On Computing Best Fly

Waqar Saleem* MPI Informatik
Wenhao Song† MPI Informatik
Alexander Belyaev‡ MPI Informatik
Hans-Peter Seidel§ MPI Informatik

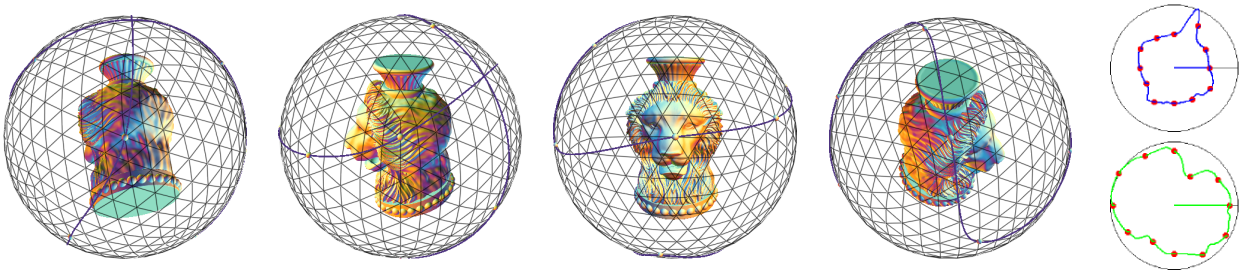

Figure 1: We present a method that, given a shape model, computes an animation of the shape based on its geometric properties and perception based, aesthetic criteria. Representative viewpoints of the shape are extracted, and an interpolating path is computed as a trajectory for a virtual camera. Speed and zoom of the camera along the path are controlled. Here, we show several scenes extracted from left to right in sequence from our animation of the Lion vase model. The blue line on the viewsphere represents the camera's trajectory and the rightmost image shows how the camera speed (top) and zoom (bottom) vary along the trajectory.

Abstract

With growing popularity of online 3D shape databases, the problem of navigation and remote visualisation of large 3D shape models in such repositories is gaining prominence. While some recent work has focused on automatically computing the best view(s) of a given model, little attention has been given to the problem's dynamic counterpart – *best fly*. In this paper, we propose a solution to this problem that extends on previous best view methods. Given a shape, we use its best views to compute a path on its viewsphere which acts as a trajectory for a virtual camera pointing at the object. We then use the model's geometric properties to determine the speed and zoom of the camera along the path.

Keywords: best fly, remote shape exploration

1 Introduction

With increased accessibility of 3D acquisition and modeling technology, the amount of 3D content available online is growing rapidly. This is evident in the existence of large shape repositories offering vast amounts of detailed, complex geometric data [3DScanRep ; PDB ; NDR ; PSB ; AIM@SHAPE ; LGMA] with some of them regularly offering new content. The surge in availability of 3D models fuels research in the geometry processing community which in turn creates demand for more high quality models.

*wsaleem@mpi-inf.mpg.de
†wsong@mpi-inf.mpg.de
‡belyaev@mpi-inf.mpg.de
§hpseidel@mpi-inf.mpg.de

SCCG 2007, Budmerice Castle, Slovakia, April 26 – 28, 2007.
© 2007 ACM 978-1-60558-956-5/07/0004 $15.00

Repositories offering such extensive data require a means to efficiently and economically visualise 3D shape content. This 'best view' problem has been addressed by the pattern recognition and computer vision community [Denton et al. 2004; Hall and Owen 2005; Lee et al. 2004; Mokhtarian and Abbasi 2000; Vázquez et al. 2001] and recently has also been popularised in computer graphics [Bordoloi and Shen 2005; Lee et al. 2005; Podolak et al. 2006; Polonsky et al. 2005; Sokolov and Plemenos 2005; Takahashi et al. 2005; Vázquez et al. 2003; Yamauchi et al. 2006]. The approach is to sample all views of the object and assign a view goodness measure to each view based on view similarity and/or stability, or properties of the visible part of the shape. Views with highest values for this measure are chosen as the best views. Some online repositories [3DoD ; AIM@SHAPE] offer a more dynamic solution in the form of 3D viewers for interactive exploration of shapes.

Both these solutions have drawbacks. While the view-based static solution cannot convey information about all of the shape and relation between different views, the dynamic solution is impractical for large shape models.

In this paper, we present an efficient, hybrid solution that is a natural extension of the best view problem. After computing representative viewpoints of the given shape [Yamauchi et al. 2006], we interpolate them on the shape's viewsphere. The resulting path is used as a trajectory for a virtual camera pointing at the object. In tradition of best view, we propose the term 'best fly' for the problem of computing the camera trajectory, where fly or fly-by refers to the virtual camera's passing over the object. The criteria that distinguish the 'best' fly from any other fly around the shape are outlined in Section 2.

Once the fly is computed, we control the camera's speed along it by taking into account the perception based view saliency measure from [Lee et al. 2005] to quickly skip uninteresting views. We use the multi-scale nature of the view saliency computation to compute a relevant viewing scale for shape features, which is used to regulate the zoom of the camera.

The main contributions of this paper are as follows.

- A new method to compute a fly around the object is presented.
- Shape properties of the object are used to determine the speed

of the camera along its trajectory.

- Appropriate viewing scales for shape features are calculated which control the zoom of the camera during the fly.

To the best of our knowledge, altering the speed and zoom parameters of the camera to provide a more informative fly has not been considered before in the literature.

Once the animation is computed, it can be put online on a shape repository's webpage in a popular web format, e.g. animated GIF or Flash. A tunable parameter can allow a user to control the length of the animation.

Some recent work related to the problem is presented in Section 2. In Section 3, we present the preprocessing steps of our method, namely view selection and saliency computation. Sections 4 to 6 then describe our computation of the trajectory, speed and zoom of the camera. Results of experiments on several models are presented in Section 7 followed by a discussion and conclusion in Sections 8 and 9. Most of the images in this paper are snapshots from the actual animations computed by our method. We refer the reader to the accompanying video for a better understanding of our results (the video will be made available online at the time of publication).

2 Previous Work

While best view methods sample all possible views of the object from its viewsphere to choose the best among them, taking such an approach to the best fly problem is not feasible as the size of the search space increases exponentially with the number of viewpoints to be visited in the fly. To guide the computation, previous approaches [Sokolov et al. 2006a; Sokolov et al. 2006b] have therefore made use of some heuristics which we mention below.

1. *Brevity* – the animation should not be long,
2. *Information* – the animation must be maximally informative,
3. *Exploration* – the camera path should avoid fast returns to already visited viewpoints
4. *Smoothness* – the path should be smooth.

Depending on how the path computation is performed, shape exploration methods are classified as either *offline* or *online*. Offline methods analyse the object once and compute the fly in advance. Online methods compute a path in real time each time they visit the object. Another classification is made on the nature of exploration conducted. *Global* methods aim to give a general understanding of the shape model. The camera stays outside the model, restricted to its viewsphere. In *local* methods, the camera enters the model and becomes part of it. Our method presented in this paper is a global, offline method that satisfies the above heuristics.

Previous methods either fail the Smoothness condition [Sokolov et al. 2006b] or deal with it artificially by putting in a damping factor where sharp turns occur [Sokolov et al. 2006a]. The path is often computed incrementally [Sokolov et al. 2006a]; at each viewpoint in the path, the next viewpoint is selected after considering each candidate viewpoint's view goodness, distance from the starting point of the path and the fraction of the model uncovered from that viewpoint. Such methods suffer from the drawback that they cannot guarantee that the computed path will pass through a given set of points, without violating any of the heuristics or complicating the computation.

Our method capitalises on the recent flurry of activity in the best view area. Our premise is that the shape is sufficiently described by the views computed by these methods. What remains is to inform the user on the relation (transition) between these views. In Section 8, we discuss how our method meets the conditions mentioned above.

3 Preprocessing

3.1 Representative views

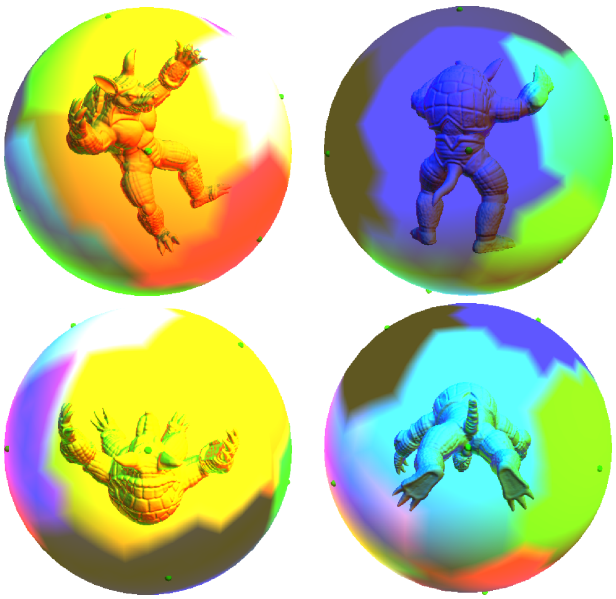

Figure 2: Stable view regions (coloured) for the Armadillo model and their representative viewpoints (green dots). We compute 12 regions, and each of the four views above is taken from the representative viewpoint of the region in the centre of the image.

We require a method that yields a set, $\mathscr{V}$, of representative views that are maximally dissimilar to each other and thus provide a good coverage of the viewsphere. Most view measures however are such that their best few views are clustered close to each other on the viewsphere. In [Sokolov et al. 2006a], such a set is constructed by iteratively picking viewpoints that make visible the most number of uncovered shape vertices, and stopping when a certain fraction or all of the vertices have been uncovered. This overlooks the shape of the model, and is sensitive to tessellation. The method in [Podolak et al. 2006] uses shape symmetries to yield the desired set, but it involves costly computations that make its use for large shape models impractical. The method we employ is closest in spirit to [Mokhtarian and Abbasi 2000] which uses shape similarity to compute $\mathscr{V}$.

We use the method described in [Yamauchi et al. 2006] which uses image comparison [Khotanzad and Hong 1990] to compute similarity of neighbouring viewpoints. After assigning the similarity values as weights to the edges of the viewsphere, a graph partitioning method [Karypis and Kumar 1998] is applied to partition the viewsphere into a desired number of non-overlapping regions, such that views from points within a region are maximally similar to each other and maximally dissimilar to those from neighbouring regions. The centroid of each of these *stable view regions* is taken as its representative viewpoint.

The representative views thus obtained constitute $\mathscr{V}$, our required set of viewpoints. An example of stable view regions along with

their representative viewpoints is shown in Figure 2. The number of regions is usually low (we compute 12 by default) but may be tuned by a user as mentioned in Section 1, leading to a longer animation. A larger number of regions will lead to a finer granularity in representation of the shape. The effect on computation time is discussed in Section 7.

In [Yamauchi et al. 2006], the computation time reported for obtaining the similarity weighted viewsphere is around 40 minutes, with the bottleneck being the unoptimised similarity computations between views of resolution 256×256. We reduce the similarity computation time to a couple of seconds by comparing not the binary views but their extracted boundaries at a resolution of 400×400. We also replace the area normalisation step of the original method with the simpler one proposed in [Kamila et al. 2005].

3.2 Mesh and view saliencies

Figure 3: Saliency values for the Lion vase and Armadillo models, increasing from cold (blue) to warm (red) colours.

View saliency is derived from mesh saliency [Lee et al. 2005] which is a function that uses perception based factors to assign to each vertex in a mesh a scalar value corresponding to the saliency of the vertex in the shape. Higher values indicate higher saliency. For each vertex, intermediate saliency values are calculated at several scales. The final value is then a weighted average of the intermediate values. The idea is that a feature is salient if it is significantly different from its neighbourhood. The saliency of a view of the mesh is the sum of saliencies of visible vertices. Figure 3 provides visualisations of mesh saliency values for some of the models used in this paper.

The intermediate saliency values are computed at progressively larger scales, where the size of a scale corresponds to the size of the neighbourhood of the vertex considered for saliency computation. A high saliency value at a low scale indicates that the vertex belongs to a small shape feature, and a high value at a higher scale indicates a large feature. We shall make use of this information in Section 6 where we compute appropriate viewing scales.

4 Computing the path

Given the set, $\mathscr{V}$, of representative viewpoints from Section 3.1, we want the computed path, $\mathscr{P}$, to interpolate its members on the viewsphere. Also, we require our animation to run continuously on a shape repository webpage without any visible breaks. We pose these requirements on the computed path as the following additional constraints:

- *Interpolation* – the camera path should interpolate a given set of viewpoints.
- *Looping* – the camera path should be a cycle.

In [Sokolov et al. 2006b], the interpolation order is determined by first defining a distance function that favours viewpoints with higher view goodness values, computing all pairwise shortest paths between the viewpoints and then applying a TSP solver. This restricts the computed path to edges of the tessellated viewsphere, and fails the Smoothness condition from Section 1.

Looping is satisfied trivially by repeating the first point at the end when computing the ordering. As the path is a cycle, the choice of first point is meaningless. Once the ordering is determined, we interpolate the points on the sphere using cubic spherical splines [Watt and Watt 1991] (Chapter 15). An example of a computed path is shown in Figure 4. Our computation of viewpoint ordering is described below. A discussion on how well the resulting path satisfies the conditions listed in Section 2 is given in Section 8.

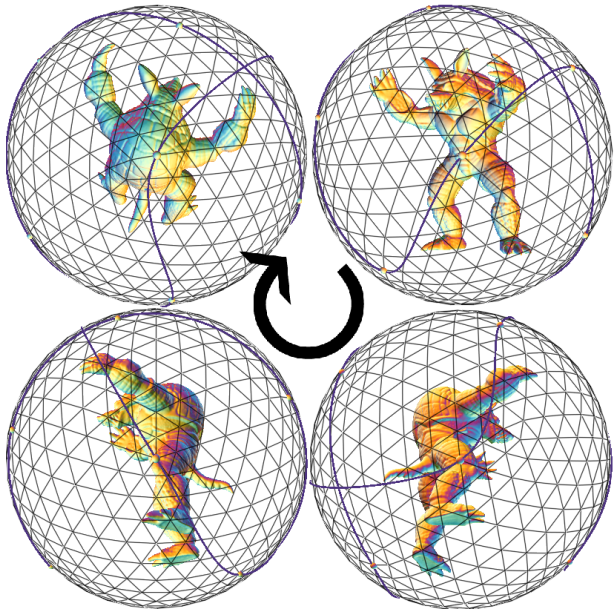

Figure 4: The computed path, $\mathscr{P}$, for the Armadillo model, shown in blue on the viewsphere. Dots on the viewsphere represent the interpolated viewpoints. The four images above are extracted from the animation in clockwise order.

For every 3 consecutive points in a potential ordering, V_i, V_j, $V_k \in \mathscr{V}$, consider the quantity

$$\Theta_{ijk} = |d(V_i,V_j) + d(V_j,V_k) - d(V_i,V_k)|,$$

where $d(A,B)$ is the spherical distance between points A and B. Θ_{ijk} gives a measure of the turn at V_j. For V_i, V_j, V_k lying on the same great circle, $\Theta_{ijk} = 0$. The computed ordering is the one which minimises $\sum_{V_j} \Theta_{ijk}$.

4.1 Up-vector consistency

Special consideration is given to the up-vectors of both the virtual camera and the models used. We use a default value for models'

up-vector, $\vec{U}_m = (0,1,0)$, which is consistent with most scanning systems. However, this is not robust and we manually fixed $\vec{U}_m$ for one of the five models used in this paper.

For proper orientation of the model in the animation, we keep $\vec{U}_c$, the up-vector of the camera, consistent with $\vec{U}_m$. At all times, $\vec{U}_c$ is chosen as the vector perpendicular to the viewing direction that is coplanar with $\vec{U}_m$. This gives two possible orientations for the up-vector. Indeed, there is a 'flip' in orientation at singular points, i.e. viewpoints with view direction parallel $\vec{U}_m$. The flip in $\vec{U}_c$ is necessary to maintain correct orientation of the model, otherwise, once the camera passes through the singular point, the model appears to be upside down.

5 Computing camera speed

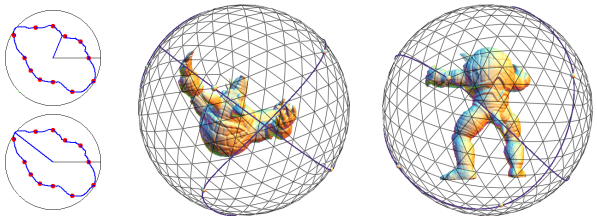

Figure 5: a) Minimum (top) and maximum (bottom) positions in the *speed clock* of the Armadillo animation. The 3 o' clock position represents the starting and ending point of the animation, during which the pointer moves clockwise. The length of the pointer represents the magnitude of the speed, and the dots represent the interpolated viewpoints. b) and c) show the views corresponding to the min and max positions respectively.

Camera speed, $\mathscr{S}$, determines the distance along $\mathscr{P}$from the current viewpoint to the next one. The motivation is that the camera should quickly fly by uninteresting views. Formally, we pose the following condition on the speed.

- *Saliency respecting* – the camera should slow down when passing over visually important regions of the shape, and speed up for uninteresting views.

We use the perception based measure of view saliency, $\mathscr{VS}$[Lee et al. 2005], to compute visual importance. The above formulation suggests an inverse relationship, $\mathscr{S} \propto \frac{1}{\mathscr{VS}}$. Taking the view that the purpose of our animation is to aid human understanding of the shape, we use the Two-Thirds Power Law (cf [de'Sperati and Viviani 1997] and references therein) from locomotion which relates tangential velocity, V, of free-hand movements to the radius of curvature, R, of the trajectory as follows:

$$V(t) = K \cdot \left(\frac{R(t)}{1 + \alpha \cdot R(t)} \right)^{1-\beta} \quad \alpha \geq 0,\ K \geq 0, \tag{1}$$

where K is a velocity gain constant, α is negligible if the trajectory does not have inflection points, and β is close to $\frac{2}{3}$ for adults. Putting in these values, we get

$$\begin{aligned} V(t) &= K \cdot (R(t))^{\frac{1}{3}} \\ &= K \cdot \left(\frac{1}{\kappa(t)} \right)^{\frac{1}{3}}, \end{aligned}$$

where $\kappa(t)$ is the curvature of the path. In our case, we want the speed to depend not on the curvature, but on $\mathscr{VS}$. Therefore we set

$$\mathscr{S}(t) = K_s \cdot \left(\frac{1}{\mathscr{VS}(t) + \gamma} \right)^{\frac{1}{3}}, \tag{2}$$

where γ is a constant offset to compensate for the 0 to 1 normalisation of $\mathscr{VS}$. Putting $\gamma = 1$ makes $\mathscr{S}$vary between K_s and $\frac{K_s}{\sqrt[3]{2}} \approx 0.79K_s$. A high value of K_s is thus needed for changes in speed to be discernible. Note that our use of cubic splines for interpolation technically invalidates the choice $\alpha = 0$ in Equation 1. However, we find that as a first approximation, the obtained results are quite satisfactory.

Equation 2 is in agreement with the inverse relation suggested earlier. The exponent dampens the effect of any irregularities in $\mathscr{VS}$. In Figure 5, we show an example of the computed speed function. As the static images only poorly convey the dynamic nature of the result, we urge the user to view the accompanying video for a better understanding.

6 Computing camera zoom

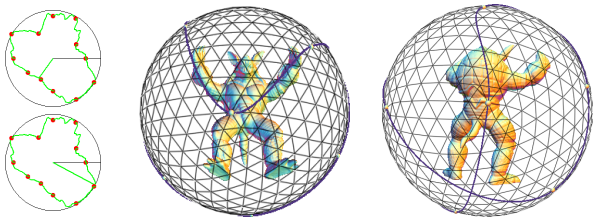

Figure 6: a) Minimum (top) and maximum (bottom) positions in the *zoom clock* of the Armadillo animation, where the zoom clock represents zoom in the same way as the speed clock in Figure 5 represents speed. b) and c) show the views corresponding to the min and max positions respectively.

The motivation behind a variable camera zoom, $\mathscr{Z}$, is the following.

- *Appropriate viewing scale* – the shape should be viewed at a scale that is in accordance with the size of the features being viewed.

In photography, zooming is achieved by changing the focal length of the camera lens. With the perspective projection of OpenGL that we use throughout his paper, this is equivalent to varying the distance between the camera and the object, i.e. placing the camera in the corresponding position on a viewsphere with a different radius; smaller radius for zooming in and larger for zoom out. Therefore, we compute $\mathscr{Z}$by computing the corresponding radius, $\mathscr{R}$.

Recall that saliency is computed in a multi-scale way (Section 3.2), where a higher value at a small scale implies a small scale feature, for which the camera should zoom in (small viewing radius). Correspondingly, a high saliency value at a high scale implies a large scale feature, which requires a large viewing radius for proper inspection. We thus define, for each scale i, a corresponding viewing radius $r_i \propto k\sigma_i$, where k is a constant and σ_i is the size of the vertex neighbourhood considered for saliency computation at scale i. The appropriate viewing radius, R, for a vertex, v, can then be computed as:

$$R(v) = \frac{\sum_i \mathrm{Sal}_i(v) r_i}{\sum_i \mathrm{Sal}_i(v)},$$

where $\mathrm{Sal}_i(v)$ is the saliency of v at scale i. For a view, V, an average viewing radius is computed as

$$\overline{R}(t) = \frac{\sum_{v \in V} R(v)}{n(v \in V)}. \quad (3)$$

After normalising $\overline{R}$ to $[0,1]$, we use Equation 1 to compute $\mathscr{R}$as

$$\mathscr{R}(t) = K_z(\overline{R}(t)^{\frac{1}{3}} + 1),$$

where K_z corresponds to the minimum value of $\overline{R}$.

Results for the Armadillo model are shown in Figure 6. Once again, for a better visualisation, we refer the reader to the accompanying video.

7 Results

We tested our method on several models and the results are shown in Figures 1, 4 and 7 to 9, and more comprehensibly, in the accompanying video. A summary of computation times is given in Table 1. Most of the time is spent on the mesh saliency calculation, which depends on the size of the mesh. For large models, e.g. the Buddha and Lion vase, this can be quite large. However, this is a one-time preprocessing step whose results are saved. This time also includes the computation of $R(v)$ from Section 6. The other preprocessing step – extracting $\mathscr{V}$ – uses image similarity and depends on the resolution of the views being compared. As we use the same resolution for all models, the time taken is the same for all models.

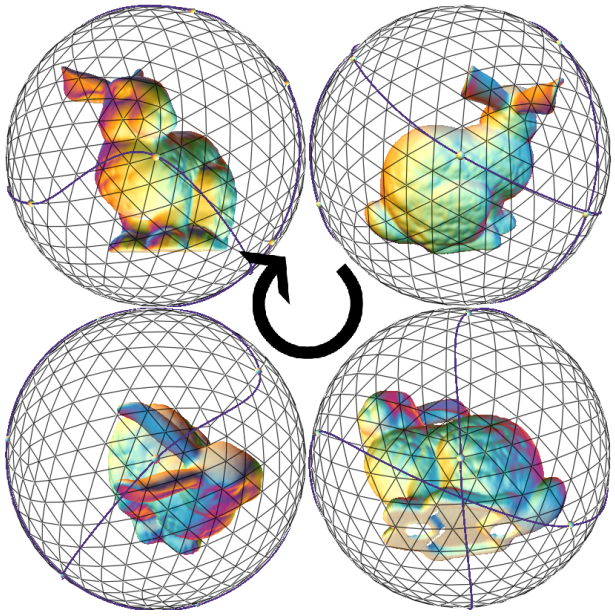

Figure 7: Scenes extracted in clockwise order from the computed animation of the bunny model.

Time taken for viewpoint ordering depends on the number of viewpoints being considered, and for constant number of viewpoints (12 in our case) is independent of model size. View saliency computation requires identifying visible mesh vertices from each viewpoint. We interpolate 12 viewpoints using 12 spherical cubic splines, and sample 50 points on each spline. We thus have to compute visible vertices for 600 viewpoints. The times for view saliency computation also include computation time for $\overline{R}$ from Section 6.

When the desired length of the animation is varied through the tunable parameter mentioned earlier, the pre-computed saliency and similarity values should be used. Extraction of stable view regions from the weighted viewsphere [Karypis and Kumar 1998] and computation of $\mathscr{V}$ then takes milliseconds. Viewpoint ordering and View saliency would have to be totally recomputed. The time for the former depends only on the chosen size of $\mathscr{V}$. In our experience, 12 viewpoints provide sufficient coverage of the object. View saliency computation, which also depends on model size, would also change markedly as varying the size of $\mathscr{V}$ varies the number of splines and hence the number of viewpoints in $\mathscr{P}$. Though this can be time consuming, we suffice with this solution as we do not aim to provide a real-time solution.

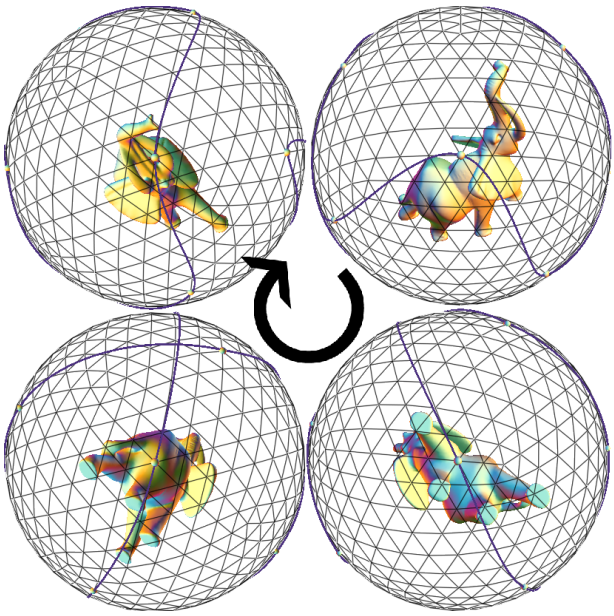

Figure 8: Scenes extracted in clockwise order from the computed animation of the Livingstone elephant model.

8 Discussion

We discuss how our method fares with respect to the four conditions mentioned in Section 1 – Brevity, Information, Exploration and Smoothness. Regarding Brevity, the length of the animation is tunable, as discussed in earlier sections. Once the points to be interpolated have been ordered, the shortest path consists of straight lines or geodesics. To ensure overall Smoothness, we interpolate using cubic splines. The extra length added by the smooth splines is compensated by speeding up the camera over low saliency views. Fulfilling the Smoothness condition and covering the viewsphere in a cyclic path invariably lead to a self-intersecting $\mathscr{P}$. This seemingly violates the Exploration condition, but we claim that by interpolating points representative of different, non-overlapping regions of the viewsphere, we have already fulfilled the condition. Lastly, we believe that by including the representative views of the shape, the path already conveys sufficient Information on the shape. Guiding it through intermediate 'good' views, as is traditionally done, will serve only to violate one of the other conditions. In addition, by allotting inspection times and viewing scales according to the visual importance of the shape features, we believe we are able to convey a lot more information about covered parts than previous methods that fly by the shape at fixed speeds and zooms.

	Vertices	Mesh Saliency	Extracting $\mathscr{V}$	Viewpoint ordering	View Saliency
Armadillo	172,974	761.42s	2s	15s	116.54s
Buddha	543,652	<1h	2s	15s	256.3s
Bunny	34,834	20.23s	2s	15s	41.0s
Elephant	20,007	12.73s	2s	15s	28.3s
Lion vase	800,002	<1.5h	2s	15s	576.5s

Table 1: Summary of computation times for a few models.

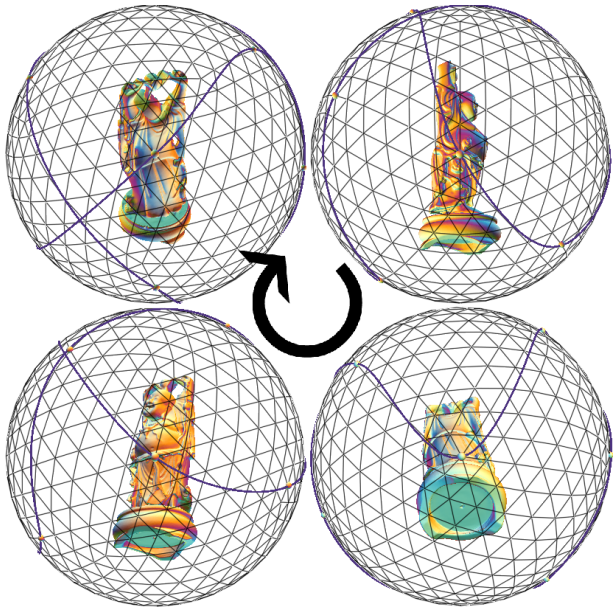

Figure 9: Scenes extracted in clockwise order from the computed animation of the Happy Buddha model.

The objective of our paper, to generate a short but informative fly around a given shape model, has a long history in the movie industry. It seems promising to study the techniques used in that area. However, caution will have to be exercised as movies often have a context – the story – which aids the determination of camera parameters. In a shape repository framework, there are no such helping factors.

One of the biggest problems we faced while developing the method was lack of feedback. In the absence of any formal measures to judge the quality of our output, it was difficult for us to ascertain whether we were on the right track. While, in principle, we can always ask a human observer to compare two different flies of a shape, the human visual system is quite lenient and seems to automatically compensate for any missing information. Indeed, the few people in our lab whom we did ask to compare such flies were unable to give a confident answer, and showing more flies served only to confuse and disorient. In fact, recent research [Henderson et al.] even puts into doubt the role of computational models of visual saliency in determining a human observer's attention. Significant advances in the fields of human cognition and psychology are required (which may just go on to verify the heuristics proposed by the graphics community) before a 'provably correct' best fly of a given shape model can be computed. Until then, the only barometer computer graphics practitioners have towards that goal is how well they fulfil their heuristics derived through observation, common sense and application requirements.

9 Conclusion and future work

In this paper, we presented a method to automatically compute a fly of a given shape model. A virtual camera pointing at the object is made to fly around the object following a computed trajectory, and with variable speed and zoom that depend on the shape's geometric features coupled with human perception factors. As there is no formal model for such a fly, we assemble the heuristics proposed in the literature and add some of our own, and compute a fly to fulfil them. In fact, altering the speed and zoom of the camera to produce a more informative fly of the shape has not been considered before.

While we follow the tradition of sampling an object's views from its viewsphere, after our experience in this paper, we are not totally convinced about the efficacy of this approach, as shape parts away from the centre of the object are seldom viewed directly. In our opinion, a viewing surface that is more faithful to the object's bounding box, e.g. an ellipsoid, is a better choice for this purpose.

Staying on the object's viewsphere also restricts us to viewing the entire object as a whole. It would be interesting to first segment the shape into meaningful parts, construct a fly around each part on its own viewsphere, and then blend all the flies together.

As our method performs a global exploration of the shape, it does not satisfactorily cover concave parts in the shape. In the future, we will extend our method to online exploration so that the virtual camera can enter the shape to visit its concave parts. Such a work has already been attempted in 2D [Brunstein et al. 2003] but a 3D analog is yet to be developed.

On a more general level, it will be interesting to see how our method scales to scenes with more than one object, and also to apply it to automatic terrain navigation.

Acknowledgements

The authors acknowledge the Stanford 3D Scanning Repository [3DScanRep] for the Armadillo, Bunny and Happy Buddha models and the AIM@SHAPE Shape Repository [AIM@SHAPE] for the Livingstone elephant and Lion vase models. We are also thankful to Boris Ajdin (MPII) for helpful discussions on up-vector consistency. This work was supported in part by the European FP6 NoE grant 506766 (AIM@SHAPE).

References

3DoD. 3D on demand. `http://www.3dod.org/`.

3DScanRep. The Stanford 3D scanning repository. http://graphics.stanford.edu/data/3Dscanrep/.

AIM@SHAPE. AIM@SHAPE shape repository. http://shapes.aimatshape.net/.

BORDOLOI, U., AND SHEN, H.-W. 2005. View selection for volume rendering. In *IEEE Visualization*, 487–494.

BRUNSTEIN, D., BAREQUET, G., AND GOTSMAN, C. 2003. Animating a camera for viewing a planar polygon. In *VMV '01: Proceedings of the Vision Modeling and Visualization Conference 2001*, 87–94.

DENTON, T., DEMIRCI, M. F., ABRAHAMSON, J., SHOKOUFANDEH, A., AND DICKINSON, S. 2004. Selecting canonical views for view-based 3-D object recognition. In *ICPR '04: Proceedings of the 17th International Conference on Pattern Recognition*, IEEE Computer Society, 273–276.

DE'SPERATI, C., AND VIVIANI, P. 1997. The relationship between curvature and velocity in two-dimensional smooth pursuit eye movements. *The Jounral of Neuroscience 17*, 10, 3932–3945.

HALL, P. M., AND OWEN, M. J. 2005. Simple canonical views. In *The British Machine Vision Conf. (BMVC'05)*, vol. 1, 7–16.

HENDERSON, J. M., BROCKMOLE, J. R., CASTELHANO, M. S., AND MACK, M. Visual saliency does not account for eye movements during search in real-world scenes. In *Eye Movements: A Window on Mind and Brain*, R. van Gompel, M. Fischer, W. Murray, and R. Hill, Eds. Elsevier. to appear.

KAMILA, N. K., MAHAPATRA, S., AND NANDA, S. 2005. Invariance image analysis using modified Zernike moments. *Pattern Recogn. Lett. 26*, 6, 747–753.

KARYPIS, G., AND KUMAR, V. 1998. MeTiS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. version 4.0. In *http://www-users.cs.umn.edu/˜karypis/metis/*. Univ. of Minnesota, Dept. of Computer Science.

KHOTANZAD, A., AND HONG, Y. H. 1990. Invariant image recognition by Zernike moments. *IEEE Trans. on Pattern Analysis and Machine Intelligence 12*, 5, 489–497.

LEE, J., MOGHADDAM, B., PFISTER, H., AND MACHIRAJU, R. 2004. Finding optimal views for 3D face shape modeling. In *International Conf. on Automatic Face and Gesture Recognition*, IEEE Computer Society, 31–36.

LEE, C. H., VARSHNEY, A., AND JACOBS, D. W. 2005. Mesh saliency. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2005) 24*, 3, 659–666.

LGMA. Large geometric models archive at Georgia Tech. `http://www.cc.gatech.edu/projects/large_models/`.

MOKHTARIAN, F., AND ABBASI, S. 2000. Automatic selection of optimal views in multi-view object recognition. In *The British Machine Vision Conf. (BMVC'00)*, IEEE Computer Society, 272–281.

NDR. National design repository. `http://www.designrepository.org/`.

PDB. Protein data bank. `http://www.pdb.org/`.

PODOLAK, J., SHILANE, P., GOLOVINSKIY, A., RUSINKIEWICZ, S., AND FUNKHOUSER, T. 2006. A planar-reflective symmetry transform for 3D shapes. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2006)*, 549–559.

POLONSKY, O., PATANÉ, G., BIASOTTI, S., GOTSMAN, C., AND SPAGNUOLO, M. 2005. What's in an image? In *Visual Computer*, Springer-Verlag, 840–847. Proc. of Pacific Graphics 2005.

PSB. Princeton shape benchmark. `http://shape.cs.princeton.edu/benchmark/`.

SOKOLOV, D., AND PLEMENOS, D. 2005. Viewpoint quality and scene understanding. In *VAST 2005: Eurographics Symposium Proceedings.*, 67–73.

SOKOLOV, D., PLEMENOS, D., AND TAMINE, K. 2006. Methods and data structures for virtual world exploration. *The Visual Computer 22*, 7, 506–516.

SOKOLOV, D., PLEMENOS, D., AND TAMINE, K. 2006. Viewpoint quality and global scene exploration strategies. In *International Conference on Computer Graphics Theory and Applications (GRAPP 2006)*, INSTICC - Institute for Systems and Technologies of Information, Control and Communication.

TAKAHASHI, S., FUJISHIRO, I., TAKESHIMA, Y., AND NISHITA, T. 2005. A feature-driven approach to locating optimal viewpoints for volume visualization. In *IEEE Visualization*, 495–502.

VÁZQUEZ, P.-P., FEIXAS, M., SBERT, M., AND HEIDRICH, W. 2001. Viewpoint selection using viewpoint entropy. In *VMV '01: Proceedings of the Vision Modeling and Visualization Conference 2001*, 273–280.

VÁZQUEZ, P.-P., FEIXAS, M., SBERT, M., AND HEIDRICH, W. 2003. Automatic view selection using viewpoint entropy and its applications to image-based modelling. *Computer Graphics Forum 22*, 4, 689–700.

WATT, A., AND WATT, M. 1991. *Advanced Animation and Rendering Techniques*. ACM Press.

YAMAUCHI, H., SALEEM, W., YOSHIZAWA, S., KARNI, Z., BELYAEV, A., AND SEIDEL, H.-P. 2006. Towards stable and salient multi-view representation of 3D shapes. In *IEEE International Conference on Shape Modeling and Applications 2006 (SMI2006)*, 265–270.