CHALMERS

Automatic Surface Reduction and Normal Correction in Large 3D Models

Master of Science Thesis in Computer Science and Enginerring

PER LINDSTRAND

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Göteborg, Sweden, January 2009

Automatic Surface Reduction and Normal Correction in Large 3D Models

PER LINDSTRAND

Examiner: ULF ASSARSSON

Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden January 2009

Abstract

Traditionally virtual representations or models are created with different software systems with different requirements. In order to build a car, designers and engineers uses so called CAD (Computer-Aided Design) modeling software to create parameterized surfaces for all parts of that particular car such as seats, buttons, engine, nuts and bolts. The result is a complete blueprint of the whole car and all of its parts. This level of detail is needed to manufacture the car, but in order to produce a poster featuring the aesthetic design of the same car, the electrical wiring, for instance, inside the front door is of no visual importance. This requires only a visually appealing (yet accurate) representation of the same car and would normally require artists (rather than engineers) to create a schematic model of the car. This process can be automated by using the blueprints of the car in order to create a visually accurate model through a tessellation process. However, the tessellated model would include all wires, nuts and bolts, which are mostly irrelevant to the final product. Furthermore, the tessellation process often introduces errors such as surfaces facing the wrong way, duplicate surfaces on top of each other, let alone the abundance of unnecessary surfaces. In order to use the tessellated model to efficiently produce computer generated images these issues must be addressed.

Sammanfattning

Traditionellt skapas virtuella representationer eller modeller med olika verktyg med olika krav. För att bygga en bil använder designers och ingenjörer sig av så kallade CAD-modelleringsmjukvara (Computer-Aided Design) för att skapa parameteriserade ytor för alla delar av den aktuella bilen såsom säten, knappar, motor, skruvar och muttrar. Resultatet är en komplett ritning av hela bilen och alla dess delar. Denna detaljnivå krävs för att bygga bilen, men för att göra en estetisk broschyrbild av densamma är t ex elektriska komponenter och kablar, skruvar och muttrar inuti en framdörr visuellt sett onödiga. Detta kräver endast en visuellt tilltalande (dock korrekt) representation av samma bil och kräver normalt att grafiska artister (snarare än ingenjörer) skapar en schematisk modell av bilen. Denna processen kan automatiseras genom att använda ritningarna på bilen för att skapa en visuellt korrekt modell genom en tesselleringsprocess. Den tessellerade modellen innefattar alla kablar, skruvar och muttrar som för det mesta är irrelevant för slutprodukten. Dessutom introducerar tesselleringsprocessen ofta fel som felvända ytor, duplicerade ytor, utöver mängden onödiga ytor. För att kunna använda den tessellerade modellen för att effektivt producera datorgenererade bilder måste dessa problem hanteras.

Preface

This master's thesis was written and performed by Per Lindstrand at the Department of Computer Science and Engineering at Chalmers University of Technology on behalf of Spark Vision AB during the fall of 2007 in Gothenburg, Sweden. The project work has been performed at Spark Vision AB who have supplied both rooms and computers. Supervisor was Johnny Widerlund at Spark Vision AB and examiner was Ulf Assarsson at the Department of Computer Science and Engineering.

The thesis is aimed at people with a good understanding of computer graphics and are fairly familiar with computers, programming and algorithms.

Acknowledgements

I would like to thank my supervisor at Spark Vision AB Johnny Widerlund for all his help and invaluable exchange of ideas. I would also like to thank my supervisor and examiner Ulf Assarsson at Chalmers University of Technology for his dedication and immense patience. Great thanks to all the people at Spark Vision AB for all the help that made this possible.

A special thanks to friends, family and girlfriend for their love, support and understanding.

Contents

1 Introduction

1.1 Assignment

The assignment for this master's thesis project was to create a set of tools for Spark Vision's *Genesis* and *Ignite* software systems to perform automatic surface reduction and surface correction. The tools should run in reasonable time for very large models (approximately 15 to 20 million triangles) and should meet the quality demands required by high resolution production images.

Specifically, reasonable time is cost-effective time and quality compared to manually performing the task at hand. High resolution production images are 9+ megapixel and printed on A4-A3 sized paper.

1.2 Background

The specialized visualization industry often uses technical designs and schematics such as CAD data as a basis for the final production. The process involves transforming a CAD model in a tessellation process [18] [20]. Tessellated CAD data often produce irregular or complex data as a result of the tessellation process itself. Some of these irregularities, such as cracks, cavities and incorrect normals can be and is often fixed locally by the tessellation software. However, overall surface orientation (surface normal coherency) and visually unimportant surfaces are traditionally not handled, the latter for obvious reasons.

In order to efficiently visualize large models (such as cars) created through a tessellation process, the models often have to be reduced in size. The reduction, however, should not affect the quality of the visualization, that is, it should not alter visually defining surfaces. Furthermore, technical surfaces are often orientation independent, as no real world materials are one-sided. The tessellated surfaces are therefore often somewhat randomly oriented.

1.3 Prerequisites

To fully comprehend and utilize the theories presented in this thesis, a basic understanding of modern computer graphics, linear algebra and algorithmic theory is required. The author recommends reading [7], [9], [10] and [11] in order to learn about or recapitulate the mentioned above subjects.

1.4 Previous Work

The academic work regarding surface reduction is very limited, there has been little public work on the subject of permanent hidden surface removal. Perhaps due to the fact that 3D models are often created and designed by graphics artists who usually don't create excessive amounts of hidden surfaces. Surface subdivision and merging are common operations in 3D modeling software but the algorithms used are not designed to be lossless.

There has been a lot of work done on the subject of Hidden Surface Removal (HSR)

and Visible Surface Determination (VSD), such algorithms are often applied in real-time rendering applications (such as games and various visualization tools), some of which contain ideas used in the algorithms developed in this thesis. The real-time hidden surface removal algorithms are primarily used to minimize rendering per frame without modifying the content [12].

Most of the work within the area of surface correction focus heavily on optimally and analytically correcting surfaces [15] [16] [17]. This often results in a lot of highly complex calculations per surface, not suitable for larger models in terms of running time and computational complexity. Basic forms of *normal* correction are often incorporated in tessellation software in order to produce coherent results whereas overall surface orientation coherency is ignored.

The problems presented in this thesis are common to the industry concerned with tessellated CAD visualization, which is highly competitive and as a result, permanent hidden surface removal and surface normal correction algorithms are developed privately and not open to the public.

2 Overview

2.1 Glossary

The following terms are used throughout the thesis.

primitive Common graphics objects such as triangles or quadrilaterals.

surface A *surface* is a set of primitives.

model A *model* is a set of surfaces, most likely a purposeful collection such as a solid geometric volume or a large coherent surface.

scene A *scene* is a space containing a set of models, surfaces or other objects.

2.2 Graphics Hardware Acceleration

Algorithms outlined in this thesis utilizes hardware accelerated occlusion queries. These effectively yields fast ray tracing operations when combined with *depth masking* (see Section 3.2), a similar technique is used in most occlusion culling schemes.

3 Visibility

3.1 Local and Global Visibility

Given that a model consists of a set of surfaces, we define visibility in the following way.

global visibility A surface in a model is *globally visible* if an infinite ray from some point on that surface can be constructed such that it does not intersect any other surface in the model.

Corollary, a surface is hidden or occluded if such a ray does not exist. This property is used in occlusion culling techniques [2] [3]. The algorithms presented in this thesis are based on surface visibility tests or *occlusion queries*. These tests are derived from the fact that a surface can be determined *visible* in this way.

local visibility A surface in a model is *locally visible* from a viewpoint p if a ray between some point on that surface and p can be constructed such that it does not intersect any other surface in the model.

It is possible to perform this type of visibility test very fast using graphics hardware. An interface for occlusion queries was defined in OpenGL 1.5 [1]. Ray intersections are performed in the rendering pass using occlusion queries, these are issued per surface during rendering of the scene. Depth testing ensures that only the closest intersections are valid and used. In this application, each frame buffer fragment corresponds to a ray intersecting the scene and if the whole scene is contained in the given view it can approximate the *local visibility* of a surface.

In surface reduction, the classification of a surface as hidden is more detriment to the visualization than visible. It is therefore preferable to consider surfaces visible than to consider them hidden. The visibility test is constructed such that a surface is hidden if and only if it is not visible in any visibility query. It is easier to show that a surface is visible than it is to show that it is hidden. This follows from the definition of *global visibility* where only a single ray intersection test is necessary to show that a surface is visible while an infinite number of tests is necessary to show that a surface is hidden.

Global visibility is equivalent to an infinite number of local visibility queries. Global visibility can be approximated by performing a number of approximated local visibility queries. The quality of the approximation is naturally dependent on the quality and amount of queries. Higher resolution and more queries yields a better approximation.

The basic visibility query algorithm performs a number of approximated local visibility queries using occlusion queries for each rendered surface. The result of the algorithm is essentially a measure of how visible each surface is from a set of viewpoints (see Figure 1).

The visibility of a surface can be expressed as a probability in terms of approximated local visibility tests. Given an enclosing sphere around the scene, the surface area of that sphere is a sample space while the combined area of all viewpoints from which that surface is visible is the event space or event. Let the sphere surface area be U and the area of all viewpoints from which the surface is visible be A. If $A = U$ then the surface is always visible and cannot be incorrectly classified. If $A = 0$ then the surface is never visible and cannot be incorrectly classified. If $A < U$, the probability of a given surface generating a result in a single visibility test is $\rho = A/U$ where $\rho \in [0, 1)$ and the probability of that surface being incorrectly classified as hidden is

$$\lim_{n\to\infty} \prod^{n} \rho \to 0 \tag{1}$$

that is, as the number of visibility test samples goes to infinity then the probability of a visible surface being incorrectly considered hidden goes to zero.

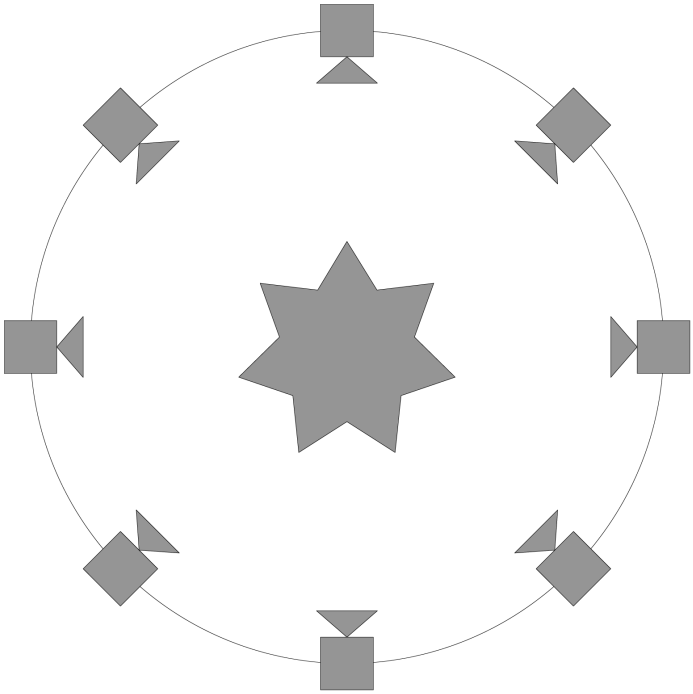

Figure 1: A two-dimensional example of spherical sampling, sample positions are generated on an enclosing sphere around the model's convex hull.

3.2 Depth Masking

Occlusion queries can be used to efficiently determine the visibility of a primitive [13]. This technique can be used to determine the visibility of a surface or a model as well by batching or grouping sets of primitives together.

An occlusion query counts and returns the number of pixels (actually *samples* as described in the specification) of a surface that passed the depth and stencil tests [13] [14]. As the name suggests this is designed and used to determine occlusion of surfaces. However, the result from an occlusion query can be used to measure and quantify the visibility of a surface as well, since these are essentially mutual qualities.

Rendering the scene to the depth buffer creates a depth map of the scene. This map can be used as a reference for a *masking* occlusion query with a equality depth test. The occlusion query result will correspond to the number of samples with a depth value equal to that in the depth reference map. This yields an approximate visibility measure for all rendered surfaces and will be correct for intersecting surfaces as well.

3.3 Algorithm

Given a set of surfaces σ and a set of viewpoints ξ, for every surface $s \in \sigma$ and viewpoint $v \in \xi$ compute visibility measure $\phi_v(s)$ such that $\phi_v(s) > 0$ if and only if s is visible from v.

Algorithm 1 returns visibility measure ϕ given a set of surface σ and a set of viewpoints ξ.

Algorithm 1 VISIBILITY-QUERY

Require: Set of viewpoints ξ
Require: Set of surfaces σ
  $\phi \leftarrow \emptyset$ {Occlusion result mapping for surfaces}
  for all v such that $v \in \xi$ **do**
    Setup viewpoint v
    Clear depth buffer
    Use depth test function `LESS`
    for all s such that $s \in \sigma$ **do**
      Render surface s
    end for
    Use depth test function `EQUAL`
    for all s such that $s \in \sigma$ **do**
      r = `occlusion-query(`s`)`
      $\phi(s) \leftarrow \phi(s) + r$
    end for
  end for
  return ϕ

4 Reduction

4.1 Theory

The number of surfaces in a model can be reduced by

(i) replacing two or more surfaces with one surface,

(ii) removing unnecessary surfaces.

The term *unnecessary* is subject to application and in this thesis an unnecessary surface does not contribute visually to the scene. Method *(i)* requires spatial analysis and restructuring of neighboring surfaces, a computationally difficult operation [18] [19] [20]. Given the generally large number of surfaces in the test data for this thesis, method *(i)* is not a viable option. Method *(ii)* requires only visibility analysis, which can be performed efficiently using the technique described previously in this thesis (see Section 3.)

4.2 Algorithm

The visibility query algorithm defined in Section 3 (see Algorithm 1) provides meaningful information used to determine whether to remove surfaces. Using Algorithm 1 it is possible to statistically classify surfaces as either visible or invisible. The visibility information can then be used to either keep or remove surfaces. This yields Algorithm 2, the simple surface reduction. The visibility query algorithm is bounded by the quality of the graphics driver implementation and graphics hardware itself. These

Algorithm 2 SIMPLE-SURFACE-REDUCTION

Require: Set of viewpoints ξ
Require: Set of surfaces σ
  $\phi \leftarrow$ VISIBILITY-QUERY(ξ, σ)
  for all s such that $s \in \sigma$ **do**
    if $\phi(s) = 0$ **then**
      Remove surface s {Surface s is invisible}
    end if
  end for

limitations are apparent in simple surface reduction algorithm above. Two or more adjacent surfaces with very small projected areas, typically a pixel or less, can produce visibility measures of zero for all but one of the surfaces (see Figure 2).

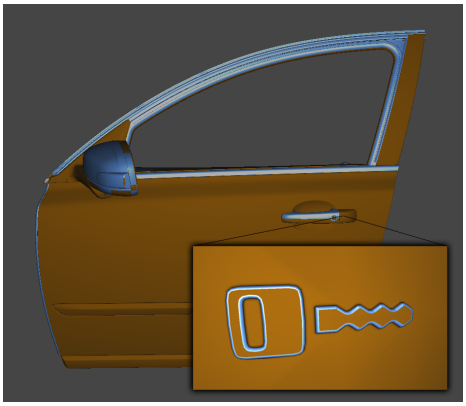

Figure 2: An example of very small surfaces on a model.

In Algorithm 2 small surfaces might be classified as invisible and therefore removed, which often results in numerous holes and cavities (see Figure 3). As mentioned, this occurs when two or more small surfaces share the same pixels and hence only a subset of them are considered visible.

This can be counteracted by grouping small adjacent surfaces and analyzing their combined visibility measure using *neighboring*.

neighboring A *neighboring* is a spatial clustering of radially adjacent surfaces. Given a surface s and a radius r, the *neighboring* of s is a set of all surfaces s' such that $distance(s, s') \leq r$.

Following the definition of neighboring we define Algorithm 3. An on-demand neighboring technique can be trivially added to Algorithm 2, see algorithm 4.

4.3 Implementation

The surface reduction algorithm was implemented and integrated with Spark Vision Genesis software suite using Microsoft Visual C++ and NVSG SDK (NVIDIA Scene

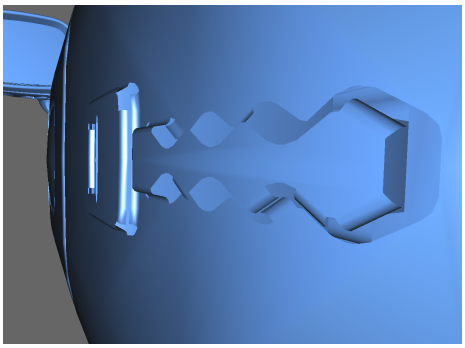

Figure 3: An example of artifacts caused by sample errors in the simple reduction algorithm.

Algorithm 3 NEIGHBORING

Require: Surfaces s
Require: Set of surfaces σ
Require: Radius r
  $\nu \leftarrow \emptyset$
  for all s' such that $s' \in \sigma$ **do**
    if $distance(s, s') \leq r$ **then**
      Insert s' into ν
    end if
  end for
  return ν

Graph Software Development Kit) with the OpenGL renderer.

The tessellated CAD models' surfaces are sets of triangle strips and triangle fans[1]. Hence, the implementation treats such sets rather than primitives such as triangles or quadrilaterals as single surfaces. This granularity limitation has no significant impact

[1] Common computer graphics primitives [5] [6].

Algorithm 4 NEIGHBORING-SURFACE-REDUCTION

Require: Set of viewpoints ξ
Require: Set of surfaces σ
Require: Neighboring radius r
  $\phi \leftarrow$ VISIBILITY-QUERY(ξ, σ)
  for all s such that $s \in \sigma$ **do**
    if $\phi(s) = 0$ **then**
      $\nu \leftarrow$ NEIGHBORING(s, σ, r)
      if $\sum_{s' \in \nu} \phi(s') = 0$ **then**
        Remove surface s
      end if
    end if
  end for

on the outcome of the algorithm since such surfaces tend to be logically (and spatially) coherent.

The final implementation uses a neighboring technique in addition to simple visibility queries. The simplistic tool interface exposes only the most frequently used and most comprehensible parameters to the user (see Figure 4).

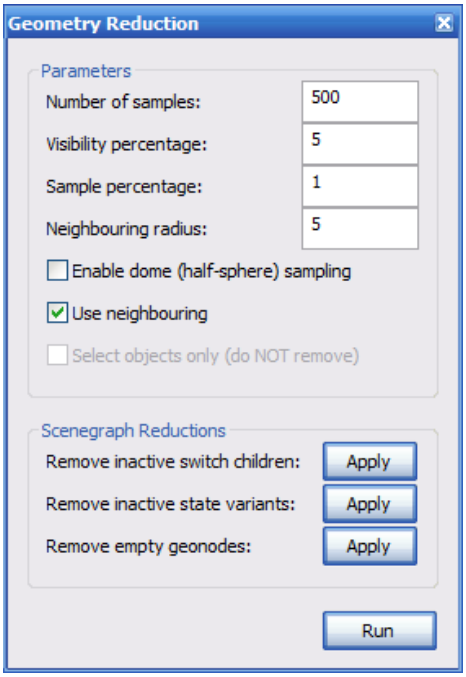

Figure 4: Surface reduction tool interface in Spark Vision Genesis. *Number of samples* is the number of view points generated for the reduction pass. *Visibility percentage* is the surface visibility requirement. *Sample percentage* is the surface view count requirement. *Neighbouring radius* sets the radius of the neighbouring sphere. *Enable dome* uses a half-sphere or dome instead of a sphere for neighbouring. *Use neighbouring* enables or disables neighbouring. The buttons below are convenience tools and not part of the algorithm.

5 Correction

5.1 Theory

In lack of better terminology *surface correction* in this thesis refers to the process of orienting surfaces in a model coherently and correctly. This problem occurs when using single-sided, as opposed to two-sided, rendering in computer graphics. Given an arbitrary model, it is impossible to determine a definite outside or inside of that model [4] (see Figure 5). Hence, any surface correction algorithm must rely on some assumption regarding what should be considered inside and outside.

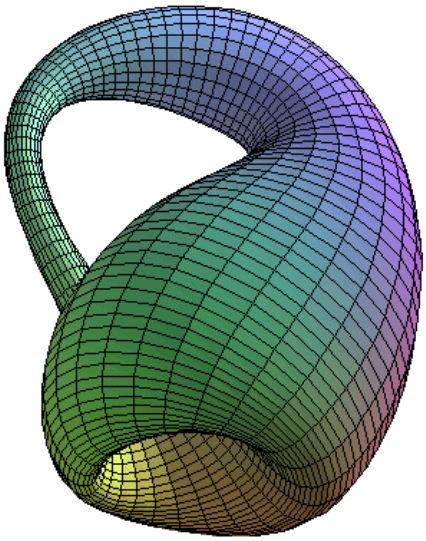

Figure 5: An example of a Klein bottle which has no inside or outside, it is impossible to determine which way its surface should face.

closed A *closed* set of primitives is a convex or concave volume with no holes or openings.

Given the ambiguous nature of general surface orientation, a fundamental assumption or axiom is needed to derive a meaningful algorithm. Assuming a model consisting of a closed set of one-sided surfaces will be perceived exclusively from the outside of its convex hull, every visible (not occluded) surface is front-facing. Suppose there is a surface that is not front-facing. There is no surface occluding the backface of that surface from the outside of the model. This means that there must be a hole in the model, which is a contradiction since the model is closed.

A surface orientation is defined by the principal normal or average normal of that surface. A plane or a triangle will have a single normal while a quadrilateral may have two in which case the average of those two is the principal normal and hence orientation of that surface. Let the general surface orientation ν for a surface consisting of N triangles be defined as

$$\nu = \frac{1}{N} \sum_{i}^{N} n_i \tag{2}$$

where n_i is the normal of the ith triangle.

A surface with center s and general orientation ν, is considered erroneously oriented in viewpoint v if it is visible from v and

$$\frac{\nu \cdot (v - s)}{\|v - s\|_2} < 0 \tag{3}$$

that is, the general surface orientation is such that it is directed away from viewpoint v. The surface is visible but back-facing and therefore considered erroneously oriented

with respect to v.

The condition in Equation 3 will not be accurate for very large, curved surfaces where the normals differ greatly across the surface as shown in Figure 6. Though a more fine-grained surface definition would remedy this.

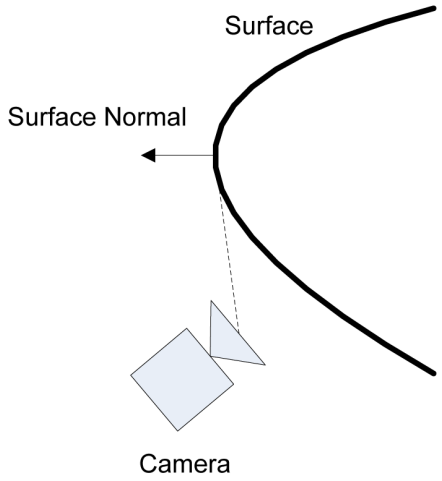

Figure 6: Errors introduced by general surface orientation and surface center for large curved surfaces. Although the surface (*thick line*) is visible and correctly oriented, the principal surface normal (*arrow*) is pointing away from the view vector (*dashed line*) between the surface center and the camera.

5.2 Algorithm

Given a set of surfaces σ and a set of viewpoints ξ, for every surface $s \in \sigma$ and $v \in \xi$ if visibility result $\phi(s) > 0$ and is s is back-facing from v then s is considered to be erroneously oriented. Let $\eta(s)$ denote the number of times s has been determined erroneously oriented.

Algorithm 5 is an extended version of Algorithm 1 that returns, in addition to visibility information ϕ, orientation statistics η given a set of surfaces σ and a set of viewpoints ξ.

5.3 Implementation

The surface correction algorithm was implemented and integrated with Spark Vision Genesis software suite using Microsoft Visual C++ and NVSG SDK (NVIDIA Scene Graph Software Development Kit) with the OpenGL renderer.

The tessellated CAD models' surfaces are sets of triangle strips and triangle fans[2]. Hence, the implementation treats such sets rather than primitives such as triangles or quadrilaterals as single surfaces. This granularity limitation has no significant impact

[2]Common computer graphics primitives [5] [6].

Algorithm 5 VISIBILITY-QUERY-WITH-ORIENTATION

Require: Set of viewpoints ξ
Require: Set of surfaces σ
  $\phi \leftarrow \emptyset$ {Visibility result mapping for surfaces}
  $\eta \leftarrow \emptyset$ {Orientation result mapping for surfaces}
  for all v such that $v \in \xi$ **do**
    Setup viewpoint v
    Clear depth buffer
    Use depth test function `LESS`
    for all s such that $s \in \sigma$ **do**
      Render surface s
    end for
    Use depth test function `EQUAL`
    for all s such that $s \in \sigma$ **do**
      o = `occlusion-query(`s`)`
      $\phi(s) \leftarrow \phi(s) + o$
      if $\phi(s) > 0$ and s is back-facing from viewpoint v **then**
        $\eta(s) \leftarrow \eta(s) + 1$
      end if
    end for
  end for
  return ϕ, η

on the outcome of the algorithm since such sets tend to be logically (and spatially) coherent.

The implementation uses a generalized visibility query internally as a foundation to generate information needed for the high-level evaluation. The algorithm itself is simply a modified visibility query and the implementation reflects that by only augmenting the results from a visibility query.

The interface to the surface correction tool is shown in Figure 7. This interface contains a few simple related tools as well that were developed in the process. The automatic "flip" uses the surface correction algorithm discussed in this section.

6 Results

6.1 Performance

In terms of running time, both algorithms require little computational overhead apart from rendering the scene due to hardware accelerated occlusion queries. The resolution of the frame buffer affects the outcome of both the reduction and correction algorithms, since both algorithms rely on visibility queries which in turn uses the depth buffer. A low resolution is more likely to produce erroneous or unwanted results. Therefore, running either algorithm using spatially large models in a low resolution will most likely produce less desirable results.

The whole is greater than the sum of its parts.

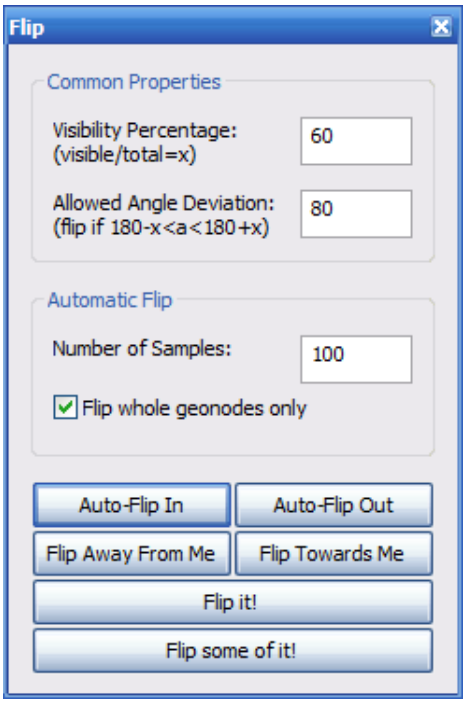

Figure 7: Surface correction tool interface in Spark Vision Genesis. *Visibility percentage* is the surface visibility requirement. *Allowed angle deviation* is the angle difference allowed between the view direction and the surface's average normal. *Number of samples* is the number of view points generated for the automated correction passes. *Flip whole geonodes only* is a platform specific optimization for the algorithm where collections of triangles or triangle strips are corrected instead of individual ones. *Auto-flip in* corrects surfaces in such a way that they face inwards. *Auto-flip out* corrects surfaces in such a way that they face outwards. *Flip away from me* performs a single correction pass and turns visible surfaces away from the viewer. *Flip towards me* performs a single correction pass and turns visible surfaces towards the viewer. The buttons below are convenience tools and not part of the algorithm.

The graphics hardware, monitor and operating system limit the resolution at which the algorithms can be run. There are several feasible solutions to address this resolution limitation:

(i) use large off-screen depth buffer instead of the frame buffer;

(ii) use a tiling approach by rendering subsets of some desired resolution in multiple passes;

(iii) dissect the model or scene and apply the algorithm on smaller parts.

Due to the nature of the input data on which these algorithms are used, given in the assignment specification, the dissecting approach is used as the models are constructed from smaller parts.

6.2 Correction

Qualifying the correction result is hard since it is a subjective matter how coherently the surfaces of a model are oriented. Visual results are presented in Figure 9 in comparison to Figure 8.

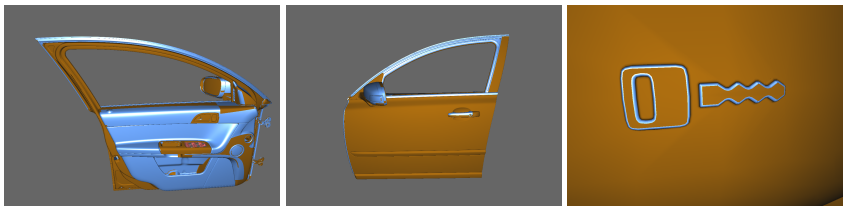

Figure 8: Before surface correction where blue/bright surfaces are front-facing (correct) and brown/dark surfaces are back-facing (incorrect).

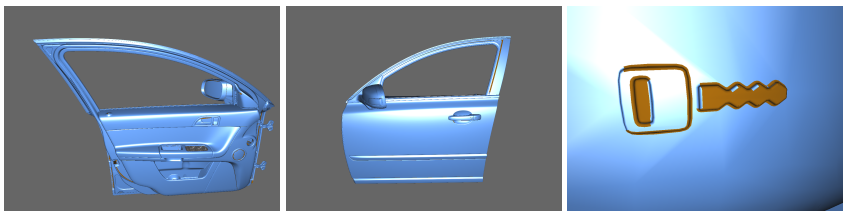

Figure 9: After surface correction where blue/bright surfaces are front-facing (correct) and brown/dark surfaces are back-facing (incorrect).

The correction algorithm relies on visibility and orientation statistics which requires a more sophisticated neighboring or grouping scheme. Neighboring surfaces do not necessarily have the same or even similar orientation. In fact, as seen in the rightmost image in Figure 8 the small surfaces around the icon differ greatly with respect to orientation. A neighboring scheme for this situation must consult topology information in order to perform a proper grouping of neighboring surfaces.

6.3 Reduction

Table 1 shows performance results of the reduction algorithm.

Technique	**Samples**	**Vertices**	**Faces**
Original	N/A	7 693 056	2 315 492
No neighboring	500	1 209 968	429 407
Neighboring	750	1 563 424	523 603
Neighboring	500	1 549 488	517 797
Neighboring	250	1 532 304	511 513
Neighboring	100	1 503 744	500 342

Table 1: Reduction algorithm results.

The algorithm not using any neighboring technique will naturally remove more surfaces, specifically small surfaces which often results in previously mentioned artifacts such as holes and cavities. As expected, more samples will marginally increase the final number of surfaces since the probability of a surface yielding occlusion query samples increases. In practice, this is highly model dependent but generally, a higher number of samples will remove less surfaces and produce a more correct result.

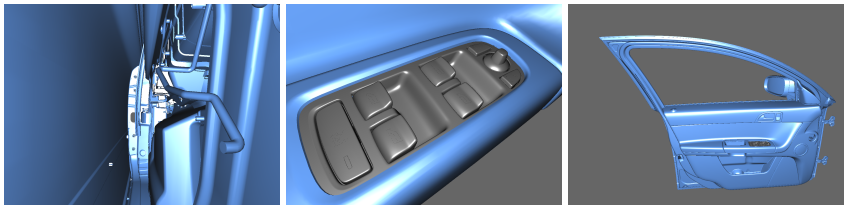

Figure 10: Before surface reduction.

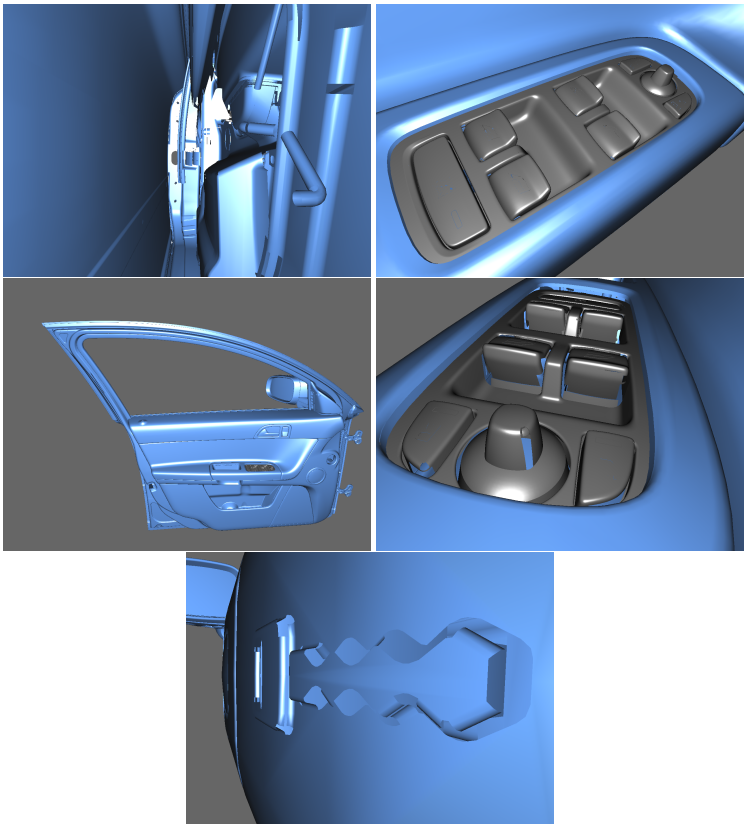

Figure 11: After surface reduction without neighboring.

In terms of running time, the neighboring extension has an insignificant (less than one percent) performance penalty using a visibility map. The map contains grouping information for small surface clusters.

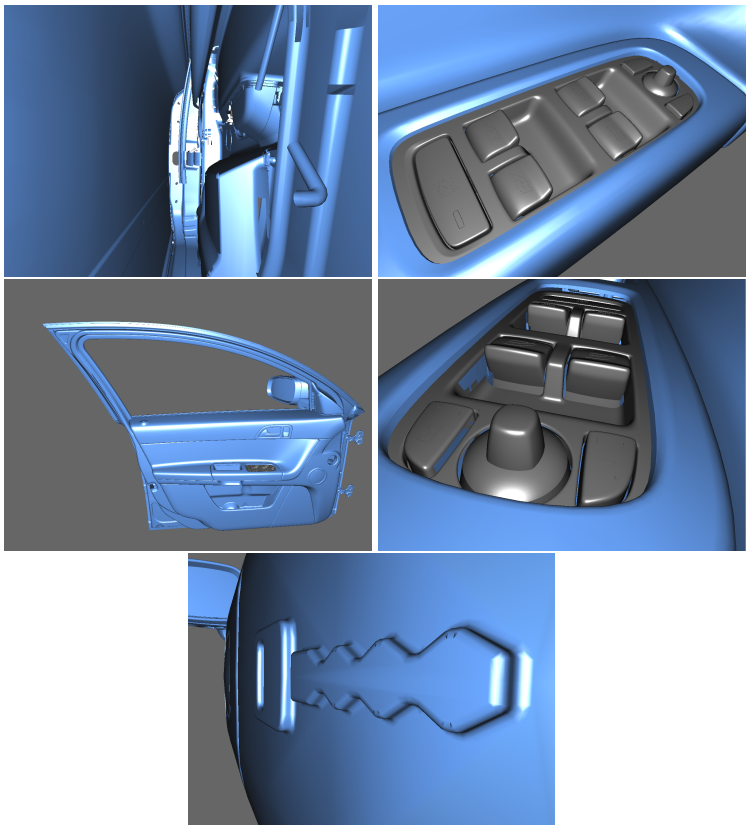

Figure 12: After surface reduction with neighboring.

6.4 Meeting the Demands

In terms of visual demands, both algorithms performs well. Specifically, the reduction algorithm (with the neighboring extension) removes surfaces conservatively in order to prevent unwanted reduction.

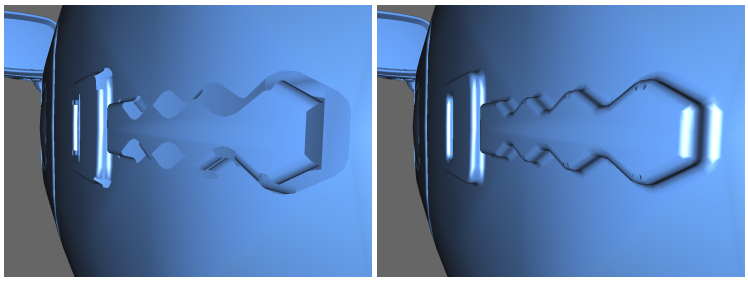

Figure 13: A comparison of (*left image*) reduction without neighboring and (*right image*) reduction with neighboring.

The correction algorithm successfully corrects a vast majority of the surfaces, although

some manual correction is still needed, mostly small or tucked away surfaces. A primitive neighboring was added which improved accuracy on small surfaces, it is however not a general solution and hence not discussed in-depth in this thesis.

Requirements of handling unpredictable data (namely surfaces generated through a tessellation process) is targeted with very generalized algorithms with practically non-existent assumptions. The input, however, is assumed to be meaningful in the sense that it is expected to be a somewhat coherent volume of surfaces. That is, the model construction is such that it has an inside and an outside.

A path and hemisphere extension was added to the reduction algorithm to be used in special cases. The path extensions will generate samples following a given path instead of generating random sample positions on an enclosing sphere around the model. The hemisphere extension will restrict the spherical sample position generator to a hemisphere, which can be used to increase performance on partial models occluded by flat or planar surfaces.

The implementation of the algorithms has almost full parameterization of almost all the steps, allowing the user to fully control the algorithms' performance and tweak their behavior depending on the input.

7 Conclusion

7.1 Discussion

Both implementations of the surface reduction and surface correction algorithms have met the original demands and are used in production. The statistical approach yielded very simple high-performance algorithms that work well in practice. Analytical, as opposed to statistical, methods and algorithms reviewed were focused on surface subdivision rather than reduction. They were soon ruled out as an approach due to the size of the input data.

The work presented in this thesis is aimed at a practical application, as opposed to a purely theoretical (proof-of-concept) approach. Algorithm and tools were developed, tested and improved iteratively. There was no need to integrate the tools as the existing software was used as a development environment. The template mechanisms of C++ allowed a policy based implementation parameterized over element granularity (model, surface, triangle) without any performance penalty. Although the granularity obviously implies memory and rendering penalties. Using the intermediate mode of OpenGL allowed quick prototyping and profiling.

The visibility algorithm was as a natural step in the surface reduction algorithm design process and was originally designed as such. However, the simplicity and versatility of the algorithm enabled it to be used more generally and eventually as the foundation for both the reduction and correction algorithms.

To reduce sensitivity to holes in the visibility algorithm, a primitive *visible percentage* mechanism was implemented to complement the original visibility measure. The potential visibility measure used the occlusion query method without a depth test which

reflected the potential or maximum number of samples a surface could possibly yield. Let τ denote the visibility measure and ω denote the potential visibility measure, then the visible percentage or visible fraction λ can be defined as $\lambda = \tau/\omega$. The algorithms could, for instance, add the requirement $\lambda \geq 0.5$, in other words that at least half of the surface should be visible. However, the visible percentage scheme fails to accommodate surfaces which are intentionally partially hidden such that the visible percentage is always less than the requirement.

7.2 Summary

This thesis has presented two algorithms addressing the problem of surface reduction and surface correction. The focus has been on performance and fidelity. Both algorithms have pros and cons, the statistical approach of the supporting visibility algorithm infers statistical errors. Holes and cavities still present a problem for both algorithms. The conservative visibility test is sensitive to holes which propagates to the governing algorithm. In the surface reduction algorithm, holes tend to classify otherwise hidden surfaces as visible. In the surface correction algorithm, holes tend to cause incorrect orientation following erroneous visibility statistics. For instance, a surface seen through a hole will be corrected if it is back-facing with respect to the current viewpoint.

The statistical visibility algorithm was the foundation for both tools presented and can perhaps be used for others as well. An extension to the reduction algorithm or perhaps a separate tool to solve the problem of duplicate surfaces might be derived from the same principle.

7.3 Future Work

The statistical method used for the surface correction algorithm might be too crude and is, as a result of the orientation evaluation, not eligible for neighboring techniques. The orientation of a surface is not a simple or *boolean* property, it is not subject to neighboring or grouping in the same way as visibility. However, an orientation analysis of neighboring surfaces might work well as a complementary post-processing step to the current surface correction algorithm.

The surface correction algorithm might be modified to use a propagation technique in order to correct neglected surfaces. As shown in images above, perhaps most notably Figure 14, the surface correction algorithm neglects small surfaces as it has no neighboring or grouping technique. By propagating the surface corrections using a post-processing neighboring step, even these surfaces might be fully corrected as well.

The surface reduction and surface correction implementations both assume some sort of meaningful data, albeit scrambled. This implies that the models should have an outside and an inside and are fairly interconnected, as opposed to being constructed from fully disjoint surfaces. This requirement is currently forced by the underlying software platform but should be relaxed or at least optional.

A common and important problem is *double surfaces*, these are two or more surface duplicates, erroneously generated or created during the model tessellation or design. Using the depth buffer method to identify surfaces cannot handle surface duplicates

Figure 14: Incorrect surfaces still present after running the surface correction algorithm.

properly. It might be possible to add a preparation step and remove duplicate surface using a modified version of the reduction algorithm by removing surfaces on a first-served basis.

References

[1] MARK SEGAL, KURT AKELEY, *The OpenGL Graphics System: A Specification (Version 1.5)*, Silicon Graphics Inc, 2003.

[2] K. HILLESLAND, B. SALOMON, A. LASTRA, D. MANOCHA, *Fast and Simple Occlusion Culling using Hardware-Based Depth Queries*, University of North Carolina at Chapel Hill.

[3] D. STANEKER, D. BARTZ, M. MEISSNER, *Improving Occlusion Query Efficiency with Occupancy Maps*, University of Tübingen, Germany, 2003.

[4] WEISSTEIN, ERIC W., *Klein Bottle*, From MathWorld – A Wolfram Web Resource, `http://mathworld.wolfram.com/KleinBottle.html`.

[5] WIKIPEDIA, *Triangle Strip*, From Wikipedia, the free encyclopedia, `http://en.wikipedia.org/wiki/Triangle_strip`.

[6] WIKIPEDIA, *Triangle Fan*, From Wikipedia, the free encyclopedia, `http://en.wikipedia.org/wiki/Triangle_fan`.

[7] DONALD HEARN, M. PAULINE BAKER, *Computer Graphics (C Version)*, Second Edition, International Edition, Prentice-Hall International Inc., University of Illinois.

[8] JOHNNY WIDERLUND, *Increasing Realism in Real-Time Visual Simulations using Hardware Accelerated Progressive Radiosity*, Department of Computer Science, Gothenburg University, 1999.

[9] DENNIS G. ZILL, MICHAEL R. CULLEN, *Advanced Engineering Mathematics*, Second Edition, Jones and Bartlett Publishers, 2000.

[10] MICHAEL T. GOODRICH, ROBERTO TAMASSIA, *Data Structures and Algorithms in Java*, Second Edition, John Wiley & Sons Inc., 2001.

[11] JON KLEINBERG, ÉVA TARDOS, *Algorithm Design*, Cornell University, Pearson Education Inc., 2006.

[12] WIKIPEDIA, *Hidden surface determination*, From Wikipedia, the free encyclopedia, `http://en.wikipedia.org/wiki/Hidden_surface_determination`.

[13] SGI, *ARB Occlusion Query*, OpenGL Extension Registry, `http://oss.sgi.com/projects/ogl-sample/registry/ARB/occlusion_query.txt`.

[14] MASON WOO, JACKIE NEIDER, TOM DAVIS, *OpenGL Programming Guide Second Edition*, Silicon Graphics Inc., 1997.

[15] VELEBA, D., FELKEL, P., *Survey of Errors in Surface Representation and their Detection and Correction*, Czech Technical University, Faculty of Electrical Engineering.

[16] MARTIN CVETANOV MARINOV, *Automatic Generation of Structure-Preserving Models for Computer-Aided Geometric Design*, Aachen, Techn. Hochsch., Diss., 2006.

[17] G. SUSSNER, G. GREINER, S. AUGUSTINIACK, *Interactive examination of surface quality on car bodies*, Computer Graphics Group, University of Erlangen, 2003.

[18] WIKIPEDIA, *Polygon triangulation*, From Wikipedia, the free encyclopedia, `http://en.wikipedia.org/wiki/Polygon_triangulation`.

[19] MARTIN FRANC, VÁCLAV SKALA, *Parallel Triangular Mesh Reduction*, Proceedings of ALGORITHMY, Conference on Scientific Computing, 2000.

[20] STAN MELAX, *A Simple, Fast, and Effective Polygon Reduction Algorithm*, Game Developer Magazine, 1998.

[21] WIKIPEDIA, *Tessellation*, From Wikipedia, the free encyclopedia, `http://en.wikipedia.org/wiki/Tessellation`.