Visual Comput (2008) 24: 239–247
DOI 10.1007/s00371-007-0201-0

ORIGINAL ARTICLE

Yotam Livny
Michael Press
Jihad El-Sana

Interactive GPU-based adaptive cartoon-style rendering

Published online: 25 December 2007

Y. Livny (✉) · M. Press · J. El-Sana
Ben-Gurion University of the Negev,
Beer-Sheva, Israel
{livnyy, pressmi, el-sana}@cs.bgu.ac.il

Abstract In this paper we present a novel real-time cartoon-style rendering approach, which targets very large meshes. Cartoon drawing usually uses a limited number of colors for shading and emphasizes special effects, such as sharp curvature and silhouettes. It also paints the remaining large regions with uniform solid colors. Our approach quantizes light intensity to generate different shadow colors and utilizes multiresolution mesh hierarchy to maintain appropriate levels of detail across various regions of the mesh. To comply with visual requirements, our algorithm exploits graphics hardware programmability to draw smooth silhouette and color boundaries within the vertex and fragment processors. We have adopted a simplification scheme that executes simplification operators without incurring extra simplification operations as a precondition. The real-time refinement of the mesh, which is performed by the graphics processing unit (GPU), dramatically improves image quality and reduces CPU load.

Keywords Cartoon rendering · View-dependent rendering · Hardware acceleration

1 Introduction

Cartoon-style drawing conveys a great deal of information by using silhouettes, sharp features, and a limited number of colors. It has been successfully used in cartoon animation for entertainment and illustration. Its high compression potential, modest memory requirement, and light computation make it also suitable for limited bandwidth wireless mobile devices.

Cartoon-style rendering is typically considered two-dimensional graphics. However, the advances in three-dimensional (3D) graphics have led to the development of various software and hardware tools that can model and render graphics primitives efficiently. These tools can be efficiently integrated into two-dimensional cartoon animations (see Fig. 1). As a result, acquisition and modeling technologies together with animation and rendering tools are replacing traditional cell-based drawing and animation. Nevertheless, current graphics hardware is optimized for standard tasks of Gouraud shading and z-buffer rendering, which does not fit the requirements of cartoon-style rendering.

The features of cartoon-style rendering, such as silhouettes, color boundaries, and sharp edges, are inherently view dependent and hence need to be computed per frame. Moreover, these special features should appear clear and smooth with minimum visual artifacts.

Our approach utilizes level-of-detail multiresolution hierarchy and feature-dependent rendering to select appropriate resolution around interesting features and low resolution elsewhere. Appropriate resolutions need to be high enough to capture the model features without overloading the CPU or exceeding the hardware rendering capability. To compensate for insufficient resolution, the silhouette and color boundaries are refined within the graphics processing unit (GPU) by programmable shaders.

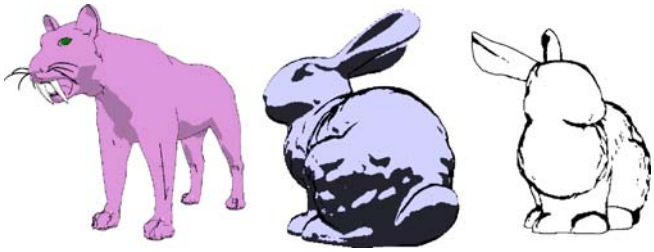

Fig. 1. Cartoon-style images

Our algorithm also utilizes temporal coherence among consecutive frames to efficiently compute levels of detail, silhouettes, and color transition boundaries. In typical level-of-detail hierarchies, the selected level reaches its maximal resolution at the leaves of the hierarchy. In contrast, our scheme allows finer resolution and color curves by using a GPU-based refinement.

In the rest of this paper, we present related work in non-photorealistic and level-of-detail rendering. Then we discuss our approach to accelerate cartoon-style rendering, followed by implementation details and experimental results. Finally, we draw some conclusions and suggest directions for future work.

2 Related work

Our approach relies on previous results in cartoon-style and level-of-detail rendering. Therefore, we will discuss closely related work in these two fields separately.

2.1 Cartoon-style rendering

Several approaches have been developed to render silhouettes of polygonal models by utilizing graphics hardware [8, 16, 22]. However, explicitly identifying such edges is a cumbersome process and usually not supported by hardware [20]. Saito and Takahashi [23] utilized image space algorithms and modified the depth over several rendering stages. Raskar and Cohen [21] also used a similar approach while scaling the polygons instead of modifying the depth. Nienhaus and Doellner [17] proposed a multipass rendering algorithm based on the edge map, a texture that encodes visually important edges of the scene's objects. On the one hand, image space algorithms are not sensitive to polygons' connectivity. On the other hand, generating images via multiple rendering stages usually slows down the overall rendering process.

Determining silhouette edges in real time involves visiting all the edges of the mesh, which implies $O(n)$ time complexity. Buchanan and Sousa [2], and Lake et al. [14] suggested simplifying the input mesh to reduce the complexity of silhouette finding. Markosian et al. [16] implemented a stochastic search for silhouette edges. Gooch et al. [8] developed a Gauss map method to find silhouette edges in an $O(k \log n)$ time for a model of n edges and k silhouette edges. Johnson and Cohen [12] and Sander et al. [24] used hierarchical trees of cones to quickly find silhouette edges. Hertzmann and Zorin [9] extended [16] by using four-dimensional (4D) homogeneous coordinates which are projected into eight 3D cubes.

Lake et al. [14] calculated one-dimensional texture coordinates on-the-fly to accelerate real-time cartoon-style rendering. Claes et al. [3] developed a cartoon-style rendering that uses two or three colors for the geometry and explicit outline. They demonstrated smooth color transition borders and silhouette lines. Raskar [20] developed an approach that adds new polygons to the mesh with the appropriate color, shape, and orientation. Hardware is then utilized to generate silhouettes and sharp features. Barla et al. [1] suggested an extended cartoon shader for the modern GPU. Several approaches [19, 26] used the GPU for interactive non-photorealistic rendering (NPR) rendering with hatching strokes. They have managed to avoid aliasing during real-time rendering.

2.2 Level-of-detail rendering

Multiresolution hierarchies have been used to allow various levels of detail to smoothly co-exist over different regions of the same surface. Such schemes usually support view-dependent rendering and take advantage of temporal coherence to adaptively refine or simplify the polygonal environment from one frame to the next.

Hoppe [10] introduced a progressive meshes scheme that provides a continuous resolution representation of polygonal meshes. Merge trees were introduced by Xia et al. [27] as a data structure built upon progressive meshes to enable real-time view-dependent rendering of an object. Hoppe [11] developed a view-dependent rendering algorithm that works with progressive meshes. Luebke and Erikson [15] defined a tight octree over the vertices of the given model to generate hierarchical view-dependent simplifications. De Floriani et al. [4] introduced multitriangulation (MT). Decimation and refinement in MT are achieved through a set of local operators that affect fragments of the mesh. Kim and Lee [13] managed to remove the dependency limitation of the split and merge operation. In their refinement scheme, each vertex split or edge collapse can be performed without incurring extra vertex split or edge collapse transformations. El-Sana et al. [5] developed Skip Strip: a data structure that efficiently maintains triangle strips during view-dependent rendering. El-Sana and Chiang [6] developed an external memory view-dependent simplification. Shamir et al. [25] developed a view-dependent approach that handles dynamic environments with arbitrary internal deformation. Pajarola [18] developed an efficient multiresolution hierarchy for view-dependent rendering which is based on the half-edge collapse operation.

3 Our approach

We present a novel approach to accelerate cartoon-style rendering of polygonal datasets while providing quality images. In contrast to previous approaches that usually process the entire input model at each frame, our approach uses level-of-detail rendering to reduce the geometric complexity of an input model. It also takes advantage of graphics hardware programmability to emphasize cartoon features, such as silhouettes, sharp edges, and color transition boundaries.

Our approach quantizes light intensity to generate different shadow colors, which are used to guide the feature-dependent level-of-detail representation. The level-of-detail representation is then sent to the graphics hardware, which detects silhouettes and raises the resolution of its curves (using a GPU shader) beyond that of the original mesh. The next sections describe each aspect of this technique.

3.1 Cartoon colors

Typical cartoon-style drawing usually includes a small number of colors. For that reason, our algorithm generates a set of predetermined shaded colors for each color in the input model. These colors are used to paint the model in real time. We will refer to these colors as the *cartoon colors*. We usually choose four cartoon colors for each color in the model: an illuminated and a shaded version of the color, white color for specular light, and a silhouette color which is usually black.

To reduce aliasing artifacts, such as staircase and straight edges in silhouette and color boundaries, several approaches use CPU-based subdivisions to refine the model geometry. In contrast, our algorithm refines the geometry and color boundaries using a programmable GPU. The GPU-based geometry refinement is required for those triangles that occupy large screen space and thus add long edges to the silhouette. These edges usually produce disturbing aliasing artifacts. The GPU-based color refinement is required for those triangles that include two or three different colors. Such a procedure replaces the CPU-based mesh subdivision to achieve higher triangle resolution.

3.2 Adaptive level-of-detail

Typical cartoon images consist of uniform ambient color regions and cartoon features, such as silhouettes, color boundaries, and sharp edges. The ambient colors occupy most of the final image, while the cartoon features cover only a small fraction. Similar ratios between uniform ambient colors and cartoon features are observed also in the 3D meshes used to generate cartoon images. We would like to represent regions that include special cartoon features in fine resolution and use a coarse level of detail to represent uniform ambient color regions. However, current multiresolution level-of-detail hierarchies have several limitations that complicate their use for such applications. First, the selection of the adaptive levels is performed in the CPU, which often fails to extract or update large levels that match the rendering capability of current graphics hardware within the duration of one frame. Second, the selected levels or even the original model may not include the appropriate resolution to enable the generation of smooth silhouette or color boundary curves. Third, multiresolutions that maintain dependences to avoid foldover, such as merge trees [27], progressive meshes [10], and view-dependent trees [7], usually use more than the required edges to achieve smooth transitions between different levels of detail over adjacent regions. We have found that this property limits the overall reduction of the number of triangles required to represent the appropriate level of detail.

To overcome the first limitation, an appropriate triangle budget is maintained carefully within the CPU and the update of active meshes is amortized over several frames, if necessary (the number of these frames is usually very small). The triangle budget, which is determined based on the capabilities of the CPU and graphics hardware, is updated in real time to reflect CPU load. Moreover, as a result of moving the refinement phase to the GPU, the generated representations are very small and the CPU is capable of handling them within the duration of one frame. The second limitation is tackled by drawing silhouette and color boundaries within the programmable GPU. We resolve the third limitation by using the *truly selective refinement of progressive meshes* scheme, by Kim and Lee [13], as the base multiresolution hierarchy. This multiresolution approach can perform valid transitions between adjacent different level-of-detail regions in a small number of edges.

The multiresolution hierarchy is constructed off-line in a preprocessing stage in a bottom-up fashion by recursively applying edge collapse. At runtime the constructed hierarchy is used to guide the selection of appropriate levels of detail in each frame with respect to illumination and view parameters while taking advantage of temporal coherence among consecutive frames. The selection of appropriate levels of detail at real time is performed by simplifying and refining different regions of the current mesh representation.

The collapse of an edge is performed by merging its two vertices and removing degenerate triangles. The collapse's dual, the vertex split operation, is achieved by adding the removed edge and reinserting its adjacent triangles.

3.3 Real-time rendering

At runtime the algorithm traverses and updates the current level of detail to match the new illumination and view parameters. In the context of cartoon-style rendering, we

are interested in maximizing the resolution around color boundaries, silhouettes, and sharp features; and minimizing it at regions painted with solid colors and back facing, without exceeding the triangle budget.

The active levels of detail are traversed in a priority manner, which means that close-to-viewer, front-facing, and high-detailed regions are traversed earlier than far-from-viewer, back-facing, and low-detailed regions. We utilize per frame traversal of active nodes to update the color of active vertices and determine silhouettes and color borders.

In the context of triangle mesh, silhouette edges usually connect back-facing and front-facing triangles. However, since we emulate smooth shading and determine the normal per vertex, silhouette and color boundary curves often pass across triangles and not along edges. Testing whether a curve passes through a triangle or not is performed by computing the dot product between the light-vector/view-director and the normals of the vertices of each triangle. Silhouette and color boundary curves can divide a triangle into two or three regions. A triangle can include no curve, one curve, or an intersection/merge of two curves as shown in Fig. 2a–c, respectively. To reduce the CPU load and to increase the visual precision, we exploit the programmable graphics hardware to refine the geometry and assign cartoon color.

3.4 Graphics hardware refinement

Current graphics hardware provides common functionality for both vertex and fragment processors which is useful to generate various effects, such as multitexturing, displacement mapping, and shading. Programmable vertex and fragment processors provide the ability to access the graphics pipeline and alter vertex or fragment properties, respectively.

We utilize the graphics hardware to increase the performance of our algorithm, as well as the precision of the rendered images. The graphics hardware is used for silhouette detection, which is performed within the vertex processors; and cartoon coloring and shading, which is performed by the fragment processors.

The subdivision of a triangle is determined at the GPU stage based on the normals of its vertices. The normal N_i at an internal point P_i on the edge E_{ab} is determined by a linear interpolation of the normals N_a and N_b at its two vertices V_a and V_b respectively, as shown in Eq. 1. The silhouette curve passes at the point where N_i is perpendicular to the view direction, as shown in Eq. 2.

$$N_i = tN_a + (1-t)N_b, \tag{1}$$

$$\begin{aligned} N_i \cdot V_d &= 0 \\ &\Rightarrow (tN_a + (1-t)N_b) \cdot V_d = 0 \\ &\Rightarrow t = \frac{N_a \cdot V_d}{N_b \cdot V_d - N_a \cdot V_d}. \end{aligned} \tag{2}$$

For each transmitted triangle, the algorithm tests the angle δ between the normals of its vertices. Whenever δ exceeds a predetermined threshold, the graphics pipeline replaces the triangle with a predefined planar triangular patch. Such a procedure increases the geometric resolution (vertices per unit) around silhouette curves and color boundaries. Since we assume that the geometric model was sampled from a smooth surface, the vertex shader elevates every vertex in the triangular patch. The elevation directions and values are determined by interpolating the normals at the vertices of the original mesh (see Fig. 3).

To carry the normals to the color-fill phase, the vertex shader encodes the normals of the vertices into color components and utilizes the graphics hardware rasterization to perform triangle coloring. The rasterization divides each rendered triangle into pixel-size fragments, each with its own properties.

The fragment processor recovers the interpolated normal of a fragment from its color. It then determines silhouette curves based on the fragment normal f_n and view direction v_d (please refer to Fig. 4). Practically, the fragment processor only marks, independently, silhouette fragments. The color boundaries are computed in a similar manner using the fragment normal, viewpoint, and light direction.

In cartoon-style rendering, silhouette curves are emphasized by a lucid color which is often black, as shown in Fig. 5. The thickness of these curves is determined by the fragment shader and based on the value fv_{dot}, which is computed by the dot product between the fragment normal and the view direction. In an ideal case, only fragments with the value zero will be on the silhouette curve. However, dealing with triangle meshes requires considering

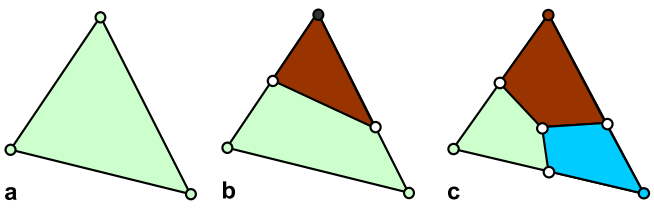

Fig. 2a–c. Cartoon curves can divide a triangle into one, two, and three regions

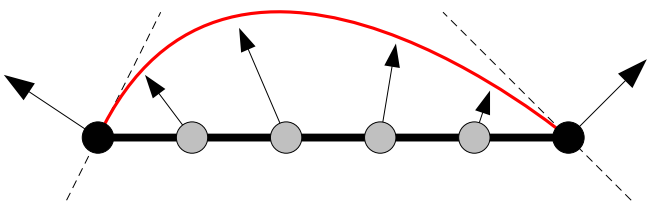

Fig. 3. GPU-based geometry smoothing. The elevation directions and values determined by the bounding normals

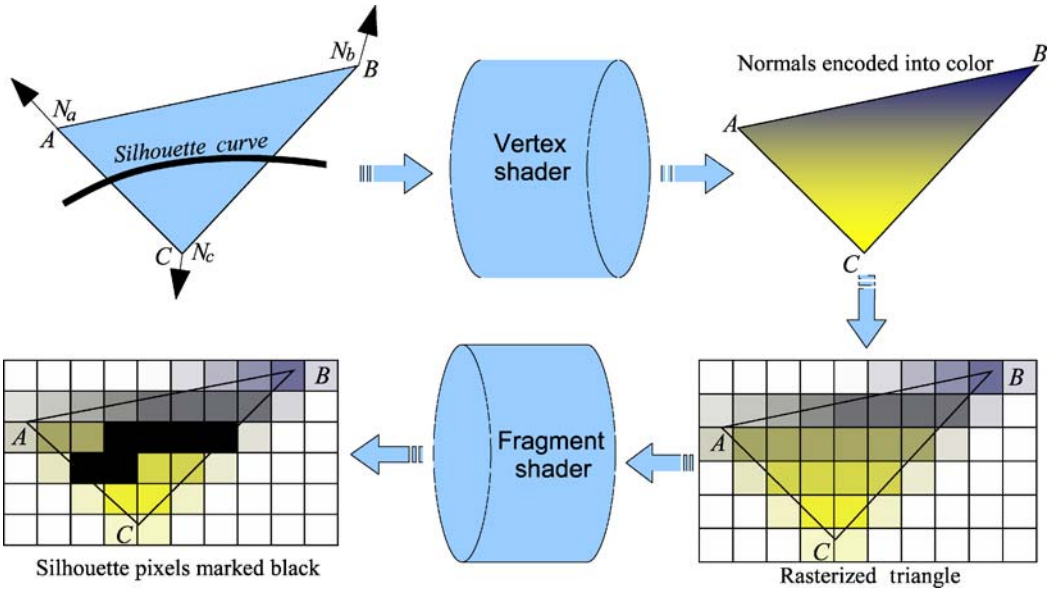

Fig. 4. GPU-based silhouette refinement

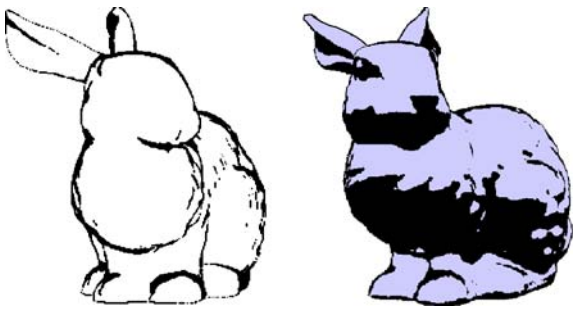

Fig. 5. The *left-* and *right-hand images* show the GPU-computed silhouette and two-level shading, respectively

fragments with values within the range $[-\epsilon, \ldots, \epsilon]$ as silhouette fragments. In our current implementation, we use a predefined thickness for all silhouette curves, which is derived from the resolution of the final image.

Exploiting the graphics hardware and the programmable GPU allows several colors to pass through the same triangle. Furthermore, this scheme increases the color curve precision to fit the screen resolution without any extra CPU load.

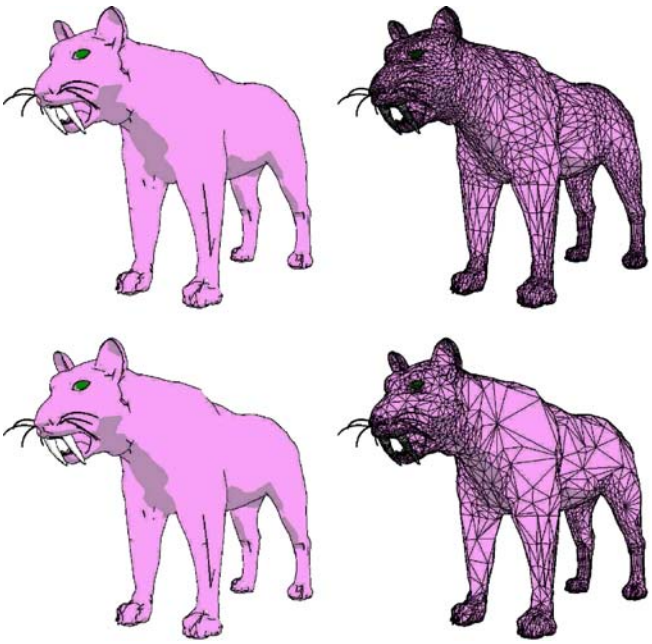

Fig. 6. The *left-hand images* show different levels of detail of a panther model and the *right-hand images* show the same model in a wire-frame representation

4 Implementation and results

We implemented our approach in C++ using the standard OpenGL API and Cg for Microsoft Windows. To support geometry and color refinement within the GPU, our implementation requires graphics hardware that supports *nVidia Shader model 2.0*. To enable efficient geometric refinement, the algorithm caches an instance of the planar patch within the GPU memory. The geometry of this triangular patch is encoded in an indexed triangle strip format and cached by using the vertex buffer object (VBO) extension.

We tested our implementation on various datasets and received encouraging results. Figure 6 shows differ-

ent levels of detail of the panther model in cartoon and wire-frame representations. The first row shows the input mesh; the second row shows the generated image received in our algorithm after using feature-dependent level-of-detail selection for cartoon-style rendering. Figure 7 shows a cartoon-style rendering of a large model – the Asian dragon – using our system. It is clear that we succeeded in reducing the number of polygons without noticeable differences.

The accuracy of the color boundaries with and without using GPU-based refinement is demonstrated by the images in Fig. 8. The left- and right-hand images show a sphere rendered in a cartoon style with and without GPU refinement, respectively. The small images above and below the spheres show the differences between the two images in more detail. Clearly, the GPU-based refinement allows silhouette curves and color boundaries to pass through triangles, and not only along edges. As a result, the generated images have smooth curves and color boundaries. Figure 9 shows the Cg code that runs in each fragment processor to compute the color of an input fragment and provides one-pixel color precision.

Fig. 7. A cartoon-style rendering of the Asian dragon model

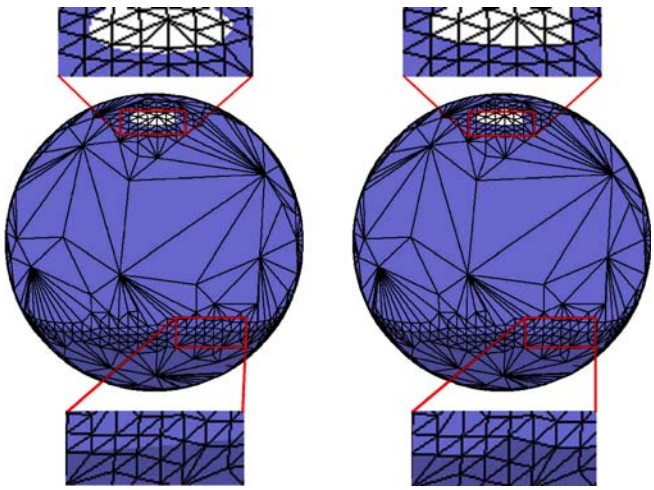

Fig. 8. Spheres (on *left*) with and (on *right*) without GPU-based refinement

```
void main( in s_FragmentIn f_in,
           uniform float3 lookPos,
           uniform float3 lightPos,
           uniform float silhouLimit,
           uniform float specLimit,
           out float4 oColor : COLOR )
{
  // Decode normal from color components
    float3 f_Normal = DECODE(f_in.iColor.xyz);

  // Detect silhouette fragments
    float3 lookDir = normalize(lookPos - f_in.iPos);
    float lookDot = dot(f_Normal , lookDir);
    float isSilhouette = step(silhouLimit , lookDot);

  // Detect darkened fragments
    float3 lightDir = normalize(lightPos - f_in.iPos);
    float lightDot = dot(f_Normal , lightDir);
    float isLighted = ceil(lightDot);

  // Detect specular fragments
    float t = SUM(lightDir) / SUM(f_Normal);
    float3 crossPoint = f_Normal * t;
    float3 diff = crossPoint - lightDir;
    float3 returnDir = normalize(crossPoint + diff);

    float specFactor = dot(lookDir , returnDir);
    float isSpec = step(specLimit , specFactor);

  // Compute final colors
    float3 light = lightColor * isLighted +
                   darkColor * (1 - isLighted);
    oColor = (1 - isSilhouette) *
             light * modelColor +
             (isSpec * specColor);
}
```

Fig. 9. A Cg code which is part of the fragment shader that determines and assigns the cartoon-style colors

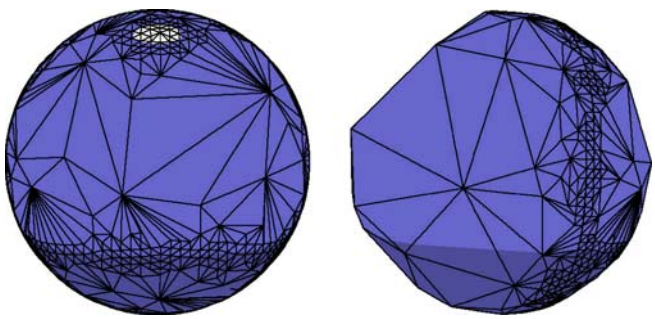

Fig. 10. The *left-* and *right-hand images* show the sphere model from two viewpoints. The fine levels of detail are selected around the silhouette and *shade curves*, as they are defined from the left-hand viewpoint

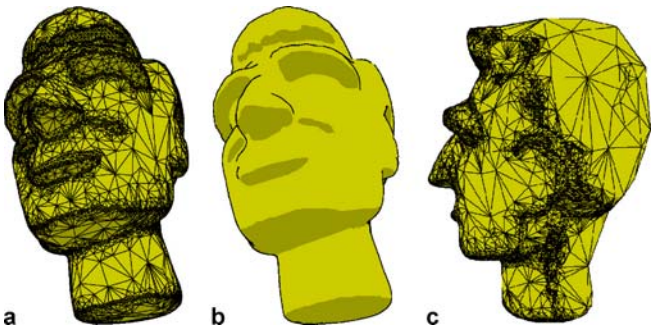

Fig. 11a–c. The head model: **a** a selected level of detail in wire frame, **b** the same model in a cartoon-style rendering, and **c** the selected level of detail from a different view

Figure 10 demonstrates the fine level of detail around silhouettes. The left-hand image shows a selected view with an appropriate level of detail that highlights the special features. The right-hand image shows the same model, with the same level of detail, but from a different viewpoint. Figure 11 shows the head model from a viewpoint that determines the level of detail and from a different viewpoint that shows silhouette edges.

Table 1 reports the time performance of our algorithm using several models of different complexities. These results were achieved on a Pentium IV 2.4 GHz machine with 1 GB and an nVidia GeForce 6800. The *PreProc* column includes the preprocessing time for each model and the *Frame* column shows the average time per frame. The time of a frame includes the adaptive refinement which is performed in the CPU to generate a feature-dependent level-of-detail representation and the GPU refinement. To generate the same images without adaptive levels of detail, one needs to send the original meshes that may exceed the rendering capability of the graphics hardware. Replacing GPU refinement with more geometry requires the rendering of dense meshes that often exceed the rendering capability, even for modest datasets.

Our approach reduces the time to determine silhouettes and color boundaries, as a result of utilizing temporal coherence and traversing only triangles on the active levels of detail. Since an active level usually includes a small fraction of the entire model, our approach manages to improve silhouette computing by the ratio of the reduction in the polygon count. In addition, silhouettes and curves are updated from one frame to the next, and not computed from scratch at each frame.

We compare two major features of our approach to other approaches: real-time performance and image quality. The real-time performance is measured by the CPU load and CPU–GPU communication load. The image quality is measured by the precision of color boundaries, and the flexibility of the coloring scheme.

The approach suggested in [3] uses the CPU to divide triangles that include two or three cartoon colors. This division increases the resolution of color boundaries beyond that of the original model. However, such a scheme increases the number of triangles, and possibly overloads the CPU and the communication to the graphics hardware. In addition, this method supports only limited coloring precision, since it uses straight edges for subdivision. In contrast, our approach uses the GPU to divide the triangles into fragments, and to assign colors. It also reduces the CPU and communication loads. Moreover, GPU-based subdivision increases the image quality, as a result of assigning different colors for fragments and not limiting the color boundaries to straight lines.

Table 1. Time performance

Model	Verts	Faces	PreProc (s)	Frame (ms)
Sphere	4 K	8 K	8	0.8
Head	17 K	34 K	10	1.7
Bunny	30 K	70 K	20	2.2
Dragon	437 K	871 K	140	3.4
Buddha	543 K	1087 K	160	4.1
A. dragon	3609 K	7218 K	160	27.1

The approach suggested in [17] uses a multipass rendering algorithm, and encodes silhouette edges into texture, to enable cartoon-style rendering. That approach requires three passes to render a single frame and demands massive communication, not only from CPU to GPU, but also from GPU to CPU. In contrast, our approach uses only a single pass, without expensive reverse communication.

5 Conclusion and future work

In this paper, we presented a novel approach that utilizes multiresolution hierarchies for real-time cartoon-style rendering. Our approach produces images with quality, comparable to much higher resolutions (of the same models). By selecting the appropriate level of detail, we reduce the rendering and processing time for all features that one would like to emphasize in cartoon-style rendering. The silhouette and color boundary curves are refined by the GPU to reach smooth curves and improve image quality.

We see the scope of future work in extending the scheme to support cartoon environments with animations. Such a development would enable the use of view-dependent schemes to animate large models, and to provide interactive cartoon-style rendering.

Acknowledgement This work is supported by the Lynn and William Frankel Center for Computer Sciences and the Israel Ministry of Science and Technology. In addition, we would like to thank the reviewers for their constructive comments.

References

1. Barla, P., Thollot, J., Markosian, L.: X-toon: an extended toon shader. In: Proceedings of the NPAR '06, pp. 127–132. ACM Press, Annecy (2006)
2. Buchanan, J., Sousa, M.: The edge buffer: a data structure for easy silhouette rendering. In: Proceedings of the NPAR '00, pp. 39–42. ACM Press, Annecy (2000)
3. Claes, J., Di Fiore, F., Vansichem, G., Van Reeth, F.: Fast 3D cartoon rendering with improved quality by exploiting graphics hardware. In: Proceedings of the Image and Vision Computing '01, pp. 13–18. University of Otago, Dunedin (2001)
4. De Floriani, L., Magillo, P., Puppo, E.: Efficient implementation of multi-triangulation. In: Proceedings of the IEEE Visualization '98, pp. 43–50. IEEE Computer Society Press, Research Triangle Park, NC (1998)
5. El-Sana, J., Azanli, E., Varshney, A.: Skip strips: maintaining triangle strips for view-dependent rendering. In: Proceedings of IEEE Visualization '99, pp. 131–138. IEEE Computer Society Press, San Francisco, CA (1999)
6. El-Sana, J., Chiang, Y.: External memory view-dependent simplification. Comput. Graph. Forum **19**(3), 139–150 (2000)
7. El-Sana, J., Varshney, A.: Generalized view-dependent simplification. Comput. Graph. Forum **18**(3), 83–94 (1999)
8. Gooch, B., Sloan, P., Gooch, A., Shirley, P., Riesenfeld, R.: Interactive technical illustration. In: ACM Symposium on Interactive 3D Graphics, pp. 31–38. ACM Press, Atlanta, GA (1999)
9. Hertzmann, A., Zorin, D.: Illustrating smooth surfaces. In: Proceedings of the ACM SIGGRAPH '00, pp. 517–526. ACM Press, New Orleans, LA (2000)
10. Hoppe, H.: Progressive meshes. In: Proceedings of the ACM SIGGRAPH '96 (Annual Conference Series), vol. 30, pp. 99–108 (1996)
11. Hoppe, H.: View-dependent refinement of progressive meshes. In: Proceedings of the ACM SIGGRAPH '97 (Annual Conference Series), vol. 31, pp. 189–198 (1997)
12. Johnson, D., Cohen, E.: Spatialized normal cone hierarchies. In: ACM Symposium on Interactive 3D Graphics, pp. 129–134. ACM Press, Research Triangle Park, NC (2001)
13. Kim, J., Lee, S.: Truly selective refinement of progressive meshes. In: Proceedings of the Graphics Interface '01, pp. 101–110. Canadian Information Processing Society, Ottawa, Ontario (2001)
14. Lake, A., Marshall, C., Harris, M., Blackstein, M.: Stylized rendering techniques for scalable real-time 3D animation. In: Proceedings of the NPAR '00, pp. 13–20. ACM Press, Annecy (2000)
15. Luebke, D., Erikson, C.: View-dependent simplification of arbitrary polygonal environments. Comput. Graph. Forum **31**(Annu. Conf. Ser.), 199–208 (1997)
16. Markosian, L., Kowalski, M., Trychin, S., Bourdev, L., Goldstein, D., Hughes, J.: Real-time nonphotorealistic rendering. In: Proceedings of the ACM SIGGRAPH '97, pp. 415–420. ACM Press, Los Angeles, CA (1997)
17. Nienhaus, M., Doellner, J.: Edge-enhancement – an algorithm for real-time non-photorealistic rendering. J. WSCG **11**(1), 346–353 (2003)
18. Pajarola, R.: FastMesh: efficient view-dependent meshing. In: Proceedings of the Pacific Conference on Computer Graphics and Applications '01, pp. 22–30. IEEE Computer Society Press, Tokyo (2001)
19. Praun, E., Hoppe, H., Webb, M., Finkelstein, A.: Real-time hatching. In: Proceedings of the ACM SIGGRAPH '01, pp. 579–584. ACM Press, Los Angeles, CA (2001)
20. Raskar, R.: Hardware support for non-photorealistic rendering. In: Proceedings of the Eurographics Workshop on Graphics Hardware, pp. 41–46. ACM Press, Los Angeles, CA (2001)
21. Raskar, R., Cohen, M.: Image precision silhouette edges. In: ACM Symposium on Interactive 3D Graphics, pp. 135–140. ACM Press, Atlanta, GA (1999)
22. Rossignac, J., Van Emmerik, M.: Hidden contours on a framebuffer. In: Proceedings of the Eurographics Workshop on Graphics Hardware, pp. 188–204. ACM Press, Annecy (1992)
23. Saito, T., Takahashi, T.: Comprehensible rendering of 3d shapes. In: Proceedings of the ACM SIGGRAPH '90, pp. 197–206. ACM Press, Dallas, TX (1990)
24. Sander, P., Gu, X., Gortler, S.J., Hoppe, H., Snyder, J.: Silhouette clipping. In: Proceedings of the ACM SIGGRAPH '00, pp. 327–334. ACM Press, New Orleans, LA (2000)
25. Shamir, A., Pascucci, V., Bajaj, C.: Multi-resolution dynamic meshes with arbitrary deformation. In: Proceedings of the IEEE Visualization '00, pp. 423–430. IEEE Computer Society Press, Salt Lake City, UT (2000)
26. Webb, M., Praun, E., Finkelstein, A., Hoppe, H.: Fine tone control in hardware hatching. In: Proceedings of the NPAR '02, pp. 53–58. ACM Press, Annecy (2002)
27. Xia, J., El-Sana, J., Varshney, A.: Adaptive real-time level-of-detail-based rendering for polygonal models. IEEE Trans. Vis. Comput. Graph. **3**(2), 171–183 (1997)

YOTAM LIVNY is a Ph.D. student of computer science at the Ben-Gurion University of the Negev, Israel. His research interests include interactive rendering of large 3D graphic models using multiresolution hierarchies and programmable graphics hardware. Livny received a B.Sc. degree in mathematics and computer science from Ben-Gurion University of the Negev, Israel in 2003. He is currently studying for a Ph.D. degree in computer science under the supervision of Dr. Jihad El-Sana.

MICHAEL PRESS received a M.Sc. degree in computer science from Ben-Gurion University of the Negev, Israel in 2006, under the supervision of Dr. Jihad El-Sana. His research interests include interactive rendering of large 3D datasets and cartoon-style rendering. Press received a B.A. degree in computer science from the Technion, Israel in 2001.

JIHAD EL-SANA is a senior lecturer in computer science at Ben-Gurion University of the Negev, Israel. El-Sana's research interests include 3D interactive graphics, multiresolution hierarchies, geometric modeling, computational geometry, virtual environments, and distributed and scientific visualization. His research focuses on polygonal simplification, occlusion culling, accelerating rendering, remote/distributed visualization, and exploring the applications of virtual reality in engineering, science, and medicine. El-Sana received B.Sc. and M.Sc. degrees in computer science from Ben-Gurion University of the Negev, Israel in 1991 and 1993. He received a Ph.D. degree in computer science from the State University of New York at Stony Brook in 1999. El-Sana has published over 40 papers in international journals and conferences.