

Shape Complexity from Image Similarity

Danyi Wang, Alexander Belyaev,
Waqar Saleem, Hans-Peter Seidel

MPI-I-2008-4-002 October 2008

Authors' Addresses

Danyi Wang
PITERION GmbH
Hanns-Klemm-Str. 5
71034 Böblingen
Germany

Alexander Belyaev
School of Engineering and Physical Sciences
Heriot-Watt University
Edinburgh EH14 4AS
United Kingdom

Waqar Saleem, Hans-Peter Seidel
Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85
66123 Saarbrücken
Germany

Acknowledgements

All shape models for this work have been taken from the AIM@SHAPE Shape Repository.

Abstract

We present an approach to automatically compute the complexity of a given 3D shape. Previous approaches have made use of geometric and/or topological properties of the 3D shape to compute complexity. Our approach is based on shape appearance and estimates the complexity of a given 3D shape according to how 2D views of the shape diverge from each other. We use similarity among views of the 3D shape as the basis for our complexity computation. Hence our approach uses claims from psychology that humans mentally represent 3D shapes as organizations of 2D views and, therefore, mimics how humans gauge shape complexity. Experimental results show that our approach produces results that are more in agreement with the human notion of shape complexity than those obtained using previous approaches.

Keywords

shape complexity, shape views, boundary contour extraction, view similarity, shape similarity, Similarity Structure Analysis, Multidimensional Scaling

Contents

1	Introduction	2
1.1	Previous Work	2
1.2	Overview	3
2	Constructing a View Similarity Matrix	6
2.1	Shape Views	6
2.2	Boundary Contour Extraction	7
2.3	Comparing Views	8
2.4	Similarity Matrix	10
3	Shape Complexity from View Similarity	11
3.1	Starting Positions	11
3.2	Checking for an SSA Solution	12
3.3	Updating Point Positions	12
3.4	Stopping Condition	13
3.5	Computing Shape Complexity	14
4	Results and Conclusion	16
4.1	Discussion	20

1 Introduction

More and more real world objects are being digitized and efforts into creation of original 3D content are also increasing. With the resulting proliferation of digital 3D shapes, it is becoming necessary to be able to organize them in some manner. One such means is to sort shapes according to their complexity. Given a collection of shapes, humans can easily estimate which shapes are more or less complex than the others. Very few such automatic measures exist for digital shapes.

1.1 Previous Work

It is not entirely clear how exactly to define and hence compute the complexity of a shape. In [17], an attempt has been made to formalize the notion of shape complexity by defining a few measures that could lead to its estimation.

- *Algebraic complexity* is the degree of the polynomial used to represent the shape.
- *Morphological complexity* is an estimate of the amount of fine details in the shape, and is computed as the largest value of r for which the shape is r -smooth or r -regular.
- *Combinatorial complexity* is the number of vertices used in the shape representation.
- *Representational complexity* is a qualitative measure of the amount of redundancy in the shape representation.
- *Topological complexity* is also a qualitative measure comprising of the genus of and non-manifold elements in the shape.

While these measures may capture some aspects of how humans gauge shape complexity, they are limited in terms of how they can be applied for automatic complexity estimation. The first two measures are restricted to specialist shape representations and all three quantitative measures are rather loose – shapes of varying complexities can easily end up having the same values for these measures.

More discriminative approaches have involved the use of information theory [19]. The canonical simplest shape is the sphere. There is no variation in its surface and its appearance is constant, up to scale, from all directions. Page and colleagues [13, 21] attribute this to the sphere’s property of having the same curvature throughout. Hence, they view the curvatures of shape vertices as a probability distribution and compute the shape’s complexity as the entropy of this distribution. This approach however is sensitive to noise. Small amounts of noise on the surface of the shape can cause large variations in curvature, thus leading to a high complexity value when computed using the above methods, whereas the shape itself does not change much.

The method proposed in [15] builds upon the observation that inside the sphere, each surface point is visible from every other surface point. A shape’s *inner complexity* is then measured in terms of the *mutual information* between regions of the shape that are mutually visible to each other through the shape’s interior. An *outer shape complexity* is also proposed that considers visibility between regions of the shape and a bounding sphere.

1.2 Overview

It is claimed in psychology [6, 10] that humans perceive 3D objects as arrangements of 2D views. This observation has been exploited for object recognition [8], to compute shape similarities [7] and to extract static [24] and dynamic [18] representations of 3D shapes.

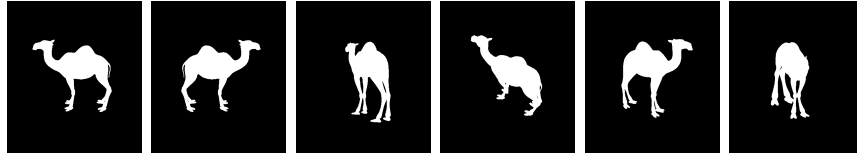
We propose that it is lack of (dis)similarity among views of the sphere, the canonical simplest shape, that leads to its low complexity, and claim that a shape is complex if the views corresponding to it are different from each other. A shape is more complex than another if the difference, or dissimilarity, between its corresponding views is greater than that for the other shape’s views.

Consequently, our approach to complexity computation is based on similarity between shape views. Given a shape, we obtain N views of it, and compare each view to the others, leading to a $N \times N$ similarity matrix, \mathbf{S} . For view comparison, we first acquire binary views from which we extract the boundary contours. These contours are then compared to each other using

the Contour to Centroid Triangulation (CCT) method [3]. In Chapter 2, we explain each of the above steps in detail. Applying Similarity Structure Analysis (SSA) [4, 5] to \mathbf{S} yields a 2D plot with N points where the distance between the points in the plot depends on the relative magnitude of entries in \mathbf{S} . Thus, a “simple” shape, whose views are relatively similar to each other, will correspond to an SSA plot containing points close to each other. A more complex shape will lead to a plot in which the points are more spread out. We therefore measure the complexity of the shape as the amount of dispersion of points in its SSA plot obtained in the above manner. In Chapter 3, we explain the SSA method and how we measure complexity values from the obtained plots. Finally, some results are presented in Chapter 4 followed by some closing remarks. A detailed version of this report has been presented in [23] and a pictorial overview of our entire pipeline is given in Figure 1.1.



(a) A shape to be analyzed. (b) The shape's view sphere.



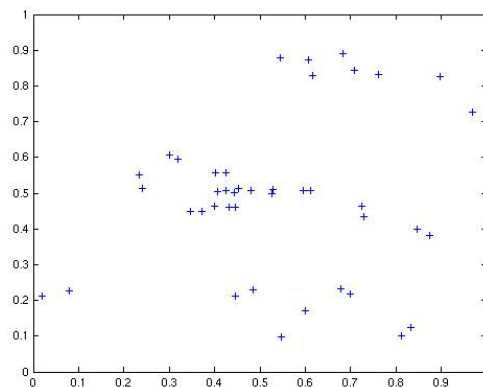
(c) Some views of the shape.



(d) Extracted boundary contours of the views.

$$\mathbf{S} = \begin{pmatrix} 0 & 0.130 & 0.246 & 0.236 & \dots \\ 0.130 & 0 & 0.222 & 0.220 & \dots \\ 0.246 & 0.222 & 0 & 0.123 & \dots \\ 0.234 & 0.220 & 0.123 & 0 & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

(e) Similarity matrix of boundary contours.



(f) SSA plot corresponding to \mathbf{S} .

Figure 1.1: An overview of our approach. Views of a shape are captured and their boundary contours are extracted and compared with each other to yield a similarity matrix, \mathbf{S} . The distances in \mathbf{S} are represented as a 2D plot using SSA. Point positions in the plot are then used to compute complexity.

2 Constructing a View Similarity Matrix

Given a shape, we first obtain N views of it from its view sphere. As a shape can be adequately represented by its silhouette, we make sure to obtain binary views as this simplifies the subsequent boundary contour extraction step. Once each view is represented as a boundary contour, we use a shape similarity method to compute similarity between all pairs of views. This yields the similarity matrix, \mathbf{S} , for the shape.

2.1 Shape Views

A shape's view sphere is a bounding sphere of the shape centered at the shape's geometric center, i.e. the center of the shape's bounding box. Each point on the view sphere is a viewpoint and the corresponding view is the one obtained by placing a camera at that point facing towards the center of the sphere. In order to view only the shape, the background is typically plain.

As the sphere has continuous surface area, placing cameras at all points on its surface is not only computationally infeasible but also impossible. We therefore approximate the view sphere with a platonic solid, namely an icosahedron, chosen for its triangle mesh structure. The icosahedron is set to have the same center and radius as the intended view sphere it is approximating. With only 12 vertices, the icosahedron provides a crude approximation of the view sphere. For finer approximations, iterations of Loop subdivision [11] followed by re-projection of vertices to the sphere are applied to it. Cameras are placed at the vertices of the resulting mesh. Views of the shape are then simply snapshots obtained from these cameras.

To obtain binary views like the ones shown in Figure 1.1, we render the shape with a white material and set the background to black before capturing

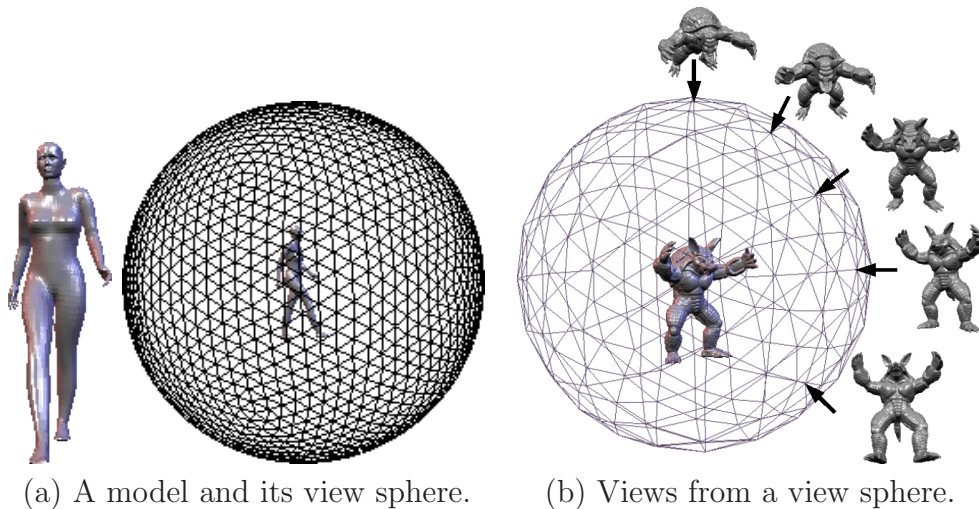


Figure 2.1: Note that the model, radius of the view sphere relative to the bounding box of the model, and the number of subdivision steps applied to the view sphere differ in each sub-figure.

snapshots from the virtual cameras.

2.2 Boundary Contour Extraction

Given a binary view image, the first step in extracting a closed, connected contour representing the boundary of the viewed shape is to identify pixels in the view image that lie on the shape boundary. This is done by scanning the image first row-wise and then column-wise. In each scan, we mark as boundary pixel the first shape pixel we come across when previous pixels belonged to the background, or vice versa. For more details, we refer the reader to [23], Chapter 3.

The row and column numbers of boundary pixels in the image give their coordinates, to which we apply the Crust method [1] to obtain boundary edges, see Figure 2.2. Given a set of points, \mathbf{P} , sampled on a curve, the method constructs the Voronoi diagram of \mathbf{P} to obtain the Voronoi vertices V of \mathbf{P} . It then performs a Delaunay triangulation [9] of $\mathbf{P} \cup V$. From the triangulation, edges that connect points in \mathbf{P} to each other then form the “crust”, C , of \mathbf{P} , which we take to be the boundary contour. This method has previously also been applied directly to 3D surface reconstruction [2]. We use the Triangle software [20] for Voronoi and Delaunay computations.

The crust, C , gives us an unordered set of edges with no connectivity information, i.e. given an edge in C , we do not know which is the next

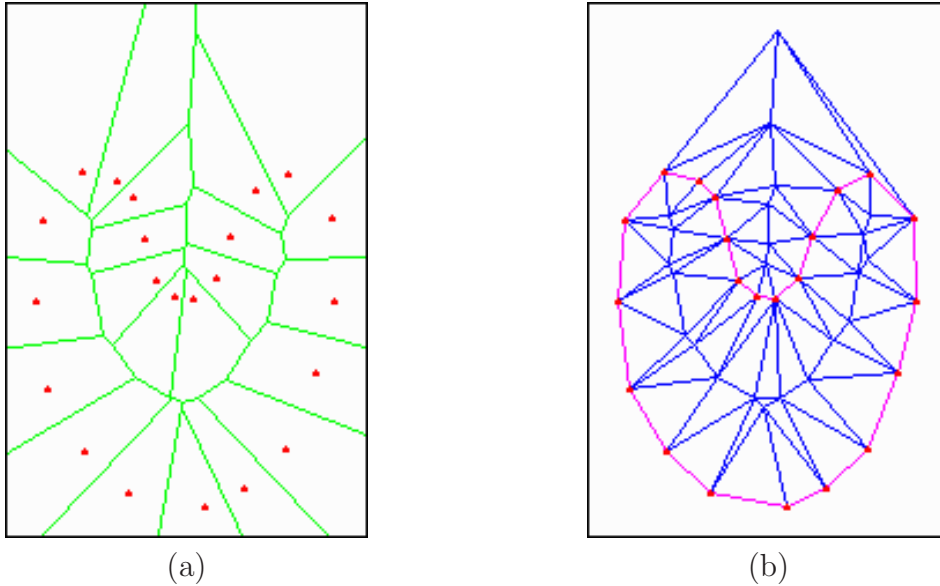


Figure 2.2: The two-dimensional crust algorithm, generated from [16]. (a) Voronoi diagram of a point set, \mathbf{P} (red), sampled from a curve. Voronoi vertices, V , and edges are in green. (b) Delaunay triangulation of $\mathbf{P} \cup V$. Crust edges are shown in pink.

or previous edge. This information is required for the similarity algorithm later. We pick the point, p_0 , in \mathbf{P} with the smallest x and y coordinates to initialize a list, $L = \{p_0\}$. Then we search C for an edge, (p_0, p_i) , for some other $p_i \in \mathbf{P}$. L is updated as $L = \{p_0, p_i\}$ and the edge is removed from C . The next incident edge (p_i, p_j) is then found, L is updated, $L = \{p_0, p_i, p_j\}$ and the edge is removed from C . This process is repeated until p_0 is reached again. At this point, L represents a closed, connected polyline representing the boundary of the shape, e.g. see Figure 1.1.

2.3 Comparing Views

To compare views, we use the Contour to Centroid Triangulation (CCT) algorithm [3] which has been reported [22] to outperform other shape similarity measures.

From a given boundary polyline, the farthest point from the center of the polyline shape is chosen and it is used as the starting point to segment the boundary. If more than one point is found, e.g. from the polyline of a circle, then any of its farthest points can act as the starting point. Going along the boundary starting from this point, the boundary is divided into

n equal length arcs where n can be between 10 and 75. A multi-resolution boundary representation can be used by using different number of segments per resolution and each resolution can describe finer details of the boundary. An overview of the descriptor is given in Figure 2.3.

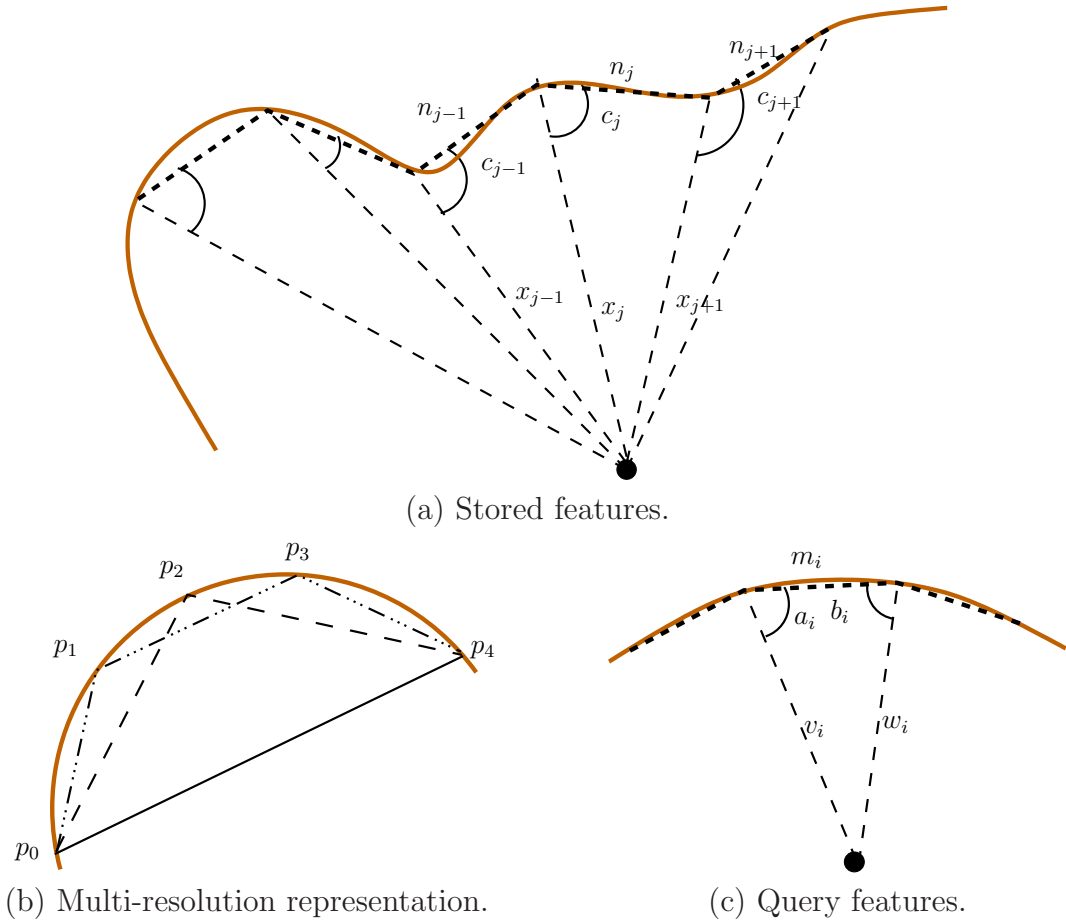


Figure 2.3: Shape descriptor for CCT, reproduced from [3]. (a) The shape boundary is segmented, triangulated with respect to the shape center and features are extracted. (b) The segment (p_0, p_4) is refined once into the segments $\{(p_0, p_2), (p_2, p_4)\}$ and a second time into $\{(p_0, p_1), (p_1, p_3), (p_3, p_4)\}$. (c) Features extracted from a query shape, to be compared with those of stored shapes.

Then three primitives (a, m, lw) per segment reading clockwise are defined as:

1. c , the start angle between the center extension and the chord. The angle that is less than or equal to 180° is used.

2. n , the ratio of chord length to arc length, denoting smoothness.

3. x , the distance from the segment's start to the center of the shape.

These parameters are normalized by dividing all by their maximum corresponding values. This normalization helps in making all the three primitives contribute equally to the descriptor. The normalized parameters are represented as a feature vector of the boundary polyline.

A similarity measure $D_f(X, Y)$, based on the sum of absolute differences, between all query local features $q_i = (a_i, b_i, m_i, v_i, w_i)$ of shape X and stored local features $s_j = (c_j, n_j, x_j)$ of shape Y , as shown in Figure 2.3, for every j rotations, where i and j are between 1 and f segments, is defined as

$$D1_f(X, Y) = \sum_i \text{Min}(|a_i - c_{j-1}| + |v_i - x_{j-1}| + |m_i - n_{j-1}|, \\ |a_i - c_j| + |v_i - x_j| + |m_i - n_j|, \\ |a_i - c_{j+1}| + |v_i - x_{j+1}| + |m_i - n_{j+1}|).$$

In order to account for reflected shapes:

$$D2_f(X, Y) = \sum_i \text{Min}(|b_i - c_{j-1}| + |w_i - x_{j-1}| + |m_i - n_{j-1}|, \\ |b_i - c_j| + |w_i - x_j| + |m_i - n_j|, \\ |b_i - c_{j+1}| + |w_i - x_{j+1}| + |m_i - n_{j+1}|).$$

The similarity measure between the query and stored shape is then the minimum of $D1$ and $D2$ or:

$$D_f(X, Y) = \text{Min}(D1_f(X, Y), D2_f(X, Y))/f,$$

where

$$D_f(X, X) = 0.$$

2.4 Similarity Matrix

Given N views, $\{v_1, v_2, \dots, v_N\}$, of the shape, and denoting the similarity between v_i and v_j as computed above by $s_{i,j}$, the similarity matrix, \mathbf{S} , for the shape is defined as

$$\mathbf{S} = \begin{pmatrix} 0 & s_{1,2} & s_{1,3} & \dots & s_{1,N} \\ s_{2,1} & 0 & s_{2,3} & \dots & s_{2,N} \\ s_{3,1} & s_{3,2} & 0 & \dots & s_{3,N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ s_{N,1} & s_{N,2} & s_{N,3} & \dots & 0 \end{pmatrix}$$

3 Shape Complexity from View Similarity

Similarity Structure Analysis (SSA), or Multidimensional Scaling (MDS) [4, 5], provides a tool for

... analyzing the structure of (dis)similarity data ... [It] represents the data as distances among points in a geometric space of low dimensionality. This map can help to see patterns in the data that are not obvious from the data matrices.

Given the $N \times N$ similarity matrix, \mathbf{S} , we apply SSA to it to obtain a 2D plot containing N points, where pairwise distances between the N points in the plot are related to the entries, $s_{i,j}, i, j \in \{1, \dots, N\}$, in \mathbf{S} .

Corresponding to the N views, $\{v_1, v_2, \dots, v_N\}$, N points, $\mathbf{P}_0 = \{\mathbf{p}_{i,0} | i \in \{1, \dots, N\}\}$, are chosen in the Cartesian plane. The original SSA method does not pose any restrictions on how these points are chosen. However, we specify a starting position of points which we describe later in Section 3.1. The distance matrix, $D(\mathbf{P}_m)$, of the set, $\mathbf{P}_m, m \geq 0$, is computed such that $d_{i,j}$ is the Cartesian distance between $\mathbf{p}_{i,m}$ and $\mathbf{p}_{j,m}$. It follows that $d_{i,i} = 0$ for all i .

In order to compute the SSA plot, the *starting configuration* is set as $\mathbf{C}_0 = D(\mathbf{P}_0)$. An iterative process then starts whereby, in each iteration $k, k \geq 1$, the configuration matrix, \mathbf{C}_{k-1} , is checked to be the *SSA solution* of \mathbf{S} . If the solution has been reached then iteration stops. Otherwise, the positions of the points in \mathbf{P}_{k-1} in the SSA plot are updated to \mathbf{P}_k , the new configuration matrix is computed as $\mathbf{C}_k = D(\mathbf{P}_k)$ and iteration continues.

3.1 Starting Positions

The original SSA method places no requirements on the initial point positions in the SSA plot. However, the choice of starting points affects our final

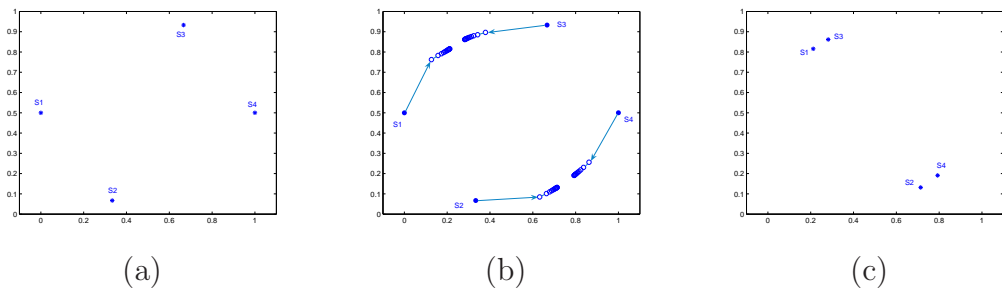


Figure 3.1: Point movement in the SSA plot. (a) starting positions of four points as determined in Section 3.1, (b) diminishing point movements with increasing iterations of the algorithm, (c) final point positions.

complexity result. A random selection of points leads to varying complexity values for the same shape each time it is computed. Therefore, we fix the initial point configuration as follows. We consider a sinusoidal function with x and y rescaled to the interval $[0, 1]$ and sample the N initial points, $\{\mathbf{p}_{i,0}\}$, $i \in \{1, \dots, N\}$, uniformly on it along the x axis, i.e. the coordinates of $\mathbf{p}_{i,0}$, are given by

$$x_i = \frac{i-1}{N-1}, \quad y_i = \frac{1}{2}(1 + \sin(2\pi x_i - \pi))$$

An example with four points is shown in Figure 3.1, and with 42 points in Figure 4.1.

3.2 Checking for an SSA Solution

To check whether a given \mathbf{C}_m , $m \geq 0$, is an SSA solution of \mathbf{S} , we need to construct the *ranking number matrices* of \mathbf{C}_k and \mathbf{S} . For a matrix, \mathbf{A} , to construct its ranking number matrix, $R(\mathbf{A})$, all entries $\mathbf{a}_{i,j}$ are sorted in descending order and given consecutive ranks. Thus, the largest entry gets a rank of one, the second largest a rank of two, and so on. Equal entries are assigned consecutive ranks. If the entries in \mathbf{A} are then replaced by their ranks, we obtain $R(\mathbf{A})$.

$$\mathbf{C}_m \text{ is an SSA solution of } \mathbf{S} \iff R(\mathbf{C}_m) = R(\mathbf{S}).$$

3.3 Updating Point Positions

Positions of points in \mathbf{P}_m , $m \geq 0$, are updated according to the *rank image matrix* of \mathbf{C}_m with respect to \mathbf{S} . We denote the rank image matrix of a

matrix, \mathbf{A} , with respect to another matrix, \mathbf{B} , as $R_{\mathbf{B}}(\mathbf{A})$. It contains the entries of \mathbf{A} permuted such that the ranking number matrices of $R_{\mathbf{B}}(\mathbf{A})$ and \mathbf{B} match, i.e. $R(R_{\mathbf{B}}(\mathbf{A})) = R(\mathbf{B})$.

For a given \mathbf{C}_m , $R_{\mathbf{S}}(\mathbf{C}_m)$ denotes the *intended* point configuration, i.e. it is desired that the Cartesian distances between the points in \mathbf{P}_m follow a similar pattern as the similarity distances in \mathbf{S} , and that their distance matrix, which will be the next configuration matrix, be an SSA solution to \mathbf{S} . To achieve this, a *correction factor* is computed for each point pair $(\mathbf{p}_{i,m}, \mathbf{p}_{j,m})$ as

$$f_{\{i,j\},m} = \frac{c'_{\{i,j\},m} - c_{\{i,j\},m}}{2c_{\{i,j\},m}},$$

where $c'_{\{i,j\},m}$ and $c_{\{i,j\},m}$ are entries in $R_{\mathbf{S}}(\mathbf{C}_m)$ and \mathbf{C}_m respectively. The correction factor for a point pair can be thought of as the force between them; the 2 in the denominator denotes how the points exert equal forces on each other. A positive value of $f_{\{i,j\},m}$ indicates that the current distance between the point pair is an underestimate and should be increased, whereas a negative value implies a shortening of the distance.

The displacement of $\mathbf{p}_{i,m}$ with respect to $\mathbf{p}_{j,m}$ is then given as

$$\vec{\mathbf{d}}_{\{i,j\},m} = f_{\{i,j\},m} \cdot (\mathbf{p}_{i,m} - \mathbf{p}_{j,m}).$$

The total displacement for $\mathbf{p}_{i,m}$ with respect to all other points is then given as

$$\vec{\mathbf{d}}_{i,m} = \frac{1}{N-1} \sum_{j=1, j \neq i}^N \vec{\mathbf{d}}_{\{i,j\},m}.$$

The averaging above ensures that points do not get displaced by too large an amount. Finally, the new point position is given by

$$\mathbf{p}_{i,m+1} = \mathbf{p}_{i,m} + \vec{\mathbf{d}}_{i,m}.$$

Figure 3.1 shows update of point positions as the SSA algorithm progresses for an example with four points.

3.4 Stopping Condition

Ideally, iteration stops when the current configuration matrix, \mathbf{C}_m , $m \geq 0$, is an SSA solution of \mathbf{S} . Indeed the update of point positions explained above aims to achieve just that. However, as each point is acted upon by all other points, the distance matrix of the new point positions is typically still not a solution to \mathbf{S} . Thus, the points are moved again and again until a stopping

condition is reached. With each iteration, the distance matrix of the point positions comes closer to the SSA solution of \mathbf{S} . This is reflected by progressively smaller values of $|f_{\{i,j\},m}|$ and $|\vec{\mathbf{d}}_{i,m}|$. Note that when the solution is reached, $f_{\{i,j\},m}$ and consequently $\vec{\mathbf{d}}_{i,m}$ will both be zero. In fact, after a certain number of iterations, $|\vec{\mathbf{d}}_{i,m}|$ becomes negligible. Therefore, iteration is stopped when the values of all $|f_{\{i,j\},m}|$ fall below a certain threshold. The point positions when iteration stops form the final configuration of the SSA plot.

3.5 Computing Shape Complexity

As we take the same number of views for each shape, the initial point configuration in the SSA plots for all shapes is the same, shown in Figure 4.1. As per the SSA method, movement of points in subsequent iterations is guided by the relative magnitudes of entries in the shape's similarity matrix. Thus, it is not possible to distinguish between plots obtained for two shapes whose similarity matrices differ only in scale. Therefore, when iteration stops, we rescale each plot according to its similarity matrix, \mathbf{S} , to obtain $\mathbf{Q} = \{\mathbf{q}_i\}$, the final set of points. Assuming the algorithm stopped after M iterations, we consider the last configuration matrix, $\mathbf{C}_M = D(\mathbf{P}_M)$, and obtain a rescaling factor

$$F = \frac{\text{largest entry in } \mathbf{S}}{\text{largest entry in } \mathbf{C}_M}.$$

The centroid of the points in \mathbf{P}_M is computed, $\mathbf{c}_M = \frac{1}{N} \sum_i \mathbf{p}_{i,M}$, and the positions of the rescaled points are updated,

$$\mathbf{q}_i = \mathbf{c}_M + F \cdot (\mathbf{p}_{i,M} - \mathbf{c}_M).$$

Complexity of the analyzed shape is now measured in terms of dispersion of the points in \mathbf{Q} . Our motivation is that a simple shape will yield only a few distinct views, leading to a handful of tight, distinct clusters in the SSA plot, whereas a complex shape will have largely varying views which will lead to loose and overlapping clusters.

We use two measures to obtain a complexity value from \mathbf{Q} . The first method measures complexity as the dispersion of the points in the x and y directions,

$$C_\sigma = \sqrt{\sigma_x^2 + \sigma_y^2},$$

where σ_x and σ_y are standard deviations of the x and y coordinates resp. of the \mathbf{q}_i . The second measure relies on the convex hull of the points in \mathbf{Q}

which is a subset, $\mathbf{H} = \{\mathbf{h}_j | j \in \{1, \dots, h\}\}$, of \mathbf{Q} . Shape complexity is then measured as

$$\begin{aligned}
C_H &= \frac{1}{2} \begin{vmatrix} \mathbf{h}_{1x} & \mathbf{h}_{1y} \\ \mathbf{h}_{2x} & \mathbf{h}_{2y} \\ \vdots & \vdots \\ \mathbf{h}_{hx} & \mathbf{h}_{hy} \\ \mathbf{h}_{1x} & \mathbf{h}_{1y} \end{vmatrix} \\
&= \frac{1}{2} [(\mathbf{h}_{1x}\mathbf{h}_{2y} + \mathbf{h}_{2x}\mathbf{h}_{3y} + \dots + \mathbf{h}_{hx}\mathbf{h}_{1y}) - \\
&\quad (\mathbf{h}_{1y}\mathbf{h}_{2x} + \mathbf{h}_{2y}\mathbf{h}_{3x} + \dots + \mathbf{h}_{hy}\mathbf{h}_{1x})],
\end{aligned}$$

where \mathbf{h}_{jx} and \mathbf{h}_{jy} are the x and y coordinates resp. of \mathbf{h}_j .

4 Results and Conclusion

We tested our approach on a set of shapes we obtained from the Internet. In Figure 4.2, we show each of these shapes alongside their corresponding SSA plots, and the obtained values for our two complexity measures, C_s and C_σ . As the shown values indicate, the shapes are sorted according to values of C_s from top to bottom and left to right, so the Bumpy Sphere is the simplest shape according to this measure, the Star is more complex and so on till the Bunny iH model. The next more complex model with respect to C_s is the Torus and then the Camel up to the Bones model. In the SSA plots shown, the points have been rescaled to fit inside the interval $x, y \in (0, 1)$ for better visualization. We use $N = 42$, i.e. we subdivide the initial view sphere once (Section 2.1). The corresponding starting position for points in the SSA plots is shown in Figure 4.1. As discussed in Section 3.1, as we use the same number of views for each shape, our strategy to assign positions to initial points in the SSA plot initializes the SSA plot for all shapes identically. How these points then move within the plot as the SSA algorithm progresses then depends on the similarity matrix, \mathbf{S} , for the shape.

Our results are summarized in Table 4.1, where the shapes are again sorted by C_s and we also show the relative complexities of the shapes, e.g. according to C_σ , the Camel is 2.6 times as complex as the Bumpy sphere.

As seen in the results, C_s and C_σ do not give mutually consistent results. This can also be seen in Figure 4.3 where we compare our results with those obtained using our implementations of previous curvature based methods [13, 21]. We see especially that relatively simple shapes like the Cone and Torus are ranked quite high with C_σ . The reason for this is that points in the SSA plots for these shapes (Figure 4.2) lie in tight, distinct clusters. As C_σ relies on deviation in one dimension only (along the x and y axes separately), the final value comes out to be large. This is corrected when we consider two dimensional information by computing the area of the convex hull to calculate C_s . In Figure 4.3, the Cone and Torus models are much higher in the ranking according to C_s . As expected, the curvature based methods of

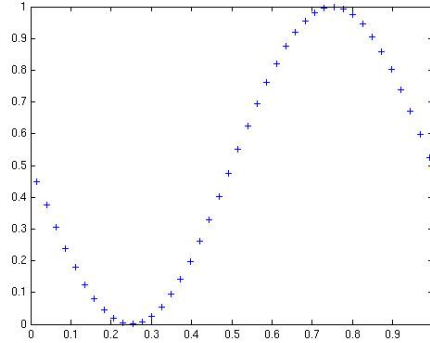


Figure 4.1: Initial point positions. As we use 42 views for each shape, our method from Section 3.1 to assign initial positions to points in the SSA plot initializes the SSA plot for all shapes identically, as shown above.

Shape	C_s	Relative	C_σ	Relative
Bumpy sphere	1.4028	1	6.0435	1
Star	1.4091	1.0	9.8827	1.6
Schwarz's Cylinder	3.1565	2.3	11.0770	1.8
Ellipsoid	3.2412	2.3	12.7204	2.1
Genus	7.3383	5.2	11.0868	1.8
Cone	7.5897	5.4	25.0723	4.1
Bunny iH	8.0565	5.7	11.9106	2.0
Torus	8.2006	5.8	19.7177	3.3
Camel	13.2013	9.4	15.9134	2.6
Dinosaur	13.3709	9.5	14.9445	2.5
Homer	15.8560	11.3	15.8468	2.6
Armadillo	15.9370	11.4	18.4462	3.1
Bones	19.5370	13.9	25.5131	4.2

Table 4.1: Shapes sorted by C_s .

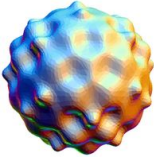
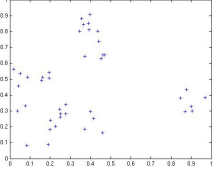

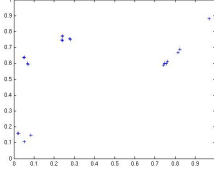
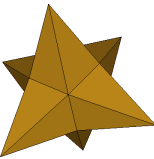
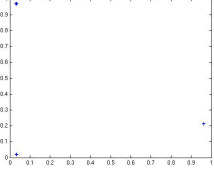

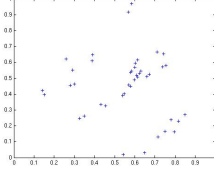
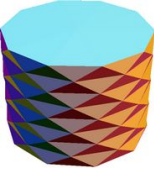
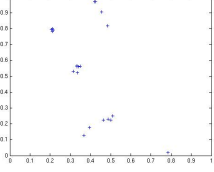

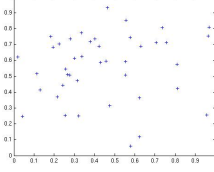
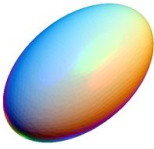
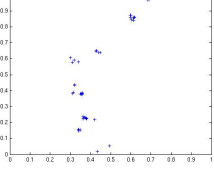

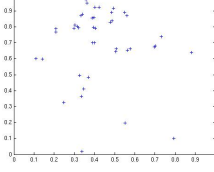

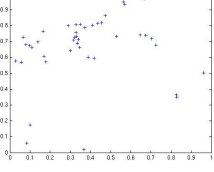

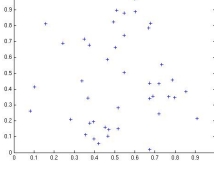

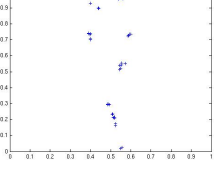

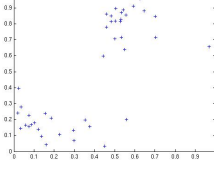
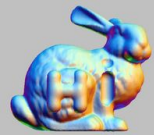
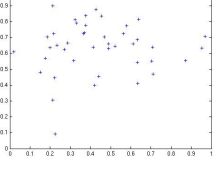
Shape	SSA plot	Shape	SSA plot
 Bumpy sphere, $C_s = 1.40$, $C_\sigma = 6.04$		 Torus, $C_s = 8.20$, $C_\sigma = 19.72$	
 Star, $C_s = 1.41$, $C_\sigma = 9.88$		 Camel, $C_s = 13.20$, $C_\sigma = 15.91$	
 Schwarz's cyl., $C_s = 3.16$, $C_\sigma = 11.08$		 Dinosaur, $C_s = 13.37$, $C_\sigma = 14.94$	
 Ellipsoid, $C_s = 3.24$, $C_\sigma = 12.72$		 Homer, $C_s = 15.86$, $C_\sigma = 15.85$	
 Genus, $C_s = 7.34$, $C_\sigma = 11.09$		 Armadillo, $C_s = 15.93$, $C_\sigma = 18.45$	
 Cone, $C_s = 7.59$, $C_\sigma = 25.07$		 Bones, $C_s = 19.54$, $C_\sigma = 25.51$	
 Bunny iH, $C_s = 8.06$, $C_\sigma = 11.91$			

Figure 4.2: Shapes sorted top to bottom, left to right according to C_s .

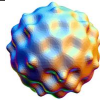
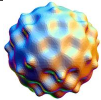




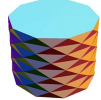
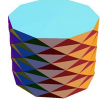
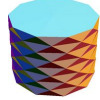
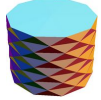


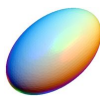













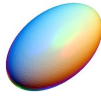















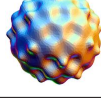
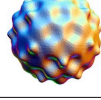

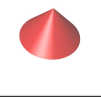

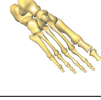




Rank	C_s	C_σ	[13]	[21]
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				

Figure 4.3: Shapes sorted according to four complexity measures – our complexity measures, C_s and C_σ , and the methods from [13] and [21].

[13, 21] are unable to deal with noise, the most prominent example of which is that they rank the Bumpy sphere as one of the most complex shapes, whereas our view based method ignores the noise and ranks the Bumpy sphere as the simplest.

4.1 Discussion

Our literature review on automatic computation of complexity of 3D shapes, presented in Section 1.1, yielded few other works. The ones among these that we tested are vulnerable to noise and slight irregularities in the shape. In contrast, our method is able to ignore these artefacts and produce a ranking of shapes that is more in agreement with human notions of shape complexity.

However, our method still has deficiencies, e.g. in the first two columns in Figure 4.3, the Bunny is ranked quite low compared to other, simpler shapes like the Torus. We believe this is because of inadequate representation of the information contained in our SSA plots. A deeper understanding of the SSA plot reflected in sophisticated measures to compute complexity from the plots will, in our opinion, relieve our method of the above problems.

The key to our complexity results is the SSA plot we obtain for each shape, which in turn depends on the shape similarity method used. A good shape similarity method, i.e. one that can compute similarities between shapes as humans perceive them, is thus crucial for the success of our approach.

As large numbers of 3D shape content become common, organizing them in a meaningful manner becomes important. Our approach can be used for this purpose to sort shapes in a 3D shape repository according to their complexities. Given a query shape, the repository can also be searched for stored shapes that are more, less or similarly complex.

One straightforward application of our SSA plots can be to compute shape symmetries [12, 14]. Symmetries in a shape are a measure of the shape’s self-similarities. A shape that has many symmetries will yield tight clusters of points in the SSA plot, e.g. the Star in Figure 4.2. This is because clusters correspond to views that are similar to each other. If views from different parts of the shape end up in the same cluster, that is indicative of a self-similarity within the shape between those parts. We could see each point in a cluster as a “vote” for a view. When different parts of a shape vote for the same view, the shape will surely be symmetric. Similar voting schemes have also been employed in previous works on symmetry [12, 14]. A significantly large number of votes for a view could also be used as a cue for the best view of the object.

Bibliography

- [1] N. Amenta, M. Bern, and D. Eppstein. The crust and the β -skeleton: combinatorial curve reconstruction. *Graphical Models and Image Processing*, 60(2):125–135, 1998.
- [2] N. Amenta, M. Bern, and M. Kamvyselis. A new Voronoi-based surface reconstruction algorithm. In *Proceedings of the 25th ACM Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '98)*, pages 415–421, 1998.
- [3] E. Attalla and P. Siy. Robust shape similarity retrieval based on contour segmentation polygonal multiresolution and elastic matching. *Pattern Recognition*, 38(12):2229–2241, 2005.
- [4] I. Borg and P. Groenen. *Modern Multidimensional Scaling: Theory and Applications*. Springer, 2005.
- [5] I. Borg and J. Lingoes. *Multidimensional Similarity Structure Analysis*. Springer, Berlin, 1987.
- [6] H. H. Bülthoff, S. Y. Edelman, and M. J. Tarr. How are three-dimensional objects represented in the brain? *Cerebral Cortex*, 5(3):247–260, 1995.
- [7] F. Cutzu and M. J. Tarr. The representation of three-dimensional object similarity in human vision. In *SPIE Proceedings from Electronic Imaging: Human Vision and Electronic Imaging II, 3016*, pages 460–471, 1997.
- [8] C. M. Cyr and B. B. Kimia. A similarity-based aspect-graph approach to 3D object recognition. *International Journal of Computer Vision*, 57(1):5–22, 2004.

- [9] B. Delaunay. Sur la sphère vide. *Izvestia Akademia Nauk SSSR, VII Seria, Otdelenie Matematicheskii i Estestvennyka Nauk*, 7:793–800, 1934.
- [10] J. J. Koenderink and A. J. van Doorn. The internal representation of solid shape with respect to vision. *Biological Cybernetics*, 32(4):211–216, 1979.
- [11] C. T. Loop. Smooth subdivision surfaces based on triangles. Master’s thesis, Department of Mathematics, The University of Utah, 1987.
- [12] N. J. Mitra, L. Guibas, and M. Pauly. Partial and approximate symmetry detection for 3D geometry. *ACM Transactions on Graphics (Proceedings of SIGGRAPH ’06)*, 25(3):560–568, 2006.
- [13] D. L. Page, A. Koschan, S. R. Sukumar, B. Roui-Abidi, and M. A. Abidi. Shape analysis algorithm based on information theory. In *Proceedings of the International Conference on Image Processing 2003 (ICIP ’03)*, pages 229–232, 2003.
- [14] J. Podolak, P. Shilane, A. Golovinskiy, S. Rusinkiewicz, and T. Funkhouser. A planar-reflective symmetry transform for 3D shapes. *ACM Transactions on Graphics (Proceedings of SIGGRAPH ’06)*, pages 549–559, 2006.
- [15] J. Rigau, M. Feixas, and M. Sbert. Shape complexity based on mutual information. In *Proceedings of the IEEE International Conference on Shape Modeling and Applications 2005 (SMI ’05)*, pages 357–362, 2005.
- [16] A. Rosner. The Crust Curve Recosntruction Applet. valis.cs.uiuc.edu/~sariel/research/CG/applets/Crust/Crust.html. last accessed on 1 Oct, 2008.
- [17] J. Rossignac. Shape complexity. *The Visual Computer*, 21(12):985–996, 2005.
- [18] W. Saleem, W. Song, A. Belyaev, and H.-P. Seidel. On computing best fly. In *Proceedings of the 23rd Spring Conference on Computer Graphics 2007 (SCCG ’07)*, pages 143–149, 2007.
- [19] C. E. Shannon and W. Weaver. *A Mathematical Theory of Communication*. University of Illinois Press, Champaign, IL, USA, 1963.
- [20] J. R. Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. *Lecture Notes in Computer Science*, 1148:203–222, 1996.

- [21] S. R. Sukumar, D. Page, A. Gribok, A. Koschan, and M. A. Abidi. Shape measure for identifying perceptually informative parts of 3D objects. In *Proceedings of the Third International Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT '06)*, pages 679–686, 2006.
- [22] R. C. Veltkamp and L. J. Latecki. Properties and performance of shape similarity measures. In *Proceedings of the 10th International Conference on Data Science and Classification 2006 (IFCS '06)*, 2006.
- [23] D. Wang. 3D shape complexity using view similarity. Master's thesis, Computer Science Department, University of Saarland, 2008.
- [24] H. Yamauchi, W. Saleem, S. Yoshizawa, Z. Karni, A. Belyaev, and H.-P. Seidel. Towards stable and salient multi-view representation of 3D shapes. In *Proceedings of the IEEE International Conference on Shape Modeling and Applications 2006 (SMI '06)*, pages 265–270, 2006.

Below you find a list of the most recent technical reports of the Max-Planck-Institut für Informatik. They are available via WWW using the URL <http://www.mpi-inf.mpg.de>. If you have any questions concerning WWW access, please contact reports@mpi-inf.mpg.de. Paper copies (which are not necessarily free of charge) can be ordered either by regular mail or by e-mail at the address below.

Max-Planck-Institut für Informatik
 Library
 attn. Anja Becker
 Stuhlsatzenhausweg 85
 66123 Saarbrücken
 GERMANY
 e-mail: library@mpi-inf.mpg.de

MPI-I-2008-5-003	F.M. Suchanek, G. de Melo, A. Pease	Integrating Yago into the Suggested Upper Merged Ontology
MPI-I-2008-5-002	T. Neumann, G. Moerkotte	Single Phase Construction of Optimal DAG-structured QEPs
MPI-I-2008-5-001	F. Suchanek, G. Kasneci, M. Ramanath, M. Sozio, G. Weikum	STAR: Steiner Tree Approximation in Relationship-Graphs
MPI-I-2008-1-001	D. Ajwani	Characterizing the performance of Flash memory storage devices and its impact on algorithm design
MPI-I-2007-RG1-002	T. Hillenbrand, C. Weidenbach	Superposition for Finite Domains
MPI-I-2007-5-003	F.M. Suchanek, G. Kasneci, G. Weikum	Yago : A Large Ontology from Wikipedia and WordNet
MPI-I-2007-5-002	K. Berberich, S. Bedathur, T. Neumann, G. Weikum	A Time Machine for Text Search
MPI-I-2007-5-001	G. Kasneci, F.M. Suchanek, G. Ifrim, M. Ramanath, G. Weikum	NAGA: Searching and Ranking Knowledge
MPI-I-2007-4-008	J. Gall, T. Brox, B. Rosenhahn, H. Seidel	Global Stochastic Optimization for Robust and Accurate Human Motion Capture
MPI-I-2007-4-007	R. Herzog, V. Havran, K. Myszkowski, H. Seidel	Global Illumination using Photon Ray Splatting
MPI-I-2007-4-006	C. Dyken, G. Ziegler, C. Theobalt, H. Seidel	GPU Marching Cubes on Shader Model 3.0 and 4.0
MPI-I-2007-4-005	T. Schultz, J. Weickert, H. Seidel	A Higher-Order Structure Tensor
MPI-I-2007-4-004	C. Stoll	A Volumetric Approach to Interactive Shape Editing
MPI-I-2007-4-003	R. Bargmann, V. Blanz, H. Seidel	A Nonlinear Viseme Model for Triphone-Based Speech Synthesis
MPI-I-2007-4-002	T. Langer, H. Seidel	Construction of Smooth Maps with Mean Value Coordinates
MPI-I-2007-4-001	J. Gall, B. Rosenhahn, H. Seidel	Clustered Stochastic Optimization for Object Recognition and Pose Estimation
MPI-I-2007-2-001	A. Podelski, S. Wagner	A Method and a Tool for Automatic Verification of Region Stability for Hybrid Systems
MPI-I-2007-1-002	E. Althaus, S. Canzar	A Lagrangian relaxation approach for the multiple sequence alignment problem
MPI-I-2007-1-001	E. Berberich, L. Kettner	Linear-Time Reordering in a Sweep-line Algorithm for Algebraic Curves Intersecting in a Common Point
MPI-I-2006-5-006	G. Kasneci, F.M. Suchanek, G. Weikum	Yago - A Core of Semantic Knowledge
MPI-I-2006-5-005	R. Angelova, S. Siersdorfer	A Neighborhood-Based Approach for Clustering of Linked Document Collections
MPI-I-2006-5-004	F. Suchanek, G. Ifrim, G. Weikum	Combining Linguistic and Statistical Analysis to Extract Relations from Web Documents
MPI-I-2006-5-003	V. Scholz, M. Magnor	Garment Texture Editing in Monocular Video Sequences based on Color-Coded Printing Patterns

MPI-I-2006-5-002	H. Bast, D. Majumdar, R. Schenkel, M. Theobald, G. Weikum	IO-Top-k: Index-access Optimized Top-k Query Processing
MPI-I-2006-5-001	M. Bender, S. Michel, G. Weikum, P. Triantafilou	Overlap-Aware Global df Estimation in Distributed Information Retrieval Systems
MPI-I-2006-4-010	A. Belyaev, T. Langer, H. Seidel	Mean Value Coordinates for Arbitrary Spherical Polygons and Polyhedra in \mathbb{R}^3
MPI-I-2006-4-009	J. Gall, J. Potthoff, B. Rosenhahn, C. Schnoerr, H. Seidel	Interacting and Annealing Particle Filters: Mathematics and a Recipe for Applications
MPI-I-2006-4-008	I. Albrecht, M. Kipp, M. Neff, H. Seidel	Gesture Modeling and Animation by Imitation
MPI-I-2006-4-007	O. Schall, A. Belyaev, H. Seidel	Feature-preserving Non-local Denoising of Static and Time-varying Range Data
MPI-I-2006-4-006	C. Theobalt, N. Ahmed, H. Lensch, M. Magnor, H. Seidel	Enhanced Dynamic Reflectometry for Relightable Free-Viewpoint Video
MPI-I-2006-4-005	A. Belyaev, H. Seidel, S. Yoshizawa	Skeleton-driven Laplacian Mesh Deformations
MPI-I-2006-4-004	V. Havran, R. Herzog, H. Seidel	On Fast Construction of Spatial Hierarchies for Ray Tracing
MPI-I-2006-4-003	E. de Aguiar, R. Zayer, C. Theobalt, M. Magnor, H. Seidel	A Framework for Natural Animation of Digitized Models
MPI-I-2006-4-002	G. Ziegler, A. Tevs, C. Theobalt, H. Seidel	GPU Point List Generation through Histogram Pyramids
MPI-I-2006-4-001	A. Efremov, R. Mantiuk, K. Myszkowski, H. Seidel	Design and Evaluation of Backward Compatible High Dynamic Range Video Compression
MPI-I-2006-2-001	T. Wies, V. Kuncak, K. Zee, A. Podelski, M. Rinard	On Verifying Complex Properties using Symbolic Shape Analysis
MPI-I-2006-1-007	H. Bast, I. Weber, C.W. Mortensen	Output-Sensitive Autocompletion Search
MPI-I-2006-1-006	M. Kerber	Division-Free Computation of Subresultants Using Bezout Matrices
MPI-I-2006-1-005	A. Eigenwillig, L. Kettner, N. Wolpert	Snap Rounding of Bézier Curves
MPI-I-2006-1-004	S. Funke, S. Laue, R. Naujoks, L. Zvi	Power Assignment Problems in Wireless Communication
MPI-I-2005-5-002	S. Siersdorfer, G. Weikum	Automated Retraining Methods for Document Classification and their Parameter Tuning
MPI-I-2005-4-006	C. Fuchs, M. Goesele, T. Chen, H. Seidel	An Empirical Model for Heterogeneous Translucent Objects
MPI-I-2005-4-005	G. Krawczyk, M. Goesele, H. Seidel	Photometric Calibration of High Dynamic Range Cameras
MPI-I-2005-4-004	C. Theobalt, N. Ahmed, E. De Aguiar, G. Ziegler, H. Lensch, M.A. Magnor, H. Seidel	Joint Motion and Reflectance Capture for Creating Relightable 3D Videos
MPI-I-2005-4-003	T. Langer, A.G. Belyaev, H. Seidel	Analysis and Design of Discrete Normals and Curvatures
MPI-I-2005-4-002	O. Schall, A. Belyaev, H. Seidel	Sparse Meshing of Uncertain and Noisy Surface Scattered Data
MPI-I-2005-4-001	M. Fuchs, V. Blanz, H. Lensch, H. Seidel	Reflectance from Images: A Model-Based Approach for Human Faces
MPI-I-2005-2-004	Y. Kazakov	A Framework of Refutational Theorem Proving for Saturation-Based Decision Procedures
MPI-I-2005-2-003	H.d. Nivelle	Using Resolution as a Decision Procedure
MPI-I-2005-2-002	P. Maier, W. Charatonik, L. Georgieva	Bounded Model Checking of Pointer Programs
MPI-I-2005-2-001	J. Hoffmann, C. Gomes, B. Selman	Bottleneck Behavior in CNF Formulas
MPI-I-2005-1-008	C. Gotsman, K. Kaligosi, K. Mehlhorn, D. Michail, E. Pyrga	Cycle Bases of Graphs and Sampled Manifolds
MPI-I-2005-1-007	I. Katriel, M. Kutz	A Faster Algorithm for Computing a Longest Common Increasing Subsequence
MPI-I-2005-1-003	S. Baswana, K. Telikepalli	Improved Algorithms for All-Pairs Approximate Shortest Paths in Weighted Graphs