Visual Comput (2005) 21: 698–706
DOI 10.1007/s00371-005-0323-1

ORIGINAL ARTICLE

Detail control in line drawings of 3D meshes

Kyuman Jeong
Alex Ni
Seungyong Lee
Lee Markosian

Published online: 1 September 2005

K. Jeong (✉) · S. Lee
Department of Computer Science and Engineering, POSTECH, Pohang, 790-784, Korea
{misterq,leesy}@postech.ac.kr

A. Ni · L. Markosian
Department of Electrical Engineering and Computer Science, University of Michigan at Ann Arbor, Ann Arbor, MI 48109, USA
{alexni,sapo}@umich.edu

Abstract We address the problem of rendering a 3D mesh in the style of a line drawing, in which little or no shading is used and instead shape cues are provided by silhouettes and suggestive contours. Our specific goal is to depict shape features at a *chosen scale*. For example, when mesh triangles project into the image plane at subpixel sizes, both suggestive contours and silhouettes may form dense networks that convey shape poorly. The solution we propose is to convert the input mesh to a multiresolution representation (specifically, a progressive mesh), then view-dependently refine or coarsen the mesh to control the size of its triangles in *image space*. We thereby control the scale of shape features that are depicted via silhouettes and suggestive contours. We propose a novel refinement criterion that achieves this goal and address the problem of maintaining temporal coherence of silhouette and suggestive contours when extracting them from a changing mesh.

Keywords Nonphotorealistic rendering · Line drawing · Level-of-detail · Progressive mesh

1 Introduction

Nonphotorealistic rendering (NPR) has emerged as an alternative to the traditional holy grail of 3D computer graphics: to produce images that are as realistic as possible. An important goal of NPR is to produce images that are *comprehensible*—they communicate information about shape effectively. The simplest kind of NPR is a *line drawing*, in which little or no shading is used, and instead just a few strokes are drawn along important shape features. Line drawings are important in many applications where a simple, clear rendering is preferred, and they often serve as the basis for more complicated styles as well.

Silhouettes (also known as contours) and sharp features are widely considered to be good candidate locations for strokes [7]. Recently, suggestive contours [1, 2] have been proposed to provide additional cues about shape with relatively few additional strokes. However, one question has been largely overlooked: how to generate strokes that provide cues about surface shape *at an appropriate scale*. For example, consider rendering a terrain that includes nearby surfaces as well as hills and mountains in the distance. If every silhouette and suggestive contour is included, their density over distant surfaces would be so great that it would destroy our ability to perceive *any* details there. The problem is that features can exist at multiple scales, and we want to omit those that are too small (or large) to be conveyed effectively by a line in image space.

The solution we propose is to convert the mesh to a continuous level-of-detail representation (specifically, a progressive mesh [5]), then dynamically adjust the mesh resolution at runtime so that triangle sizes are controlled in image space. The result is that we control the scale of shape features, e.g., to achieve a uniform level of detail across the image, or to manipulate detail according to some arbitrary "detail map." We propose a novel refinement criterion that achieves this goal and a strategy for maintaining temporal coherence of suggestive contour and

silhouette paths when extracting them from a changing mesh.

2 Related work

Most NPR renderers of 3D models address level of detail in some way. Winkenbach and Salesin [18, 19] describe algorithms to render 3D scenes in the style of pen and ink drawings, using "prioritized textures" to control the resulting level of detail. Praun et al. [13] and Webb et al. [16] describe a way to efficiently render hatching strokes on 3D models, varying the density of strokes to account for changes in camera and lighting. The procedural "graftal textures" of Kowalski et al. [9] and Markosian et al. [11] view-dependently generate 3D elements that suggest stylized leaves, tufts of fur, bumps, and so on, with a controlled level of detail. Deussen and Strothotte [3] present a related method for rendering trees in a pen and ink style with a user-selectable degree of abstraction. The more abstract renderings resemble low-detail sketches.

While the above systems rely on procedural models that generate detail as needed, Grabli et al. [4] and Wilson and Ma [17] both describe systems that render *arbitrary* complex geometry (i.e., meshes) with control over stroke density and visual "clutter." Both systems combine 3D operations with 2D image processing to detect regions of visual complexity (e.g., dense silhouettes) and remove detail where it exceeds a threshold. Both systems require significant processing time, and neither addresses temporal coherence. While these systems do address the problem of controlling stroke density in image space, neither specifically addresses the problem of depicting shape features at a desired scale in image space.

Kirsanov et al. [8] propose a method for extracting "simple silhouettes" from dense meshes based on first extracting the silhouette from a simplified version of the mesh. Their goal is different from ours: they do not address the problem of detecting silhouettes at a controlled scale in image space or of achieving interactive rates and temporal coherence of extracted lines.

Most closely related to our work is another paper we have written that addresses the same problem using a different approach [12]. The method we propose in that paper is to first generate from the original mesh a sequence of smoothed meshes (with the same connectivity) with progressively larger shape features removed. At runtime, for each vertex we compute an interpolated position and curvature from two corresponding vertices in the mesh sequence, depending on the desired feature scale. Because of its relative simplicity, this method can be implemented entirely on the GPU, and so is quite fast (e.g., 70 fps for a 400,000-polygon model on the same test machine used here). It effectively controls the scale of depicted shape features and, because mesh connectivity does not vary across detail levels, achieves a high degree of temporal coherence. An important limitation is that multiple versions of the original mesh (smoothed at different scales) must be stored simultaneously. As a result, it does not scale well as more levels of detail are added and larger meshes are used (e.g., in the case of large terrain datasets). The method we describe in this paper theoretically scales much better in these cases. It still provides effective control over the scale of depicted features and reasonably good temporal coherence, despite actively changing mesh connectivity.

3 Our approach

Our high-level goal is to provide a general means of controlling the scale of features depicted in an image. For instance, the rendering might be controlled by a "detail map" according to which one object (some background scenery) is rendered with significantly less detail than an object of interest (e.g., the central character in a story).

In signal processing terms, we seek to band-limit a "signal" (i.e., the shape) and control our sampling (i.e., the mesh) of that signal accordingly. We can achieve an approximation of this by smoothing and simplifying the mesh. Then, a coarser mesh will yield a line drawing that depicts coarser-scale features and a finer mesh will yield finer-scale features. We achieve this using a progressive mesh scheme, as we describe next.

3.1 Progressive mesh refinement criterion

To dynamically control the mesh resolution, we use the view-dependent refinement scheme for progressive meshes described by Hoppe [6]. Progressive meshes provide mesh decimation and smoothing via edge collapse operations and mesh refinement via vertex splits. The decision of when to invoke each type of operation is made according to a *criterion*, i.e., a function that evaluates a vertex and returns true if the vertex should be split and false if its child edge (in the progressive mesh hierarchy) should be collapsed.

The original goal for view-dependent refinement of progressive meshes was to improve rendering speed by reducing geometric detail while minimizing perceptible error in the rendered image. Thus the criterion proposed by Hoppe acts to refine the mesh only where its image space deviation from the original model exceeds a threshold.

Our goal is different. We thus propose a criterion that acts to refine the progressive mesh only where its triangles exceed a target size in image space.[1] More specifically, our criterion returns "true" for a vertex if the average size of triangles in its 1-ring exceeds the target size (in

[1] We could have based our criterion on *edge length* instead. Either formulation effectively captures the notion of feature scale.

image space) by a given threshold. The result is to promote uniform triangle sizes in 2D or, more generally, triangle sizes that can vary in 2D according to a given detail map.

We map a given 3D area A, located some distance d from the camera, to the corresponding image space area A_{img} (ignoring foreshortening due to orientation) as follows. For perspective projection, $A_{img} = kA/d^2$, where k is a scaling factor determined by the image and camera parameters (e.g., field of view). For orthographic projection, $A_{img} = kA$.

Hoppe's criterion for evaluating a vertex depends only on the vertex itself (its position) and precomputed constant values associated with the vertex. Our criterion, however, depends on the average triangle size around the vertex, which increases with each edge collapse operation, and decreases with each vertex split operation. We thus need two thresholds: $t_c < 1$ and $t_s > 1$. Denoting the target area by T, we invoke an edge collapse when the average triangle area around a vertex falls below $t_c T$ and a vertex split when the area exceeds $t_s T$. To avoid a cycle where a collapse immediately triggers the reverse (split) operation, the two thresholds must be sufficiently separated.

In general, the average triangle area around a vertex can increase arbitrarily during an edge collapse. In practice, however, we observe that the increase tends to stay well below a factor of 2. We thus require that $2t_c < t_s$. Taking $t_c t_s \approx 1$ for symmetry, we therefore need $t_c < 1/\sqrt{2}$ and $t_s > \sqrt{2}$. We use $t_c = 0.7$ and $t_s = 1.42$, which in our observations is sufficient to prevent cycles.

Alternatively, we could use a strategy similar to Hoppe's and precompute the area around each vertex at different levels of the progressive mesh hierarchy. This would eliminate the dependency of the refinement criterion on the neighborhood of each vertex, and so prevent cycles. We intend to investigate this strategy in future work.

3.2 Temporal coherence of geometry

Edge collapse and vertex split operations lead to abrupt visual changes ("popping"). To address this, Hoppe introduced geomorphs [6], which interpolate the shape of one progressive mesh approximation to another over a short period of time. Hoppe's method prevents sudden changes in the shape of the mesh, but it is restrictive—it generates a sequence of intermediate meshes between an initial and final configuration, and no further changes may be initiated until the final configuration is reached. This is not well-suited to a real-time system under interactive user control.

To solve this problem, we introduce a new type of progressive mesh geomorph called a *real-time geomorph*. Instead of morphing the entire mesh between an initial and final configuration, we morph individual parts independently and concurrently. That is, in any frame, new geomorphs may be initiated while others, already in progress, either continue or complete their transitions.

When an edge collapse is invoked, instead of directly performing the operation, each of the edge's vertices are temporarily "locked," so that no further operations that directly involve them may be initiated. The vertices are then moved over a transition period to meet at the position of the edge's parent. When the vertices meet, the logical edge collapse operation is executed, and the vertices are unlocked.

Whenever a vertex split operation is invoked, the logical vertex split is performed immediately, but the child vertices are initially positioned at their parent (the source of the split). The children then move over the transition period to their final positions and are unlocked upon completion.

The result is that level-of-detail changes in the mesh happen continuously, but a delay is introduced. If the viewpoint changes rapidly, the displayed progressive mesh configuration may lag significantly behind the desired one. We address this by increasing the speed of the interpolation based on the discrepancy between the current and target areas associated with each vertex. Increasing the speed of the geomorphs does reduce the perceived coherence of mesh transitions, but this happens only for rapid view changes, when coherence is difficult to notice.

3.3 Temporal coherence of suggestive contours

Though real-time geomorphs prevent sudden changes in the shape of the mesh, suggestive contours extracted from the mesh can still change abruptly. The reason is that with geomorphs, vertex *locations* change smoothly but their *connectivity* does not—and suggestive contours extracted from meshes depend on an approximation of curvature, which in turn depends on both vertex locations *and* connectivity [14]. We could try to avoid this dependency by using "exact" curvature (from the original shape) or curvature computed over the original mesh, but neither would be useful in depicting shape features at a desired scale. Instead we need a notion of "curvature at a scale." In practice, if the mesh triangle size is controlled appropriately, the "approximation" of curvature we compute from the mesh is a good substitute for the curvature at a scale that we really want.

To address the problem of popping of suggestive contours, we do not directly replace the old per-vertex curvatures and normals in each frame with the new values computed over the progressive mesh. Instead, we take a weighted average of the new and old values, with a larger weight assigned to the old value. The result is that both the interpolated curvature and its derivative (used in suggestive contour extraction) change gradually over time.

Note that each edge split or vertex collapse affects curvature values at *multiple* vertices (specifically, in the 2-ring neighborhood of the operation). To simplify bookkeeping, we choose a strategy of interpolating curvature values multiplicatively (instead of addivitively), as follows. Let p_k denote the value already stored in the mesh at frame k, and let q denote the newly computed value. For purposes of analysis, assume that q is fixed (e.g., the camera and progressive mesh have recently stopped changing). If w is the weight assigned to q, the interpolation $p_{k+1} = (1-w)\,p_k + wq$ forms a geometric series and after n frames $p_n = (1-w)^n\,p_0 + (1-(1-w)^n)q$. Therefore, if we want to reach a given fraction α of q over time interval t, and the frame rate is f frames per second, then $n = ft$, and $w = 1-(1-\alpha)^{\frac{1}{n}}$. For example, if $t = 0.8$ sec, $f = 25$ fps, and $\alpha = 0.9$, then we should take $w \approx 0.1087$.

We apply this interpolation to surface normals (used in extracting silhouettes) and also to radial curvature and its derivative in the radial direction (used in extracting suggestive contours). Since radial curvature is view dependent, this means that the interpolated radial curvature in the current frame is influenced by the viewing direction of recent frames. This introduces a small delay in the response of suggestive contours to the changing view vector during camera or object rotations. This effect is not readily apparent in most cases but can sometimes be noticed when the camera stops abruptly—suggestive contours continue to evolve for a brief time until they "catch up" to the current view.

4 Results and discussion

The still images in this paper and the accompanying video demonstrate our method for controlling detail in line drawings of 3D meshes. We used Rusinkiewicz's publically available *real-time suggestive contour* source code [15] for computing curvatures and rendering suggestive contours. The line drawings are augmented with a simple toon shader [10] that helps to convey shape. Note that our technique for controlling detail in the number and placement of strokes also controls detail in the toon shader.

Our test machine has a 2.8 GHz Pentium 4 processor with 1 GB main memory and a 128 MB ATI Radeon 9800 Pro Graphics Card. Our system achieves interactive rates (5 to 30 fps) for models up to around 100,000 polygons. The video clips were captured in real time.

The Buddha model on the left in Fig. 1 contains over 1 million polygons. The definition of suggestive contours, based on curvature, does not consider the scale at which the model is viewed, and so in this full-detail rendering many small (subpixel) "features" appear as specks in the

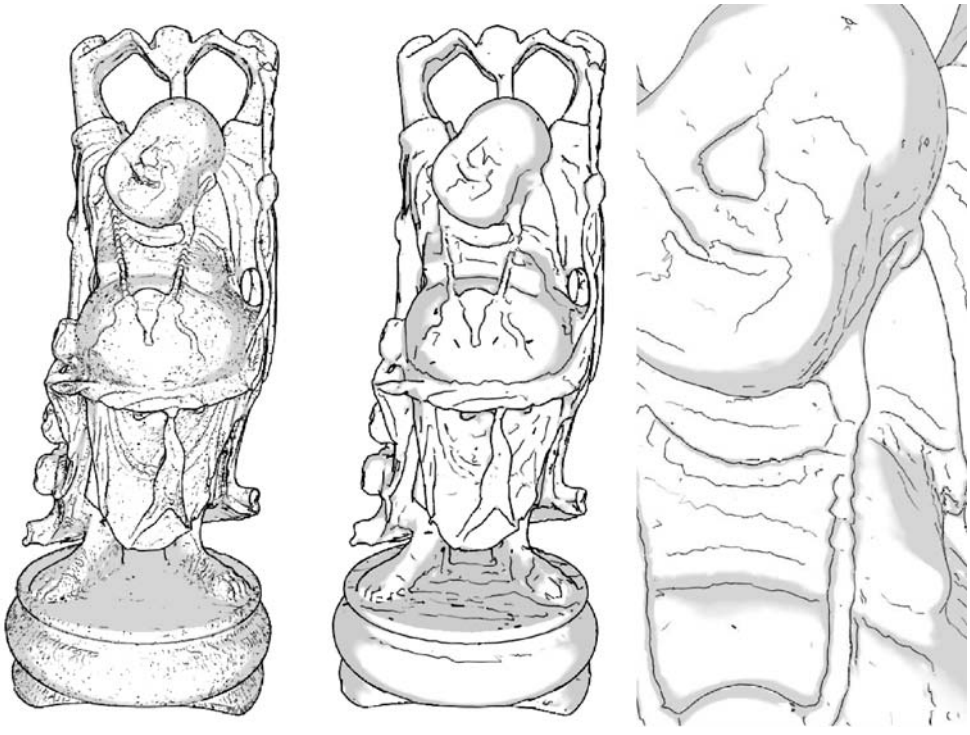

Fig. 1. *Left*: Full-detail Buddha model (∼1 million polygons). Silhouettes and suggestive contours at fine scales appear as many small *specks*. *Middle*: Controlled detail (∼70,000 polygons). *Right*: More detail is generated when the camera zooms in

image. The image in the middle was generated with a controlled level of detail. The approximation contains roughly 70,000 triangles, which is sufficient to convey its important features. On the right the camera has zoomed in. In response, the system has increased the level of detail accordingly.

The skull in Fig. 2 shows another example of a high-detail mesh that leads to many suggestive contours corresponding to small undulations in the surface. With a controlled level of detail these small features are eliminated while larger ones are retained.

Figure 3 shows how our method varies the amount of detail in the rendered image with distance. The full-resolution elephant model contains wrinkles along the length of the trunk. The tip of the trunk in the left image is far enough away that lines there have been eliminated. The middle and right images, rendered from further away, show even less detail. Figure 4 shows additional examples of detail varying inversely with distance from the camera.

The landscape in Fig. 5 is rendered with high detail on the left and reduced detail on the right. For this image we did not use a constant 2D target size for triangles. Instead, target sizes grow with distance from the camera. The effect is that detail is reduced disproportionately for distant parts of the model compared to those in the foreground.

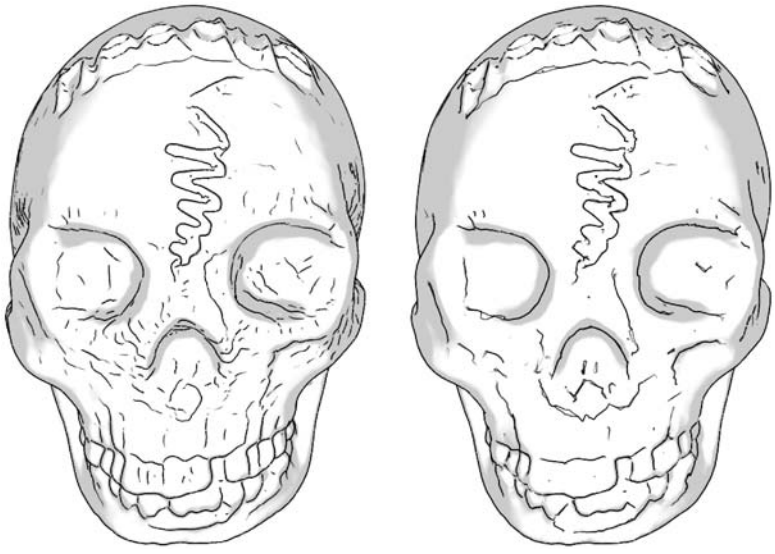

Fig. 2. In the full-detail model on the *left*, suggestive contours convey many small undulations in the surface. The reduced-detail version on the *right* elminates these small-scale shape features but preserves larger ones

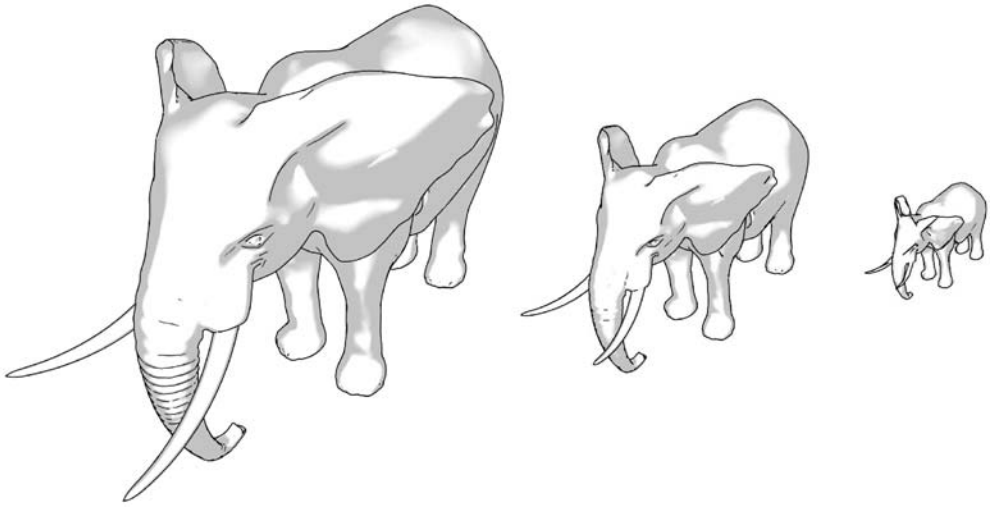

Fig. 3. Detail of this elephant model is reduced with increasing distance from the camera

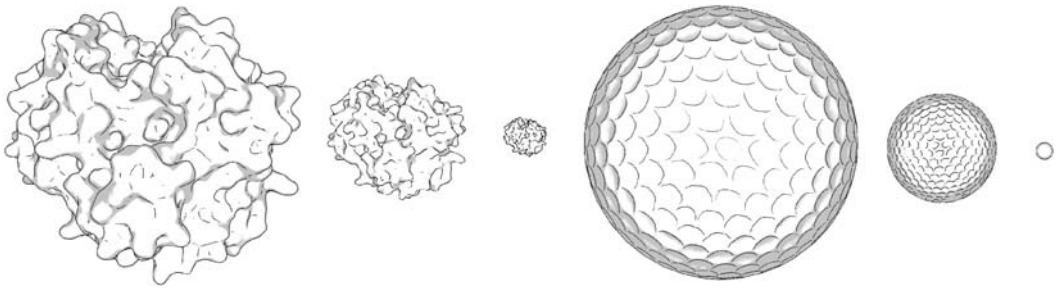

Fig. 4. Detail on this molecule model (*left*) and golfball model (*right*) is reduced incrementally as the camera zooms out

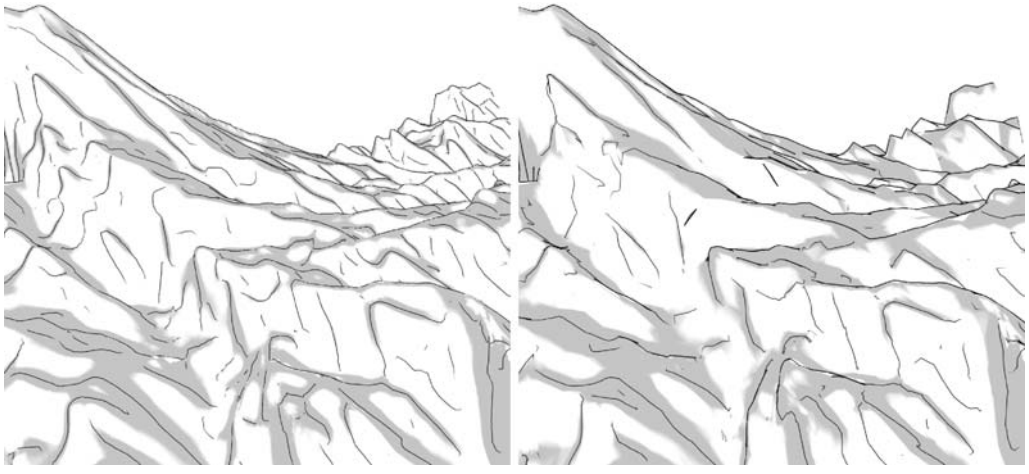

Fig. 5. A terrain model rendered with full detail (*left*) and reduced detail (*right*). On the *right*, the image space target size was made to increase with distance from the camera. Detail thus drops off disproportionately for distant regions compared to the foreground

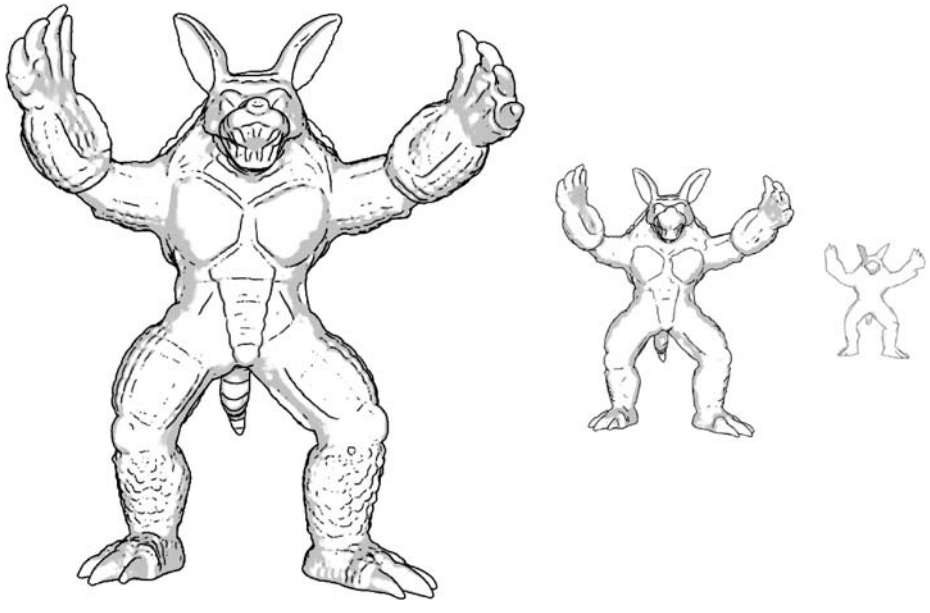

Fig. 6. The armadillo model rendered with different levels of detail. Small features in the arms and legs disappear first as the camera zooms out

Fig. 7. The feline model rendered with different levels of detail. The fine details in the wings fade out according to the distance from the viewer

Figures 6 and 7 show additional results. Small bumps on the arms and legs of the armadillo model in Fig. 6 are eliminated first, followed by larger shape features as the camera zooms out. In Fig. 7, fine details in the wings of the feline model fade out as distance to the viewer increases.

The accompanying video demonstrates additional results, as well as the temporal behavior of silhouettes and suggestive contours in our system. Although strokes are not entirely stable, it is clear that the combination of geomorphs and smoothing of normals and curvatures over time does produce more temporally coherent line drawings.

In comparison, the temporal behavior of our other method [12] (based on a precomputed sequence of smoothed meshes) is significantly more stable.[2] Maintaining a stable distribution of radial curvature values in the presence of edge collapses and vertex splits is challenging, and our approach is not completely successful. On the other hand, the high memory overhead of the other method means that it does not scale well for large models or models requiring many levels of detail.

An additional limitation of the progressive-mesh-based method is that it cannot be applied effectively to models with complex structure (e.g., the Eiffel Tower), since progressive meshes do not work well with such models.

[2] The bibliography reference provides a URL from which the paper and accompanying video can be downloaded.

4.1 Future work

We envision several avenues for future work. We think the idea of rendering shape features at a desired scale deserves more attention than it has received so far in the computer graphics literature. More often, the focus of level-of-detail algorithms has been to increase efficiency without reducing the quality of the image. Instead, we are concerned with manipulating detail *in the image*.

While the algorithm we presented is reasonably effective at controlling detail in the rendered image and providing control over the scale of shape features that are depicted, it does have some limitations. In particular, using triangle size to control detail is only suitable for some size ranges. When triangles grow too large, visual quality is reduced. A better approach would be to use a multiresolution shape representation that could eliminate detail below a desired scale while producing consistent (small) triangle sizes in image space. Where detail is eliminated the surface should be smooth. Such a representation could also solve the excess memory problem of our other method. We are thus interested in pursuing an alternative representation that has these qualities.

While the algorithm we described can be combined with a detail map that varies over the image or in 3D, we have so far not explored this possibility very far. Ideas for doing so include increasing detail in the center of the image, in a range of depths determined by a model of the depth of field of the camera, or for slowly moving objects compared to fast-moving ones. We expect that addressing the topic of detail in the rendered image more generally,

for styles beyond line drawings, will remain an active area of research, especially in nonphotorealistic computer graphics.

Acknowledgement We thank Szymon Rusinkiewicz for making the real-time suggestive contours source code available on the Web and Junho Kim for the implementation of selective refinement of progressive meshes. The bunny, armadillo, and "Happy Buddha" models are courtesy of the Stanford Computer Graphics Laboratory. The Venus, feline, and elephant models are courtesy of Cyberware, the Caltech Multiresolution Modeling Group, and Espona, respectively. The terrain model is courtesy of the United States Geological Survey and the University of Washington. This work was supported in part by the Korean Ministry of Education through the BK21 program, the Korean Ministry of Information and Communication through the ITRC support program, and the NSF (CCF 0447883).

References

1. DeCarlo, D., Finkelstein, A., Rusinkiewicz, S.: Interactive rendering of suggestive contours with temporal coherence. In: NPAR 2004: 3rd International Symposium on Non Photorealistic Rendering, pp. 15–24 (2004)
2. DeCarlo, D., Finkelstein, A., Rusinkiewicz, S., Santella, A.: Suggestive contours for conveying shape. ACM Trans. Graph. **22**(3), 848–855 (2003)
3. Deussen, O., Strothotte, T.: Computer-generated pen-and-ink illustration of trees. In: Proceedings of ACM SIGGRAPH 2000, Computer Graphics, pp. 13–18 (2000)
4. Grabli, S., Durand, F., Sillion, F.: Density measure for line-drawing simplification. In: Proceedings of Pacific Graphics (2004)
5. Hoppe, H.: Progressive meshes. In: Proceedings of SIGGRAPH 96, Computer Graphics, pp. 99–108 (1996)
6. Hoppe, H.: View-dependent refinement of progressive meshes. In: Proceedings of SIGGRAPH 97, Computer Graphics, pp. 189–198 (1997)
7. Isenberg, T., Freudenberg, B., Halper, N., Schlechtweg, S., Strothotte, T.: A developer's guide to silhouette algorithms for polygonal models. IEEE Comput. Graph. Appl. **23**(4), 28–37 (2003). http://dx.doi.org/10.1109/MCG.2003.1210862
8. Kirsanov, D., Sander, P.V., Gortler, S.J.: Simple silhouettes for complex surfaces. In: SGP '03: Proceedings of the 2003 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing, pp. 102–106. Eurographics Association, Aire-la-Ville, Switzerland (2003)
9. Kowalski, M.A., Markosian, L., Northrup, J.D., Bourdev, L., Barzel, R., Holden, L.S., Hughes, J.F.: Art-based rendering of fur, grass, and trees. In: Proceedings of SIGGRAPH 99, Computer Graphics, pp. 433–438 (1999)
10. Lake, A., Marshall, C., Harris, M., Blackstein, M.: Stylized rendering techniques for scalable real-time 3d animation. In: NPAR 2000: 1st International Symposium on Non Photorealistic Animation and Rendering, pp. 13–20 (2000)
11. Markosian, L., Meier, B.J., Kowalski, M.A., Holden, L.S., Northrup, J.D., Hughes, J.F.: Art-based rendering with continuous levels of detail. In: NPAR 2000: 1st International Symposium on Non Photorealistic Animation and Rendering, pp. 59–66 (2000)
12. Ni, A., Jeong, K., Lee, S., Markosian, L.: Multi-scale line drawings from 3D meshes. http://graphics.eecs.umich.edu/npr/msld/ (2005)
13. Praun, E., Hoppe, H., Webb, M., Finkelstein, A.: Real-time hatching. In: Proceedings of ACM SIGGRAPH 2001, Computer Graphics, pp. 579–584 (2001)
14. Rusinkiewicz, S.: Estimating curvatures and their derivatives on triangle meshes. In: Symposium on 3D Data Processing, Visualization, and Transmission (2004)
15. Rusinkiewicz, S.: Real-time suggestive contours, v. 1.2. http://www.cs.princeton.edu/gfx/proj/sugcon/ (2004)
16. Webb, M., Praun, E., Finkelstein, A., Hoppe, H.: Fine tone control in hardware hatching. In: NPAR 2002: 2nd International Symposium on Non Photorealistic Rendering, pp. 53–58 (2002)
17. Wilson, B., Ma., K.L.: Representing complexity in computer-generated pen-and-ink illustrations. In: NPAR 2004: 3rd International Symposium on Non Photorealistic Rendering (2004)
18. Winkenbach, G., Salesin, D.H.: Computer-generated pen-and-ink illustration. In: Proceedings of SIGGRAPH, Computer Graphics, pp. 91–100 (1994)
19. Winkenbach, G., Salesin, D.H.: Rendering parametric surfaces in pen and ink. In: Proceedings of SIGGRAPH, Computer Graphics, pp. 469–476 (1996)




KYUMAN JEONG is a Ph.D. student in the Computer Graphics Laboratory at POSTECH, Korea. He received his B.S. in computer science from the Korea Institute of Science and Technology in 1998 and his M.S. in computer science and engineering from POSTECH in 2000. His research is in controlling detail in NPR and developing an oriental black-ink rendering system.

ALEX NI received his B.S.E. degree in computer science and electrical engineering from the University of California, Berkeley, in 2003. He went on to earn a master's degree in computer science at the University of Michigan in 2005. He currently works as a developer at Visual Concepts in San Rafael, CA. His research interests include nonphotorealistic rendering, level of detail methods, and hardware-accelerated rendering techniques.

SEUNGYONG LEE is an associate professor of computer science and engineering at Pohang University of Science and Technology (POSTECH), Korea. He received his B.S. degree in computer science and statistics from the Seoul National University in 1988 and his M.S. and Ph.D. in computer science from the Korea Advanced Institute of Science and Technology (KAIST) in 1990 and 1995, respectively. From 1995 to 1996 he worked at the City College of New York as a research associate, where he further extended his Ph.D. work on image morphing. Since 1996 he has been a faculty member and leading the Computer Graphics Group at POSTECH. From 2003 to 2004 he spent a sabbatical year at MPI Informatik in Germany as a visiting senior researcher. His current research interests include geometry and mesh processing, nonphotorealistic rendering, and mobile graphics systems. He is a member of ACM and the IEEE Computer Society.

LEE MARKOSIAN is an assistant professor of computer science at the University of Michigan. He received his Ph.D. in computer science from Brown University in 2000, spent the next 3 years as a postdoctoral research associate at Princeton University, and joined the faculty of the University of Michigan in 2003. His research interests include nonphotorealistic rendering and sketch-based modeling of 3D shapes.