Computing Similarities of Images: A Point Cloud-based Approach

Les A. Piegl[1] and Zachariah J. Beasley[2]

[1]University of South Florida, lespiegl@usf.edu
[2]University of South Florida, zjb@usf.edu

Corresponding author: Les A. Piegl, lespiegl@usf.edu

Abstract. This paper presents a methodology to measure the degree of similarity between two images quantitatively. The research was motivated by a medical application to search a database of MRI/CT images for comparative medical diagnosis. The proposed algorithm converts the image into a point cloud, fits a NURBS (planar) surface to the points, and uses symbolic algebra on these surfaces to determine which image is most similar to a given query image.

Keywords: point cloud, medical images, NURBS, similarity.
DOI: https://doi.org/10.14733/cadaps.2023.507-521

1 INTRODUCTION

The main idea behind medical imaging is to send invisible rays through the body and see what percentage of the energy is lost as it passes through different body parts. Or, in engineering terms, one measures the linear attenuation of the rays after passing through the body. The measure that is widely used is the Hounsfield unit of radiodensity, called HU [10]:

$$HU = 1000\left(\frac{\lambda - \lambda_{water}}{\lambda_{water} - \lambda_{air}}\right)$$

λ *linear attenuation coefficient*

$$\Delta HU = 1 \rightarrow 0.1\%\ of\ \lambda$$

Air	$-1{,}000$
Water	0
Kidney	$+30$
Soft tissue	$+100\ to\ +300$
Bone	$+700\ to\ +3{,}000$

That is, one HU represents 0.1% of attenuation. The scale goes from -1,000 (air) through 0 (water) to about 3,000 (dense bone). In other words, the CT scanner can return about 4,000 levels of intensities. Typically, these numbers are converted to gray-scale intensities, and a gray-scale image is created, which is what the doctor looks at. Gray-scale images on most computer displays have 256 intensities, i.e., the CT data of 4,000 intensities is reduced to 256. While this seems like a tremendous reduction in resolution, our eyes are terrible visualization instruments. We have spatial and temporal resolution limitations, and we can see only about 30-40 different intensities in a given color spectrum (draw two gray boxes on the display and increase the intensity by one until it is clear that the two boxes are different – it may take a jump of 5-8 intensity levels to distinguish the two boxes). The spatial and temporal limitations turn out to be a blessing in other areas: we enjoy color on the computer screen (the red, green, and blue dots are fused together into a new color), and the movie industry keeps us entertained with images that change more than about 30 times per second.

Unfortunately, the intensity limitation is a curse when analyzing CT scans: while the CT device returns a lot of information, the naked eye can see only about 1% of it. The information is there; however, its interpretation with the naked eye is helpful only in the most obvious cases.

This paper argues that medical diagnosis using images only is not only inadequate; it can save or lose a life under certain conditions. We propose a more objective and intelligent image analysis with the following elements:

- Create a database of images of past patients.
- Tag each image with relevant diagnostic information such as surgical and pathological reports.
- Search the image database for each new patient and each image to find the best match.
- Study the diagnostic information of the best match for a more intelligent medical assessment.

Even though the new CT image may not show anything suspicious, the accompanying information might be helpful for more in-depth investigations, instead of declaring that "nothing is wrong" because "nothing looks to be wrong."

The prior art relevant to this paper can be found in several US patents and a few publications. Three of these patents are worth mentioning, although they are only loosely related to this paper. Abdel-Mottaleb et al. [1] use clustering to store images. The similarity is established using clusters, and a search is done within each cluster. Chong et al. [2] use the color composition of areas to determine similarity. The metric that is used is the number of times a color appears in an area. Manson et al. [5] turn the image into a point cloud and cluster the points to form a triangle. The cluster-generated triangles are used to establish similarity.

Perceptual image hashing [12][13] creates a hash vector out of an image. The Hamming distance between the hash vector is used to establish similarity.

The organization of the paper is as follows. Section 2 shows the overview of the proposed approach. Section 3 illustrates how little data is needed for similarity computation. Section 4 deals with point cloud generation, followed by Section 5 on how to map point clouds on top of each other. Section 6 is devoted to NURBS surface fitting to planar point clouds, followed by Section 7 on how to use surfaces to establish similarity. In Section 8, several examples are given with numerical data illustrating the accuracy of the approach. A conclusion section summarizes the paper and offers some future directions.

2 PROPOSED APPROACH

We propose a multi-stage approach to assist medical diagnosis using images. The raw images are converted to NURBS surfaces in the first stage, as shown in Fig. 1. Because these images will be used for similarity detection, not for visualization, many simplifications can be done. The image should be clipped, the resolution can be reduced, and the intensities can be compressed. The following section gives details on how much savings are possible.

Once the simplification is done, a point cloud is generated: each picture element is represented by a set of points based on the intensity. This step is critical as large point clouds can produce an explosion of NURBS data. A NURBS surface then fits the point cloud, and both the image and the NURBS surface are saved in the database for the next step of the multi-stage process.

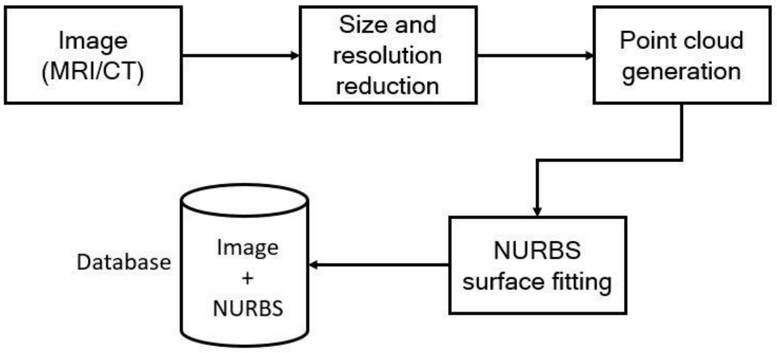


Figure 1: Pre-processing of images.

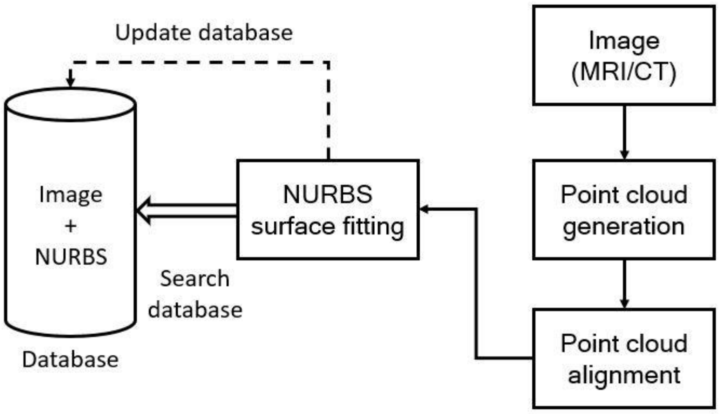


Figure 2: Searching for similar images.

Having completed the conversion of images into NURBS forms, searching for similar images in the data proceeds, as shown in Fig. 2. The new image, after simplification, is converted into a point cloud, and the point cloud is aligned to a particular position (patients are usually scanned in certain positions, so if the image is not in that position, e.g., rotated and/or scaled, alignment is necessary). A NURBS surface is then fit to the aligned point cloud for searching the database of NURBS surfaces of the previously converted images. The corresponding information is used for comparative diagnosis if a sufficiently similar image is found. Once the patient is diagnosed and treated, all relevant information is tagged to the image as well as the NURBS surface for future similarity search and diagnosis.

The NURBS surfaces generated from the images drive the similarity search in a database where images are augmented, as shown in Fig. 3. The database consists of three main parts. The first contains the raw images, used mainly for visualization. The second part has the NURBS surfaces fitted to the images, and they are used for similarity search. And the third contains all relevant medical data associated with the patient's images, used to assist diagnosis as well as treatment.

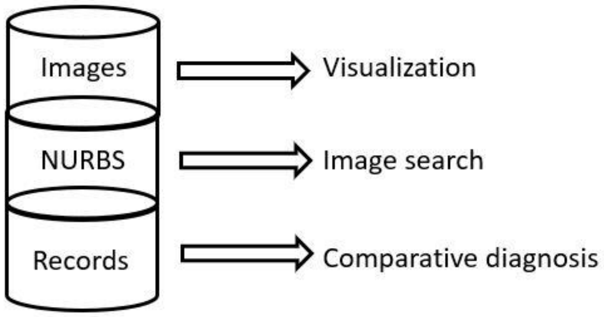


Figure 3: Partitioning an image-based database.

3 IMAGE DATA REDUCTION

While images are used mostly for visualization purposes, when it comes to performing quantitative analysis on them, the redundant data, not necessary for the analysis, needs to be removed. Fig. 4 shows three images from the Visible Human Dataset (courtesy of the National Library of Medicine, National Institute of Health) used in this research for testing purposes. All of them are shown as they come from the database. As it is evident, there is a lot of unnecessary information that can be and must be removed.

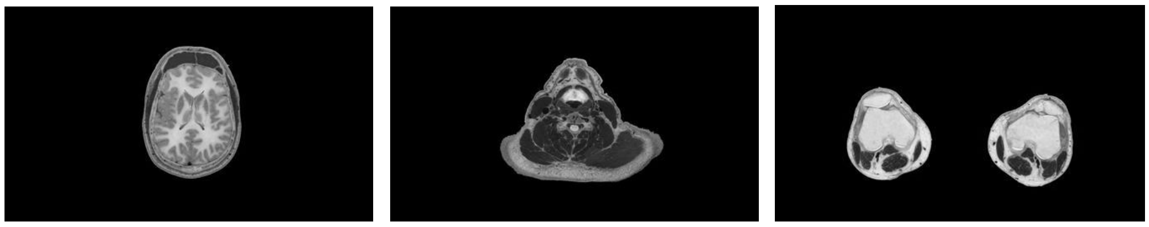


Figure 4: Examples from the Visible Human Dataset.

The images come in 1,760 x 1,024 resolution, with a grayscale depth of 256. Taking 128 as the average grayscale intensity and representing each intensity with the same number of points, e.g., intensity 25 is represented by 25 points, the number of points required to turn these images into a point cloud is 1,760 x 1,024 x 128 = 230,686,720. Feeding over 230 million points into the NURBS surface fitter would cause an explosion of control points, rendering the entire method nearly useless.

Fortunately, we do not need all these points as very little of this information is needed to establish similarity. First, we clipped the images: the head section to 750 x 750 and the other two to 1,500 x 750. Second, we reduced the sizes to 100 x 100 and 200 x 100, respectively. Third, the resolution was dropped by 2 in both directions, providing images of 50 x 50 and 100 x 50. This was a significant reduction; however, the full 256 grayscale was not necessary either, so we went with only 64 gray scales. Since on average only about 32 points are needed for each image element, the point clouds had 50 x 50 x 32 = 80,000 and 100 x 50 x 32 = 160,000 points. A reduction from the original size by more than 99%, i.e., only less than about 1% of the data, was necessary to establish similarity!

4 POINT CLOUD GENERATION

Having reduced the image to a more appropriate size and resolution, we next converted it to a point cloud. There are two issues to be addressed: (1) how many points to assign to each intensity and (2) how to place the points within each picture element. Since the intensity has been reduced from 256 to 64, we decided to keep the number of points the same as the intensity, e.g., intensity 16 receives 16 points.

The placement of the points within each picture element turned out to be a challenge. At first glance, it appears that a random distribution is called for since, visually, the random point set produced a more pleasant image. An example of the head section is shown in Fig. 5, left.

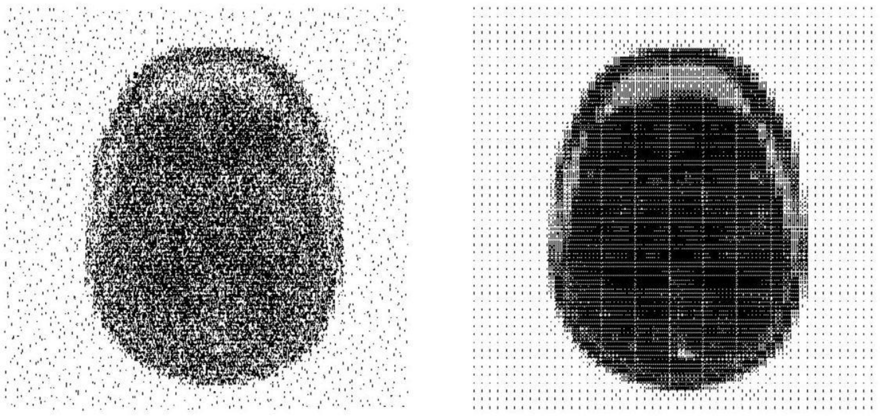


Figure 5: Random (left) and clustered dot pattern (right).

It turns out that even though the images were similar (or even identical), the fitted surfaces were different due to the randomness of the point cloud. When the random points were fit, similar sections produced poor similarity measurements, i.e., highly similar images were not recognized properly. What we needed was a point cloud generation method that produced similar (or identical) point clouds for similar (or identical) images. After some experimentation, the method using clustered dot patterns [4] produced good quality and consistent results.

Clustered dot pattern works with certain intensity levels: 4 (2x2), 16 (4x4), 64 (8x8), and 256 (16x16). Since we chose to reduce the 256-maximum intensity to 64, the dot pattern is defined by an 8 x 8 matrix called the dither matrix [11]. Dither matrices are defined recursively as follows:

$$M_2 = \begin{bmatrix} 0 & 2 \\ 3 & 1 \end{bmatrix} \qquad M_{2n} = \begin{bmatrix} 4M_n & 4M_n + 2 \\ 4M_n + 3 & 4M_n + 1 \end{bmatrix}$$

For each picture element, an 8 x 8 dither matrix is used to generate a clustered dot pattern as follows. The picture element is divided uniformly into 8 x 8 cells. For any given intensity, the cell receives a point if the value in the corresponding dither matrix is less than the intensity. As an example, let us consider the smaller 4 x 4 case with intensity 9:

$$M_4 = \begin{bmatrix} 0 & 8 & 2 & 10 \\ 12 & 4 & 14 & 6 \\ 3 & 11 & 1 & 9 \\ 15 & 7 & 13 & 5 \end{bmatrix} \quad \rightarrow \quad \begin{bmatrix} \bullet & \bullet & \bullet & \\ & \bullet & & \bullet \\ \bullet & & \bullet & \\ & \bullet & & \bullet \end{bmatrix}$$

The dots on the right are placed inside the picture element. Once all picture elements are processed, the image shown in Fig. 5 right is obtained. Naturally, it is not as pleasant to the human eye as the one on the left; however, this point arrangement is precisely what was needed to define and quantify similarity.

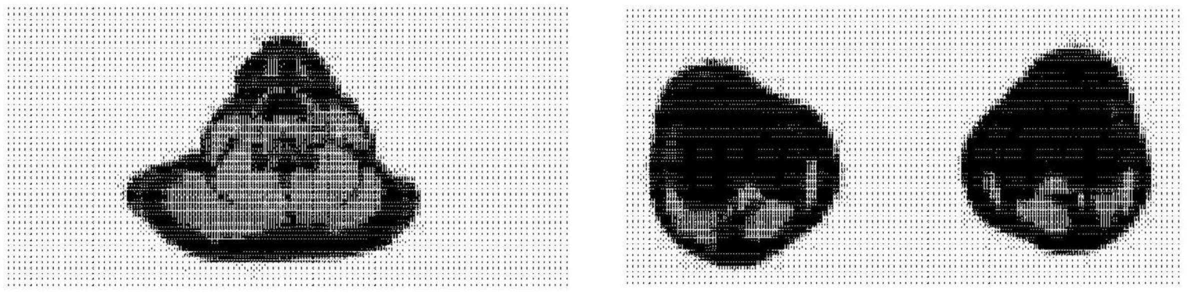


Figure 6: Clustered dot pattern of test data in Fig. 4.

Clustered dot patterns of the neck and the leg sections of Fig. 4 are shown in Fig. 6. Note how different the dot patterns are in the background parts of the images in Fig. 5. The random-dot arrangements in the background parts are exactly what threw off the similarity computation. The background color tends to be the same for each image; however, turning it into a random dot pattern gives it an identity that makes similarity challenging to compute.

5 POINT CLOUD ALIGNMENT

Given two sets of point clouds obtained from converting the images into points, we want to align these points to lay on top of each other. We need four transformations: translation, scaling, rotation, and window-to-viewport (box to box) mapping. Best fitting lines and circles [9] have been used to facilitate the first three, whereas the box mapping needs the bounding boxes only.

While the computation of best-fitting lines is relatively straightforward, computing the best circles can be a challenge. The first challenge is to decide what to minimize: the residuals, the minimum distances, or the power of points. We decided to minimize the power of points, which can be expressed as follows:

$$\begin{aligned} f_i(u_1,\ldots,u_3) &= r^2 - d_i^2 \\ &= r^2 - (x_i - a)^2 + (y_i - b)^2 \end{aligned}$$

where $C = (a,b)$ is the center of the circle, r is the radius, and $u_1 = a, u_2 = b, u_3 = r$. This formulation can be used to solve the minimization problem by linear least-squares and produces pretty good results. However, for high-quality similarity tests, a better-quality circle fit is needed [9]. It can be obtained by further improving this initial circle fit by minimizing actual distances:

$$\sum_{i=1}^{n} d_i(\mathrm{u})^2 = \min$$

where $\mathrm{u} = (u_1, u_2, u_3) = (a, b, r)$ and $d_i(\mathrm{u}) = r_i - r = \sqrt{(x_i - a)^2 + (y_i - b)^2} - r$. This is a non-linear least-squares problem that can be solved using the Gauss-Newton method [3]:

$$J\Delta = D$$

$$J = \begin{bmatrix} (a - x_1)/r_1 & (b - y_1)/r_1 & -1 \\ (a - x_2)/r_2 & (b - y_2)/r_2 & -1 \\ \vdots & \vdots & \vdots \\ (a - x_n)/r_n & (b - y_n)/r_n & -1 \end{bmatrix}$$

$$\Delta = \begin{bmatrix} \Delta a \\ \Delta b \\ \Delta r \end{bmatrix} \quad D = \begin{bmatrix} r - r_1 \\ r - r_2 \\ \vdots \\ r - r_n \end{bmatrix}$$

While this formulation seems computationally straightforward, there are a few issues to be addressed: (1) the convergence is linear, (2) the iteration may not converge, and (3) convergence may not guarantee acceptable results. An algorithmic solution was deployed that uses the Gauss-Newton iteration; however, it does not take the results for granted. The steps are as follows:

Step 1: check if the point cloud is collinear and, if yes, fit a line and exit.

Step 2: get an initial circle by linear least-squares, and set $\mathrm{u}^{(0)} = (a, b, r)$.

Step 3: compute the average relative error $av_{err}^{(0)}$ of the initial fit.

Step 4: iterate, while the maximum allowed iteration limit is not reached, and do:

Step 4.1: solve the above equation to get $\mathrm{u}^{(k+1)} = (a + \Delta a, b + \Delta b, r + \Delta r)$.

Step 4.2: compute the new average error $av_{err}^{(k+1)}$.

Step 4.3: if $av_{err}^{(k+1)} < av_{err}^{(k)}$, save the center and radius as the best fit so far.

Step 4.4: check for convergence:

$$\text{If } \left| av_{err}^{(k+1)} - av_{err}^{(k)} \right| < \gamma, \text{ done}$$

$$\text{Else } \left| \mathrm{u}^{(k+1)} - \mathrm{u}^{(k)} \right| < \delta, \text{ done}$$

Step 5: output center, radius, and average error.

The tolerances γ and δ are used to measure convergence (set to 10^{-5} in our implementation). The algorithm stops if the average error does not change much or if there is no improvement in the u-vector. Figure 7 shows the results of line and circle fits to two random sets of points.

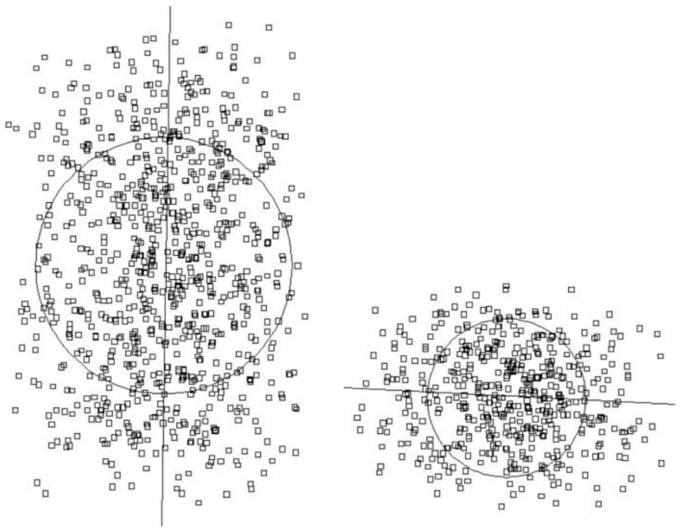


Figure 7: Fitting lines and circles to point clouds.

Testing results indicate that for image point-cloud data sets, the method above yields excellent results — both the line as well as the circle, capture the orientation of the point-cloud quite well.

Using the line and the circle fits, the point clouds are aligned as follows. A translation is performed so that the circle centers overlap. A scaling is then used to make the two circles identical (same center and radius). Finally, a rotation is employed so that one fitted line is rotated into the other. Denoting the angle between the two lines by alpha, three rotations may be needed to align the point sets: one with the angle alpha, another with 180+alpha, and a third with 90+alpha. This is because some point sets can be axially symmetric (must try rotation with alpha and 180+alpha) or can be themselves symmetric, e.g., forming a square shape (must try alpha and 90+alpha).

We tested several data sets, and this method worked well. Dissimilar data sets could be aligned well and agreeable to any qualitative measures. Data sets that were identical with respect to a scaling factor could be aligned perfectly well.

The final step in the alignment process is to bring the bounding boxes together, i.e., to map one point cloud so that its bounding box is identical to the other. This should not be necessary; however, to account for asymmetric scaling and assist the NURBS fitting, the point clouds were subjected to a window-to-viewport transformation. Such a transformation, making minimal changes to prepared medical images, does not impact the similarity of point clouds.

6 B-SPLINE SURFACE FITTING

The two sets of point clouds, brought in alignment in the previous step, are now fitted with two *planar* B-spline surfaces. We review a few B-spline formulas to make the discussion below more understandable. A B-spline surface of degrees (p,q) is defined as follows [6]:

$$S(u,v)=\sum_{i=0}^{n}\sum_{j=0}^{m}N_{i,p}(u)N_{j,q}(v)P_{i,j}$$

where $P_{i.j}=(x_{i,j},y_{i,j},z_{i,j})$ are the control points, and $N_{i,p}(u)$, $N_{j,q}(v)$ are the normalized B-spline basis functions defined over the knot vectors:

$$U = \left\{ \underbrace{u_0 = \cdots u_p}_{p+1}, u_{p+1}, \cdots, u_n, \underbrace{u_{r-p} = \cdots = u_r}_{p+1} \right\} \quad r = n + p + 1$$

$$V = \left\{ \underbrace{v_0 = \cdots v_q}_{q+1}, v_{q+1}, \cdots, v_m, \underbrace{v_{s-q} = \cdots = v_s}_{q+1} \right\} \quad s = m + q + 1$$

The fitting algorithm is looking for a B-spline surface that interpolates/approximates the point clouds in the B-spline sense, i.e., for any given data point P_k, there is a pair of parameters (u_i, v_j), so that the surface passes through or near that point: $S(u_i, v_j) = P_k$ *or* $S(u_i, v_j) \approx P_k$. The fitting is performed in two steps: (1) data preparation and (2) surface fitting.

In order to prepare the data for surface fitting, the data needs to be arranged to form a rectangular topology. The algorithm does that as follows:

Step 1: lay a grid over the points so that there is only one point in every cell on average. This can be accomplished by choosing the size of the grid to be: $size = \sqrt{xd \cdot yd / K}$ where xd and yd denote the side lengths of the bounding box and K is the number of points.

Step 2: bin the points into the cells while eliminating coincident points.

Step 3: add phantom points to empty cells. This step is not required for image data as the background color is also represented with a dot pattern. However, to make the method more general, we added the midpoints of empty cells to the point set. This aids the B-spline fitting; however, it does not alter the similarity (which is based on point distribution, and adding the midpoints of a uniform grid will not change that).

Step 4: collect and sort the points column-by-column.

The result of this point processing is a set of rectangularly arranged points in the form:

$$P_{i,j}, i = 0, \cdots, n; j = 0, \cdots, m_i$$

That is, for each column, the number of points can be different. Traditional surface fitting to a non-$N \times M$ data set will not work. However, this kind of data is a perfect setup for a cross-sectional design as follows [7]:

Step 1: fit curves of the same degree to the column of data points independently.

Step 2: make the curves compatible in the B-spline sense, i.e., merge knot vectors.

Step 3: fit a surface through the compatible network of curves.

Because the curves fit independently, each one may end up with a different knot vector. This, in turn, can produce a control point explosion during the knot merging process. Assume that there are n columns and that each column has on average m number of internal knots. If all these knots are different, then after merging, the maximum number of control points can be as high as $(m \times n) \times n = mn^2$. Whereas if the internal knots are the same, the number of control points will be in the order of mn. That is, even for a simple 100 x 100 image, this adds up to 100 times more control points than desired. Because the cross-sectional curves are all independently obtained, we can never get the ideal mn configuration. However, we can come close to this number by using a master knot vector and fuzzy knots (knot intervals) as follow [8]:

Step 1: find the column of points with the largest number of points.

Step 2: compute the knot vector to fit this point set and call it the master knot vector.

Step 3: create brackets around each master knot u_i^p where p is the degree of the fitting:

$$u_{i-1}^{p-1} < u_i^p < u_i^{p-1}$$

Step 4: generate a membership interval (a,b) around each master knot:

$$a = (1-per)\cdot u_i^p + per \cdot u_{i-1}^{p-1} \qquad b = (1-per)\cdot u_i^p + per \cdot u_i^{p-1}$$

where per denotes the percentage of membership (100% provides maximum flexibility, whereas 0% gives none – a good default is 75%). Figure 8 illustrates the idea.

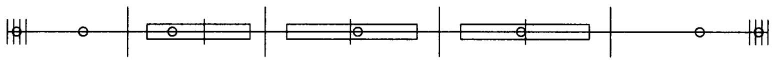


Figure 8: Fuzzy knots for curve fitting: per = 75%.

Step 5: for each column of points, do:
- *Step 5.1*: compute the knots for fitting.
- *Step 5.2*: if a knot falls into a membership interval, use the master knot.
- *Step 5.3*: otherwise, use this knot, and update the master knot vector by inserting this knot.
- *Step 5.4*: fit the curve with a mix of master knots and newly computed knots.

This method makes use of existing knots to minimize the introduction of new knots. As the fitting proceeds from column-by-column, the master knot vector gets dynamically updated to the point where no new knots need to be used. In the end, the common knot vector is the master knot vector, and all the knot merging process needs to do is add those master knots that are not present in individual curve knot vectors. To appreciate this fitting method, let us look at Table 1 below.

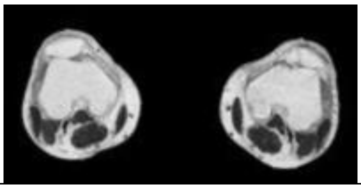
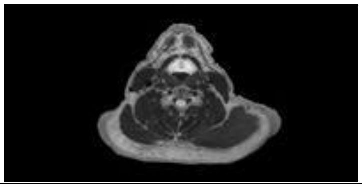
Images			
Data points	114,625	80,621	32,230
Control points	338 x 1,561	283 x 1,657	179 x 1,342

Table 1: Fitting efficiency using fuzzy knots.

The first column shows that to fit 114,625 points requires 1,561 control points in the fitting direction. Traditional fitting with straight control point merging would have needed around 114,000 control points (mn^2). Extensive testing shows that, on average, savings of over 90% can be obtained with $per \in [50\%, 75\%]$. Going higher than 75% or so may result in the matrices, to be solved for fitting, becoming poorly conditioned. On the other hand, as one approaches zero with the percentage of membership, the number of control points increases considerably. For the first image above, the number of control points can go from the current 527,000 to over 38 million!

This is not only prohibitively large; it is absolutely unnecessary because the extra control points do not contribute to the accuracy or the quality of the fit.

7 IMAGE SIMILARITY

The fitting method discussed in the previous section is now used to compute the similarity of two sets of points (obtained from images). Given point sets $P = P_0,\cdots,P_n$ and $Q = Q_0,\cdots,Q_m$, we first fit surfaces $S_P(u,v)$ and $S_Q(u,v)$ to these points. Then the difference surface is computed via symbolic algebra:

$$D(u,v) = S_P(u,v) - S_Q(u,v)$$

The difference surface is just the origin for identical surfaces, i.e., all the control points of $D(u,v)$ being $R_{i,j} = (0,0,0)$. For surfaces that are not identical, the distances of the control points from the origin are greater than zero. So, the similarity is defined as follows:

$$sim = \frac{Ave\ \|R_{i,j}\|}{Diag}$$

where $\|R_{i,j}\|$ is the vector norm of the control points, $Diag$ is the diagonal of the bounding box of the surfaces, and Ave denotes the average of the control vectors. That is, the similarity is not defined by the largest deviation but rather by the average deviation. Also, the average deviation is scaled by the diagonal of the bounding box to make it independent of the size of the image.

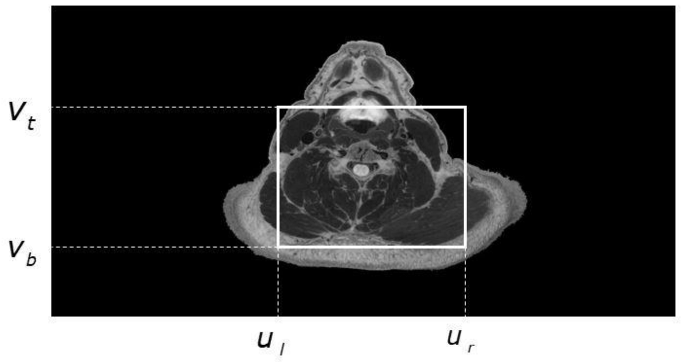


Figure 9: Extracting the domain of interest.

A powerful capability of this similarity method is that it can restrict the computation to the area of interest, as in Figure 9. When analyzing images, especially medical images, there is always an area of interest, and there are areas that are just background noise. These areas of background noise can be excluded by extracting the sub-surface $D_{[u_l,u_r]\times[v_b,v_t]}(u,v)$ by specifying the parametric sub-domains $[u_l,u_r]$ and $[v_b,v_t]$ in u- and v-direction. Similarity computation is then performed only on this sub-surface.

8 TESTS AND EXAMPLES

To test the feasibility and effectiveness of the proposed method, we used slices from the Visible Human Dataset. If the algorithm works, neighboring slices should be more similar than far apart slices. Table 2 compares three slices: 1081, 1093, and 1094. It is assumed that the surfaces are parametrized over the unit square [0,1] x [0,1].

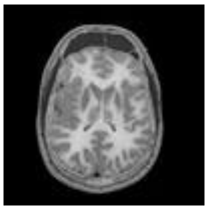
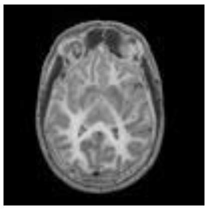
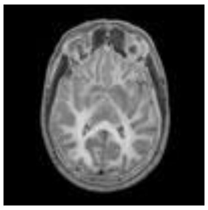
Areas of interest	 P=1081	 Q=1093	 R=1094
	$P \leftrightarrow Q$	$Q \leftrightarrow R$	$P \leftrightarrow R$
[0.0,1.0] x [0.0,1.0]	0.019363	**0.011520**	0.017232
[0.4,0.6] x [0.2,0.8]	0.031804	**0.013924**	0.028479
[0.0,0.2] x [0.0,1.0]	0.015742	**0.005033**	0.014998
[0.0,0.1] x [0.0,0.1]	0.009528	**0.002450**	0.009087
[0.0,0.5] x [0.0,0.5]	0.021718	**0.008188**	0.022999
[0.5,1.0] x [0.5,1.0]	0.016638	0.015238	**0.011216**

Table 2: Similarities among slices 1081, 1093, and 1094.

Interestingly, the algorithm declares slices 1093 and 1094 the most similar images in all cases, except when the top right corners are compared. The discrepancy can be due to noise or too low sampling. Next, we investigated similarities between neighboring slices in Table 3.

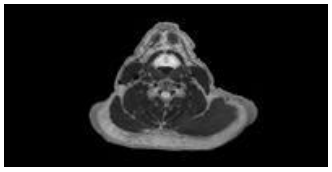
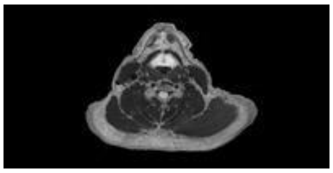
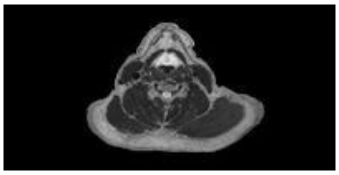
Areas of interest	 P=1240	 Q=1241	 R=1242
	$P \leftrightarrow Q$	$Q \leftrightarrow R$	$P \leftrightarrow R$
[0.0,1.0]x [0.0,1.0]	0.029672	0.026142	**0.021195**
[0.3,0.7]x [0.2,0.8]	0.041341	**0.037936**	0.040668
[0.4,0.6]x [0.4,0.6]	0.032846	**0.031072**	0.037612
[0.3,0.5]x [0.6,0.8]	0.035156	**0.030187**	0.033072
[0.0,0.2]x [0.0,0.2]	0.026512	0.016042	**0.007827**
[0.4,0.5]x [0.2,0.8]	0.021805	**0.019417**	0.026757

Table 3: Similarities among neighboring slices, 1240, 1241, and 1242.

There are two things worth mentioning. The first is that the similarities are very close and understandably so. The second is that the background color (noise) tends to throw off the metric, although the discrepancy is minimal.

The third set of examples is shown in Table 4, using slices 2300, 2301, and 2309.

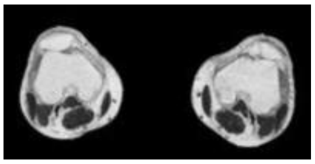
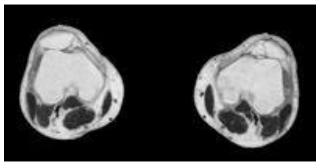
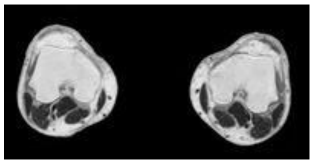
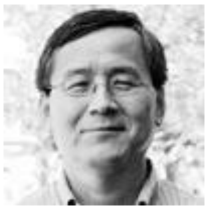
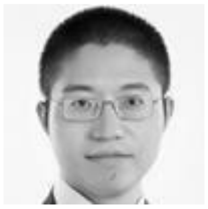
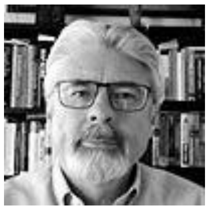
Areas of interest	 P=2300	 Q=2301	 R=2309
	$P \leftrightarrow Q$	$Q \leftrightarrow R$	$P \leftrightarrow R$
[0.0,1.0]x [0.0,1.0]	**0.015913**	0.032858	0.034466
[0.2,0.4]x [0.2,0.6]	**0.009216**	0.019940	0.020032
[0.6,0.8]x [0.2,0.7]	**0.017321**	0.040250	0.041599
[0.45,0.55]x [0.45,0.55]	**0.033740**	0.054789	0.059058
[0.48,0.52]x [0.0,1.0]	**0.034673**	0.053971	0.059147
[0.0,1.0]x [0.4,0.6]	**0.014911**	0.033235	0.034976

Table 4: Similarities among neighboring slices, 2300, 2301, and 2309.

The results are consistent; neighboring images are more similar no matter how we zoom in. The final set of images is shown in Table 5. So far, we have tested slices from the Visible Human Dataset. We were interested in testing the method on images of faces. The faces shown in the table below do not represent a close similarity. In fact, they seem to be entirely dissimilar. Yet, the question remains: which pair of faces would a human declare the most similar, and which one would the algorithm choose? We leave it to the reader to decide before looking at the decision of the algorithm.

Areas of interest	 P	 Q	 R
	$P \leftrightarrow Q$	$Q \leftrightarrow R$	$P \leftrightarrow R$

[0.0,1.0] x [0.0,1.0]	0.050683	**0.038743**	0.065256
[0.4,0.6] x [0.1,0.9]	0.057151	**0.032226**	0.100219
[0.0,0.2] x [0.0,1.0]	0.059920	**0.052283**	0.066503
[0.4,0.6] x [0.4,0.6]	0.057688	**0.032273**	0.100367
[0.45,0.55] x [0.48,0.52]	0.056100	**0.032873**	0.103830
[0.0,1.0] x [0.0,0.2]	0.049431	**0.037884**	0.064119

Table 5: Similarities of different facial images.

The results are quite compelling: no matter how we zoom in, images Q and R are declared the most similar, and in many cases, by a large margin. Even though the facial characteristics (the hair, the glasses, the facial hair, and the smiles) are noticeably different, the method declares Q and R the winners. There may be a discrepancy between visual appeal and the dynamics that is inherent in the point distribution.

9 CONCLUSIONS

A point cloud-based method of similarity computation is presented in this paper. The approach addresses a few technical challenges: (1) how to turn an image into a point cloud, (2) what is the optimum number of points to obtain acceptable similarity metrics, and (3) how to arrange the points for B-spline surface fitting, and (4) how to use the B-spline surfaces to get a similarity metric. The advantage of the method lies in its simplicity and generality: it works with points, and one can choose the required level of detail (accuracy) by varying the intensities and the resolution.

There are a few disadvantages to the approach. First, it requires a powerful suite of B-spline functions to fit the point cloud. And the second is that the number of points in the cloud can be quite large, making the algorithm run slowly. On a 100,000-strong data set, the algorithm takes a few seconds to compute similarity using an off-the-shelf desktop machine. This includes point cloud generation, point cloud alignment, B-spline fitting, and the computation of symbolic difference. The similarity is computed quickly once the images are converted and stored as B-splines.

Les A. Piegl, https://orcid.org/0000-0003-0629-8496
Zachariah J. Beasley, https://orcid.org/0000-0002-0146-2739

REFERENCES

[1] Abdel-Mottaleb, M. S.; Krishnamachari, S.: Image retrieval system using a query image, US Patent No: 6,285,995 B1, September 4, 2001.
[2] Chong, K. M.; Li, Y.: System for computationally quantifying similarities between images, US Patent No: 8,498,473 B2, July 30, 2013.
[3] Fletcher, R.: Practical Methods of Optimization, John Wiley & Sons, New York, NY, 1987.
[4] Lau, D. L.; Arce, G. R.: Modern Digital Halftoning, Marcel Dekker, Inc., New York, NY, 2001.
[5] Manson, S. J.; Trumbull, T. L.: Image recognition system and method for identifying similarities in different images, US Patent No: 9,092,967 B2, July 2015.
[6] Piegl, L. A.; Tiller, W.: The NURBS Book, Springer-Verlag, Heidelberg, Germany, 1997. https://doi.org/10.1007/978-3-642-59223-2
[7] Piegl, L. A.; Tiller, W.: Surface approximation to scanned data, The Visual Computer, 16(7), 2000, 386-395. https://doi.org/10.1007/PL00013393
[8] Piegl, L. A.; Tiller, W.: Reducing control points in surface interpolation, IEEE Computer Graphics, and Applications, 20(5), 2000, 70-74. https://doi.org/10.1109/38.865883

[9] Piegl, L. A.; Tiller, W.: Fitting NURBS spherical patches to measured data for reverse engineering, Engineering with Computers, 24, 2008, 97-106. https://doi.org/10.1007/s00366-007-0076-8

[10] Romans, L. E.: Computed Tomography for Technologists: A Comprehensive Text, Wolters Kluwer, Philadelphia, PA, 2019.

[11] Ulichney, R.: Digital Halftoning, The MIT Press, Cambridge, MA, 1987.

[12] Venkatesan, R.; Koon, S.-M.; Jakubowski, M. H.; Moulin, P.: Robust image hashing, Proc. IEEE Int. Conf. Image Processing, 3, 2000, 664-666.

[13] Zeng, J.; A novel block-DCT and PCA based image perceptual hashing algorithm, IJCSI International Journal of Computer Science Issues, 10(3), 2013, 399-403.