COMPUTER GRAPHICS TECHNICAL REPORTS

CG-2003/2

Fat Borders: Gap Filling For Efficient View-dependent LOD Rendering

Á. Balázs
Universität Bonn. edhellon@cs.uni-bonn.de

M. Guthe
Universität Bonn. guthe@cs.uni-bonn.de

R. Klein
Universität Bonn. rk@cs.uni-bonn.de

Institut für Informatik II
Universität Bonn
D-53117 Bonn, Germany

ISSN 1610-8892

Fat Borders: Gap Filling for Efficient View-dependent LOD Rendering

Ákos Balázs* Michael Guthe† Reinhard Klein‡

University of Bonn
Institute of Computer Science II, Computer Graphics
Römerstraße 164, 53117 Bonn, Germany

Abstract

Real-time high quality rendering of complex models remains a big challenge. Simply splitting the models into several parts which can be simplified and rendered independently introduces disturbing gaps along the common borders. Recent approaches for view-dependent rendering of huge models either neglect the artifacts introduced by the gaps or try to maintain the connectivity of the models. Unfortunately, in the second case the computational complexity and storage requirements of the algorithms are tremendous.

In this work we present a novel solution to the gap problem. Vertex programs are used to generate appropriately shaded fat lines that fill the gaps between neighbouring patches. Based on our method the development of new algorithms for out-of-core simplification and view-dependent rendering of complex triangle meshes is straightforward. Due to its generality, simplicity and effectiveness our method has the potential of becoming part of future graphics APIs.

To demonstrate the capabilities of our method we apply our new algorithm to the rendering of complex trimmed NURBS models, where the gap problem becomes especially apparent as these models consist already of thousands of independent trimmed patches. Combining the new gap filling algorithm with an on-the-fly tessellation of the individual trimmed NURBS patches we are able to visualize deformable trimmed NURBS models with dynamic neighbourhood in real-time.

CR Categories: I.3.3 [COMPUTER GRAPHICS]: Picture/Image Generation—Display algorithms; I.3.7 [COMPUTER GRAPHICS]: Three-Dimensional Graphics and Realism—Animation

Keywords: Level of detail, view-dependent rendering, gap closing, NURBS rendering, NURBS animation, vertex program, fragment program

1 Introduction

The recent developments in 3D acquisition systems and computer-aided design technologies steadily increase the size of geometric models. The enormous size of these models calls for sophisticated rendering techniques that support view-dependent level of detail (LOD), culling techniques and efficient memory management. There are essentially two types of algorithms dealing with this problem:

*e-mail: edhellon@cs.uni-bonn.de
†e-mail: guthe@cs.uni-bonn.de
‡e-mail: rk@cs.uni-bonn.de

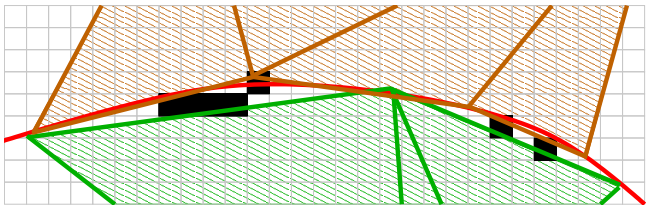

Figure 1: The lightgray grid denotes the pixels of the display hardware, the red curve denotes the original boundary curve, the tessellations of the neighbouring patches are denoted by green and brown triangles respectively, and the pixels that remain in background are denoted by black.

1. One approach to deal with this problem is to subdivide the models hierarchically into subparts which are simplified and rendered independently. This results in simple yet efficient algorithms. Unfortunately, subdividing models and processing the subparts individually introduces disturbing artifacts due to gaps along the cuts during view-dependent rendering, even if the gaps are less than half a pixel wide. The reason for this is, that the graphics hardware samples the geometry in the midpoint of the discrete pixels, see Figure 1.

2. Other approaches employ highly complex and memory intensive data-structures to manage connected triangle meshes over all LODs. Due to the elaborate dependencies of data in these hierarchical data-structures the design and implementation of really fast algorithm remains a challenging task. Furthermore, to handle the tremendous memory requirements special out-of-core techniques have to be devised.

In this paper we introduce a method that abandons the idea of maintaining the connectivity of triangle meshes over the LOD hierarchy. Instead we suggest to eliminate the gaps in a perceptually driven manner by drawing appropriately shaded fat lines along the borders of subparts using vertex programs running in state of the art graphics cards. The only requirement we impose is the control of the error along the boundaries of the subparts. We neither assume knowledge about the representation of the subparts nor about their adjacency. This gap filling algorithm allows us to devise a simple but efficient view-dependent level of detail rendering algorithm since we do not have to take care of shared triangulations of the subpart borders. Therefore, our method opens new possibilities for the development of further out-of-core simplification and view-dependent rendering algorithms for complex triangle meshes and complex models composed of higher degree surfaces. We believe that due to its generality, simplicity and effectiveness our method has the potential of becoming part of future graphics APIs.

To demonstrate the capabilities of our new method we apply it to the rendering complex trimmed NURBS models, where the gap problem becomes especially apparent as these models consist already of thousands of independent trimmed patches. Furthermore, real-time high quality rendering of complex trimmed NURBS models is of great interest for industry, since this is the most widely used format for geometric modeling in this field. Recent approaches suggest to overcome the gap problem in this case by geometrically sewing together neighbouring patches either at runtime or in

a preprocessing step based on static neighbourhood information. Like the second type of algorithms used for view-dependent rendering, their computational complexity and storage requirements are tremendous. In contrast to this, we are able to visualize deformable trimmed NURBS models with changing neighbourhoods in real-time combining our new gap filling algorithm with an on-the-fly tesselation of the individual trimmed NURBS patches.

2 Related Work

2.1 View-dependent rendering of triangle meshes

View-dependent mesh simplification and rendering methods have drawn the attention of researchers for several years now. Different simplification strategies are used to generate level-of-detail hierarchies supporting view-dependent rendering. The approaches presented in [El-Sana and Varshney 1999], [Xia et al. 1997], [Xia and Varshney 1996], [Hoppe 1997],[Hoppe 1998], [Pajarola 2001], are based on a binary vertex hierarchy derived from iteratively simplifying the input mesh by edge collapse operations [Hoppe 1996]. The main difference between these approaches are the different data structures used to represent the hierarchy. De Floriani et. al. [Floriani et al. 1998] describe a method that is based on simple vertex insertion and removal operations which can easily be extended to edge collapse hierarchies as well. Generalized view-dependent rendering frameworks based on arbitrary vertex hierarchies are presented in [Luebke and Erikson 1997] and [Schmalstieg and Schaufler 1997]. While not optimized for storage cost or rendering performance these generic approaches support a wide range of simplification operations.

El Sana and Chiang [El-Sana and Chiang 2000] presented the first method for out-of-core triangle mesh simplification and view-dependent rendering with different LODs from external memory. They introduce a new spanned sub-meshes simplification technique to subdivide the original mesh into smaller parts which can be loaded into memory at runtime while preserving the correct edge collapsing order which is necessary to guarantee the run-time image quality. Special meta-node trees facilitate the run-time level-of-detail rendering. De Coro and Pajarola [DeCoro and Pajarola 2002] present an improved approach with respect to memory requirements by partitioning the in-memory data structure used in [Pajarola 2001] into detail blocks that can be efficiently stored and loaded from external memory.

All these methods for view-dependent rendering of large triangle meshes assure that the mesh remains connected during the view-dependent refinement process, however, at the cost of tremendous additional storage requirements, complex data structures and elaborate algorithms. In the context of digital elevation data the natural parametrization of the height fields over a rectangular grid can be exploited to cut the data into smaller pieces with nearly no storage overhead and to devise much simpler and more efficient out-of-core algorithms. Specialized view-dependent terrain triangulations based on height-field models are presented in [Duchaineau et al. 1997], [Klein et al. 1997], [Lindstrom et al. 1996], [Hoppe 1998] and [Pajarola et al. 2002]. In addition to the natural parametrization these approaches take advantage of the regular grid structure to process the data.

Baxter et. al. [Baxter et al. 2002] introduce a similar cutting algorithm to render gigabyte sized, arbitrary triangle meshes. Their algorithm cuts the original data into small initial grid-aligned cells that are combined hierarchically as suggested in [Erikson et al. 2001] but neglect the gaps between the static LODs introduced by the cutting. Based on this hierarchy they proposed a simple, easy to parallelize yet very efficient view-dependent rendering algorithm, however, at the expense of sacrificing high visual quality. Note, that by including our fat border technique introduced in this paper, this algorithm can easily be modified to guarantee high visual fidelity of the output at nearly no costs and additional programming efforts.

2.2 NURBS Rendering

Due to its industrial relevance, the issue of rendering trimmed NURBS surfaces has been devoted large interest from researchers. Different approaches emerged for visualization, e.g. ray-tracing the surfaces (e.g. [Nishita et al. 1990]), pixel level subdivision (e.g. [Shantz and Chang 1988]), or polygon tessellation (e.g. [Herzen and Barr 1987; Rockwood et al. 1989; Forsey and Klassen 1990]), of which the triangle based methods are generally much faster due to recent advances in graphics hardware. While these traditional approaches deal with individual curves or surfaces and usually make no or little attempt at overcoming the problems caused by individual tessellation of patches, the more recent approaches try to handle this problem. An example for the latter is the work of Kumar et. al. [Kumar et al. 1997], which introduces the notion of super-surfaces. Based on a priori known connectivity information they statically cluster sets of trimmed NURBS patches into so-called super-surfaces. An individual view-dependent triangulation is generated at run-time for each super-surface and in a final step these view-dependent triangulations are sewn together in order to avoid cracks. The computationally complex sewing part is parallelized to achieve real-time frame rates even for huge models. Another approach [Kumar et al. 2001] only deals with very specific configurations of trimmed NURBS surfaces that are stacked on top of each other.

[Guthe et al. 2002] shifts the complex sewing part to a preprocessing step and introduces the notion of the seam graph. The seam graph consists of all trimming curves contained in a model and manages all connectivity information between the individual LODs. At runtime a view-dependent LOD of the seam graph is generated in each frame and the individual patches in the model are re-tesselated according to the LOD of the seamgraph. Although the authors show high quality results and report good framerates the storage overhead needed for the seamgraph is high. This restricts the applicability of this algorithm.

These methods all have the common disadvantage of either relying on connection information to be supplied, and/or requiring significant preprocessing time. Therefore, the connectivity information between patches cannot be changed at runtime, which makes these algorithms unsuitable for animated models with dynamic neighbourhood relations.

3 The Gap Filling Algorithm

In our new gap filling algorithm the gaps between adjacent patches introduced by independent triangulations are filled by appropriately shaded fat borders. These fat borders consist of several triangles with predefined connectivity. Their orientation and width as well as their colors are view-dependent and calculated in each frame using a vertex shader program.

The input for our gap filling algorithm consist of an arbitrary number N of LOD-hierarchies $H_i = \{\hat{M}_i = M_{n_i}, \ldots, M_{0_i}\}, i = 1, \ldots, N$, of independent patches $\hat{M}_i$ with border. The only requirement on these LOD-hierarchies is, that for each patch $\hat{M}_i$ we always can choose a LOD M_{k_i} such that the distance between the approximate surface M_{k_i} and the original $\hat{M}_i$, when projected onto the screen, is everywhere less than $\frac{1}{2}$ pixel, especially along the border of the patch.

As described in [Klein et al. 1996] this can be achieved by guaranteeing that the condition $\mathscr{H}(\hat{M}_i, M_{k_i}) \leq r$ holds for the Hausdorff distance $\mathscr{H}$, where r is chosen in such a way that the screen-space projection of the ball $B(r)$ is less than half a pixel.

Note that the way in which the different LODs are represented and generated is irrelevant as long as the above conditions are satisfied. This implies that we can gather the current LOD directly by tesselating a NURBS patch guaranteeing the required error tolerance, by using a progressive mesh representation [Hoppe 1996], by loading a static level of detail of the patch from disk or even by using a geometry image [Gu et al. 2002] with appropriate resolution

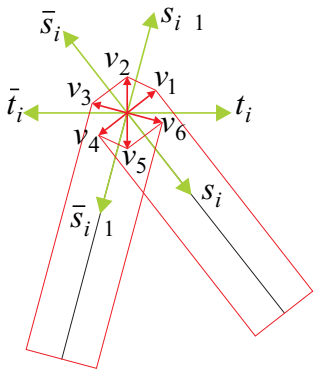

Figure 2: Concept of the vertex program. Vertices are moved to build a fat border.

to represent the LOD of the patch.

3.1 Fat Border Construction

We are given a set of polylines each representing a boundary curve. For each line segment l we generate small surfaces s_j perpendicular to the current viewing direction by extending the line segment in such a way that the projection of l_j onto image space extends the projected line segment by half a pixel in each direction, see the left drawing in Figure 3. In order to shade the newly introduced triangles exactly like the adjacent surfaces we utilize the shading parameters of the original vertices of the borderline for the newly generated vertices.

This leads to the following algorithm. We iterate through the vertices of polylines, processing each polyline in the following manner:

1. calculate the normalized orientation s_{i-1} and s_i of the respective previous and the following line segment at the current vertex v_i along with their negated counterparts $\overline{s_{i-1}} = -s_{i-1}$ and $\bar{s_i} = -s_i$, respectively, see Figure 2.

2. calculate the normalized tangent $t_i = \frac{s_{i-1}+s_i}{||s_{i-1}+s_i||}$ of the poly line at the current vertex, and its negated counterpart $\bar{t_i} = -t_i$.

3. Generate six new vertices by displacing v_i perpendicular to each of the directions computed in the above steps and the viewing direction $d_i = \frac{c-v_i}{||c-v_i||}$ (see Figure 3), where c is the location of the camera:

$$v_{i1} = \varepsilon \frac{(s_i \times d_i)}{||(s_i \times d_i)||} \quad (1)$$

$$v_{i2} = \varepsilon \frac{(t_i \times d_i)}{||(t_i \times d_i)||} \quad (2)$$

$$v_{i3} = \varepsilon \frac{(s_{i-1} \times d_i)}{||(s_{i-1} \times d_i)||} \quad (3)$$

$$v_{i4} = \varepsilon \frac{(\bar{s_i} \times d_i)}{||(\bar{s_i} \times d_i)||} \quad (4)$$

$$v_{i5} = \varepsilon \frac{(\bar{t_i} \times d_i)}{||(\bar{t_i} \times d_i)||} \quad (5)$$

$$v_{i6} = \varepsilon \frac{(\overline{s_{i-1}} \times d_i)}{||(\overline{s_{i-1}} \times d_i)||} \quad (6)$$

where ε is the object space geometric error guaranteeing the half pixel error in screen space.

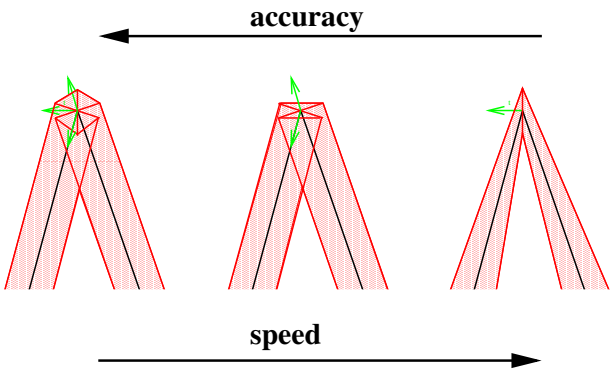

Figure 3: Different fat border generation algorithms. From left to right 6, 4, 2 new vertices. The black polyline is the boundary, and the red polygons represent the generated fat lines. Note how the fat lines become thinner from left to right.

Figure 4: Left: Aliasing artifact due to partly intersecting fat borders. Right: Pushing back the fat borders reduces the artifact.

4. Push the newly generated vertices away from the viewer along the viewing direction again by ε.

5. Generate new triangles by connecting the resulting vertices as shown in Figure 2. Note, that due to the simple structure of our fat borders we can define a triangle strip for each of it.

6. Calculate the color of each of the new vertices by assigning the shading parameters of the original border vertices. Note, that the orientation of the fat borders serve to fill the gaps, however they do not influence the actual shading which is computed based on the original normals.

As the viewing direction changes from frame to frame the position of the new vertices has to be updated for each frame. This can easily be done using a vertex program. The only prerequisite for this is to provide for each border vertex six dummy vertices and their connectivity. The vertex program should be conducted only for the border vertices. Therefore, we disable its execution while rendering the patch itself. See Figure 2 for illustration, and Figure 7 for an example.

3.2 Optimization

Note, that although six points are required to ensure the extend of the borders even at sharp corners, in practice using only 4 or even 2 new vertices delivers good results (only minor visible artifacts) and can have a huge impact on the rendering performance since only one or two thirds of fat border triangles have to be rendered. The corresponding fat border generation schemes for 6, 4 and 2 vertices are illustrated on Figure 3. If we use 4 vertices, the vertices v_2 and v_5 in equation 1 are left out, if 2 vertices are used only v_2 and v_5 are generated.

3.3 Problems

Unfortunately, the fat borders of neighboring patches might intersect each other. This does not cause any problems if their shading

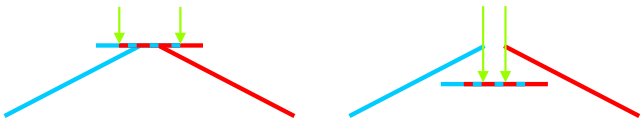

Figure 5: Sine the patches on the left and right have different colors the intersecting fat borders lead to artifacts that occur in the range of one pixel. If we push the fat borders away from the viewer they are only visible through the gap itself.

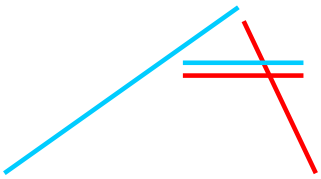

Figure 6: The spike trough artifact occurs at sharp angles of the original geometry. In this case the fat border intersects a neighbouring patch and introduces shading artifacts.

results in the same color. But if their colors are different, for example due to different materials or shading aliasing artifacts occur, see Figure 4. This artifact is greatly reduced if we push the fat lines away from the viewer. This is shown in Figure 5. Since after this push operation the fat borders are only visible through the gap and this is in general much less than a pixel the aliasing artifacts are greatly reduced.

A further problem is shown in Figure 6, where at sharp angles of the geometry a fat border intersects a neighboring patch. This artifact can not be avoided by repositioning the fat border since we do not have any information about the location of the neighboring patch. Fortunately, the size of the visible spike through is always less than half a pixel.

Both artifacts introduced by our method are are restricted to at most one pixel in width. They become apparent since the hardware samples the pixel in its midpoint introducing aliasing artifacts. Thus they can be reduced by using standard super sampling. Using the 6x6 super sampling of the ATI Radeon 9700 the artifacts cannot be perceived.

4 Application to NURBS Models

Our method would also fit nicely into the framework of [Kumar et al. 1997]: one could avoid their computationally complex sewing part and use our solution instead. One of the reasons we did not choose this path is that the problem of clustering the model into super-surfaces still remains and it makes it rather difficult to employ

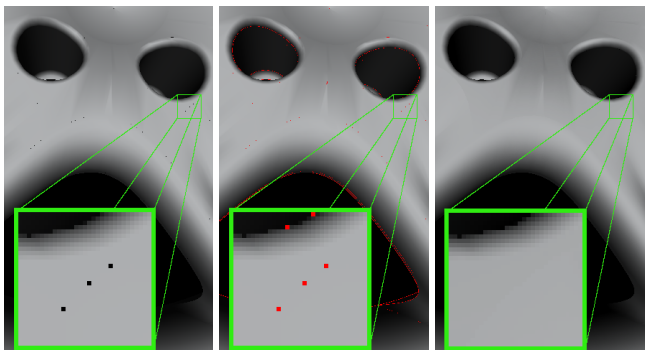

Figure 7: a) Part of the wheel rim model rendered without wide lines (note the gaps.) b) The same part rendered with the wide lines superimposed. c) Result: the wide lines cover up the gaps.

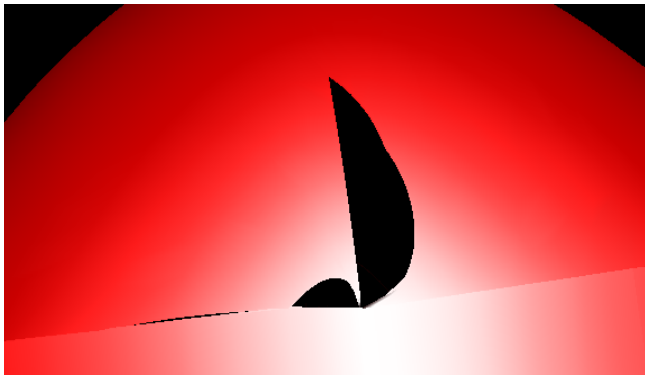

Figure 8: Classic modelling error: wrongly specified trimming curves, still very visible!

this approach to render dynamic models. Instead we tessellate on the fly the individual trimmed NURBS patches independently with a guaranteed approximation error and fill the emerging gaps with our new procedure.

An important issue in rendering trimmed NURBS models is that error caused by erroneous modeling should be visible and not be concealed by the visualization method. Our gap filling method works well considering this aspect. As shown in Figure 8 modeling errors can be easily detected, as the width of the fat borders becomes smaller compared to the modeling error in case of a close-up.

4.1 Tessellation of trimmed NURBS Patches

In our NURBS rendering system we use a tessellation process that guarantees that the final tessellation does not deviate more than a prescribed error from the original analytical surface in 3D Euclidean space as required by the gap filling algorithm. This error changes from frame to frame. Therefore, the tessellation of the individual patches has to updated accordingly on-the-fly. This implies that we need a very efficient on-the-fly tessellation for trimmed NURBS. There are several methods available for this purpose. In our implementation, we follow the approach described in [Guthe et al. 2002] which we found to be efficient enough.

4.2 Load Balancing

Since complex models typically contain at least 8000-10000 patches or more, it could mean that we have to re-tessellate thousands of patches. In order to ensure interactive frame rates we restrict the time available for re-tessellation to 40ms per frame. This implies that we possibly will not have enough time to re-tessellate every patch. To overcome this problem, we first reduce the number of patches considered by taking into account only those patches that were visible in the last frame. In addition we insert the remaining patches into a priority queue according to the following weight function:

$$w = \begin{cases} (\varepsilon_{act}/\varepsilon_c)^2, & \varepsilon_c < \varepsilon_{act} \\ \varepsilon_c/\varepsilon_{act}, & \varepsilon_c > \varepsilon_{act} \end{cases}$$

where ε_{act} is the current error, and ε_c is the desired error.

This means patches for which the current error is higher than the desired error (and thus are likely to cause visible artifacts) will have a much higher priority than those for which the error is too low and we only want to re-tessellate them to save memory and rendering time. Our experimental results show that the screen-space error converges to one pixel with only a short delay. It usually takes less than 3-4 seconds to have no noticeable visibility errors on screen after fixing the camera parameters.

Figure 9: View-dependent rendering of a car model consisting of 8036 trimmed NURBS patches using environment mapping. The model was kindly provided by Volkswagen.

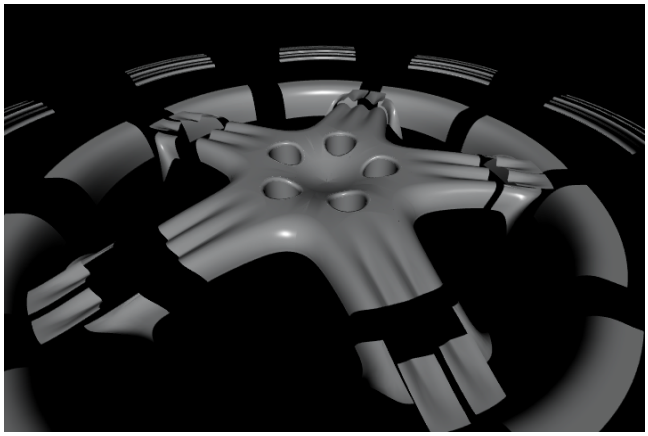

Figure 10: Snap shot of an implosion of a wheel rim. The neighbourhood information changes dynamically. Nevertheless, no cracks are visible using our fat borders.

4.3 Further Enhancements

As our approach seamlessly integrates into the normal graphics pipeline we can use just about any established technique for further enhancing the resulting image. As an example we chose to implement normal maps as a fragment program. Normal maps are used to calculate the colors on a per pixel basis instead of using normals to do interpolation of colors between vertices [Guthe and Klein 2003].

Our experiments indicate that the rendering is not GPU limited, which implies that we could use the GPU to perform more image enhancements.

5 Results

We applied our method to render static and dynamic NURBS models. In Figure 9 an example of a car consisting of 8036 trimmed NURBS patches is shown. The individual patches are tesselated independently on the fly according the view-dependent error metric, resulting in frame rates of about 10 frames per second on a Pentium 4, 1.8 GHz with 512 MB memory and an ATI Radeon 9700 Pro without any visible artifacts.

In our second example we show a snapshot from an implosion video of a wheel rim. In this example the neighborhood changes dynamically while the model is assembled. No precomputation was performed to ensure crack free tessellations. Although in this case no artifacts become visible which is hardly achievable with previous methods. In this example the frame rate is about 25 frames per second on the same PC as above. In this case the retessellation of the individual patches is the bottleneck.

6 Conclusions and Future Work

In this paper we described a novel method for efficient view-dependent LOD rendering via gap filling. We introduced the notion of ”fat borders”, a perception based rendering aid that covers the cracks between unconnected components of complex models that have different LOD levels.

Our implementation runs on a Pentium IV. processor with the ATI Radeon 9700Pro card and for models containing more than 11000 components achieves 8-10 frames per second. While our implementation focuses on the rendering of trimmed NURBS models (hence the relatively large number of components) the concept we introduced is general enough to be applicable to most present boundary representations.

As future work we intend to extend our dynamic view dependent renderer to to handle other kinds of model representations (e.g. subdivision surfaces, isosurfaces) and investigate how our method performs in the field of out-of-core simplification.

7 Acknowledgements

This project was partially funded by the German Federal Ministry of Education and Research (BMBF) under the project of OpenSG Plus. We thank Volkswagen for providing us with the trimmed NURBS models.

We thank ATI for kindly providing us with a Radeon 9700Pro graphics board prior to its official release.

We would also like to thank Andreas Schilling for many fruitful discussions on this topic.

References

BAXTER, W. V., SUD, A., GOVINDARAJU, N. K., AND MANOCHA, D., 2002. Gigawalk: Interactive walkthrough of complex environments.

DECORO, C., AND PAJAROLA, R. 2002. Xfastmesh: fast view-dependent meshing from external memory. In *Proceedings of the conference on Visualization '02*, IEEE Press, 363–370.

DUCHAINEAU, M. A., WOLINSKY, M., SIGETI, D. E., MILLER, M. C., ALDRICH, C., AND MINEEV-WEINSTEIN, M. B. 1997. ROAMing terrain: real-time optimally adapting meshes. In *IEEE Visualization*, 81–88.

EL-SANA, J., AND CHIANG, Y.-J. 2000. External memory view-dependent simplification. 139–150.

EL-SANA, J., AND VARSHNEY, A. 1999. Generalized view-dependent simplification. 83–94.

ERIKSON, C., MANOCHA, D., AND WILLIAM V. BAXTER, I. 2001. Hlods for faster display of large static and dynamic environments. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, ACM Press New York, NY, USA, 111 –120.

FLORIANI, L. D., MAGILLO, P., AND PUPPO, E. 1998. Efficient implementation of multi-triangulations. In *IEEE Visualization '98*, D. Ebert, H. Hagen, and H. Rushmeier, Eds., 43–50.

FORSEY, D. R., AND KLASSEN, R. V. 1990. An adaptive subdivision algorithm for crack prevention in the display of parametric surfaces. In *Graphics Interface '90*, Canadian Information Processing Society, 1–8.

GU, X., GORTLER, S. J., AND HOPPE, H. 2002. Geometry images. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM Press, 355–361.

GUTHE, M., AND KLEIN, R. 2003. Efficient NURBS Rendering using View-Dependent LOD and Normal Maps. In *Journal of WSCG*, vol. 11.

GUTHE, M., MESETH, J., AND KLEIN, R. 2002. Fast and memory efficient view-dependent trimmed nurbs rendering. In *proceedings of Pacific Graphics 2002*, IEEE Computer Society, 204–213.

HERZEN, B. V., AND BARR, A. H. 1987. Accurate triangulations of deformed, intersecting surfaces. In *Computer Graphics (Proceedings of SIGGRAPH 89)*, vol. 21, 103–110.

HOPPE, H. 1996. Progressive meshes. *Computer Graphics 30*, Annual Conference Series, 99–108.

HOPPE, H. 1997. View-dependent refinement of progressive meshes. *Computer Graphics 31*, Annual Conference Series, 189–198.

HOPPE, H. H. 1998. Smooth view-dependent level-of-detail control and its application to terrain rendering. In *IEEE Visualization '98*, D. Ebert, H. Hagen, and H. Rushmeier, Eds., 35–42.

KLEIN, R., LIEBICH, G., AND STRASSER, W. 1996. Mesh reduction with error control. In *IEEE Visualization '96*, R. Yagel and G. M. Nielson., Eds., 311–318.

KLEIN, R., COHEN-OR, D., AND HUTTNER, T. 1997. Incremental view-dependent multiresolution triangulation of terrain. In *Pacific Graphics 1997*, 127–136.

KUMAR, S., MANOCHA, D., ZHANG, H., AND HOFF, K. E. 1997. Accelerated walkthrough of large spline models. In *1997 Symposium on Interactive 3D Graphics*, ACM SIGGRAPH, 91–102. ISBN 0-89791-884-3.

KUMAR, G. V. V. R., SRINIVASAN, P., SHASTRY, K. G., AND PRAKASH, B. G. 2001. Geometry based triangulation of multiple trimmed nurbs surfaces. *Computer-Aided Design 33*, 6 (May), 439–454. ISSN 0010-4485.

LINDSTROM, P., KOLLER, D., RIBARSKY, W., HODGES, L. F., FAUST, N., AND TURNER, G. A. 1996. Real-time, continuous level of detail rendering of height fields. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, ACM Press, 109–118.

LUEBKE, D., AND ERIKSON, C. 1997. View-dependent simplification of arbitrary polygonal environments. *Computer Graphics 31*, Annual Conference Series, 199–208.

NISHITA, T., SEDERBERG, T. W., AND KAKIMOTO, M. 1990. Ray tracing trimmed rational surface patches. In *Computer Graphics (Proceedings of SIGGRAPH 90)*, vol. 24, 337–345. ISBN 0-201-50933-4.

PAJAROLA, M. R., ANTONIJUAN, M., AND LARIO, R. 2002. QuadTIN: quadtree based triangulated irregular networks. In *IEEE Visualization*, 395 – 402.

PAJAROLA, R. 2001. Fastmesh: Efficient view-dependent meshing. In *Pacific Graphics 2001*, 22–30.

ROCKWOOD, A. P., HEATON, K., AND DAVIS, T. 1989. Real-time rendering of trimmed surfaces. In *Computer Graphics (Proceedings of SIGGRAPH 89)*, vol. 23, 107–116.

SCHMALSTIEG, D., AND SCHAUFLER, G. 1997. Smooth levels of detail. In *VRAIS 97*, 12–19.

SHANTZ, M., AND CHANG, S.-L. 1988. Rendering trimmed nurbs with adaptive forward differencing. In *Computer Graphics (Proceedings of SIGGRAPH 88)*, vol. 22, 189–198.

XIA, J. C., AND VARSHNEY, A. 1996. Dynamic view-dependent simplification for polygonal models. In *IEEE Visualization '96*, R. Yagel and G. M. Nielson, Eds., 335–344.

XIA, J. C., EL-SANA, J., AND VARSHNEY, A. 1997. Adaptive real-time level-of-detail-based rendering for polygonal models. *IEEE Transactions on Visualization and Computer Graphics 3*, 2, 171–183.