds an object’s surface from MRI’s, CT scan images, or other volumetric data gathered from different sources [6][13][16]. Conventional 3D reconstruction methods are based on hand-traced contours [10][17], which are widely used in many commercial reconstruction packages. However, they require too much manual work and the result may not be realistic if the number of contours is not enough. We have an existing computer-based reference model — a high fidelity cadaver knee surface model, which is what an ordinary knee should be. Despite small variances, ordinary human knees have same appearance and features. Our method uses physics-based similarity transformation, as discussed in Section 2, to align the MRI model (a set of image contours) with the reference model to reconstruct the patient’s knee model. Specifically, we fix the MRI model, define feature control points in the reference model and the MRI model, and build up constraint forces according to the distances of the corresponding control points so that the reference model will be transformed to align with the MRI model as close as possible. The reference model only goes through an affine transformation so that its orientation, location, and size may change, but its shape will be preserved. After this, we will deform the reference model and recover the lost features between the MRI slices, as described in the next section.

4.3 Warping

In order to recover the lost features between MRI slices, after we align the corresponding features of the MRI model with the reference model, we deform the reference model according to surface warping interpolations. First, each MRI model’s vertex is paired with a reference model’s vertex as shown in Figure 8_d. Because there are more vertices in the reference model than in the MRI model, the search process for vertex pairs is reversed — for a given reference model’s vertex, the MRI model’s vertices are searched for the corresponding vertex. This way many reference model’s vertices can be eliminated immediately after a few comparisons. The following function is maximized at the paired vertices: $(r \bullet n)^d/|r|$, where $r$ is the vector from
the MRI model’s vertex to the reference model’s vertex, \( n \) is the normalized surface normal at the reference model’s vertex, and \( d \) is a factor to attenuate the emphasis on distance. When the function is at maximum, the distance between the paired vertices (\(|r|\)) and the angle between \( r \) and \( n \) are at minimum. A “porcupine spine” will be visible in 3D from the MRI model’s vertex pointing in the same direction as \( r \). The user can interactively modify poorly chosen paired vertices by manually rotate the spine around the MRI model’s vertex, and the rest of the spines (i.e., all other paired vertices) will be adjusted automatically. After this, each of the reference model’s paired vertices (that are on the spines) is translated to the corresponding contour point, and each of the reference model’s other vertices (that are not on the spines) will be interpolated to its corresponding position by bi-linear interpolation according to its nearest “paired” neighbor vertices. After this stage, we have a patient’s 3D surface model.

![Diagram](image)

Fig. 8: Construct a patient’s knee MRI model and 3D surface model
5 Affine Invariant Alignment

Affine transformation includes rotation, translation, as well as shears and non-uniform scales along \(x\), \(y\), and \(z\) axes. In general, the following matrix is used for shear transformation:

\[
\text{shear} = \begin{bmatrix}
1 & a & b & 0 \\
0 & c & d & 0 \\
e & f & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]  

(9)

We can modify the alignment algorithm in Section 2 to include the scales and shears:

\[
\text{AffineAlignment} (P, Q) \{
\begin{align*}
& \text{Alignment} (P, Q); \\
& \text{while} \ (a+b+c+d+e+f+sx+sy+sz > \text{threshold1}) \ {\}
\end{align*}
\]

\[
\begin{align*}
& \quad \text{translatexyz}(P, -C); \\
& \quad \text{sheara}(P, a); \ F\text{new} = \text{CalculateCurrentForce}(); \\
& \quad \quad \text{if} \ (F\text{new} > F) \ a = -a/2; \ F = F\text{new}; \\
& \quad \quad \text{shearb}(P, b); \ F\text{new} = \text{CalculateCurrentForce}(); \\
& \quad \quad \quad \text{if} \ (F\text{new} > F) \ b = -a/2; \ F = F\text{new}; \\
& \quad \quad \text{shearc}(P, c); \ F\text{new} = \text{CalculateCurrentForce}(); \\
& \quad \quad \quad \text{if} \ (F\text{new} > F) \ c = -a/2; \ F = F\text{new}; \\
& \quad \quad \text{sheard}(P, d); \ F\text{new} = \text{CalculateCurrentForce}(); \\
& \quad \quad \quad \text{if} \ (F\text{new} > F) \ d = -a/2; \ F = F\text{new}; \\
& \quad \quad \text{sheare}(P, e); \ F\text{new} = \text{CalculateCurrentForce}(); \\
& \quad \quad \quad \text{if} \ (F\text{new} > F) \ e = -a/2; \ F = F\text{new}; \\
& \quad \quad \text{shearf}(P, f); \ F\text{new} = \text{CalculateCurrentForce}(); \\
& \quad \quad \quad \text{if} \ (F\text{new} > F) \ f = -a/2; \ F = F\text{new}; \\
& \quad \quad \text{scalex}(P, sx); \ F\text{new} = \text{CalculateCurrentForce}(); \\
& \quad \quad \quad \text{if} \ (F\text{new} > F) \ sx = -sx/2; \ F = F\text{new}; \\
& \quad \quad \text{scaley}(P, sy); \ F\text{new} = \text{CalculateCurrentForce}(); \\
& \quad \quad \quad \text{if} \ (F\text{new} > F) \ sy = -sy/2; \ F = F\text{new}; \\
& \quad \quad \text{scalez}(P, sz); \ F\text{new} = \text{CalculateCurrentForce}(); \\
& \quad \quad \quad \text{if} \ (F\text{new} > F) \ sz = -sz/2; \ F = F\text{new}; \\
& \quad \quad \text{translatexyz}(P, C); \\
& \quad \}
\]

\[
\begin{align*}
& \quad \text{output}(P) \\
& \}
\end{align*}
\]

Acknowledgement

Thanks to Xusheng Wang, Bobbie Pew, Douglas Pew, Ying Zhu, and Xiaodong Fu for helping with the programming.

References


