This document contains the draft version of the following paper:

A.K. Priyadrashi and S.K. Gupta. Finding mold-piece regions using computer graphics hardware. *Geometric Modeling and Processing Conference*, Pittsburgh, PA, July 2006.

Readers are encouraged to get the official version from the conference proceedings or by contacting Dr. S.K. Gupta (skgupta@umd.edu).

Finding Mold-Piece Regions Using Computer Graphics Hardware

Alok K. Priyadarshi[1] and Satyandra K. Gupta[2]

[1] Solidworks Corporation, Concord, MA 01742, USA
apriyadarshi@solidworks.com
[2] University of Maryland, College Park, MD 20742, USA
skgupta@eng.umd.edu

Abstract. An important step in the mold design process that ensures disassembly of mold pieces consists of identifying various regions on the part that will be formed by different mold pieces. This paper presents an efficient and robust algorithm to find and highlight the mold-piece regions on a part. The algorithm can be executed on current-generation computer graphics hardware. The complexity of the algorithm solely depends on the time to render the given part. By using a system that can quickly find various mold-piece regions on a part, designers can easily optimize the part and mold design and if needed make appropriate corrections upfront, streamlining the subsequent design steps.

1 Introduction

While designing injection molds, there are often concerns about disassemblability of the mold as designed. An important step in the mold design process that ensures disassembly of mold pieces consists of identifying various regions on the part that will be formed by different mold pieces. These regions are called *Mold-Piece Regions*.

Most of the literature in mold design is focused on detecting undercuts and finding undercut-free directions. For an overview of mold design literature, the reader is directed to [Priy03,Bane06]. Ahn et al. [Ahn02] presented a provably complete algorithm for finding undercut-free parting directions. Khardekar et al. [Khar05] implemented the algorithm presented by Ahn et al. [Ahn02] on programmable GPUs. They also describe a method to highlight the undercuts.

We use GPUs to find mold-piece regions on a part efficiently and robustly. The basic idea behind the algorithm is similar to shadow mapping. The near-vertical facets are handled by slightly perturbing the vertices on those facets and visibility sampling. We describe an implementation of our algorithm that can be executed on any OpenGL 2.0. The complexity of our algorithm solely depends on the time to render the given part.

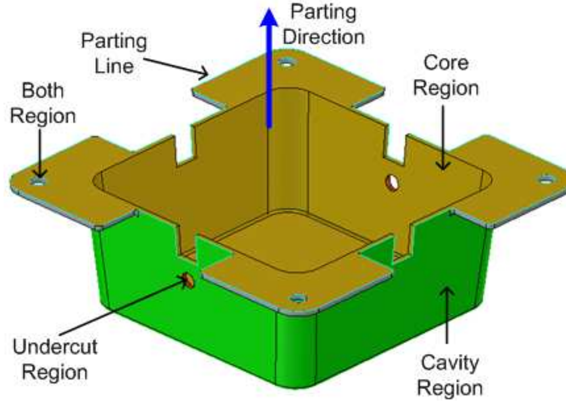


Fig. 1. Mold-Piece Regions

2 Finding Mold-Piece Regions

A *Mold-Piece Region* of a part is a set of part facets that can be formed by a single mold piece. Given a polyhedral object P and a parting direction $\boldsymbol{d}$, there are four mold-piece regions with the following property:

1. *Core* is accessible from $+\boldsymbol{d}$, but not $-\boldsymbol{d}$
2. *Cavity* is accessible from $-\boldsymbol{d}$, but not $+\boldsymbol{d}$
3. *Both* is accessible from both, $+\boldsymbol{d}$ and $-\boldsymbol{d}$
4. *Undercut* is not accessible from either $+\boldsymbol{d}$ or $-\boldsymbol{d}$

Hence, the problem of finding mold-piece regions reduces to performing accessibility analysis [Dhal03] of P along $+\boldsymbol{d}$ and $-\boldsymbol{d}$. Figure 1 shows various mold-piece regions for a part.

2.1 Overview of Approach

We use programmable GPUs to highlight the mold-piece regions on a part. The basic idea is very similar to hardware shadow mapping [Kilg01]. The given part is illuminated by two directional light sources located at infinity in the positive and negative parting directions. The regions that are lit by the upper and lower lights are marked as 'core' and 'cavity' respectively. The regions lit by both the lights are marked as 'both', while the regions in shadow are marked as 'undercuts'.

For a given parting direction, our approach highlights the mold-piece regions on a part in two steps:

1. *Preprocessing*: We create two shadow maps by performing the following procedure. First the part is rendered with the camera placed above the part and view direction along the negative parting direction. The resulting z-buffer is transferred to a depth texture (shadow map). The current orthogonal view

matrix is also stored for the next step. The same procedure is repeated with the view direction along positive parting direction.
2. *Highlighting*: The user can then rotate the camera and examine the mold-piece regions of the part from all directions. A vertex program transforms the incoming vertices using the two model-view matrices stored in the pre-processing stage. The fragment program determines the visibility of each incoming fragment by comparing its depth with the depth texture values stored in the preprocessing stage and colors it accordingly.

If the algorithm is implemented as described, all the vertical facets will be reported as undercuts. Also, like any method based on shadow mapping, it needs to handle aliasing and self-shadowing. Section 2.2 and Section 2.3 describe techniques to handle these issues.

2.2 Handling Near-Vertical Facets

There is a slight difference between the notion of visibility in computer graphics and accessibility. The mathematical conditions for visibility and accessibility of a facet with normal $\boldsymbol{n}$ in direction $\boldsymbol{d}$ are the following:

Visible if: $\boldsymbol{d} \cdot \boldsymbol{n} > 0$

Accessible if: $\boldsymbol{d} \cdot \boldsymbol{n} \geq 0$

In other words, a facet perpendicular to a direction (vertical facet) is not visible, but accessible. This means that all the vertical facets will be reported as undercuts.

In addition to vertical facets, we also need to handle facets whose normals are very close to being perpendicular to the parting direction. These near-vertical facets are usually produced as a result of the approximation introduced by faceting vertical curved surfaces. The robustness problems in geometric computations are usually handled by slightly perturbing the input. But we cannot adopt this approach here as perturbing the vertices of the part will change it's appearance on the computer screen. We solve this problem by visibility sampling. To determine the accessibility of a rasterized fragment, the neighborhood of the corresponding texel in the shadow map is sampled in the image space. If any sample passes the visibility test, the fragment is marked as accessible. Incidentally, percentage closer filtering (PCF) [Reev87] used to produce anti-aliased shadows does just that.

For a given parting direction $\boldsymbol{d}$, we divide the part facets into three categories:

1. Up facets: $\boldsymbol{d}.\boldsymbol{n} \geq \tau$
2. Down facets: $\boldsymbol{d}.\boldsymbol{n} \leq -\tau$
3. Near-vertical facets: $|\boldsymbol{d}.\boldsymbol{n}| < \tau$

where $\boldsymbol{n}$ is the facet normal and τ is normal tolerance whose value is dependent on the surface tolerance introduced by faceting the part. It is usually set between 1-2 degrees.

Up and down facets are tested for accessibility along $-\boldsymbol{d}$ and $+\boldsymbol{d}$ respectively. The near-vertical facets are tested in both the directions with PCF enabled. In

our implementation, we used the OpenGL extension ARB_shadow that samples the neighborhood of a fragment and returns the average of all the depth comparisons. If the returned value is greater than zero, we mark the fragment as accessible. The PCF kernel that determines the size of the sampling neighborhood should be adjusted according to the surface tolerance of the given part and resolution of the shadow map. We found that 3x3 kernel (9 samples) worked fine for the parts that we tested.

2.3 Preventing Self-Shadowing

Our algorithm, being based on shadow mapping, is prone to self-shadowing due to precision and sampling issues. The focus of the currently available algorithms to prevent self-shadowing is mainly on producing aesthetically pleasing results. They may not be physically correct. We decided not to use the most popular polygon offset technique [Kilg01] after extensive experimentation. We found that it is very difficult to specify an appropriate bias for a part automatically. If the bias is too little, everything begins to shadow. And if it is too much, shadow starts too far back i.e., some of the fragments that should be in shadow are incorrectly lit. We found that this problem is exaggerated in case of mechanical parts with regions of high slope. We developed an adaptation of the second depth technique [Wang94] that prevents self-shadowing and robustly handles the near-vertical facets.

Second depth technique [Wang94] is based on the observation that in case of solid objects there is always a back facet on top of a shadowed front facet. It renders only the back facets into the shadow map and avoids many aliasing problems because there is adequate separation between the front and back facets. But it may show incorrect results when used with PCF for near-vertical facets. As explained in Section 2.2, we use PCF to sample the neighborhood of a point on a near-vertical facet. If any sample passes the visibility test, we mark the point as accessible. Because the shadow map only partially overlaps the PCF kernels for both points A and B, they will be reported as only 50% shadowed and hence accessible. This is the intended result for point B, but incorrect for point A.

To solve this problem, we use a visibility theorem for polyhedral surfaces based on the results published in [Kett99] and [Ahn02].

Definition 1. *An edge is a contour edge if it is incident to a front-facing facet and a back-facing facet for a given viewing direction.*

Theorem 1. *For a given polyhedron and a viewing direction, if the edges and facets of the polyhedron are projected into the viewing plane, the visibility of the projected facets can only change at the intersection with convex contour edges.*

The proof of the above theorem follows from the results presented in [Kett99] and [Ahn02]. We exploit the corollary of the above theorem that the visibility of projected facets cannot change at the intersection with *concave* contour edges.

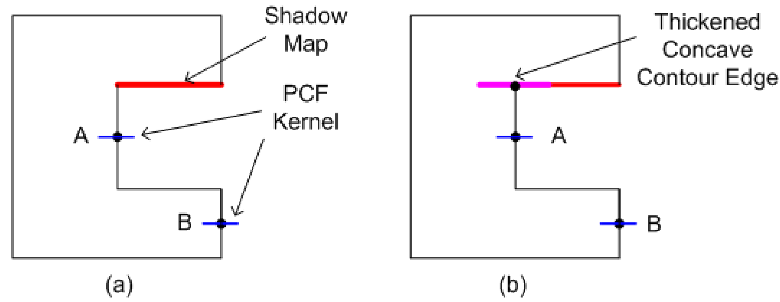


Fig. 2. The problem with the second-depth technique when used with PCF (the PCF kernel and the contour edge has been exaggerated for illustration purposes); (a) Both point A and point B are reported as only 50% shadowed and hence accessible; (b) The problem is solved by rendering thick concave contour edges into the shadow map

When creating the shadow map, we also render thick concave contour edges along with the back facets. As can be seen in Figure 2(b), now that the shadow map fully overlaps the PCF kernel for point A, it will be correctly reported as fully shadowed and hence marked as inaccessible. It can also be seen that thickening the concave contour edges does not affect the accessibility of point B.

2.4 Transferring Results from the GPU to CPU

The previous sections describe how to find and highlight the mold-piece regions using GPUs. This section describes how the information on mold-piece regions can be transferred back to the CPU for other purposes such as designing molds. We describe a simpler two-pass algorithm to accomplish the same.

We first assign a unique ID (color) to each facet of the given part. Almost all the currently available graphics cards support at least 24-bit color palette that can generate over 16 million unique colors. Then we follow the following procedure to obtain the results on the CPU. The part is first rendered with the camera placed above the part and view direction along the negative parting direction. The resulting frame buffer (image) is read back to the CPU. The facets whose IDs are present in the resulting image constitute the 'core' region. The same procedure is followed with the view direction along the positive parting direction to obtain the 'cavity' region. The facets missing from both the images are undercuts.

The problem with the above approach is that it cannot find the 'both' region. None of the facets will be present in both the frame buffers and all of the vertical facets will be reported as undercuts because being perpendicular to the viewing direction, they cannot be rendered. But now since the part is not rendered for visualization purposes, we can perturb the vertices of the part. For both the viewing directions (negative and positive parting direction), we slightly perturb the vertices of the near-vertical facets such that it becomes a front-facing facet

for that viewing direction and hence an eligible candidate for being rendered. This perturbation is similar to adding a draft to the near-vertical facets and can be done on either the CPU or by a vertex program loaded on the GPU. A reference plane is first located at the top-most vertex with respect to the viewing direction and then each vertex on the near-vertical facets is slightly moved along the surface normal at that point. The perturbation amount is in proportion to the distance of the vertex from the reference plane and is given by $d = z. \tan(\tau)$, where τ is a small user-defined angle, which depends on the average length of facets and resolution of the frame buffer. We found that for a 512x512 buffer, $\tau = 0.5°$ was appropriate for the parts that we tested.

The algorithm for transferring the results from the GPU to CPU is based on the assumption that the each facet belongs to only one mold-piece region. Sometimes a front facet needs to be split into a core and an undercut facet, or a vertical facet needs to be split into all the four mold-piece regions. A brute-force approach to overcome this limitation could be splitting each facet into very small facets. Another approach could be projecting each facet into the viewing plane and splitting them at the intersection with convex contour edges [Kett99], and performing trapezoidal decomposition of vertical facets [Ahn02].

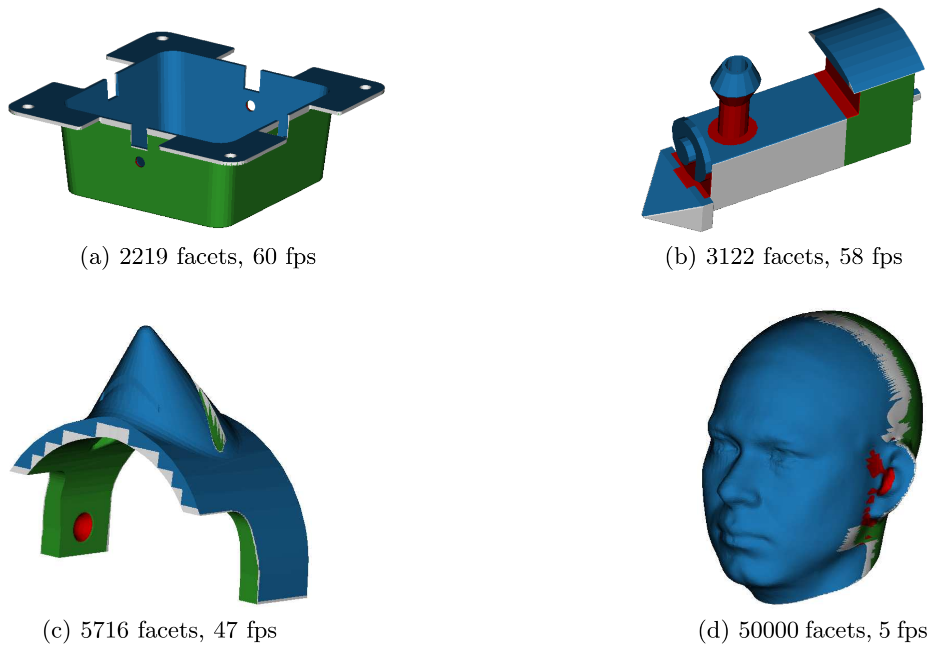
3 Implementation and Results

The latest GPUs allow users to load their own programs (shaders) to replace some stages of the fixed rendering pipeline. We have implemented our algorithm as shader programs using OpenGL Shading Language (GLSL). The implementation has been successfully tested on more than 50 industrial parts. It currently supports Stereolithography (STL) and Wavefront (OBJ) part files. Figure 3 shows the screenshot of four example parts. Figure 4 shows the performance of our implementation on 128 MB NVIDIA Fx700Go card. It shows the obtained frame rates when simply rendering the part using fixed OpenGL pipeline (without highlighting) and with highlighting. It can be seen that the overhead imposed by the highlighting algorithm does not significantly affect the time taken by the GPU to render a frame. The observed drop in performance when highlight is at most one fps. In other words, the complexity of the algorithms solely depends on the time to render the given part.

4 Conclusions

In this paper we describe a method that utilizes current-generation GPUs to find and highlight mold-piece regions on a part. We presented techniques for robustly handling the near-vertical facets by slightly perturbing the vertices on those facets and visibility sampling. We also presented a technique that prevents self-shadowing and robustly handles the near-vertical facets.

Our algorithm exploits the computational power offered by the GPUs. Moreover, an efficient implementation of our algorithm does not impose any significant overhead on the GPU. The mold-piece regions even for parts with more than



(a) 2219 facets, 60 fps

(b) 3122 facets, 58 fps

(c) 5716 facets, 47 fps

(d) 50000 facets, 5 fps

Fig. 3. Screenshots of four example parts. The color scheme for highlighting is following. Core region is blue, cavity region is green, both region is gray, and the undercuts are red. Number of facets and obtained rendering speed is reported against each subfigure.

50,000 facets can be highlighted at interactive rates. We believe that in the current scenario when the data size is growing at exponential rates because of the advances in scanning technology, such a system that provides real-time information about mold-piece regions will be very useful to the part and mold designers alike. They can easily optimize the part and mold design and if needed make appropriate corrections upfront, streamlining the subsequent design steps.

Acknowledgments

This work has been supported by NSF grant DMI-0093142. However, the opinions expressed here are those of the authors and do not necessarily reflect that of the sponsor. We would also like to thank the reviewers for their comments that improved the exposition.

References

[Ahn02] Ahn, De Berg, Bose, Cheng, Halperin, Matousek, and Schwarzkopf, Separating an object from its cast. *Computer-Aided Design*, 34, 547-59, 2002

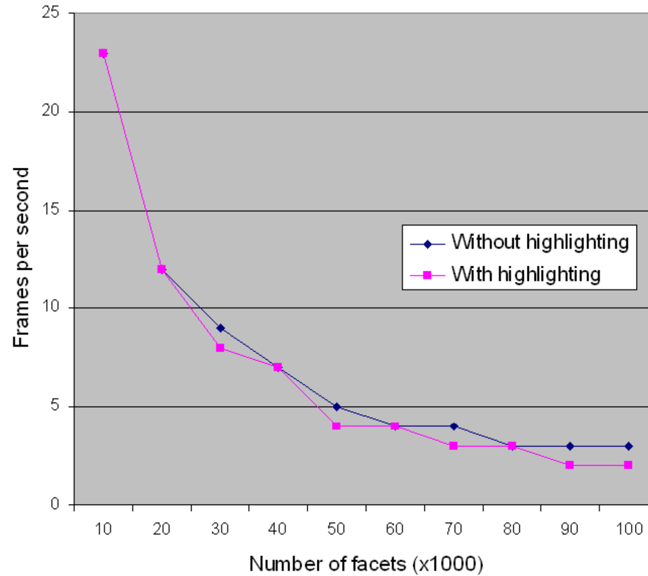


Fig. 4. Performance of the algorithm on 128 MB NVIDIA Fx700Go card. The plot shows the obtained frame rates when simply rendering the part (without highlighting) and those when also highlighting the mold-piece regions

[Bane06] A.G. Banerjee, and S.K. Gupta, A step towards automated design of side actions in injection molding of complex parts. In *Proceedings of Geometric Modeling and Processing*, Pittsburgh, PA, 2006

[Dhal03] S. Dhaliwal, S.K. Gupta, J. Huang, and A. Priyadarshi. Algorithms for computing global accessibility cones. *Journal of Computing and Information Science in Engineering*, 3(3):200–209, September 2003

[Kett99] Lutz Kettner. Software Design in Computational Geometry and Contour-Edge Based Polyhedron Visualization. PhD Thesis, ETH Zrich, Institute of Theoretical Computer Science, September 1999

[Kilg01] Mark Kilgard. Shadow Mapping with Today's Hardware. Technical presentation, http://developer.nvidia.com/object/ogl_shadowmap.html

[Khar05] Khardekar, Burton, and McMains. Finding feasible mold parting directions using graphics hardware, *Proceedings of the 2005 ACM symposium on Solid and Physical Modeling*, Cambridge, MA, June 2005, pp. 233-243
2004 ACM Workshop on General-Purpose Computing on Graphics Processors (p. C-19) and SIGGRAPH 2004 poster sessions, Los Angeles, Aug. 2004

[Priy03] A. Priyadarshi, and S.K. Gupta. Geometric algorithms for automated design of multi-piece permanent molds. *Computer Aided Design*, 36(3): 241–260, 2004.

[Reev87] W. T. Reeves, D. H. Salesin, and R. L. Cook. Rendering antialiased shadows with depth maps. In *Computer Graphics (SIGGRAPH 87 Proceedings)*, pages 283-291, July 1987

[Wang94] Y. Wang and S. Molnar. Second-Depth Shadow Mapping. *UNC-CS Technical Report TR94-019,* 1994