Automatic 2D Shape Orientation by Example

Waqar Saleem, Danyi Wang, Alexander Belyaev, Hans-Peter Seidel
Max Planck Institut Informatik
Computer Graphics group
Stuhlsatzenhausweg 85
66123 Saarbrücken, Germany
{wsaleem, danyi, belyaev, hpseidel} @mpi-inf.mpg.de

Abstract

As large shape repositories become more common, the problem of automatically generating good views of shapes has recently gained prominence. However, very few of the proposed methods take into account the orientation of the shape in the resulting view, and none presents a satisfactory solution. In this paper, we present a simple, example based method to correct the orientation of a shape in a query image. Our method depends on the availability of a database of classified images containing correctly oriented shapes. In the first step, a candidate class for the query shape is identified, and in the second, the query shape is aligned with a target shape from the candidate class.

1 Introduction

For visualisation of shapes in large shape repositories, it is important to present users with a few representative images for each shape. Automatic generation of such images, the so-called best-view problem, has recently attracted interest from the computer graphics [3, 5, 10, 11, 15, 17, 20], computer vision [2, 8] and pattern recognition [6] communities. None of these methods consider shape orientation during view selection. Some of them [11, 15] fix orientations of selected views by hand. In [20], a heuristic is proposed to correct shape orientation in the chosen views but the results are unsatisfactory.

The above methods utilise techniques that are intrinsically rotation invariant — view entropy, shape similarity, view saliency, symmetry. Hence, they cannot distinguish between views in which the shape is oriented differently, e.g. the views in Figure 1.

This leads us to consider the following problem: given a view of a 3D shape, e.g. selected by a best view method, find the correct orientation of the shape in the view.

Figure 1. Two views each with different orientations of two shapes. Given the left image of each shape, our method automatically computes the right image, which has a more natural orientation for the shape.

We could not find previous work on automatic shape orientation. There has recently been some work on image orientation (cf. [7, 16, 19]), whereby the correct orientation of a natural image (0°, 90°, 180° or 270°) is decided by performing statistical analyses of image features. However, the shape orientation problem is different. Given a view of a shape, we want to automatically determine the correct orientation of the shape for the view.

Humans know the correct orientation(s) of a shape through experience with objects of that shape. It is through previous interaction that we know, for example, that the correct orientation for a horse is one in which the legs are down, instead of up. Such information is difficult, if not impossible, to compute from the shape alone.

Therefore, in this paper, we propose an example based approach to shape orientation. We note that as a shape can be represented aptly by its silhouette, it is sufficient to represent views of shapes by their binary images. This reduces the problem to a 2D one. In the rest of this paper, we use the term ‘shape’ to refer to a projection of a 3D shape to a 2D image. In a preprocessing step, we set up a database of classified, correctly oriented shapes. Then, for a new query shape, we suggest a correct orientation for it. Orientation correction takes place in two steps; first, we find for the query shape a candidate class from our database, and then, we align the query shape to a target shape from the candi-
date class.

To the best of our knowledge, this is the first attempt to automatically solve the problem of shape orientation in views. However, we found that many of the required components of our method already exist or can easily be derived from existing sources. Instead of re-inventing the wheel, we use existing methods when necessary, and report on their performance in Section 7.

2 Training: Setting up the Database

We need to set up a database of classified, correctly oriented shapes. There are several large image datasets available on the Internet. We choose the MPEG-7 dataset [4, 18], which contains 1400 binary images organised into 70 classes with 20 members per class, where each image contains a single shape. As the dataset was compiled to test shape similarity techniques, members of the same class differ from each other in shape features. With regards to orientation, there is a lot of redundancy and false information which had to be filtered out.

We found several classes of non-orientable shapes. These are artificial shapes for which no notion of correct orientation exists. These classes were removed. In some of the remaining classes, we found images with incorrect orientations. These images mostly contain natural objects for which a human observer can easily specify a correct orientation. Such images were also removed, and later used as queries for our method.

The dataset still contained redundant images. These are class members that differ from each other only in fine shape features and not necessarily in orientation. We removed redundancies automatically by identifying groups of very similar images within a class, and retaining one image from each group. This is done by choosing a class member initially at random and removing all other members within a certain shape similarity distance from it. The process is then repeated with the class member with the highest similarity distance. Iteration stops when all class members have been either selected or removed. The shape similarity technique used is described in Section 5. Figure 2 shows examples of the kinds of images that are removed from the database.

After the above filtering, our database contains 237 images in 56 classes. The smallest and largest class memberships are 1 and 14 respectively with a median of 4. The choice of ‘similar’ images is very much dependent on the shape similarity method used. We talk more about this in Section 7. Within the database, we represent each shape by its boundary contour. Our boundary extraction method is presented in Section 3, and the simplified database can be viewed at the web interface mentioned in Section 6, where our method, described in Section 4 can also be tested.

Figure 2. Training requires manual removal of incorrectly oriented images (top row) and images containing non-orientable shapes (middle row). Redundantly oriented images (bottom row) within a class can be removed automatically.

3 Boundary Extraction

We scan the image for boundary pixels and obtain the pixels’ Crust [1]. This typically leads to a closed boundary contour of the shape as desired, except in a few cases where the boundary of the shape contains noise or fine features and the computed Crust contains a large number of disconnected edges. This is easily detected, and solved by applying blurring followed by morphological opening on the original image, and then computing its Crust. This process is repeated with increasingly blurred versions of the original image until the resulting Crust is a closed contour as required.

From the 237 images in our database, 20 images required blurring iterations, with only 4 images requiring 3 or more iterations.

4 Example based Shape Orientation

As mentioned earlier, our method proceeds in two steps – a classification followed by an alignment step. We use nearest neighbour classification. Using a shape similarity technique, we retrieve a list of database images sorted according to their similarity distance from the query image. The candidate class is then chosen as the class containing the best match, i.e. the image least distant from the query. The shape similarity technique we used is described in Section 5.

From the candidate class, we choose the shape most similar to the query shape. This is the target shape. Recall that database shapes are correctly oriented, and that all shapes are represented by their boundary contours. To align the query shape to the target shape, we rotate the query shape
such that the directions of its Principal Components (PCs) match those of the target shape.

To remove bias because of point density, the contours are uniformly sampled before PCA calculation. PCs of 180° rotations of a shape have the same direction, though the shapes are oriented differently. To avoid errors because of this, once the query shape has been rotated as described above, we match it with both the target shape and a 180° rotation of the target shape. If the 180° rotated target shape gives a better match, we rotate our final image by 180°. The matching method is explained in Section 5.

5 Shape Similarity Technique

For all shape similarity computations, the MPEG-7 standard, Curvature Scale Space (CSS) method [9] is used. It involves smoothing a shape’s boundary contour until it is fully convex. A plot of amount of smoothing against positions of curvature inflection points on the boundary contour is generated. This plot is the shape’s CSS image. The shape is then represented by the positions of maxima in its CSS image. When comparing two shapes, the highest maxima in their CSS images are aligned and the shapes’ similarity distance is then computed as the cost of matching the remaining maxima. Smaller distances imply greater shape similarity.

The method is rotation and translation invariant, and by normalizing boundary contours, is also made scale invariant. In Section 4, when a rotation sensitive method is required, we use the same method, but this time assign a cost to the initial alignment of maxima as well.

6 Results

We query our method with the incorrectly oriented shapes removed from the original MPEG-7 dataset, with user input images, and with commonly used shapes from the Computer Graphics literature. Our system can be tested online at http://shapes.graphics.mpi-inf.mpg.de/shapeOrient, where users can upload or use an interface to sketch query shapes. We used images of commonly used models that were presented as best-view results in [20]. Results are shown in Figures 3 to 5 (some artefacts may be visible because of image resizing). The columns show query images, corresponding target shapes chosen from the database and our final results. Recall that database shapes are assumed to be correctly oriented. The correctness of a result is evaluated visually, and is taken to be correct if the resulting image is similarly oriented as a database image of the same class, or if it agrees with the human notion of correct orientation for the contained shape.

We have the most success when using incorrectly oriented shapes previously removed from the dataset, Figure 3.
as they have similar class members still in the database. The tree-3 case shows the efficacy of the PCA rotation correction presented at the end of Section 4, while the chicken-7 case illustrates the limitations of a shape’s PCs in estimating its orientation, and the apple-1 case underlines our use of object boundaries for similarity computation. Results for user sketches, Figure 4, are fairly good when the sketch already matches an existing database shape closely. Limitation of orientation estimation by PCs is again illustrated in the heart case, and the car case shows the deficiency of our method when queried with a shape different from database exemplars (our database contains a personal_car class whose members are differently shaped than this query).

Major deficiencies of the method are fully exposed in Figure 5. Our database already contained a horse exemplar very similar to the query horse, thus yielding a good result. We would have expected the camel also to match the horse, but its curvature properties caused our similarity method to deem it more similar to the frog, resulting in an incorrect output orientation. (Notice that the camel’s front and hind legs are not apart in the boundary image). Finally, our database simply does not contain suitable exemplars for many of the other models, so the results are more or less arbitrary. Santa and rocker-arm are examples of non-orientable shapes. No orientation can be regarded as correct for these shapes. The target shapes chosen are irrelevant, as any orientation imposed on these shapes will do.

### 7 Discussion & Future Work

Our method is quite slow, with typical running times of over a minute, and complex shapes (whose boundaries have many concave/convex pairs) up to two minutes. The bulk of the running time is consumed in retrieving the similarity sorted list. We plan to investigate optimisation of the database search by an initial one-time probabilistic analysis of the database [14] and approximate nearest neighbour searches [12]. The search can further be optimised through indexing. Instead of matching a query image with the entire database, we could extract a ‘prototype’ for each class and match the query with the prototypes to find the candidate class. The target shape could then be found by matching the query with all members of the candidate class.

Many aspects of our method are dependent on the shape similarity method used. The most important of these is the retrieval of the similarity sorted list mentioned above. As seen in Section 6, accuracy of this list is crucial to our method. Artefacts in the similarity method lead to questionable sortings causing our method to yield implausible results. Throughout this paper, we have used the CSS method, which had been reported earlier [4] to perform well and has in fact been included in the MPEG-7 standard. However, recent results [18] and our experience in this paper with the method indicate that the method can still be substantially improved. We also plan to investigate other shape similarity techniques, possibly using different ones at different stages of our method, and enhanced by probabilistic analyses of the data [14].

Perhaps the most important ingredient of our method is the database used. It currently contains only 56 classes, extracted from an already compiled dataset. The limitations arising from this are visible in Figure 5. While the need for extending the dataset is obvious, it is not entirely clear how to do so without manual interaction. One solution could be to crawl the Internet for images. As these images are (mostly) posted by human users, we can assume them to be correctly oriented. However, automatically sorting through these images to extract the ones containing single objects against a plain background is non-trivial. Furthermore, each extracted image will need to be either classified into one of the existing classes in the database, or added as an example of a new class. This could be done using a similarity threshold. We plan to test this approach in the future.

Also, as highlighted in Section 6, PCs are not robust estimators of shape orientation. We plan to investigate other methods for the alignment step, e.g. Iterative Closest Point (ICP).

One drawback our method suffers from is the absence of

<table>
<thead>
<tr>
<th>Query</th>
<th>Boundary</th>
<th>Best match</th>
<th>Aligned</th>
</tr>
</thead>
<tbody>
<tr>
<td>horse</td>
<td>horse</td>
<td>horse</td>
<td>horse aligned</td>
</tr>
<tr>
<td>David</td>
<td>face</td>
<td>David</td>
<td>David aligned</td>
</tr>
<tr>
<td>bunny</td>
<td>heart</td>
<td>bunny</td>
<td>bunny aligned</td>
</tr>
<tr>
<td>camel</td>
<td>frog</td>
<td>camel</td>
<td>camel aligned</td>
</tr>
<tr>
<td>rocker-arm</td>
<td>personal_car</td>
<td>rocker-arm</td>
<td>rocker-arm aligned</td>
</tr>
<tr>
<td>Santa</td>
<td>fork</td>
<td>Santa</td>
<td>Santa aligned</td>
</tr>
</tbody>
</table>

Figure 5. Results for common Computer Graphics models.
a systematic means of evaluation. The correctness of our result can be judged only by visual inspection by a human user, or by similarity to stored instances of the same class. But we believe that this flaw is intrinsic in our problem description, and will be resolved once we have a representative enough reference dataset.

A natural extension of our method is to work directly with 3D models. On closer inspection, this problem amounts to choosing a ‘correct’ up-vector in 3D for the given shape. We believe an example based approach as the one we have presented will be a fruitful line of research to follow.

8 Conclusion

This paper has addressed the novel and pertinent research problem of automatic orientation of 2D shapes. Motivated by the human approach to the problem, we use an example based method. As discussed in Section 7, there are many factors involved in our method, and further work is still needed to better the preliminary results reported in this paper.

We believe that the problem we tackle is a hard one, as we try, in essence, to mimic the human notion of correct orientation, which is a complex mix resulting mostly from user experience and partly from the object’s shape. Such a notion is not computationally replicable and we would like the reader not to let the exploratory nature of our current results divert attention from the promise of example based methods for this problem.

Acknowledgements

We are thankful to Remco Veltkamp, Utrecht University, for helpful comments and for providing code from SIDESTEP, to Longin J. Latecki, Temple University, for useful discussions, and to Shin Yoshizawa, MPII, for suggesting the Crust method and Triangle software [13] for boundary extraction. This research is partially supported by European FP6 NoE grant 506766, AIM@SHAPE. Some preliminary results were presented by the first author in an AIM@SHAPE workshop, for the organisation of which, he would like to thank the project partners.

References


