View-dependent Object Simplification and its Application in the case of Billboard Clouds

Elmar Eisemann

Université Joseph Fourier de Grenoble DEA IVR 2004

View-dependent Object Simplification and its Application in the case of Billboard Clouds

Elmar Eisemann

Rapport de projet de DEA

Artis/GRAVIR/IMAG-INRIA. UMR CNRS C5527.

Composition du jury :

Xavier	Décoret	Directeur de stage
Augustin	Lux	
Peter	Sturm	Rapporteur externe

Acknowledgements

I would like to thank Sylvain Lefebvre for his help concerning the pBuffers and 3DSLib and other OpenGL questions.

I would like to thank Gilles Debunne for libQGLViewer. Probably one of the most useful libraries available.

I would like to thank Samuel Hornus for his comments and proof reading and Martin Eisemann for all his comments as well as Stephan Camerer for all the discussions we had.

Special thanks go to Xavier Décoret for his support. I am really lucky to have had him as my advisor. Thank you for all the chats we had, and that you always had time for me, whenever I had had questions. Thanks for your great input and this fantastic opportunity.

Finally I would like to thank all members of INRIA, as well as the guys from E101.

Contents

Chapter 1

Introduction

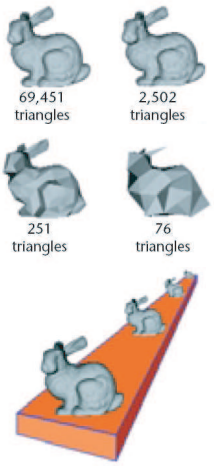
In this Master thesis we will present a new approach to simplify a model representation based on a supplementary knowledge of a region in which the observer is allowed to move, the so-called *view cell*. The simplified representation should be faster to render, without loosing the similarity to the original objects. To assure this we will present our main contribution, a new error bounding method, which to our best knowledge, allows for the first time to restrain the error of a representation for a given view cell. In particular, a lot of common assumptions that were widely accepted are proven to be inexact. We will show several properties for the 3D case and solve a particular case, as well as a numerical solution for points in 3D. For the 2D case we were able to obtain an exact solution which allows our method to be applied on 2.5 dimensional scenes.

Our error bounding method is then used in the context of Billboard Clouds [DDSD03]. Still, our result is more general and thus not at all restricted to this particular usage. The view-dependent Billboard Clouds that we will introduce in this master thesis have several advantages. The error of the representation can be bound and the simplification is very successful; to mention one example a 4480 triangle scene has been simplified to approximately 40 billboards (80 triangles) with a $\approx 5\%$ representation error[1], for a centered view cell inside of the scene with a size approximately 1/10 of the bounding diagonal. Most algorithms add ad-hoc criteria to preserve silhouettes, our algorithm preserves them automatically. Other advantages of our method are that it works for any kind of triangulated input and that it is easy to use; only two parameters are needed (simplification error and texture quality). It is completely independent of the view frustum that is used for the observer, which is not very common, as most image based view-dependent simplification methods need a fixed view frustum. Also our method does not share a common problem with most other view cell approaches, where the representation becomes worse when the observer approaches the border of the view cell. The construction of view-dependent Billboard Clouds is mostly based on the original Billboard Cloud approach [DDSD03], but some slight improvements were made with respect to the original algorithm.

[1] three degrees angular error

Chapter 2

Previous Work



Simplification has a very long history and over the years more and more algorithms have been developed. The main goal is to transform an input representation into an output representation which should be faster to display and/or less complex, but still 'close' to the original, where 'close' has to be interpreted in an intuitive way, as the exact definition may vary from method to method. Error bounds, polygon counts, or even runtime bounds to assure a certain frame rate are possible criteria.

Today lots of different simplification techniques exist and it is almost impossible to give an overview over all of them, but we will describe those which, in our eyes, seem the most innovative and important. In particular we will concentrate on the error/quality measures which are used to ensure a convincing representation.

Simplification becomes more and more important. Objects obtained from a laser scanner often contain much more triangles than actually necessary or wanted. As a laser is only able to examine the surface point-wise, it has to take lots of samples, which are then triangulated to build a surface. This triangulation is probably not optimal, neither in the number of triangles nor in the way the surface is triangulated, leading to a very recent branch of research, *remeshing*, which will be examined shortly towards the end of this chapter.

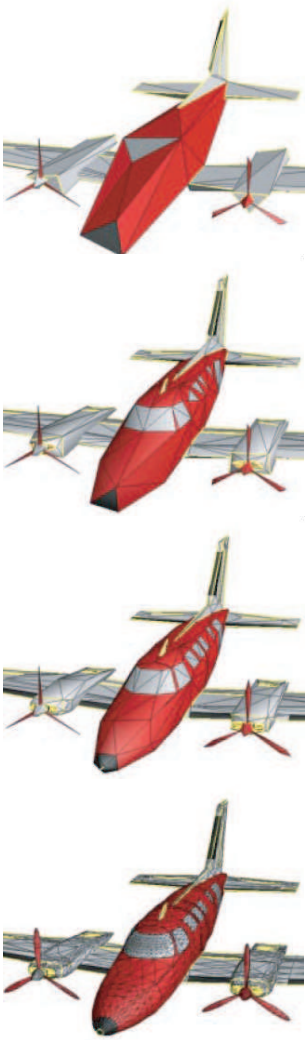
Another important application is the rendering of objects on the screen. If one imagines an object that is very close to the observer and a second one which is farther away, it is intuitively clear that the second one does not have to be as detailed as the closer one. Clark[Cla76] was the first to underline the advantage of using different representations of the same object. This was in 1976 and for a long time lots of video games followed this approach, mostly by using manually created multi-resolution models. Given the fast development of graphic cards today a current trend is to enrich a simple model via special techniques which are well suited for the hardware to obtain a result that looks like a geometrically much more detailed model, but is still fast to render. In particular, texture mapping, bump mapping and recently displacement mapping have proven to be very useful to represent geometry with a lower rendering cost. To accelerate the last id software game Doom 3[Sta04], the developers decided to almost completely rely on texture and bump mapping. The models are created at a very high resolution (approx. 250.000 polygons), then simplified to a reasonable representation (approx. 1500 polygons) and the lost geometry

is put back via bump mapping deduced from the finer detailed models. It is important to mention that texture mapping and bump mapping are *output sensitive*, meaning that the cost is related directly to the output size (in pixels) on the screen, which makes these techniques especially interesting for games. Still these two cannot replace geometry completely; a surface on which both of these techniques are applied will, when seen from the side, still look as 'flat' as the original surface. This problem is partly solved by displacement mapping. Unfortunately it is still very costly to work with displacement mapping, although [WWT+03] showed that it is now possible to achieve real time frame rates on common graphic cards. Nevertheless, all of these techniques are limited to the resolution of the map. Knowing current graphics cards is an important point when talking about simplification because the goal ('faster to render') is indeed directly linked to the hardware. We, too, will exploit graphics hardware to optimize our simplified output.

In this master thesis we deal with a special sub group of simplification algorithms, those which are view-dependent. In the first part we will take a look at algorithms whose output results in triangulated scenes. In the second part we will mention techniques that are based on alternative representations, in particular the so-called *impostor* methods. As mentioned before the main application of simplification remains in optimizing the rendering speed while ensuring a certain quality, but recently even in areas which do not seem to be directly related, view-dependent simplification became important. To mention one example, Cornish et al. [CRL01] used view-dependent simplification for non-photo realistic rendering. The goal was to obtain a more or less uniform sampling in screen space, which is a nice property when placing strokes, therefore they attached samples to the vertices based on a view-dependent simplification of the initial model.

Interestingly most of the algorithms resulting in triangulated objects follow the same framework. A hierarchy of the original representation is created, and during run-time a model is chosen from the hierarchy which satisfies quality criterions based on the current viewpoint.

One classic approach are the progressive meshes introduced by Hoppe [Hop96]. The main idea is to create a sequence of more and more simpler models. At each step an edge of the previous object is removed based on an energy minimization. It is represented as a sum of sub functions taking into account the distance of vertices to the surface, but also a spring energy for each of the edges and energy functions encoding scalar and discrete values like normals or material properties. The result allowed then to transform the models in a smooth continuous way, a so-called *geomorph*. To add an edge, a vertex is split and the edge continuously grown until reaching its original size. The edge collapse could be done continuously, too, by shrinking the edge slowly until the extremities coincide. This approach already allowed for view-dependent simplification. The idea was that for each vertex a query has to be answered whether the vertex should be subdivided. If yes, the corresponding moment in the sequence is found where the vertex was split and it is applied, if the current mesh contains the necessary neighboring vertices. This query approach was later refined in [Hop97b].This simple approach based on the edge lengths had several issues; no exact error bound was provided and the refinement query was expensive to evaluate. Still, it is very important to mention that Hoppe's technique was one of the first to take surface properties into account. A general problem of the method is that

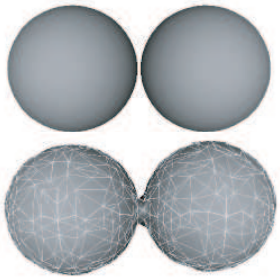


A sequence of edge collapsing operations simplifies the model

it is not capable of changing the genus of a surface, therefore disqualifying the algorithm for extreme simplification.

Xia and Varshney [XV96] developed a view-dependent simplification algorithm based also on edge collapsing. They develop a hierarchical structure the so-called *merge tree*. A merge tree encodes a sequence of edge collapsing operations. At each level in the tree edge collapsing is applied to different regions of influence. A region of influence consists of all triangles incident to a vertex. This ensures that no overlapping/mesh folding occurs, when unfolding the edges during runtime. Starting with the smallest edges, collapsing is performed, until there is no more possible collapse on the current level. Then the algorithm reiterates on the obtained mesh. At each vertex the maximum distance to the child vertices and the distance to the parent vertex is stored. On runtime the subdivision in the merge tree is performed by ensuring that the projected size of these distances obey a given threshold. A global geometric error is not obtained, as the distances are given level by level, the direction of the edges is not taken into account, nor is the error for the faces, as the bound is only established on the edge. Unfortunately the assumption that the projection error of an edge implies a bound on the projection error of a face is wrong as we will show in chapter 6. To overcome some of the problems, more subdivision criteria were added that are not explicitly described in the text, such as laying more importance on the silhouettes and subdivision based on the lighting situation, for example around a specular spot, or where the illumination creates the strongest variation (An approach that had already been used for a long time for global illumination and radiosity. [SP]). A problem, remaining for almost all algorithms based on edge collapsing, is that only a gradual change is permitted from regions of high refinement to regions of low refinement.

A very similar approach had been developed by El-Sana et al. [ESV99] The main contribution was to overcome the problem that the genus of objects remains unchanged. The solution was the introduction of virtual edges, similar to the idea of α-Hulls introduced by the same authors that are used to close holes smaller than a given value. This allows even for the application on non manifold input data. These edges which are not present in the original model are obtained via a Voronoi diagram. Neighboring cells in the Voronoi diagram may be collapsed. Involving a quality measure to prefer equilateral triangles, assuring to introduce no fold over cases and a spline based error measure, they derive what they call a *view-dependence tree*. On which they apply a spline based metric that takes not only the viewpoint distance, but also the light source distance into account. Our approach, too, deals with objects of arbitrary genus.

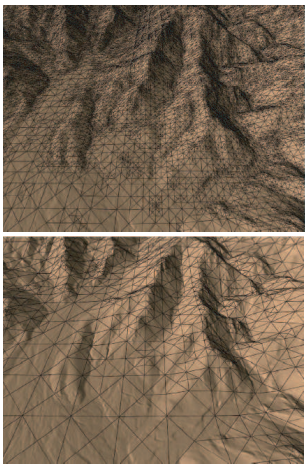


El-Sana et al. introduce virtual edges, allowing vertices to collapse, even when not connected by an edge.

Later El-Sana et al. extended the approach to support visibility information [ESSS01]. A grid encoding opacity/solidity in the cells is precalculated and used to estimate visibility. Two approaches are suggested, the area of the projection of the faces onto the grid cell with respect to the grid cell surface and an estimation using a discrete sample set of rays. The accumulation for an arbitrary viewpoint takes the length of the ray segments inside a grid cell into account. This kind of approaches were already used in radiosity calculations [SP]. Still it is interesting because no algorithm used visibility information before as its exact value is expensive to evaluate.

Another approach we want to mention was presented by Lindstrom et al [LKR+96]. Their algorithm was suited solely for height fields a common rep-

resentation for surfaces. A height field can be seen as a 2 dimensional planar mesh, on which a function is applied, which elevates each of the vertices along the normal direction of the mesh to the corresponding height. The approach is very interesting, as the subdivision takes the orientation of the viewpoint and perspective projection into account. The terrain is represented using a quadtree like structure, a quad can be subdivided into four child quads to obtain a higher resolution. They were able to derive a maximum (vertical) distance between two different levels in the hierarchy. The goal is to measure this distance along the normal direction of the terrain measured in screen space corresponding to the viewpoint. Applying several simplifications/assumptions they obtain an equation whose sign indicates whether or not a subdivision has to be performed given a threshold. Interestingly geometrically this equation results in an object that they call *bialy*, which is a torus without hole, which we will encounter in the discussion of our method, too, although the calculus leading to this result is completely different. In a final step to accelerate the computation, Lindstrom et al. find for each block/quad a smallest value that when projected from a certain point in the block exceeds the threshold and the biggest value which when projected from any point in the block remains underneath the threshold, leading to an *uncertainty* interval. Therefore it is sufficient to compare the actual distance value in such a block with the uncertainty interval to know, whether one surely has to subdivide, or whether no subdivision at all will be necessary. For the remaining vertices the bialy equation has to be evaluated. This leads to an almost conservative algorithm (still the assumptions made before do not allow to pronounce an exact bound, not even in this restricted case of height fields). A nice property is that the algorithm, like ours, is not affected by tessellation. Other than approaches that do not take the relative orientation between observer and surface into account, as most runtime view-dependent optimizations, as the approximation is mostly based on spheres of size corresponding to an edge length value.



Lindstrom et al. perform view-dependent simplification of a height field upper image ≈0.5 lower ≈4 pixels error

Another important article by Hoppe is [Hop97b] where he also exploited the idea of starting with a very simple mesh and refining it on the fly depending on the viewpoint of the observer. As already explained in [Hop96] a refinement query is used, which depending on the viewpoint decides whether parts have to be refined or not. The function to do so still seems intuitive, as no correct bound on the error is obtained. A view frustum test is performed, followed by a back-facing test which is applied on the normal cone created by the normals beneath in the hierarchy. In a final step the screen projection error is approximated. The main test is performed by projecting a sphere of size *Hausdorff distance*[1] between the simplified and original model onto the screen, if the size of the projection remains below a certain threshold, a subdivision does not have to be performed. This test is then improved, by a generalization to an arbitrary surface of [LKR+96]. Although, to ensure validity for curved surfaces a threshold sphere is placed inside of the bialy. The hierarchical structure obtained may take up to several hours of precalculations and although the subdivision is performed on run time, the error bound is not exact. Still the results are very impressive and lead to a speed up of a factor of almost seven. Also for many approaches 'popping' artifacts occur when changing from one representation to

[1]Let M, N be two sets in a metric space, $h(M,N) := max_{m \in M} min_{n \in N} d(m,n)$, where d denotes the distance, the Hausdorff distance $\mathcal{H}$ is then defined as $\mathcal{H}(M,N) := max(h(M,N), h(N,M))$

Hoppe refines the object depending on the view frustum

a second one. This problem was nicely solved using the geomorphing described earlier in [Hop96], only this time the differently refined parts are morphed independently, instead of the whole model. Compared to [XV96] the order in which the edge collapsing is performed is no longer restricted to influence regions, as the problem of overlapping is solved during runtime.

A special view-dependent simplification algorithm for terrains has also been proposed by Hoppe [Hop97a] including several important contributions. The geomorph is based on an estimation of where the observer might move next. Allowing for a very smooth transition with lower cost. He had the insight that for irregular subdivision schemes it is insufficient to test the error of reprojection just at the vertices of the height field and he develops a new formula to find an exact bound. He also describes a data structure that is output sensitive. It is very interesting that the approach still decides the level of refinement during runtime, which implies that an extrapolation of the observer's movement has to be done to make the geomorphs work. Instead of performing such an extrapolation some approaches, especially methods for "impostors" which we will examine more closely later, work with valid representations for a region around the observer.

Coming back to more general approaches it is worth mentioning Luebke et al.'s approach [LE97]. The main motivations for their algorithm are to make as few assumptions about the input model as possible, the algorithm should be completely automatic and adjustable via a fine-grained user interaction to trade off performance and fidelity. All these points are also fulfilled by our algorithm that we are going to present in the chapter 7. The main idea of Luebke et al.'s algorithm is to cluster vertices together. Corresponding to the current viewpoint clusters are refined or regrouped to bigger clusters. Therefore the final representation just contains the vertices that are necessary to fulfil the user specified fidelity for a given viewpoint. It deals with arbitrary scenes not only with single objects. The algorithm works as follows. In each cluster a bounding sphere volume is saved. For all clusters used in the scene this sphere is projected onto the projection plane of the current viewpoint. If the size exceeds a user specified threshold the same test is performed for all the sub cluster, if the size is below the threshold the cluster is collapsed and the algorithm is applied recursively. To optimize the result Luebke et al. also take normals into account, to further refine at silhouettes. As in [XV96] normal cones are used for the test. If any of the normals in the cone is orthogonal to a normal of the view cone from the current viewpoint a subdivision is performed. To optimize the speed several accelerations are used. A special visibility culling is performed working only on the higher clusters, to avoid a complete tree traversal. The algorithm is very effective and more or less easy to implement. The idea of grouping together vertices is rather intuitive, as it is completely independent of the topology. But unfortunately the method has some major drawbacks; the error bound is very simple, due to the fact that it has to be evaluated several times for each frame during execution. The subdivision at silhouettes seems rather ad-hoc and the screen space error is only valid for vertices, which is insufficient, as we will show in chapter 6.

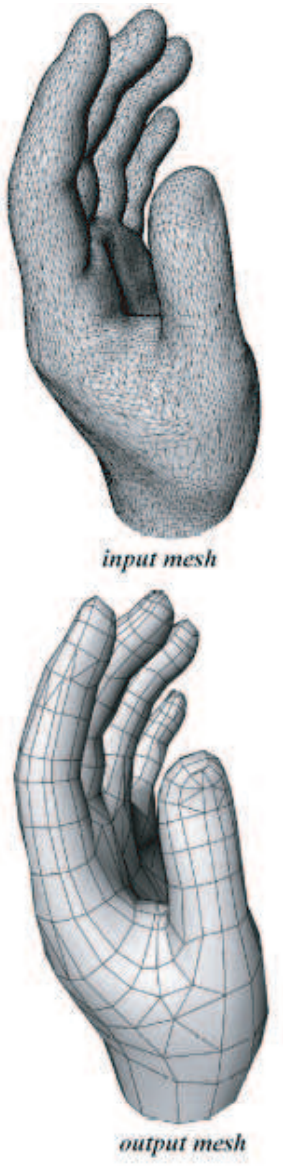
The idea to cluster together vertices is also used in [RCRB], although the algorithm does not provide really new insights it shows that for foliage simplification topology does not play a huge role. Based on several aforementioned algorithms Remolar et al. combine small details (in this case leaves) together

to form simpler shapes in a multi-resolution form. This shows nevertheless an important issue concerning simplification. It is possible to develop special algorithms for very particular objects. In particular for trees a whole lot of work has been developed. As an example, Deussen [Deu03] presented a way to render plants efficiently. When far away, the geometry is simply replaced by points for leafs and lines for trunks. Leading to very convincing results that still run in real time even for bigger scenes. Similarly Meyer and Neyret developed a hierarchical representation in the case of pine trees ([MN00]). In this document we want to focus on general simplifications.

Recently remeshing algorithms have become very popular. Remeshing works as follows. Given a triangulated object, one tries to infer something which could be seen as an underlying surface. This underlying surface is then used to create a new polygonal model, which is approximating this surface. This is very important nowadays, for meshes resulting from a laser scan, as aforementioned. The sampling is mostly uniform and therefore even planar regions will be tesselated. To overcome this problem remeshing algorithms create a mesh whose vertices and edges are placed in a way to be conform with the shape. Without mentioning all work that has been done in this area we still want to point out some important works, as it has interesting impacts on simplification algorithms.

The article [ACSD+03] exploits the local curvature of an object to place vertices. The algorithm works only on objects topologically equivalent to a disc. This is because in a first step the model is parameterized in 2D, a curvature tensor field is estimated and the so-called *umbilics* are extracted (points where the curvature is identical in all directions). From these points min and max curvature lines are developed over the whole surface, spaced according to a user defined precision. These lines create a grid (locally orthogonal) that follows the curvature of the object. The intersection points between min and max curvature lines are then used to define the mesh. This approach exploits very well the local behavior of the surface. An exact error bound of course is impossible to deduce. Still the observation that not only vertex positions, but also edges and their orientation play a huge role in the appearance of an object is very important. Most simplification algorithms work with the Hausdorff distance, which is purely point-based and does not include any information about orientation of edges, we will come back to this point in chapter 3. The observation was not new though ([Sim94]), but usually the solutions were restricted to perform sequences of edge flipping operations, to achieve a local minimum, instead of exploiting curvature directly during a reconstruction step.

In [Fre00] Frey presents not directly an algorithm but a very general framework for simplification/remeshing/refinement. The main idea is to infer from the given triangulation an underlying continuous and smooth surface. For numerical stability, this step involves what Frey calls a *geometric mesh*, which is a simplification of the input mesh based on Hausdorff distance and quality criteria (equilateral triangles are preferred, strong G^1 discontinuities are kept). One cannot suppose that the underlying surface is G^1 as sharp features could be present in the model. The algorithm detects these and marks them. The other edges are assumed to be discrete representations of Bezier patches. In a second step he develops a locally varying metric, which is usually based on the curvature (and the orientation), but could be completely arbitrary. Based on this metric edges are split and collapsed until a unit mesh is obtained. A unit mesh has the property that all edge lengths are included in an interval



Alliez et al. align edges with the principal curvature directions

$[1/\sqrt{2}, \sqrt{2}]$. The introduced vertices are not placed on the original mesh, but on the assumed underlying surface, therefore it is even possible to 'create' more information than what was present in the input. On the other hand edges are simply split and not reoriented. He mentions that edge flipping can be performed, but it is not clear in what way it should be applied. And still it is not sufficient to achieve the best result. Nevertheless the article describes a very nice frame work, and although this is not mentioned in the text, we believe that it should be applicable in a view-dependent context, too. Our bound for reprojection errors (see chapter 5) could be directly transferred into the metric. A unit mesh would then represent a view-dependent simplification.

Turk [Tur92] does not perform remeshing, but his approach is also based on a sampling of the original model. Points are spread on the object based uniformly with respect to area, but with preference of high curvature areas. A relaxation places/moves the points based on repelling forces. Different levels of detail are obtained by adding sample points.

Other important algorithms work on subdivision/spline/implicit surfaces. All these surfaces have the property that they are not triangulated and therefore not limited to a certain resolution. Anyway, arbitrarily precise polygonal models could be deduced. The problem posed in this context is that one would like to infer a triangulated representation which is nicely approximating the underlying surface for a given viewpoint/view cell. For subdivision and spline surfaces it is possible to obtain a triangulated approximation, such that the area of all triangles are smaller than a given threshold, but this would lead to subdivisions on back-facing triangles, as well as to over refinement in some areas that are e.g. very far away from the viewpoint. Therefore the goal is to drive the subdivision based on the observer's positions.

In [CK01] Chugani et al. describe an algorithm to visualize spline surfaces that is interesting, as the triangulation is completely done on the fly. Starting with lists of well chosen sampling points, which locally minimize the error of the spline patch, the whole object is placed in an octree. The octree blocks sizes are used during runtime to decide which of the sampling points are used in the mesh, which is triangulated during runtime. If the sampling was insufficient in a region for the current viewpoint more sampling points are added on the fly. Several optimizations are applied to improve speed and also to prevent 'cracks' that could appear between two spline patches that are sampled differently. The goal of the algorithm was not to have an exact error bound, which in fact is not provided by this method, but to have a sufficiently detailed representation in form of a triangulated surface and to achieve acceptable frame rates. The on-the-fly-triangulation keeps the algorithm from being applicable in situations where high performance is needed. Another problem is that texture information is probably difficult to use in these conditions.

A different approach is [ALSS] by Alliez et al. concerning subdivision surfaces. The algorithm is far from providing realtime performance. The goal is to obtain a refinement that corresponds to the current viewpoint and is still the result of the refinement operator used for the subdivision surface. More precisely, they show their results on surfaces, refined using the $\sqrt{3}$ method. This operator involves several edge flips, which is why a precalculated hierarchical structure cannot be applied. They extract regions of interest, which could be classified as front facing, or silhouette regions of the object. These are potential areas for refinement. They do not precise the method to decide when a refinement has

to occur, their goal was just the general framework, still they point out that projection error, a subdivision limit, curvature and silhouette criteria should be taken into account. This framework, too, could benefit greatly from a correct error bound.

For completeness, concerning iso-surfaces, we only want to mention an article by Gregorski et al. [GDL+02]. Using a tetrahedron based subdivision scheme the iso-surface is approximated via a mesh. The error measure is based on a simple approximation using spheres with a radius corresponding to the current distance of the mesh to the iso-surface.

The last examples show that simplification is not equivalent to 'less triangles'. In general it is a transformation of some input data into an output form which is more convenient. Which leads us to impostor algorithms, which use different possibilities to represent geometry, so-called *alternative representations*.

Billboards, which could be considered as a simple textured quad facing the observer, have been used successfully in computer games for a long time. Lately they have also been used for volume rendering(e.g. [RMD04]). Billboard Clouds [DDSD03] use several textured quads that might interpenetrate each other to approximate a model. As this technique is the base of our approach we will take a closer look at it in chapter 4.

Other well known approaches like Light Field [LH96] and Lumigraph [GGSC96] represent an object indirectly via a plenoptic function. Simply saying, in this case, two planes are used to parameterize rays leaving the eye. Based on the two intersection points on these planes a corresponding color value is taken from a look-up-table.

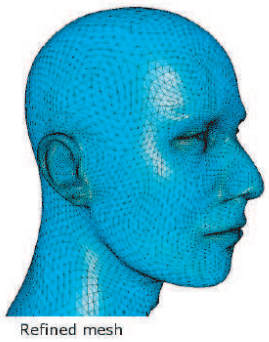
Volume rendering, is usually applied when the input is given in form of an iso-surface. As we will mostly be interested in triangulated inputs we will skip this point.

Finally shaders have become very popular nowadays in particular due to the success of shader languages, like Cg[2], that are now even working on standard graphics hardware in real time. For example, Goldman [Gol97] used a statistical model to deduce a shader to represent fur on animals, which found application in a movie and Neyret et al. [Ney00] used shaders to represent clouds. Shaders have also been applied in Billboard Clouds to perform relighting of the simplified objects.

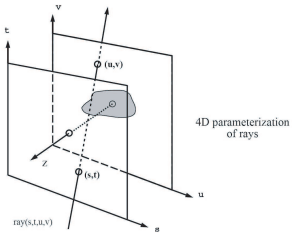
Impostors are themselves a special form of alternative representation.

Sillion et al. introduced the mesh impostor in [SDB97]. The idea is to represent the geometry farther away by one single mesh grid. The algorithm was proposed in the context of city simplification. Therefore the observer is supposed to walk inside of streets, which restricts the movement more or less to a forward/backward translation. For each street an impostor is created which replaces the far away geometry. The mesh is obtained using the z-buffer. An image is shot at the entry point of the view cell, so in this case a segment, the image contains a color and a distance information. Using the distance information (the Z-image) a simple mesh is created. Strong discontinuities in the Z-image are detected and combined to edges of discontinuities. These edges of discontinuities are used together with a constrained Delaunay triangulation to obtain a final mesh grid. On this simple triangle grid the texture is applied. This leads to a very simple model, which still contains the strongest discontinuities

[2] Although that it seems as if Cg is no longer supported by NVidia



Alliez et al. perform surface subdivision based on the viewpoint



Goldman's fur shader

'Rubber sheet' effect due to occlusions

and therefore parallax effects of the original mesh. An error bound is not provided. The most important point is the brilliant idea to rebuild geometry from what is actually seen at a given resolution instead of transforming the geometry, which is a simple still effective procedure. Nevertheless all these approaches will suffer from geometric errors that result from an insufficient sampling resolution. Other problems particular for this approach are that geometry not visible from the sampling point will not appear in the impostor, even if it is visible from other viewpoints. This is particularly problematic for hidden areas. This leads to *skins*, or *rubber sheets* at the borders of the object, as shown in the margin.

An improvement was then suggested in [DSSD99], Décoret et al. propose to subdivide the mesh impostor explained before into several layers. Objects are grouped together in layers based on a parallax measure in 2D. The parallax measure corresponds to the difference between biggest and the smallest angle under which two vertices, one from each object, can be seen from a segment view cell. A graph is created between all the objects that overlap for a viewpoint in the view cell, weighted by their angular distance. Based on this graph objects are grouped together until in each group the sum of all weights from one object to another remains underneath a threshold value. Each of the layers is then created independently. Another important contribution is the way textures are calculated. A very coarse texture is saved for each impostor. A finer version is calculated on-the-fly, while the observer is moving inside of the view cell. Although the approach is very elegant and works well in practice, one problem remains, the fact that the reasoning is vertex based, which does not assure exact bounds on single faces. Moreover, the error created by the transformation to an impostor is not taken into account. Artifacts due to the use of a single viewpoint to create the impostor also remain.

An important approach [JWS02], but also limited by a fixed viewpoint to create the impostors, was presented by Jeschke et al. Their impostor-based approach ensures a correct approximation for a very special case, assuming the view cell is a cube, contained in a near field cube of a certain size, in which the original geometry is used. Around these cubes impostor cubes are placed at distances appropriate to remain below a certain error threshold. Unfortunately the whole calculation concerning the choice of the distances is missing in the paper, and was not publicly available.We show in chapter 6 that, to obtain a correct error bound, there has to be a limit on the minimum size of the near field cube because they predict in their approach of englobing cubes, like many others, that the viewpoints maximizing a reprojection error are always on the extremity of the view cell. We proved this assumption wrong in the general case, and the counterexample shows that it is wrong in this case, too, if a limit size does not exist. Unfortunately the paper does not answer to this question. What they mention briefly is that they bound the error for a very special case, where the observer moves in a straight line from the center of the view cell to a corner, the replaced geometry is assumed to be entirely on a parallel plane behind the impostor, on which the projection is performed. Leading to a correct representation only for a part of the view cell and only for geometry that is actually already planar and aligned with view cell and impostor, which sounds very restricting. Apart of this problem, when assuming the error bound were correct, due to the projection on a cube, faces that are not aligned will lead to a huge error, therefore enforcing the impostors to be very close to each other. This is why in our approach we want to take the orientation/shape of the geometry

Figure 2.1: *A comparison image showing the problems of approaches with a creation based on a fixed viewpoint (TDM) and the incremental textured depth-mesh approach which involves several viewpoints (IDTM). The latter is already very close to results obtained by level of detail methods(LOD), which are much slower, due to the on-the-fly simplification.*

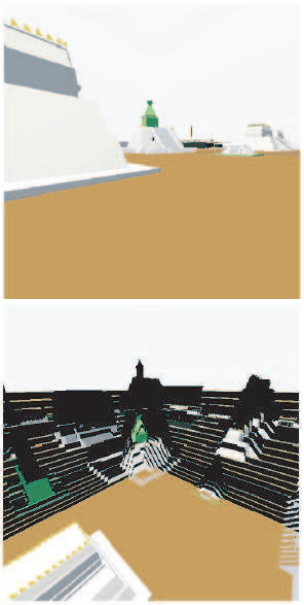
into account too.

A second article [JW02] by Jeschke et al. improves these results slightly. Instead of using planar impostors a coarse geometry is deduced, involving a voxel approximation. The texture is still created from the center point, therefore still leading to skins. The error bound used, is the same as in the aforementioned article.

In general for the impostor representations based on a single point of view the artifacts increase the more one moves away from the view cell center. This leads to severe 'popping' artifacts when changing between one view cell to the other because two different extremes of artifacts are exchanged. Therefore it is more convenient to have a representation that distributes the error uniformly over the view cell. Our method meets this criteria, as well as some more methods, all based on an approach involving several viewpoints during the construction, which also helps to avoid skins.

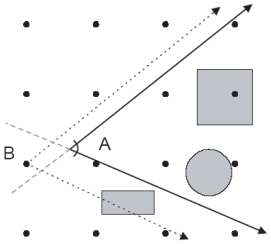
A recent article on impostors was written by Wilson and Manocha [WM03]. Their approach improves Textured Depth Meshes drastically. A comparison can be found in figure 2.1.

They pointed out that the main problem of Textured Depth Meshes are regions, that are invisible from the sampling point. This leads to severe artifacts. Therefore they chose several sampling points inside the view cell. These sampling points are selected based on a heuristic trying to evaluate which sampling point uncovers the maximum invisible region for the other sampling points, the so-called *void regions*. As this problem is very hard to solve, they apply a heuristic. A Voronoi decomposition of the view cell is created and the voronoi points are designated to be potential new sampling points. For these points, optimized via graphics hardware an "objective function" is approximated, which measures the projected extent of the void regions unveiled by the sampling point. Sampling points are added until the objective function remains below a certain threshold. As for other impostors, a triangle grid is deduced from the image, which is shot at a sampling point. This grid is first simplified by a standard simplification and then further on, based on the view-dependent approach by Luebke et al. ([LE97]). To optimize storage, a tree is created, based on the distances of the sampling points from the centroid of the view cell. Having this hierarchy all redundant information, meaning pixels/triangles, which are visible for an ancestor node are deleted from the children. During rendering the algo-



The lower image shows how the impostor is constructed

rithm descents in the tree, until a cost limit is reached (based on triangles, or nodes). Though the algorithm delivers high frame rates and visually pleasing results, no error bound can be given. The approach is completely based on the images shot at the sampling points, not on the geometry itself. Another problem, which remains true for most impostor methods, is the huge storage amount necessary. For the famous power plant model, which contains 13 Million triangles, covering 15 % of the terrain by view cells lead to the amount of 1,4 GB of texture, at a resolution of 512*512. But the authors mention that for a similar quality for the original Textured Depth Mesh approach 9,4 GB would be necessary. Another point interesting to mention is, that a complete exhaustive separation of the power plant in view cells would have needed more than a week of calculation on a Pentium IV processor, although this cannot be seen as a disadvantage, as it is always time consuming to simplify extremely complicated models.



A *layered depth image* approach to perform simplification is described in [AL99]. A layered depth image (LDI) contains in one pixel all the intersections with the scene (see [GwHC] for an overview). The algorithm needs lots of storage space but achieves an almost guaranteed frame rate. Given a fixed number of triangles that are allowed to remain in the scene for a given viewpoint a representation is achieved obeying to this constraint. The main observation is that for a viewpoint A, if its view frustum/viewcone is completely contained inside the one of a viewpoint B, B already sees all the geometry A does. Therefore given sample viewpoints, for which an alternative representation is available and a new viewpoint it is sufficient to look for the closest sample viewpoint which is contained in the inverse view frustum of the new viewpoint, we will refer to this point as the *corresponding sample point*. The representation for the new viewpoint will be based on its corresponding sample point. The choice of sample viewpoints is not easy. First the position of the projection plane plays a role, as for a new viewpoint which is in front of the projection plane of its corresponding sample point geometry behind the observer would appear in front. Ensuring with sufficient sampling points, that this case does not occur the actual simplification is started. At each sample point a new representation of the scene is created in form of an octree with limited depth. There are only a finite number of viewing directions with different octree cells in the view frustum. For all these views a Cost-Benefit function is evaluated, that is measuring the effect, when replacing a contiguous subset of octree cells in the view frustum via layered depth images, taking into account the complexity of the geometry and its screen projection and the distance. Preferring dense, small, far away cells. The algorithm continues, until for all views, the polygonal budget is met. Layered depth images are efficient in the representation, as McMillan pointed out ([GwHC]), that a back to front ordering of the elements (which is necessary, to deal correctly with occlusions) could be highly optimized by treating the image in a special order. Nevertheless LDIs are not amenable to hardware acceleration. One problem of the approach are the extreme amount of storage space necessary (an example was a 2M triangles model, where the constraint was fixed at 250,000 triangles leading to 3.4 GB of LDI's) Another limitation is that the algorithm assumes the observer to move in the (x,y)-plane and rotate only around the z-axis. This is necessary to solve the projection plane problem and probably also because of computation time; the aforementioned example took more than 20 hours to calculate.

All algorithms until now suffer from some artifacts and most view-dependent simplifications assure a quality for a single given viewpoint, as it is simpler to evaluate. Impostor approaches usually do not provide an exact error bound, but the result will still remain reasonably close to the original geometry, as the representation is based on an evaluation of a view. There is also an ambiguity between appearance and geometry, which will be further discussed in E. Where we will also shortly explain another impostor algorithm [WWS01] based on points, encoding a plenoptic function.

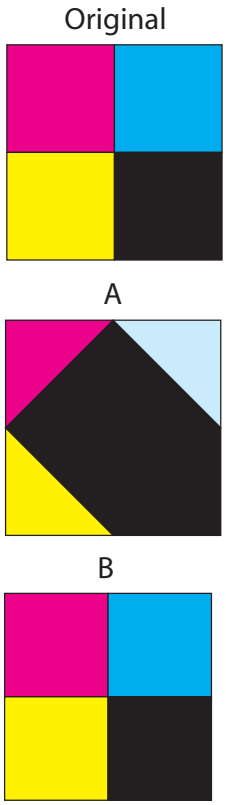
Capturing appearance is a rather new development. Lindstrom et al. [LT00] simplify the mesh via edge collapses and shoot several images of the simplified model to compare them to images of the original model. The algorithm is accelerated by rerendering only parts that are affected by simplification. In particular texture distortion is measured indirectly and optimized for using a geometric heuristic. In our approach a correct bound for all the points on the surface and therefore also the texture is derived. Therefore we treat texture distortion as an inherent part of simplification, which to our best knowledge has not been done before.

Lindstrom et al. use images as the basis for the simplification

A last approach we want to mention because it describes a trend leaving simply geometric distances towards more meaningful distances. Williams et al. [WLC+03] presented an algorithm which takes human perception into account. This is very important, e.g. perception of detail is very weak in high frequency regions, in low frequency regions the introduction of high frequency would result in artifacts that are very evident. We will come back to this point during the future work.

Chapter 3

Motivation



3.1 Properties of the Algorithm

In our algorithm we want to combine the benefits of several approaches. Concerning the input, we want to be able to deal with any triangular input from non-manifold to polygon-soups. Our output should be fast to render and extremely simple.

Silhouettes of objects, which play an important role for the appearance should be represented other than by polygons, as this would be too expensive.

View cell approaches benefit from the fact that no hierarchy has to be established for any potential viewpoint and there is no function to be evaluated at runtime which modifies the representation. On the other hand most view cell approaches do not measure the error on the real geometry, but more often on the alternative representation. We want to use a view cell approach, but establish an error bound with respect to the original geometry. It has to be pointed out that a view cell approach still allows for walkthroughs, as one could define several view cells and switch or blend between the solutions during the transition. The error in representation will still be bounded. We want our error measure not to be based on the Hausdorff distance, as it is a pure shape closeness measure. It is expensive to calculate exactly and therefore almost always approximated. Texture information is not taken into account. Imagining two models $\mathcal{M}_1$ and $\mathcal{M}_2$. If the Hausdorff distance $\mathcal{H}(\mathcal{M}_1, \mathcal{M}_2)$ between $\mathcal{M}_1$ and $\mathcal{M}_2$ equals ε, this assures only that for each point in the original model there is a point in the new model which is at most at a distance ε and vice versa. The image in the margin shows an example of the problem that arises from this approach. The object A will be considered closer to the object *original* as B because B is slightly smaller. Neither texture distortion, nor distance between corresponding vertices plays a role. Our error measure should therefore be defined on a per point basis, meaning that each point of the original model[1] is only allowed to move in a certain region, what we call the *validity region*. Our algorithm should have the property that the distance, as seen by any viewpoint in the view cell, between a point in the validity region and the original mesh point remains below a certain threshold. Parallax effects should be taken into account during simplification. Far away objects should be simplified more than

[1] where point does not necessarily mean vertex

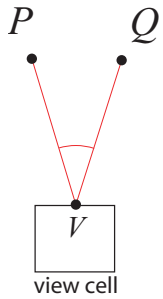
closer ones.

All of the mentioned goals were possible to achieve in one single algorithm, which is described in this master's thesis.

3.2 Angular Distance

We started from the question: What does it mean that two points are close, given that the observer moves within a certain view cell? The choice of a metric like L_2 or L_∞ are unsatisfying choices, as far away distances project to a smaller area on the screen than distances closer to the view cell. Our definition of closeness should take this effect into account. One possibility would be to use a projectional distance, meaning that the distance of the two points projected on the screen should be close. This approach would be perfect, if the projection matrix, including view frustum, that will be used to display the model is known a priori. To be independent of this constraint we decided to define an angular distance. Given a view cell $\mathcal{V}$, the distance between two points P and Q will be defined as:

$$\max_{V \in \mathcal{V}} \widehat{PVQ}$$



where $\widehat{PVQ}$ denotes the smallest positive angle between the three points[2]. Assuming for the moment that the optical center of the projection matrix coincides with the viewpoint. Then this expression means, that if P and Q are at a distance of Θ to each other, when replacing P by Q, the two rays passing through the optical center and one of the points will create an angle smaller or equal to Θ, for any viewpoint in the view cell.

It is possible to bound the distance in the reprojection via the angular distance and vice versa. A 'problem' is that, due to the more general angular expression, points at the boundary of the view frustum with an angular distance Θ will be farther away in the reprojection, than points in the center. This is not really problematic: first of all the angular error could be decreased and secondly human visual perception degrades toward the periphery. Therefore it is even desirable to have a degradation of quality towards the boundary of the view frustum.

Indirectly we also capture the parallax effect of the scene. The parallax effect is the distance between the smallest and the biggest signed angle seen between two mesh points, when the observer moves in the view cell. Let's imagine that two mesh points who have a common point at a distance $< \Theta$. This implies that the angle between the two mesh points is smaller than 2Θ for all viewpoints, therefore the parallax effect between the two will be small. On the other hand, if the angle between two mesh points exceeds for one single viewpoint 2Θ it is impossible to find a point to which the two could be simplified.

Another advantage of the angular approach is that a simplification should be more aggressive for distant objects than for close ones. Therefore a correspondence between distance and level of detail exists.

Unfortunately it is all but trivial to apply the angular distance in practice, but we present a solution in this document.

[2]This expression could also be defined via smallest geodesic distances on a sphere centered at V, but this would be less intuitive and unnecessarily more complicated in notation. But it shows that this definition results in a distance function in the mathematical sense.

Chapter 4

Billboard Clouds

In this chapter we want to give an overview of the technique presented at Siggraph 2003, by Décoret et al. which is the basis for the new algorithm we propose in this document.

The main idea is to simplify 3D models onto a set of planes with texture and transparency maps. Based on a heuristic "optimal" planes are chosen which are more or less tangent to the model. Geometry that is close enough is then projected onto these planes and stored in form of a texture and transparency information.

The goal of the algorithm is to obtain an alternative representation of the input model, such that the L_∞ distance of each point of the model to the corresponding simplified point on the plane remains below a user-defined distance ε. Planes are added in a "greedy fashion" until all faces of the object are simplified. To evaluate the importance of a particular plane $\mathcal{P}$ a *density* $d(\mathcal{P})$ is calculated. The density consists actually of two values:

$$d(\mathcal{P}) = C(\mathcal{P}) - P(\mathcal{P})$$

where C corresponds to a "coverage" value indicating the amount of faces that could be simplified onto this plane. $P(\mathcal{P})$ represents a penalty to favor tangent planes.

A face f is *valid for a plane* if all its points are at a distance inferior to ε. Let $valid_\varepsilon(\mathcal{P})$ denote the set of valid faces for $\mathcal{P}$. Therefore the goal of the

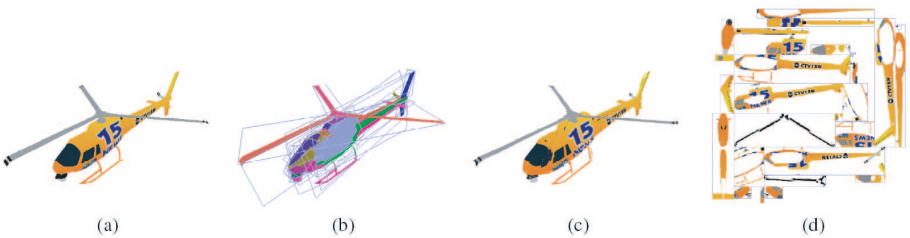


Figure 4.1*: The input model (a) is approximated using planes (b), which are textured with images of the original model, leading to the so-called Billboard Cloud(c), (d) shows how the billboards look like.*

algorithm is to find a minimal set $\{\mathcal{P}_i\}$ such that for all faces f there exist i such that $f \in valid_\varepsilon(\mathcal{P}_i)$, a problem which is NP-hard.

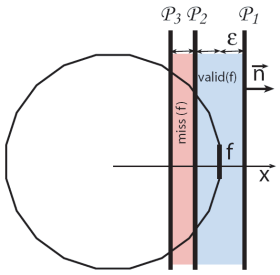
The coverage of a plane $\mathcal{P}$ is calculated as:

$$C(\mathcal{P}) = \sum_{f \in valid_\varepsilon(\mathcal{P})} area_{\mathcal{P}}(f)$$

where $area_{\mathcal{P}}(f)$ denotes the (orthogonally) projected area of the face f onto $\mathcal{P}$. Therefore more weight is put on planes on which a large amount of the models surface could be simplified and which is more or less tangent to the model.

It can be shown that using this approach a sphere would be badly represented, as the first, best plane for a sphere cuts off a small part which cannot be simplified on this plane. To avoid this case and to force the algorithm to start with tangent planes, a penalty term is subtracted from the density. The penalty term is given by:

$$P(\mathcal{P}) := \omega_{penalty} \sum_{f \in miss(\mathcal{P})} area_{\mathcal{P}}(f)$$



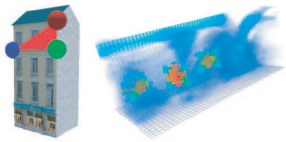
Penalty and valid region for a sphere

where $\omega_{penalty}$ is a constant, usually set to 10 and $miss(\mathcal{P})$ denotes the set of faces that miss the plane $\mathcal{P}$ slightly. More precisely, a plane $\mathcal{P}$ misses a face f, if there exists a parallel plane $\mathcal{P}'$ such that $f \notin valid_\varepsilon(\mathcal{P}))$ and $f \in valid_\varepsilon(\mathcal{P}')$. Furthermore let $\vec{n}$ be the normal of $\mathcal{P}$, then there exist $a \in (0, \varepsilon)$ such that $\mathcal{P}' + a\vec{n} = \mathcal{P}$. The figure in the margin depicts the valid and penalty region at a sphere.

As the problem is NP-hard, the selection of the planes is based on a 'greedy' optimization approach. The space of planes is represented by a discretisation of the *Hough space* [HOU]. In the Hough space a plane is represented by a normal direction $\vec{n}(\phi, \psi) = (\cos(\phi)\cos(\psi), \sin(\phi)\cos(\psi), \sin(\psi))^\top$ and a distance to the origin ρ, the plane resulting equation is $P(x) :=< \vec{n}(\phi, \psi)|x > -rho$. A discretized version of the plane space can therefore be obtained from a discretization of the Hough space. The Hough space is non-uniform and has singularities at the poles, but this does not matter in this context as some areas are simply oversampled. The discretisation results in a set of bins $B_{(\phi,\psi,\rho)}$. For each bin a density is computed, such that a face contributes to a bin if the bin contains a plane for which the face is valid. This can be computed efficiently by looping only on ϕ, ψ. The distance of a point p to a plane at the origin with normal $\vec{n}(\phi, \psi)$ implies the ρ value of the plane passing through p. Therefore the ρ values of valid planes for this point will lie in $[\rho - \varepsilon, \rho + \varepsilon]$. To obtain the valid region for a face f its vertices are tested and the valid region for the face is deduced. Assuming f is a triangle one obtains $[\rho^1_{min}, \rho^1_{max}]$, $[\rho^2_{min}, \rho^2_{max}]$ and $[\rho^3_{min}, \rho^3_{max}]$ as valid regions for the vertices. The valid region for the face is given by $[max(\rho^1_{min}, \rho^2_{min}, \rho^3_{min}), min(\rho^1_{max}, \rho^2_{max}, \rho^3_{max})]$. To obtain a conservative estimate for a bin $B_{(\phi,\psi,rho)}$, the four plane directions given by the borders of the bin results in four ρ-regions for a face $[\rho^{p1}_{min}, \rho^{p1}_{max}]$, $[\rho^{p2}_{min}, \rho^{p2}_{max}]$, $[\rho^{p3}_{min}, \rho^{p3}_{max}]$ and $[\rho^{p4}_{min}, \rho^{p4}_{max}]$. A conservative estimate is therefore given by:

$$[min(\rho^{p1}_{min}, \rho^{p2}_{min}, \rho^{p3}_{min}, \rho^{p4}_{min}), max(\rho^{p1}_{max}, \rho^{p2}_{max}, \rho^{p3}_{max}, \rho^{p4}_{max})]$$

and the face contributes to the bin, if the interval between the lower and upper ρ-value of the bin intersects this ρ-region of the face. Having densities in the



Discretized representation of the density for the house model

bins, planes can be chosen based on the highest density. The density of the simplified faces is then removed from the bins and the algorithm iterates until no more faces are left to simplify.

This naive description would lead to a very high computation time, when choosing a very fine discretization, or a very poor behavior concerning the choice of planes, when choosing a very coarse discretization. Therefore an adaptive refinement has been developed. The bin with the highest density is taken, if all the faces are valid for the plane given by the center of the bin, then this plane is chosen. Otherwise all the faces that contributed to this bin are collected in a set $valid_\varepsilon(B)$. The bin and its 26 neighbors are subdivided and this new density array is filled up with the densities from $valid_\varepsilon(B)$. The algorithm reiterates until an acceptable center plane is found, or a user-specified limit for the subdivisions is reached.

Having specified a set of planes, bounding rectangles in each plane are calculated, which contain the entire geometry that has been simplified onto the plane. A texture is then created by placing an orthogonal camera such that the view frustum corresponds exactly to the rectangle and by placing clipping planes at distances ε. The alpha channel is used to capture the parts, which do not contain geometry.

The billboard is then displayed by using the textured rectangles. The transparent parts are created using alpha blending.

To optimize texture usage a slight modification is done. A problem is that distant objects might end up on the same plane, but therefore leaving lot of transparent space on the billboard. To prevent this behavior, during the subdivision step of the bin, all the faces that contributed are projected onto the center plane. Clustering is then performed to group close faces together, the algorithm continues by only selecting the biggest cluster instead of all faces that remain valid. It has to be pointed out, that faces discarded in this way will be handled in subsequent iterations.

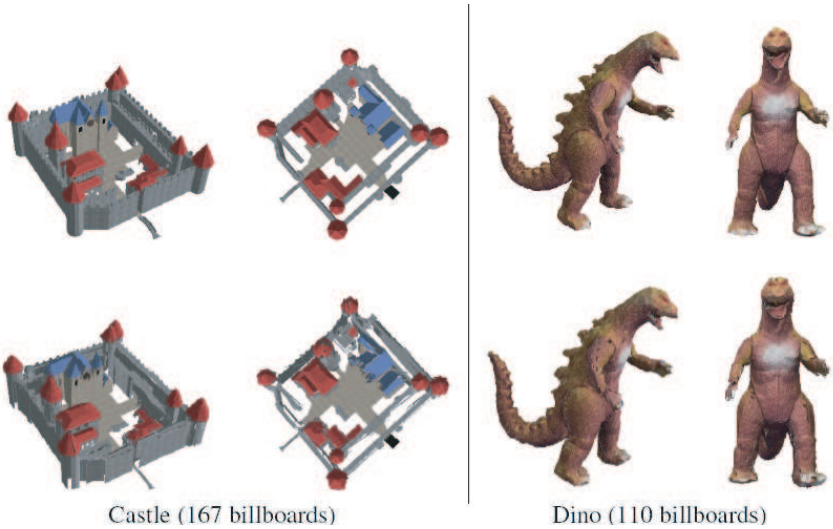


Figure 4.2*: Upper row: originals, lower row: simplifications*

Chapter 5

Deriving an Error Measure for view-dependent Simplification

This chapter deals with the main theoretical results of our work. We are interested in an appropriate simplification of a model given that the observer moves within a given region, the view cell.

Our method consists in defining for each mesh point M on the model a region in space such that for any point P of this region the *reprojection error* will be below a certain threshold. The reprojection error has to be understood as the observed distance of M and P from an arbitrary viewpoint within the view cell.

Starting with the development of formulae for points in space, we will interest ourselves afterwards to the case of faces and we will develop an exact solution for the 2D case.

To our best knowledge, this theoretical result has never been achieved before. Although lots of possible approximations exist (see chapter 2), no exact measure for the reprojection error had been established up to now.

5.1 Mathematical Definition of the Problem

The input of the problem is a mesh $\mathcal{M}$ and a view cell $\mathcal{V}$. The mesh is defined by vertices V_i and faces $\mathcal{F}_j$, and is made of all the *points* on those faces. The view cell is a set of *viewpoints*.

The goal is to simplify $\mathcal{M}$ into another mesh $\mathcal{S}$. A simplification is a mapping of the points of $\mathcal{M}$ to the points of $\mathcal{S}$:

$$\begin{aligned} s : &\mathcal{M} \mapsto \mathcal{S} \\ &M \to s(M) \end{aligned} \tag{5.1}$$

The simplification s is a surjection but not an injection i.e. several points can be "simplified" to the same place.

5.2 Simplification Error Definitions

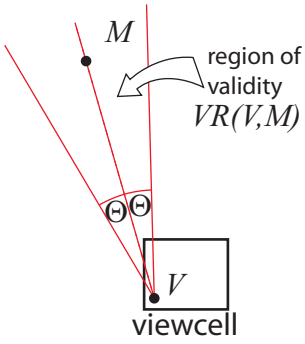
The problem of measuring the error due to the simplification is crucial. A classical approach is to measure the distance between $\mathcal{M}$ and $\mathcal{S} = s(\mathcal{M})$ using the Hausdorff distance for example. As explained before, these approaches only consider the geometric difference between the mesh and its simplification. If every point on the mesh has a color (e.g. via texturing), two meshes can have the same shape (i.e. a null Hausdorff distance) but look very different. For that reason, we want to measure the extent to which each individual point on the mesh is allowed to "move" during the simplification. In other words we want the distance between M and $s(M)$ to be bounded by a certain metric for all points on the mesh.

Definition. ***Reprojection error***
We define the *reprojection error* at a viewpoint V of a point M simplified to $S = s(M)$ as the angle under which the segment $[MS]$ is seen from V.

$$re(V, M) := \widehat{MVS} \tag{5.2}$$

The reprojection error for the view cell is simply the maximum reprojection error when the viewpoint moves in the view cell :

$$re(M) := \max_{V \in \mathcal{V}}(re(V, M)) \tag{5.3}$$



We decided to measure an angular error, as it allows to remain independent of the observers view frustum, which might be unknown by the time of simplification. Nevertheless, having fixed a view frustum it is possible to bound the error via the angle and vice versa. (compare 3.2)

Definition. ***Validity region***
For an error bound[1] Θ, we define the *validity region* of a point M for viewpoint V as the cone of apex V, aperture 2Θ and opened in the direction of M and whose axes pass through M. The validity region for the view cell is the intersection of all these cones :

$$VR_\Theta(M) := \bigcap_{V \in \mathcal{V}} VR_\Theta(V, M) \tag{5.4}$$

Clearly, simplifying/moving a point in its validity region yields a reprojection error less than Θ, that is :

$$s(M) \in VR_\Theta(M) \implies re(M) \leq \Theta \tag{5.5}$$

The shape of the validity regions is complex even in 2D. Figure 5.1 shows some examples. Notice in particular that it is not necessarily bounded.

To extend this idea to the whole mesh, we will define the following:

Definition. ***Valid simplification***
For an error bound Θ, a simplification is said to be *valid* for a set $\mathcal{F}$ of points of the mesh (e.g. a face of the mesh, or the mesh itself) if :

$$\forall M \in \mathcal{F}\ s(M) \in VR_\Theta(M) \tag{5.6}$$

that is; no simplified point will reproject "farther" than Θ from where the original point projects.

[1] For convenience we will assume that $\Theta < \frac{\pi}{2}$. For practical applications this will be absolutely sufficient, as it corresponds to a field of view of 180◦. Mathematically this avoids to treat redundant cases.

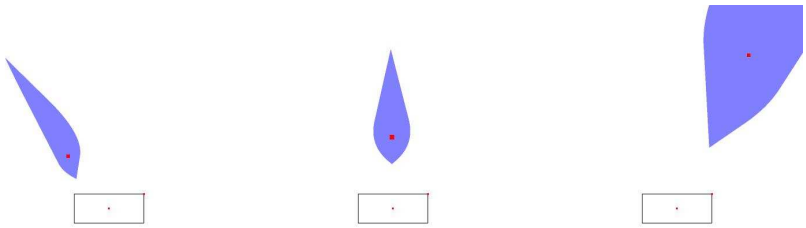


Figure 5.1: *Example of validity regions in 2D for a rectangle. Images are computed by sampling the view cell and performing the intersection of a finite number of cones.*

5.3 Properties of the Validity Region

From the definition we can deduce several properties.

Remark. The intersection of validity regions of different view cells corresponds to the validity region of the union of these view cells, this is why in the future we will assume connectivity of the view cell.

Lemma 5.1. *The validity region of a point M is convex and star shaped*[2] *around M.*

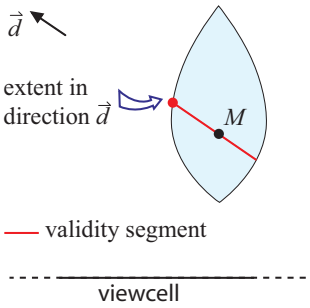
Proof. So let's consider the validity region around a point M. As an intersection of convex sets, a cone is convex, the validity region will be convex. Convexity then implies star shape, which is in particular true for the point M. □

This property implies that it is sufficient to know the extent of the validity region for all directions to reconstruct it properly.

This leads to the following definition:

Definition. ***Validity Segment, Extent of the Validity Segment***

The *validity segment* $VR^{\vec{d}}_{\Theta}(M)$, given a point M, a view cell, a threshold angle Θ and a direction $\vec{d}$ is the intersection of the line passing through M in direction $\vec{d}$ with $VR_{\Theta}(M)$. The *extent of the validity segment/region in a direction* $\vec{d}$ will be the intersection of the ray starting in M in direction $\vec{d}$ with the boundary of the validity region. The *extents of a validity segment* $[E_1, E_2]$ are simply E_1 and E_2.



We see, that the extents in direction $\vec{d}$ and $-\vec{d}$ are sufficient to calculate the validity segment and vice versa. In the following we will therefore focus on one fixed direction $\vec{d}$.

In a first step we want to consider a view cell containing a single viewpoint. From what we have just seen, it is possible to parameterize the validity region via a spherical parametrization. Knowing this we can restrain ourselves to find a solution for a given direction $\vec{d}$. The proof of the theorem could be skipped

[2] A set S is called star shaped, if it contains a particular point C, such that for all points $P \in S$ the segment $[C, P] \in S$

during a first reading, as the resulting formula is not crucial for the further understanding.

Theorem: 5.1. *The extents of the validity segment for a point M in a given direction $\vec{d}$ and view cell containing* ***one*** *viewpoint V are given by $E_i = M + \alpha_i \vec{d}$ $(i = 1, 2)$, where*

$$\alpha_{1,2} = ||M - V|| \frac{-\sin^2\Theta < \frac{M-V}{||M-V||}|\vec{d} > \pm \sin\Theta\cos\Theta\sqrt{||\vec{d}||^2 - < \frac{M-V}{||M-V||}|\vec{d} >^2}}{< \frac{M-V}{||M-V||}|\vec{d} >^2 - \cos^2\Theta||\vec{d}||^2}$$

in the case where the signs of the two alpha differ. Otherwise the alpha with the smaller absolute value is valid and the other should be set to $\pm\infty$ respectively.

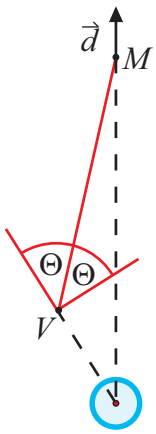
Proof. The two points $E1, E2$ we are looking for lie on the line passing through M in direction $\vec{d}$ and are leading to an angle of Θ between $E_i - V$ and $M - V$, where $i = 1, 2$.

We therefore obtain the equation:

$$\frac{| < M - V|(M + \alpha\vec{d}) - V > |}{||M - V||||(M + \alpha\vec{d}) - V||} = \cos\Theta$$

As Θ is supposed to be smaller than $\frac{\pi}{2}$ squaring this equation leads to the same set of solutions. Straightforward transformations lead to a quadratic expression:

$$A(B + C\alpha) + D\alpha^2 = 0$$



where $A = \sin^2\Theta||M - V||^2$, $B = ||M - V||^2$, $C = 2 < M - V|\vec{d} >$ and $D =< M - V|V >^2 - cos^2\Theta||M - V||^2||\vec{d}||^2$

There is only one solution to this equation when $\frac{<M-V|\vec{d}>^2}{||M-V||^2||\vec{d}||^2} = \cos^2\Theta$, the interpretation is that if the angle between the direction $\vec{d}$ and the viewing direction towards M creates an angle Θ, one side of the cone defining the validity region will become parallel to the direction $\vec{d}$, therefore there exists no intersection, which means, that even if we place the approximation of M at infinity along $\vec{d}$, the projection will still be close to the projection of the original point M. More precisely we see that the intersection, to be in the direction of $\vec{d}$, the angle between $\vec{d}$ and $V - M$ has to be smaller than Θ. By construction we would expect two different values, one positive, the other one negative. If this is not the case, we have a "false" intersection on the "wrong" side of the view frustum as depicted on the image in the margin. But as we supposed that $\Theta \in (0, \frac{\pi}{2})$ a false intersection can only happen if the two alpha values obtained share the same sign and more importantly the one with the bigger absolute value will correspond to the 'wrong' intersection, which makes the calculation a lot easier. If one does not accept this distinction, a formula for one of each rays could be derived, what will actually be necessary in section 6, but we decided to stick to this much more efficient proof. In the end of this section this particular problem will also be further clarified.

We will now focus on the case where we have two solutions:

$$\begin{aligned}\alpha_{1,2} &= -\frac{||M-V||^2 \sin^2\Theta < M-V|\vec{d}>}{< M-V|\vec{d}>^2 - \cos\Theta||M-V||^2||\vec{d}||^2} \\ &\quad \pm\sqrt{\frac{||M-V||^4 \sin^4\Theta < M-V|\vec{d}>^2}{(< M-V|\vec{d}>^2 - \cos^2\Theta||M-V||^2||\vec{d}||^2)^2} - \frac{\sin^2\Theta||M-V||^4}{< M-V|\vec{d}>^2 - \cos^2\Theta||M-V||^2||\vec{d}||^2}} \\ &= -\frac{||M-V||^2 \sin^2\Theta < M-V|\vec{d}>}{< M-V|\vec{d}>^2 - \cos^2\Theta||M-V||^2||\vec{d}||^2} \\ &\quad \pm||M-V|||\sin\Theta|\cos\Theta\frac{\sqrt{||M-V||^2||\vec{d}||^2 - < M-V|\vec{d}>^2}}{|< M-V|\vec{d}>^2 - \cos^2\Theta||M-V||^2||\vec{d}||^2|}\end{aligned}$$

Exploiting once more the fact that $\Theta < \frac{\pi}{2}$ we obtain:

$$\alpha_{1,2} = ||M-V||^2\frac{-\sin^2\Theta < M-V|\vec{d}> \pm \sin\Theta\cos\Theta\sqrt{||M-V||^2||\vec{d}||^2 - < M-V|\vec{d}>^2}}{< M-V|\vec{d}>^2 - \cos^2\Theta||M-V||^2||\vec{d}||^2}$$

Due to the Cauchy-Schwartz inequality we see that there is always at least one solution to the equation, showing that we treated all cases possible.

□

Remark. From the theorem 5.1 it is possible to calculate the resulting extent for an arbitrary mesh point. Interestingly the resulting equation (theorem 5.1) leads to a ruled surface. Implying that only the relative angle ($< \frac{M-V}{||M-V||}|\vec{d}>$) and the distance ($||M-V||$) have an influence on the result, where the distance has a linear influence on the extent. These results correspond exactly to the intuition.

In real-life a human observer encounter this effect regularly. The farther an object is away, the more difficult it becomes to estimate its actual size. This becomes more evident when looking at far away mountains. Although the heights might differ by several hundred meters it is very hard to estimate the exact difference without a reference point.

Lemma 5.2. *Given a point M a view cell and a direction $\vec{d}$ the validity segment of M will be unbounded if and only if the cone with apex M and aperture 2Θ opened in direction $-\vec{d}$ does contain all points of the view cell. (figure 5.2)*

Proof. Following the remark in theorem 5.1 we know that the validity region is bounded if and only if the cosine of the angle between $\vec{d}$ and $P-V$ is smaller than $\cos\Theta$. This is equivalent to the construction of the cone. □

5.4 Iso Values

We have seen in the last section how to calculate the validity region given that the view cell only contains one point.

In this section we will be interested in understanding which viewpoints in space are actually responsible for the extents of a validity region. That is to say given a point of the mesh and a simplification point we would like to determine the viewpoints for which those two points are at an angular distance of Θ.

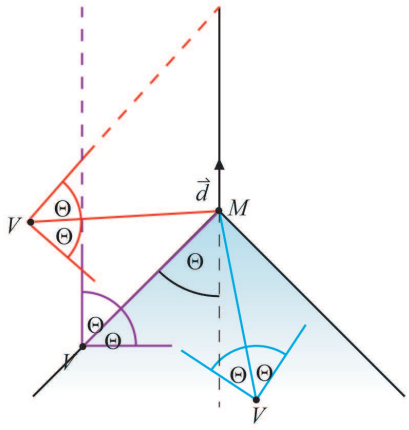
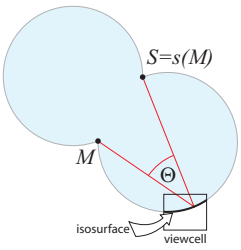


Figure 5.2: *The viewcones of the viewpoints inside the cone will not intersect with the ray from M in direction $\vec{d}$, therefore the validity segment will be unbounded. For viewpoints on the cone we have the special case where the viewcone aligns with the ray. For viewpoints outside we will have an intersection and therefore a limited validity segment.*



Definition. ***Iso-points, iso-viewpoints***
For a point M and a simplification $S = s(M)$, we call *iso-points* Θ the points in space that see $[MS]$ under the angle Θ, and *iso-viewpoints* Θ the viewpoints that sees $[MS]$ under the angle Θ.

In the 2 dimensional case, the set of these iso-viewpoints can be described geometrically. From the Thales theorem, and remembering that $\Theta < \pi/2$, these iso-viewpoints are on the intersection of an eight shape and the view cell. Furthermore the theorem implies that points inside the eight shape would see M and S in with an angle superior and outside inferior to Θ.

In the 3 dimensional case this set quite similar. Due to the following lemma:

Lemma 5.3. *Two points M and S all viewpoints which lie on a circle orthogonal to the direction $M - S$ with its center on the line passing through M in the direction $M - S$ share the same angle of view for [MS].*

Proof. The formulation of this problem is actually more difficult than the proof. The problem is depicted in figure 5.3. Imagining that we fix a viewpoint V, which is rotated around the axis described by M and S, the angle under which the segment [MS] is seen remains the same.

Mathematically the proof can be performed using the formula we found before. It is clear that the observation is also valid if we apply a rigid transformations. Therefore we assume $\vec{d} = (0, 0, 1)^\top$, $M = (0, 0, 0)^\top$ and V is replaced by $-V$ leading to the much simpler form:

$$sin\Theta||V||^2\left(\frac{-\sin\Theta < V|\vec{d} > +\cos\Theta\sqrt{||V||^2 - < V|\vec{d} >^2}}{< V|\vec{d} >^2 - \cos^2\Theta||V||^2}\right)$$

If one now expresses V in coordinates conform to the rotational invariance we

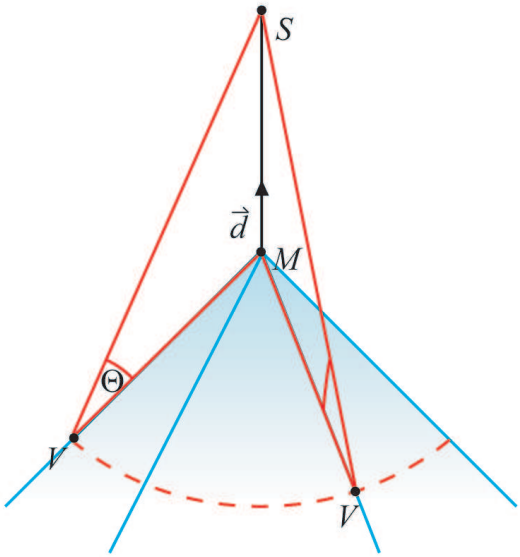


Figure 5.3*: The angle under which two points are seen does not change if the viewpoint is rotated around the segment created by these points.*

want to prove, $V(\phi, r, z) = (r\cos\phi, r\sin\phi, z)^\top$, we obtain:

$$\sin\Theta(r^2+z^2)\frac{\cos\Theta r - \sin\Theta z}{z^2 - \cos\Theta(r^2+z^2)}$$

which is independent of ϕ □

Unfortunately when looking at the formula in the proof of lemma 5.3, one sees that a (still) complicated relationship remains between the two other parameters of the parametrization, which makes the problem very difficult in three dimensions.

Still it is possible to describe the iso-viewpoints in 3D geometrically. Knowing that we have a rotational symmetry for the iso values we can place ourselves back into R^2. This is done by intersecting with a plane passing through the point M on the mesh and a point S which represents the simplification (see figure 5.4). We know that on this plane the iso-points correspond to an eight-shape. Rotating this plane around the axis defined by M and S we get an idea of the 3D shape. We see that we obtain a torus without hole in the middle. This shape, called *bialy*, was obtained by Lindstrom et al. [LKR+96] (page 4) to measure the error, but with a completely different approach. We will refer to this set further on as the *iso-torus*, even in the 2D case, as the cut of the iso-torus by a plane results in the aforementioned eight-shape (figure 5.4).

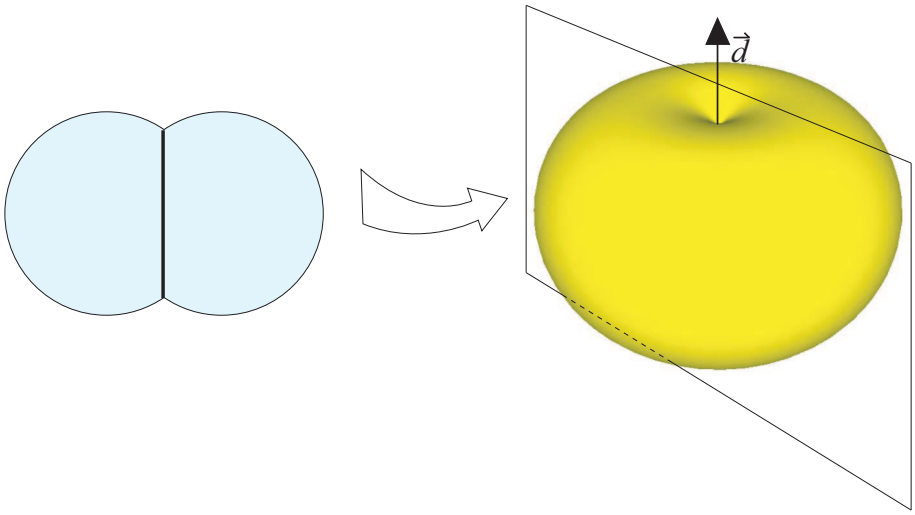
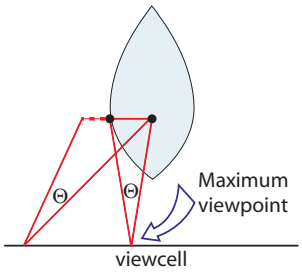


Figure 5.4: *As the 3D shape is rotationally invariant along the axis M and $\vec{d}$ it is sufficient to understand the 2D case to get an idea of the 3D shape. The 2D shape is rotated around the aforementioned axis.*

5.5 Maximum Viewpoints



Definition. ***Maximum viewpoints***
Let M be a point and $\vec{d}$ a direction. We call a *maximum viewpoint* a viewpoint V for which the validity region border of M in direction $\vec{d}$ coincides with a point on the border of the view cone of V. The view cone is the cone with apex V, aperture 2Θ and the axis is given by $M - V$. In other words as $VR_\Theta(M)$ is constructed via the intersection of all view cones. For each given direction there is at least one viewpoint whose cone limits the validity region in that direction, these are exactly the maximum viewpoints.

To assure the existence, from now on, we will assume that our view cell is a closed set. For 'real life' applications this is not a restriction. Most of the times view cells will be defined by polygonal borders. Having a closed set we know from the definition of the validity region, that for a given direction $\vec{d}$ there exists a point the *maximum viewpoint* in the view cell whose cone created the border of the view cell in this direction.

Remark. Existence of maximum viewpoints: Usually for the existence one would need compactness, but we will see that closeness is sufficient. Objects become smaller the farther the observer moves away, so when the distance to the mesh point becomes very large, the angle under which two points are seen will approach zero. Given a sequence of viewpoints restricting the validity region more and more until the validity region corresponds to the validity region for the whole view cell, the sequence is necessarily bounded. Therefore we have a sequence in a bounded and closed set, which in R^n is equivalent to compactness. The theorem of choice gives us a converging subsequence. The limit of this subsequence has to be a maximum viewpoint. It has to be pointed out, that the original sequence does not necessarily converge. Given as view cell the (x,y)-plane we are looking for the maximum viewpoints of $M = (0, 0, 1)$ and

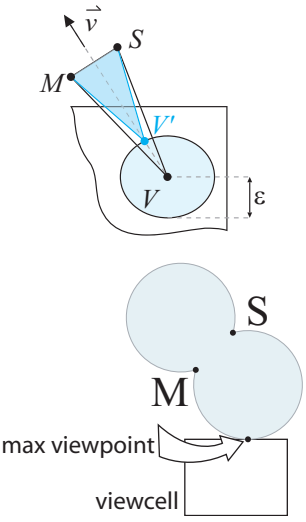
$S = 0, 0, 2$. Due to the rotational symmetry, if (x,y,0) is a maximum viewpoint, so is $(\cos(\phi)x - \sin(\phi)y, \sin(\phi)x + \cos(\phi)y)$ for an arbitrary ϕ. This exceptional case, where $M - S$ is orthogonal to the view cell is actually the only case, as we will see later.

Now we can prove a very important result:

Lemma 5.4. *For a volume view[a] cell the maximum viewpoints are necessarily on the boundary.*

[a] In 2, resp. 3 dimensions a 2, resp. 3 dimensional viewcell

Proof. Intuitively we will show that a close viewpoint to the point M restricts the validity region more than a viewpoint farther away. Meaning that for a viewpoint that is very close to the object the simplification has to be less aggressive than for a far away viewpoint. Given a direction $\vec{d}$, we call M the point on the mesh and S the extent of the validity region in direction $\vec{d}$. Assuming we have a maximum viewpoint V in the interior. By definition of the interior of a set, we can find a sphere $B_\varepsilon(V)$ of radius ε around V which lies completely inside the view cell. Let's call $\vec{v}$ the direction of the bisector of $\widehat{MVS}$ and define $V' = V + \varepsilon/2\vec{v}$[3]. Clearly the angle $\widehat{MV'S}$ is greater than $\widehat{MVS} = \Theta$ which is a contradiction, as it implies that shrinking the angle back to Θ one could not cover the whole segment $[MS]$ and therefore S cannot lie on the border of $VR_\Theta(M)$, leading to a contradiction. □



This proof implies, that for volume view cells only the boundary is important. This does not imply that the maximum viewpoints are always "extremities", which is falsely assumed in several articles. (see chapter 2). The image to the right shows a counter example. Following the false assumption that the error is maximized at the extremities, leads to a validity region which is a polyhedron. We will see later that the validity region is more properly described using hyperboles.

5.6 Finding Maximum Viewpoints for a Hyperplane View Cell

From the previous sections we have seen that it is sufficient to know the border of the validity region for a given direction due to its star shape (Lemma 5.1). Then we showed how to calculate this validity segment, if the view cell contains no more than a single point (Theorem 5.1). Afterwards we were interested in the shape of iso-points which correspond to points in space which create the same validity segment for a given direction. By using the Thales theorem we got a good insight in the shape of this set (page 23 - 25). In the next step we have introduced the notion of a maximum viewpoint (5.5), those viewpoints which actually restrict the validity region, and we have shown that these points are always on the view cell's boundary (lemma 5.4).

Having all this information we are now able to calculate the validity region of a point given a hyperplane view cell. We will show how to determine the maximum viewpoints and derive a geometric interpretation of iso-viewpoints.

[3] $V' = V + \varepsilon/2\frac{V-M}{||V-M||}$

As aforementioned we will fix a direction $\vec{d}$ for which we will determine the extent of the validity region, this means we will detect the extent only in the direction $\vec{d}$, not in the direction $-\vec{d}$.

To ease the comprehension we will restrain ourselves to the 2D case first, before solving the problem in 3D. As mentioned before it is sufficient to find the maximum viewpoint for a given direction $\vec{d}$. To achieve this goal the intuition is to grow the iso-torus of iso-points until it becomes tangent to the view cell (see figure 5.5).

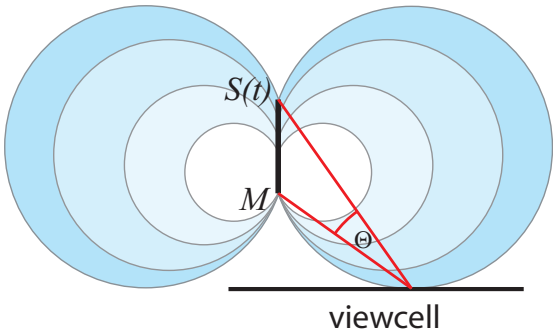
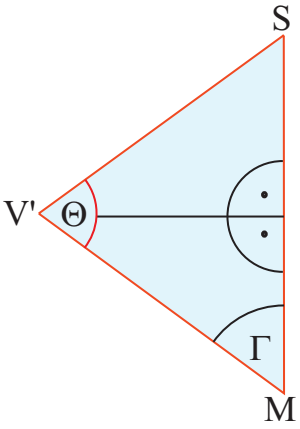


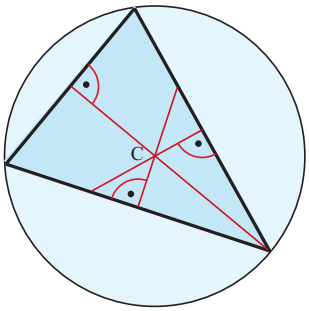
Figure 5.5: *The idea to find the maximum viewpoint is to grow the iso-torus until it becomes tangent to the view cell. Due to the Thales theorem the tangent point is the one we are looking for.*



To get started we want to parameterize the iso-torus using a parameterized simplification point $S(\alpha) = M + \alpha\vec{d}$. Let's first assume S has been chosen. The first observation is that in 2D the iso-torus correspond to the union of two circular parts therefore we are first of all interested in creating each of those circles independently for each side. To find the corresponding iso-values we will start by constructing a third point V', such that the angle $\widehat{MV'S}$ equals Θ. This will be useful, to construct a circle through these three points[4]. Once again the Thales theorem tells us, that all the points which create an angle Θ with M and S and lie on the same side as V' will be found on this circle. The point V' is given by the intersection of the mediator and the line passing via M in direction R_Γ, where R_Γ is a rotation matrix of angle Γ. The angle Γ equals $\frac{\pi-\Theta}{2}$ or $2\pi - \frac{\pi-\Theta}{2} \sim -\frac{\pi-\Theta}{2}$, depending on which side of the eight-shape we are interested in. The angles come from a simple geometric reasoning; the sum of the angles inside of a triangle equals π). The situation is depicted in the margin. If one assumes that the middle point of the segment $[MS]$ has coordinates $M + t\vec{d}$ we obtain the following line equations: $l_1(\alpha) := M + \alpha(R_\Gamma(\vec{d}))$ and $l_2(\beta) := (M + t\vec{d}) + \beta\vec{d}^{\perp}$. Those lead to the intersection point $V' = M + t\vec{d} + \tan\Gamma\vec{d}^{\perp}$. One realizes, that the result is always valid except for $\Gamma = \pm\frac{\pi}{2}$, which implies that $\Theta = 0$ or $\Theta = \frac{\pi}{2}$ both of these cases had been excluded. More precisely the result for the concrete situation as depicted before would be $V' = M + t\vec{d} + \cot(\frac{\Theta}{2})\vec{d}^{\perp}$ and $V' = M + t\vec{d} - \cot(\frac{\Theta}{2})\vec{d}^{\perp}$ for the 'mirrored' one.

Now having one parameterized point we can construct a parameterized circle. To construct the circle we use basic geometry, as depicted in the margin the

[4] slightly more efficient would be to use the fact that the angle for the circle's center with M and S is of 2Θ, but we preferred this more intuitive approach.



intersection of the mediators gives the surrounding circle's center $C(t)$. The result is given by $C(t) = M + t\vec{d} - t\cot(2\Gamma)\vec{d}^{\perp} = M + t(\vec{d} \pm \cot(\Theta)\vec{d}^{\perp})$ and exists under the same conditions as in the step before[5] Having the center one can deduce the radius using the distance of the center to any of the other points of the triangle. Involving several trigonometric transformations the radius is given via $r(t) = \frac{t}{2cos\Gamma|\sin\Gamma|} = \frac{t}{\sin\Theta}$ in both cases the radius is the same, which can be explained geometrically, as one case is the mirrored of the other.

Having a parameterized circle, we now apply the idea of growing it until it touches the view cell. To further simplify the problem we will assume for the moment that the view cell is actually a hyperplane and without loss of generality we consider it to be $y = 0$. We assume for the moment, that the viewpoint that is given by the tangent circle is actually the maximum viewpoint we are looking for. The problem is, that the Thales theorem predicts an eight shape, in 3D the iso-torus, so when growing a circle and calculating the intersection with the view cell, we might encounter an intersection with the part of the circle which would have been excluded by the Thales theorem. We will show in chapter Aan exact classification of when these viewpoints are actual maximum viewpoints. To ease the understanding we will restrict ourselves to look for a potential maximum viewpoint.

Starting to grow the circle, we now would have to distinguish two cases, the one where the point M lies in the half space given by $y \geq 0$ and the one where M lies in the half space given by $y \leq 0$. Both cases are solved in the same way, therefore we will concentrate on the case for M in the half space $y \geq 0$ and announce simply the result for the other half space. For each circle the point with the smallest y value on the circle will be described by $P(t) = C(t) + r(t)(0,-1)^{\top}$

To find the tangent point, we only have to solve a linear equation given by:

$$C(t) + r(t)(0,-1)^{\top} = (x,0)^{\top} \tag{5.7}$$

Resulting in

$$t = \frac{\sin\Theta m_2}{\pm\cos\Theta d_1 - sin\Theta d_2 + 1} \tag{5.8}$$

where $\vec{d} = (d_1, d_2)^{\top}$ and $M = (m_1, m_2)^{\top}$. $\pm$ corresponds to the two different sides. To see for which values this equation is undefined we have to take a closer look at the denominator. As $||\vec{d}|| = 1$ we can write $\vec{d} = (\cos\gamma, \sin\gamma)^{\top}$. The denominator thus simplifies in the first case to $\cos(\Theta + \gamma) + 1$ which is the case for $\gamma = \pi - \Theta$ and without a surprise in the other case we obtain $\gamma = \Theta$. 'Without a surprise', as it corresponds to the 'mirrored case' and the angle is mirrored at the y-axis (the normal of the view cell). The situation is depicted in the figure 5.6.

This case implies that even if the real view cell were the whole hyperplane, it would not be possible for any viewpoint to see the segment $[MS]$ from this direction. On the other hand we have proven in this step, that there is always one side, from where the segment can actually be seen.

In the case where M is in the half space $y \geq 0$, we would have to replace

[5] To distinguish the two cases, one might remember that for the case in direction of $\vec{d}^{\perp}$ the coefficient in front has to be positive.

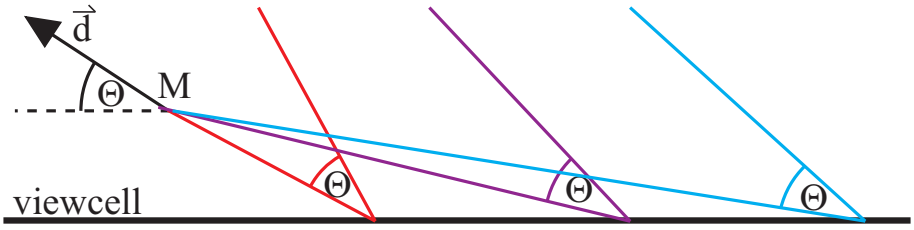


Figure 5.6: *The image shows the case where we are looking for a maximum viewpoint which is situated to the right of the ray defined by M and $\vec{d}$. One sees that in the situation where the angle of $\vec{d}$ exceeds $\pi - \Theta$ (measured at the x-axis) there cannot be any maximum viewpoint, as for all viewpoints in the half plane we do not have an intersection with the ray. Actually for the point at infinity we would encounter the case where the view cone aligns with the direction $\vec{d}$.*

$(0,-1)^\top$ by $(0,1)^\top$ in equation 5.7. Leading to almost the same equation

$$t = \frac{\sin\Theta m_2}{\pm\cos\Theta d_1 - sin\Theta d_2 - 1}$$

Remark. In practice we will never need this second equation, as it is sufficient for each face of the view cell to consider only the points which lie in 'front' of the face[6]. The proof is analog to lemma 5.4 and will therefore be skipped[7].

When considering these equations we might end up with a negative result for t, which means that there is no maximum viewpoint for M, S and the direction v on the considered side of the segment $[MS]$. This can be seen because the numerator of t is greater than zero. Therefore the sign change can only result from the denominator. We have seen at which moment the denominator changes its sign; it is the moment when there are no more maximum viewpoints for this side of the segment $[MS]$.

It is now very interesting that a very beautiful parametrization of the extent of the validity border can be achieved, which is not very useful in practice, but very nice from a theoretical point of view.

Theorem: 5.2. *The validity region of a point for a plane view cell for a given direction $\vec{d}$ is the intersection of two hyperboles.*

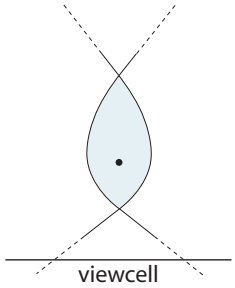
Proof. Looking at equation 5.8 one could imagine to observe the variation of t (which corresponds to the extent of the validity region) in dependance of the direction $\vec{d}$. The resulting equation is

$$t((\cos\gamma, \sin\gamma)) = \frac{\sin\Theta m_2}{\pm cos(\Theta \pm \gamma) - 1}$$

This is a very well known equation. It is a parametrization of a hyperbole. □

[6] In the case that the view cell just consists of one view cell face, it would have to be considered twice, once for each orientation.

[7] Short proof: The idea is to assume that it is not the case, then the connecting segment between the point to simplify and maximum viewpoint intersects the view cell. This is a contradiction, as all viewpoints on this segment will restrict the validity region more than the 'maximum viewpoint'.



This result reveals, that the assumption of having a polyhedron (which would be the case if the maximum error is obtained on the extremities of the view cell) is false. This result is also interesting as it shows once again, that a polygonal sampling of the validity region would remain in the interior of the validity region due to the curvature of hyperboles, allowing for a conservative estimate in the case of hyperplane view cells.

The result for the a view cell line will now be transferred to the 3D case, where the view cell is a plane. The following lemma will show that it is possible to treat the 3D case having solved the 2D case.

Lemma 5.5. *Given a direction $\vec{d}$, a mesh point M and a view cell which is a plane, the maximum viewpoint can be found on the line l, which is the orthogonal projection of the line passing through M in direction $\vec{d}$ onto the view cell. If $\vec{d}$ is orthogonal to the plane, the maximum viewpoints, are all situated on a circle, around the orthographic projection of M onto the view cell. (The situation is depicted in figure 5.7)*

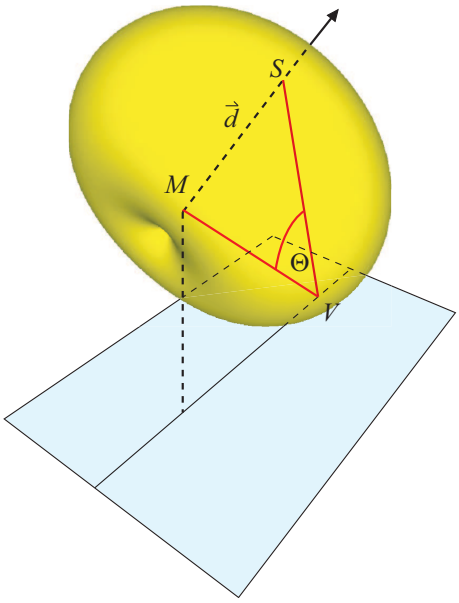


Figure 5.7*: The maximum viewpoints for a plane are restricted to the orthographic projection of the line passing through M in direction $\vec{d}$ onto the view cell plane. The maximum viewpoint in this case is indicated with a V.*

Proof. We have shown that the iso-points are situated on an object which is rotationally invariant around the axis A defined by M and $\vec{d}$ (see lemma 5.3) and looks like an eight shape when intersected with a plane containing the line through M in direction $\vec{d}$. We also know, that the maximum viewpoint V corresponds to a tangent point at the view cell of an isotorus containing V. We have seen that the validity region of M will not change if we replace the plane view cell via the half space, bounded by the plane and not containing M (see lemma 5.4). If V is not on l and $\vec{d}$ is not orthogonal to the view cell, by construction of l a rotation of V around the aforementioned axis A

would penetrate the interior of the half space view cell. This means that there would also be a maximum viewpoint in the interior, which is impossible due to lemma 5.4. In the case where $\vec{d}$ is orthogonal to the view cell a maximum viewpoint rotated around the axis would result in a circle on the plane. □

Due to the lemma above it is now sufficient to restrain the 3D case to the 2D case by working in the plane passing through M and containing the directions $\vec{d}$ and the normal of the view cell plane.

5.7 Finding the Maximum Viewpoint for a Polygonal View Cell

Maximum viewpoints are always situated on the border of volume view cells. Therefore it is possible to consider only the faces of the view cell. Following the implies that

In this section we will use this result to solve the case of a polygonal view cell (not necessarily bounded)[8]. As mentioned before it is sufficient to restrict ourselves to the faces creating the view cell[9](remark on page 21 and lemma 5.4). The idea in this part, is to use the maximum viewpoints approach, to define a finite set of viewpoints, that have to be considered.

Remark. Given a point M, a direction $\vec{d}$ and a polygonal view cell $\mathcal{V}$, which is contained in a hyperplane $\mathcal{H}$. If the maximum viewpoint for $\mathcal{H}$ is inside $\mathcal{V}$ then the extent in direction $\vec{d}$ for $\mathcal{V}$ equals to the one of $\mathcal{H}$.

The difference will thus be in the cases where the maximum viewpoint for the hyperplane falls outside of the polygonal view cell.

It is actually the polygonal shape of the view cell which makes the whole problem of bounding the error difficult. We will encounter this problem again in the chapter 6. The validity region will be calculated indirectly via the classification of maximum viewpoints. The idea is to find always the viewpoint which minimizes the extent of the validity region in a given direction. As usual we will solve the problem in the 2D case before explaining how to deal with the 3D case.

The advantage of the indirect computation of the validity region lies also in the fact, that it is sufficient for a given direction to determine a finite set of viewpoints which will be needed to determine the validity segment. If some of those viewpoints are useless or falsely classified as maximum viewpoints the result will still be coherent, as by definition the validity region is given by the intersection of ALL view cones. As on page 28 we will consider both sides of the eight shaped isotorus separately. We will then determine 'maximum viewpoints' for each side and the validity region will be given by the intersection of the cones of this finite set. Abusing the notation we will also refer to these 'pseudo' maximum viewpoints as maximum viewpoints, as they represent a maximum viewpoint for one side. Figure 5.8 shows this situation where a maximum viewpoint for one side does not correspond to the 'real' maximum viewpoint.

[8] more precisely a view cell given by the finite union of finite intersections of closed half-spaces

[9] in 2D a segment, in 3D a planar polygon (not necessarily bounded)

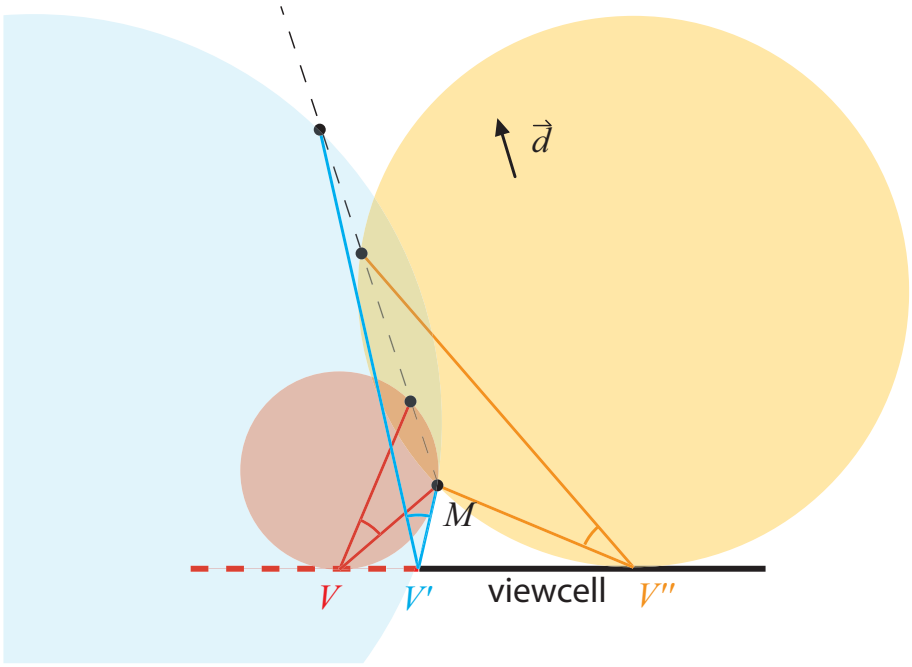


Figure 5.8: *Although the maximum viewpoint for the hyperplane view cell lies to the left V, the maximum viewpoint for the view cell lies to the right V'', as can be seen when compared to the maximum viewpoint to the left V'.*

To obtain the maximum viewpoint of a polygonal view cell we will first go back to the case where the view cell is a hyperplane. We have seen, how the 3D case can be solved by reducing the problem to the 2D case (see lemma 5.5).

In the following we determined the validity segment for a hyperplane view cell. During the calculus we have not exploited all the information. The equation given by the first line of equation 5.7 will allow us to calculate the coordinates of the maximum viewpoint by inserting the result of equation 5.8 in equation 5.7. Due to the assumptions we made ($M = (m_1, m_2)^\top$ is in the halfspace $y > 0$, the view cell is given by $y = 0$) mathematical transformations lead to the point:

$$V_{max} = (m_1 + m_2 * \frac{\sin\Theta d_1 \pm \cos\Theta d_2}{\pm\cos\Theta d_1 - \sin\Theta d_2 + 1}, 0) \tag{5.9}$$

If this point falls inside the polygonal view cell it represents a potential maximum viewpoint. If it falls outside we have to determine where the actual maximum viewpoint is located. As before we have seen that we can grow the isotorus until it becomes tangent to the view cell. The following lemma will show, that in this case the maximum viewpoint will be on the view cell's extremities.

Lemma 5.6. *If the maximum viewpoint for a hyperplane view cell containing the original view cell does not fall into the interior of the original view cell it has to be on the extremities*[10]

[10] In 2 dimensions, extremities correspond to the extremal points of a segment view cell, and in 3 dimensions to the bounding segments.

Proof. The proof is not complicated, and easier to understand when following it on the figure 5.9.

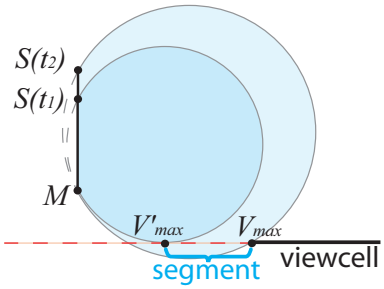


Figure 5.9*: The maximum viewpoint for a polygonal view cell is on the border, if the corresponding maximum view cell for the hyperplane falls outside of the polygonal view cell.*

To show this we assume that the maximum viewpoint V'_{max} for the hyperplane falls outside. Therefore we know that there is an isotorus passing through this point and being tangent to the hyperplane. Due to its construction via the Thales theorem, we know that smaller isotorus' are contained in the bigger ones (The Thales theorem states that points inside the circle form a bigger, outside a smaller angle). Now we assume that we have grown the isotorus until the concerned side of the eight shape comes into touch with the view cell at V_{max}. So segment $[V'_{max}, V_{max}]$ has to lie in the interior of the isotorus. If V_{max} is not the boundary of the view cell, there has to be a point V''_{max} which lies on the segment $[V'_{max}, V_{max}]$ and is closer to V'_{max}. The isotorus passing through V''_{max} would be included in the isotorus passing through V_{max} as V_{max} was chosen to be minimal we get $V''_{max} = V_{max}$ proving that V_{max} is on the border. □

In the 2D case this is sufficient to find the complete solution to our problem because the extremities of a segment are two points. So if, as mentioned in the lemma 5.6, the maximum viewpoint falls outside of the segment, the closer border is the maximum viewpoint for the segment view cell.

Unfortunately this is insufficient to solve the 3D case because then boundaries correspond to segments, leaving us still with an infinity of possible maximum viewpoints to test. For more details and a numerical solution one could refer to chapter B.

Chapter 6

Validity Regions and Projections

In the last chapter a new approach was presented to calculate the exact validity region of a point. Unfortunately the validity region of vertices cannot simply be used to ensure a validity for faces. In other words, there is no direct relationship between the validity region of vertices of a face and the validity region of the face itself.

Let's consider two points M and N of the mesh and pick valid simplifications for these points $S \in VR_\Theta(M)$ and $T \in VR_\Theta(N)$. Then the segment $[ST]$ is not a valid simplification for $[MN]$. This can be seen on the counter-example of Figure 6.1.

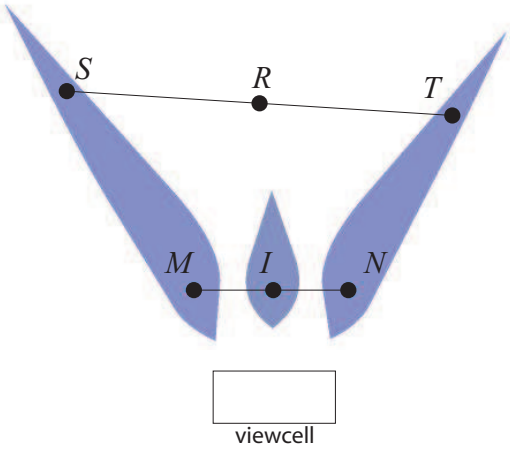


Figure 6.1*: Validity regions cannot be linearly interpolated. When M and N move within their validity regions, their middle point does not remain necessarily in its validity region.*

In particular, in a Billboard Cloud like approach, if the projection (orthog-

onal or perspective) of M and N on a plane are valid simplifications for M and N, the projection of $[MN]$ is not necessarily valid.

Still our results concerning validity regions for points are not useless, as a face could be sampled. The error that is created is bounded in relation to the distance of the sampling points. A sampling scheme, e.g. Turk in [Tur92] and [Tur01], could be used to offer an approximation of the validity for a face.

As we were more interested in a correct approximation with exact error bound we further explored the validity of faces. In the following we will see that it is a very difficult task, we will show that the polygonal character of the view cell is the actual problem. For a hyperplane view cell the validity at vertices can be used to ensure validity of the whole face, making our 3D solution work for this restricted case. Finally we will provide a complete exhaustive solution in the 2D case which has applications not only in the context of Billboard Clouds (for which we developed this approach) but also for other simplification algorithms.

6.1 Ensuring Validity of Points

As described in chapter 4 the simplification approach used by billboard clouds is that elements are projected onto a plane. This projection corresponds to the movement of several points along the plane normal. To ensure the validity for view-dependent Billboard Clouds, it is necessary to ensure that the plane intersects all the validity segments in direction of the plane normal of all points that are supposed to be projected onto the plane. It is insufficient that the plane just passes via the validity regions, the way the points are transferred/projected onto the plane influences the validity. If the projection were to be a projective projec-

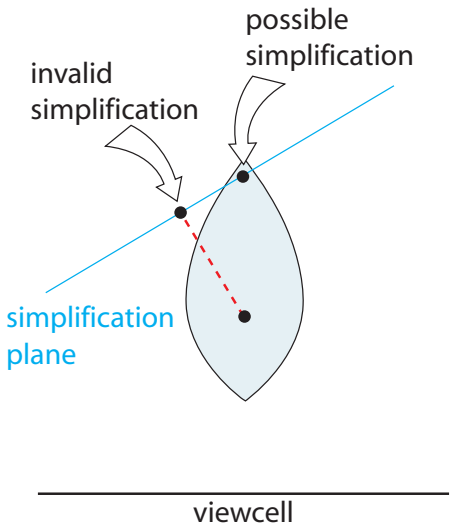


Figure 6.2*: Although the simplification plane passes via the validity region of the point M, an orthogonal projection will not necessarily lead to a valid simplification. All that one can deduce is that there exists a projection direction for which it would be a valid simplification.*

tion for a fixed viewpoint, then the projection direction would change depending

on the position. As the validity region is not spherical this would involve another step of complexity to the problem. But as we will show in section D, it is not convenient to use a fixed projection point. Having different points to shoot the texture from might lead to another problem. Big faces might be more likely to show distortion problems, when a perspective projection is used. Therefore the distortion might already create an error superior the specified threshold Θ for certain viewpoints. To be able to simplify the model, those big faces would have to be cut in smaller ones, for which different projection centers are used, leading to possibly higher complexity than the original model and possibly disturbing texture discrepancies at the border of such patches.

These reasonings lead us to the decision to use orthographic projections, nevertheless even for this case it is not a simple task.

Our goal is to ensure validity of a face given a plane, which is defined as follows:

Definition. ***validity for a (simplification) plane, projection direction***
A face $\mathcal{F}$ is called *valid for a plane* $\mathcal{P}$ (under an orthographic projection), or simply *valid*, if the validity segment for each point on the face given in direction of the normal of $\mathcal{P}$ is intersected by $\mathcal{P}$. To distinct this kind of plane we will refer to it in the following as the *simplification plane*. In the same sense a simplification plane will be called *valid* for a face, if the face is valid for the simplification plane.

This is equivalent to the formulation: The simplification mapping $\mathcal{S}_{|\mathcal{F}}$ defined by the orthogonal projection onto $\mathcal{P}$ is a valid simplification for the face $\mathcal{F}$.

We will refer to the normal of the simplification plane as the *projection direction*. (We will speak of a simplification 'plane', and mesh 'face' also in the 2 dimensional context)

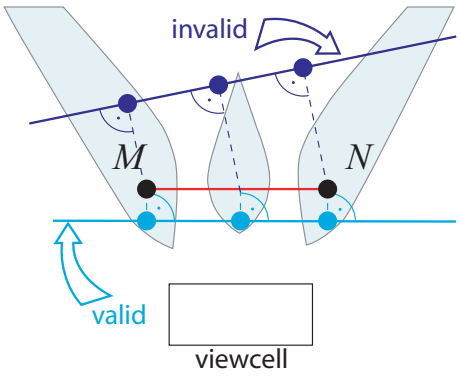
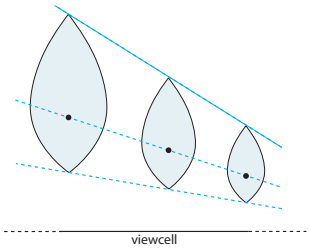


Figure 6.3*: The upper plane represents an invalid simplification plane for its projection direction, although it passes through the validity regions of all points. The lower plane represents a valid simplification plane for its projection direction.*

A first interesting observation can be made in a particular case. In section 5.6 we have seen how to calculate the validity region given that the view cell is a hyperplane. The obtained equation (5.8) for the extent of the validity region in



direction (d_1, d_2) was :

$$t = \frac{\sin \Theta m_2}{\pm \cos \Theta d_1 - sin\Theta d_2 + 1} \tag{6.1}$$

where t corresponded to the extent of the validity region, the point $M = (m_1, m_2)^\top$ to the concerned mesh point and $\vec{d} = (d1, d2)^\top$ to the direction and the view cell was assumed to be $y = 0$. The important observation is that the extent depends linearly on the distance of the point M to the hyperplane view cell, leading to the following lemma.

Lemma 6.1. *In the case of a hyperplane view cell a valid simplification plane for the vertices will also be a valid simplification plane for the face.*

Proof. For the 3D case it is sufficient to assume that the face is a segment. If the lemma were wrong for faces, but correct for segments, there would be a point on the face which is not valid. It cannot lie on the border, as for segments the lemma is true. But then it cannot lie in the interior, as it is sufficient to apply the lemma for a segment passing through this point, with extremities on the border of the face, which are proven to be valid. To show the result for a segment we will show that the linearity expressed in equation 6.1 remains true in the 3D case. We know from lemma 5.5 that in the case of a hyperplane view cell, and a single mesh point it is sufficient to solve the 2D case for a plane containing the mesh point, the normal of the view cell and the projection direction. If the segment is already contained in this plane, we are done. Otherwise, the plane is translated along the segment, therefore the intersection between plane and segment varies linearly in the plane. The sum of two linear functions (change of the extent of the validity region and the change of distance along the segment) remains linear, therefore we also have linearity in the 3D case. A simplification plane passing through the validity segment in the projection direction at the vertices will, due to the linearity, remain in the validity region for all the points on the segment. □

The lemma 6.1 ensures a solution for a special subcase of our problem. The general situation is more complicated, therefore for the remaining of this chapter we will consider only the 2D case. A 3D solution remains future work, but is very complicated and will possibly remain an open problem for a longer time. The approach we present will heavily depend on the possibility to calculate maximum viewpoints exactly. Nevertheless we gained a good insight into the 3D case in the last chapters. It is probably even interesting to point out that a straightforward approach, trying to solve all questions via sampling, would lead to an arbitrarily huge reprojection error (lemma 5.2 told us that a single viewpoint, or a small region of ε size can change the validity region from ∞ to a bound value[1]).

To solve our problem in the 2D case we will show that it is possible to subdivide the faces into a finite number of subsegments such that each subsegment is valid if and only if the simplification of the extremities of the subsegments are valid.

For example we have seen, that it is sufficient to test the vertices if the view cell is a hyperplane, as the behavior is linear. In the same way, when looking

[1] For a polygonal view cell though it is sufficient to add the extremities to get a more reasonable bound.

at the coordinates of the maximum viewpoint in equation 5.9 we see that the position also depends linearly:

$$V_{max} = (m1 + m2 * \frac{\sin\Theta d_1 \pm \cos\Theta d_2}{\pm\cos\Theta d_1 - \sin\Theta d_2 + 1}, 0)^\top$$

Therefore moving a point M continuously will move the maximum viewpoints accordingly. If the maximum viewpoints fall into the view cell the validity region behaves, as if we had a hyperplane view cell. This leads to the first idea; to detect those parts of the face for which the maximum viewpoint falls inside the view cell. The segmentation of the face will therefore start by isolating the linear part and it is sufficient to compute the validity for its two extremities. For the other cases we have already seen(lemma 5.6), that the maximum viewpoint corresponds to an extremity of the view cell. Therefore it is inevitable to gain further inside in the variation of a validity segment for points on a segment when there is only one viewpoint. We will see that this behavior can be described via a rational function, which will be the basis of how to decide where the segment has to be further subdivided. (Actually we will see, that (at most) one further subdivision is necessary.)

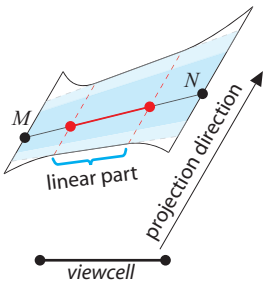
In the final step the simplification plane will be valid for a face if it has been proven valid for all subsegments.

6.2 Detecting the Linear Part

It might still be confusing, that we are working on different 'sides' of the direction $\vec{d}$. If the view cell was a hyperplane, the 'side' would be simply such that the angle with the direction $\vec{d}$ becomes smaller. For a polygonal view cell this is insufficient. (see chapter 5.7)

Following equation 5.9:

$$V_{max} = (m1 + m2 * \frac{\sin\Theta d_1 \pm \cos\Theta d_2}{\pm\cos\Theta d_1 - \sin\Theta d_2 + 1}, 0)^\top$$



it is possible to calculate the coordinates of the maximum viewpoint. One sees, that if the projection direction $\vec{d} = (d_1, d_2)^\top$ remains fixed, which is the case for an orthogonal projection, the existence of a maximum viewpoint only depends on the angle between the view cell and $\vec{d}$ (this had been pointed out in section 5.7).

If there is no maximum viewpoint we know that no viewpoint can see this side of a segment $[M, M + \alpha\vec{d}]$, therefore the validity segment is infinite in direction $\vec{d}$. So let's assume that there is a maximum viewpoint for a point on the face. Then there will be a maximum viewpoint for all points on the face, as the relationship is linear. It is therefore possible to find the two points on the face, which have the maximum viewpoints that are the borders of the view cell, as shown in the margin. All points in between these two will have maximum viewpoints in the view cell. The part of the face which is linear has therefore been detected successfully.

Having this idea it is still necessary to transform it to a mathematical method. As before, without restriction, but to ease the calculation we will assume the view cell to be a segment which is parallel to the x-axis. The face

$\mathcal{F}$ is assumed to be in the halfspace $y > 0$. Thanks to equation 5.9 it is possible to find the coordinates of the maximum viewpoint for a given point M.

$$V_{max}(m_1, m_2) = (m_1 + m_2 * \frac{\sin\Theta d_1 \pm \cos\Theta d_2}{\pm\cos\Theta d_1 - \sin\Theta d_2 + 1}, 0)^\top$$

We will now replace M by the line equation that corresponds to the line containing $\mathcal{F}$. Assuming the line is given by : $l(\alpha) := M + \alpha(M_2 - M) = M + \alpha\vec{w}$ and one border of the view cell is given by $(r, 0)^\top$. We can therefore solve the equation: $V_{max}(l(\alpha)) = (r, 0)^\top$ which is a linear system. We have:

$$\begin{aligned} & V_{max}(l(\alpha)) = (r, 0)^\top \\ \Leftrightarrow \quad & (m_1 + cm_2) + \alpha(w_1 + cw_2) = r \end{aligned}$$

The solution in the case where $(w_1 + cw_2) \neq 0$ is therefore

$$\alpha = \frac{r - (m_1 + cm_2)}{w_1 + cw_2}$$

, where $c := \frac{\sin\Theta d_1 \pm \cos\Theta d_2}{\pm\cos\Theta d_1 - \sin\Theta d_2 + 1}$. If $(w_1 + cw_2) = 0$ we see that all points on the face share the same maximum viewpoint, so either this point is inside the view cell, therefore it is sufficient to assure validity at the vertices, or it is completely outside and therefore there is no linear part.

An interesting observation can be drawn from this equation, too. One sees that there exists a parametrization such that all points along one axis share the same maximum viewpoint, in other words, the maximum viewpoints are given by the intersection with a cone. This is a nice theoretical result, that does not ease the formulas derived before, but it offers the possibility to derive a solution independent of the coordinate system.

6.3 Dealing with the Non-linear Part

Having detected the linear part, we now consider faces, such that for all points on the face the maximum viewpoint for a line view cell does not fall into the view cell itself. In this case we already know, that the maximum viewpoint will actually be one of the borders of the view cell and it will be the same border for all points, as otherwise the face would contain a linear subsegment[2]. Therefore we have to get a deeper insight into the case where the view cell only consists of a single viewpoint. In this section, we will observe that, the variation of the validity segments can be described by rational functions. To ensure validity for all points of the non-linear part on the face, we will show, that we have to calculate the tangent in the orthogonal direction of the projection direction at this curve. As before, we will distinct the cases where this tangent point corresponds to a point on the face and when not. We will then explain how to handle both situations.

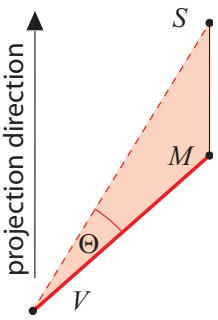
This section is structured as follows, first we will describe the mathematical situation and derive the function representing the validity extents for points

[2]This can be seen, as the maximum viewpoints for a hyperplane view cell are displaced linearly along the face, therefore there has to a part on the face where the maximum viewpoint falls inside the view cell, but then this mesh point on the face would lie in the linear part of the face.

on a line face. During a first lecture the following exhaustive discussion of the function can be skipped and the reader will be informed to do so in the text. At the end of the discussion we will see, that only one additional subdivision has to be performed to ensure validity of all subsegments just by assuring validity at its vertices. During the discussion, we will point out, that due to our approach 'false' intersections might be present in the graph. An exact classification of these cases, leading to an optimization in computation time and more importantly to a complete understanding of the non-linear situation can be found in C.

For the remainder of this section we are only interested in the non-linear part and assume that the linear part of the face has already been detected. Let's suppose we have a face $\mathcal{F}$, and a viewpoint V we are interested in the curve that is described via the validity region extents in a given direction $\vec{d}$. We examine instead of the face itself a line passing through the face. Assuming for the moment that the face is actually this line.

A first approach could consist in using the equation obtained in theorem 5.1 and to add these lengths to the actual points on the face. Unfortunately this approach is very complicated and did not lead to a result. Instead we will develop a direct representation and we will see, that this function is a lot simpler than the one from theorem 5.1. Unfortunately we will have to divide the approach into two. Instead of considering a view cone of aperture 2Θ as we did before, we will be interested in the intersection between a line passing through the observed point and a ray shot at an angle Θ from the viewing direction. (The ray corresponds to one border of the view cone.) The resulting graph of all these intersections may contain points that correspond to 'false' intersections, as we called them in the theorem 5.1. Those represent the moment when there is no longer an intersection in 'front' of the viewpoint. The image in the margin depicts the new situation we are interested in. Of course we will have to deal with the problem that only a part of the curve is actually valid and is not based on false intersections, but for the moment we will keep it simple and not bother about this because the resulting curve, though much simpler than the formulation of Theorem 5.1 still remains quite complicated.



Having this curve we actually visualize the validity region extents created by all the points on the face. To have a valid simplification plane, it has to be below the curve, where below refers to the opposite projection direction $\vec{d}$. It is therefore necessary to find the tangent with direction $\vec{d}^{\perp}$ at the curve in the valid area.

The reader might want to skip parts of this section and come back later if he wants, as we will concentrate in this section only on the resulting function and we will show that it is possible to find a point on the face which minimizes the validity region and therefore assures the simplification plane's validity. Curious readers might continue until further notice, if they want to get an idea of how this is done one.

The following reasonings will in practice not only have to be performed for Θ but also for $-\Theta$. The reasoning is very similar and we will therefore only concentrate on the case of Θ at the moment[3] Indirectly this distinction was already present when looking for maximum viewpoints for a given direction (meaning one of the rays in 2D of the maximum viewpoint was actually useless).

[3]The cases are not identical, as for Θ we have the property that $0 < \Theta < \frac{\pi}{2}$.

When M moves on a line we obtain a graph which we will examine more closely. Figure 6.4 shows an example of how the curve looks like.

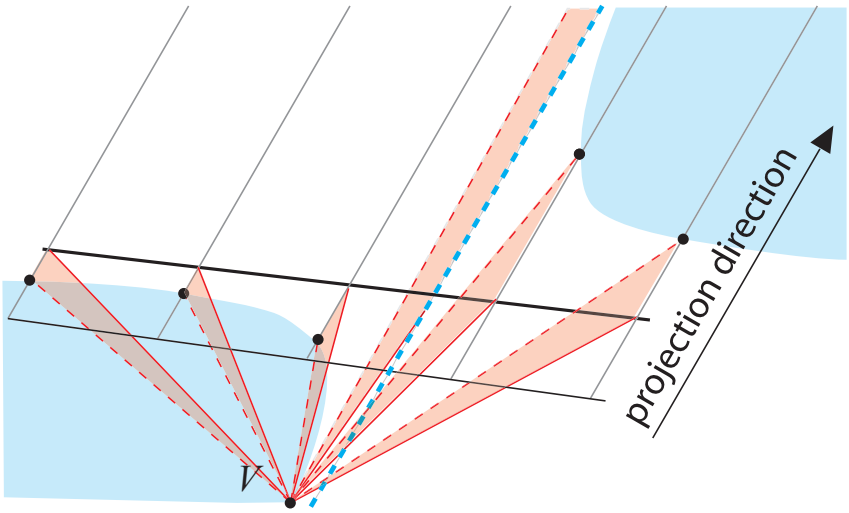


Figure 6.4*: The figure shows the graph, obtained by intersecting two lines, one corresponding to a view cone ray, the other to a line in projection direction, passing through the points on the mesh face. One could think of this as some kind of half (the second view cone ray is missing) validity region for the face's points.*

To describe this function mathematically we will start by describing the situation more precisely. Given a point M, the viewpoint V and the projection direction $\vec{d}$ we are looking for the intersection between the two lines

$$l_1(\beta) := M + \beta\vec{d}$$

and

$$l_2(\alpha) := V + \alpha R_\Theta(V - M)$$

, where R_Θ describes a rotation of angle Θ

To ease the calculus, but without loss of generality, we are going to assume that $V = (0,0)^\top$ and $\vec{d} = (0,1)^\top$. Calculating the intersection leads to

$$I((m_1, m_2)^\top) := (m_1, m_1 \frac{\sin\Theta m_1 + \cos\Theta m_2}{\cos\Theta m_1 - \sin\Theta m_2})^\top$$

This expression is not always defined, but we will examine this later. We now want to discuss the curve we obtain if M is not fixed, but moves on a line. Let's represent this line as $M(\alpha) := (m_1, m_2)^\top + \alpha(\cos\gamma, \sin\gamma)^\top$, we are therefore interested in the graph of $G := I \circ M(\alpha)$. We obtain the following expression:

$$G(\alpha) := (m_1 + \alpha\cos\gamma, (m_1 + \alpha\cos\gamma)\frac{sin\Theta(m_1 + \alpha\cos\gamma) + \cos\Theta(m_2 + \alpha\sin\gamma)}{\cos\Theta(m_1 + \alpha\cos\gamma) - \sin\Theta(m_2 + \alpha\sin\gamma)})^\top$$

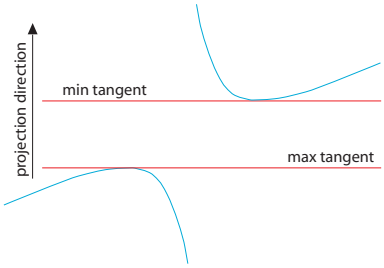
To further simplify we will assume that $M_0 = (0, m)^\top$. This is actually a restriction, as it is now impossible that M describes a vertical line which

is not the y-axis. This is not crucial though, as the projection direction was assumed to be be vertical. A face would be projected to a line (point in 2D) if an orthogonal simplification plane were valid, which would not be a reasonable way to simplify the face. Therefore we will simply assume that simplification planes are not allowed to be perpendicular to the face they are supposed to simplify. It has to be pointed out, that this choice has been made because it makes sense and simplifies the calculus. Interestingly this special case will play a special role in chapter D. Applying the simplification we obtain:

$$G(\alpha) := (\alpha\cos\gamma, \alpha\cos\gamma\frac{sin\Theta\alpha\cos\gamma + \cos\Theta(m_2 + \alpha\sin\gamma)}{\cos\Theta\alpha\cos\gamma - \sin\Theta(m_2 + \alpha\sin\gamma)})^\top$$

The geometry of a curve is not modified when applying a linear transformation to the parametrization. Therefore substituting the parameter α by $\frac{\alpha}{\cos\gamma}$, where $\cos\gamma \neq 0$ as we excluded vertical faces.

$$G(\alpha) := (\alpha, \alpha\frac{sin\Theta\alpha + \cos\Theta(m_2 + \alpha\tan\gamma)}{\cos\Theta\alpha - \sin\Theta(m_2 + \alpha\tan\gamma)})^\top$$



Now this curve can actually simply be represented as the graph of the function

$$X(\alpha) := \alpha\frac{sin\Theta\alpha + \cos\Theta(m_2 + \alpha\tan\gamma)}{\cos\Theta\alpha - \sin\Theta(m_2 + \alpha\tan\gamma)}$$

As mentioned before we were interested in the tangent with normal equal to the projection direction. As the projection direction has been chosen to be $(0,1)^\top$, we are actually interested in local minima and maxima.

Due to the complicated coefficients calculating derivatives is an easy source of error. On the other hand if one solves the problem for a more general function it gets a lot easier. Assuming that the denominator of the function is not constant, as X is a rational function we can rewrite[4] X as

$$X(\alpha) := A\alpha + B + \frac{C}{E\alpha + F} \tag{6.2}$$

Deriving the function 6.2 two times leads to:

$$\frac{d^2X(\alpha)}{d\alpha} := \frac{G}{(Ex + F)^3}$$

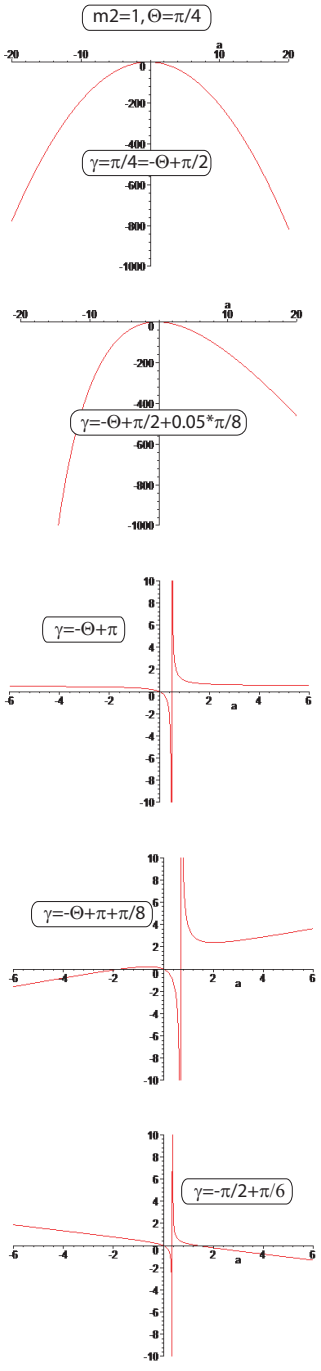
Therefore the curvature of the function can only have two different signs, one for each side of the definition gap. Thus any point for which the derivative vanishes is automatically an extremum, there cannot be any saddle points.

So all that has to be calculated are the roots of the first derivative. The resulting expression is at most quadratic and therefore possible to solve.

We start by transforming X:

$$\begin{aligned} X(\alpha) &= \alpha\frac{sin\Theta\alpha + \cos\Theta(m_2 + \alpha\tan\gamma)}{\cos\Theta\alpha - \sin\Theta(m_2 + \alpha\tan\gamma)} \\ &= \alpha\frac{(sin\Theta + \cos\Theta\tan\gamma)\alpha + \cos\Theta m_2}{(\cos\Theta - \sin\Theta\tan\gamma)\alpha - \sin\Theta m_2} \\ &= \alpha\frac{\alpha\sin(\Theta + \gamma) + \cos\Theta\cos\gamma m_2}{\alpha\cos(\Theta + \gamma) - \sin\Theta\cos\gamma m_2} \end{aligned}$$

[4]This result can be found in almost any book on analysis. It is actually a result of simple polynomial division. An example would be the excellent book by Heuser[Heu].



First one sees that for the two solutions to exist, we need $AB \neq 0$ (in equation 6.2 and the values under the square root have to be positive. Unfortunately this part is quite tricky. As the function behaves very differently for different values of γ changing from a line, to a parabola, to a hyperbole and to a 'real' rational function. Therefore we have to treat all of these cases. Fortunately we already know a lot about the function, the curvature does not change its sign on each *branch* of the function, where branch means a part which lies on one single side of the definition gap. We will exploit this information to examine all possible cases.

The discussion will be quite lengthy and technical. During the first reading of this document it is to skip this part. The reader should just know, that it is possible in all cases to calculate the tangent to the curve and therefore we can decide whether the simplification plane is valid.

First we will treat the case where the numerator is of degree 2 the denominator of degree 1. Our first observation will be to find out when the denominator divides the numerator.

Assuming that they would divide leading to $c\alpha$ (with $c \neq 0$).

$$\begin{aligned} & \frac{\alpha \sin(\Theta+\gamma) + \cos\Theta\cos\gamma m_2}{\alpha\cos(\Theta+\gamma) - \sin\Theta\cos\gamma m_2} = c \\ \Leftrightarrow \quad & \alpha\sin(\Theta+\gamma) + \cos\Theta\cos\gamma m_2 = c(\alpha\cos(\Theta+\gamma) - \sin\Theta\cos\gamma m_2) \\ \Leftrightarrow \quad & \alpha(\sin(\Theta+\gamma) - c\cos(\Theta+\gamma)) + \cos\gamma m_2(\cos\Theta + c\sin\Theta) = 0 \end{aligned}$$

as $\cos\gamma \neq 0$, we will only need to distinct the cases $m_2 = 0$ and $m_2 \neq 0$. Assuming $m_2 \neq 0$ then the constant part implies $c = -\cot\Theta$. Inserting this into the linear part leads to $\sin(\Theta+\gamma) + \tan\Theta\cos(\Theta+\gamma) = 0$, implying that $\cos\gamma = 0$. But this case had been excluded, as then the face would be vertical. If now $m_2 = 0$, X simplifies to $tan(\Theta+\gamma)\alpha$. A linear function, except that two cases have to be distinguished (an example can be seen in figure 6.5). The one where $\gamma = -\Theta + \pi/2$ and the one where $\gamma \neq -\Theta + \pi/2$. We would also need to consider $-\Theta + \pi/2 + \pi$, but remembering that γ describes the angle, the lines for γ and $\gamma + \pi$ coincide, we will therefore reason modulo π. The case $\gamma = -\Theta + \pi/2$ corresponds to what we already mentioned several times. The ray in direction Θ becomes parallel to the projection direction $(0,1)^\top$.

So let's deal with the case that we have a quadratic denominator and a linear denominator and the expression cannot be simplified.

As before we know that we can write the function as follows:

$$X(\alpha) := A\alpha + B + \frac{1}{C\alpha + D} \tag{6.3}$$

Due to the linearity of the denominator we know that at the poles the function 'jumps' from ∞ to $-\infty$. On the other hand due to linear part we know that the function approaches ∞ and $-\infty$ when $|\alpha|$ goes to ∞. Let's assume the pole is at τ and that $left\lim_{\alpha\to\tau} X(\alpha) = \infty$, there has to be a minimum if $\lim_{\alpha\to-\infty} X(\alpha) = \infty$, which corresponds to the case that $A < 0$. We therefore only have to compare the signs of A and C. If they are opposing we cannot have an extremum, and the function is monotone. If they are the same, the function will have extremities.

Calculating the polynomial division leads to:

$$X(\alpha) = \tan(\Theta+\gamma)\alpha + \frac{m_2\cos^2\gamma}{\cos^2(\Theta+\gamma)} + \frac{m_2^2\cos^3\gamma\sin\Theta}{\cos^2(\Theta+\gamma)(\cos(\Theta+\gamma)\alpha - m_2\sin\Theta\cos\gamma)}$$

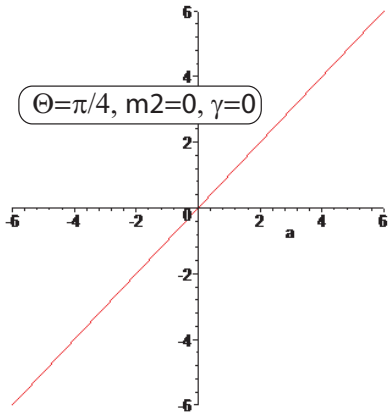


Figure 6.5*: An example of the linear case.*

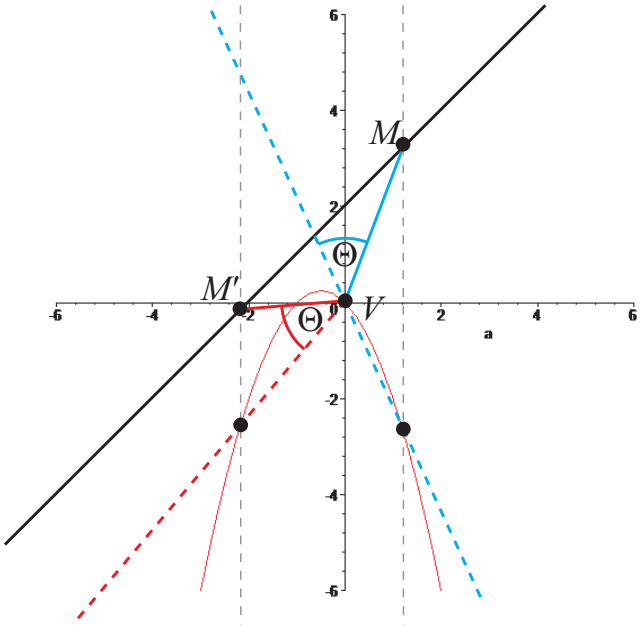


Figure 6.6*: An example of the parabola case.*

We see that the sign of the coefficients is determined via $\tan(\Theta+\gamma)$ in the numerator and $\cos(\Theta+\gamma)/(\cos\gamma\sin\Theta)$. Therefore the relationship between the signs can be determined via $\sin(\Theta+\gamma)$ (numerator) and $(\cos\gamma\sin\Theta)$ (denominator).

We are now looking for the extrema.

$$\frac{dX}{d\alpha} = \tan(\Theta+\gamma) - \frac{m_2^2\cos^3\gamma\sin\Theta}{\cos(\Theta+\gamma)(\cos(\Theta+\gamma)\alpha - m_2\sin\Theta\cos\gamma)^2}$$

$\frac{dX}{d\alpha} = 0$, implies

$$\begin{aligned}
& \tan(\Theta+\gamma) - \frac{m_2^2\cos^3\gamma\sin\Theta}{\cos(\Theta+\gamma)(\cos(\Theta+\gamma)\alpha - m_2\sin\Theta\cos\gamma)^2} = 0 \\
\Leftrightarrow\quad & (\cos(\Theta+\gamma)\alpha - m\sin(\Theta)\cos(\gamma))^2 - \frac{m_2^2 cos^3\gamma\sin\Theta}{\sin(\Theta+\gamma)} = 0 \\
\Leftrightarrow\quad & \cos^2(\Theta+\gamma)\alpha^2 - 2\cos(\Theta+\gamma)\alpha m_2 sin(\Theta)\cos(\gamma) \\
& + m_2^2 sin^2\Theta\cos^2\gamma - \frac{m_2^2\cos^3\gamma sin\Theta}{\sin(\Theta+\gamma)} = 0 \\
\Leftrightarrow\quad & \cos^2(\Theta+\gamma)\alpha^2 - 2\cos(\Theta+\gamma)\alpha m_2 sin(\Theta)\cos(\gamma) + \\
& m_2^2\sin^2\Theta\cos^2\gamma - \frac{m_2^2\cos^3\gamma\sin\Theta}{\sin(\Theta+\gamma)} = 0 \\
\Leftrightarrow\quad & a^2 - 2\frac{m_2\sin\Theta\cos\Theta}{\cos(\Theta+\gamma)}a + \\
& \frac{m_2^2\sin^2\Theta\cos^2\gamma}{\cos^2(\Theta+\gamma)} - \frac{m_2\cos^3\gamma\sin\Theta}{\sin(\Theta+\gamma)\cos^2(\Theta+\gamma)} = 0
\end{aligned}$$

Applying the Mitternachts formula leads to the roots of the derivative.

$$\alpha_{1,2} = \frac{m_2\cos\gamma}{\cos(\Theta+\gamma)}(\sin\Theta \pm \sqrt{\frac{\cos\gamma\sin\Theta}{\sin(\Theta+\gamma)}}) \tag{6.4}$$

So we have seen how to deal with the case where the numerator and denominator do not simplify. As we can see, the intuitive explication for when the maxima exist correspond exactly to the mathematical formulation in equation 6.4.

It suffices now to examine the cases where they simplify. These two cases correspond to $\sin(\Theta+\gamma) = 0$ (numerator) and $\cos(\Theta+\gamma) = 0$ (denominator), realize, that they cannot both vanish at the same time.

The case $\sin(\Theta+\gamma) = 0$ implies $\gamma = -\Theta$. As before the function can now be written in the form:

$$X(\alpha) = A + \frac{D}{B\alpha + C}$$

This is a real hyperbole and we know that both branches are monotonic. There cannot exist a tangent with normal $(0,1)^\top$ at a hyperbole. The only exception is when the numerator is completely zero which implies that $m_2 = 0$ in this case $D = A = 0$ and therefore we have a linear function, the x-axis. This is because the view cone ray aligns with the x-axis for all points in this configuration.

The second case $\cos(\Theta + \gamma) = 0$, corresponds to $\gamma = \pi/2 - \Theta$. This leads to a parabola (an example is shown in figure 6.6) except for $m_2 = 0$, when the function is always undefined. The latter situation occurs if for all points on the face the view cone ray aligns with the projection direction.

Let's assume $m_2 \neq 0$, modifying equation 6.3 according to the situation, we obtain:

$$\begin{aligned} X(\alpha) &= \alpha \frac{\alpha \sin(\Theta + \gamma) + \cos\Theta \cos\gamma m_2}{-\sin\Theta \cos\gamma m_2} \\ &= -\alpha^2 \frac{\sin(\Theta + \gamma)}{\sin\Theta \cos\gamma m_2} - cot\Theta\alpha \end{aligned}$$

Therefore $\frac{dX}{d\alpha} = 0$ if

$$\begin{aligned} & -\alpha \frac{2\sin(\Theta + \gamma)}{\sin\Theta \cos\gamma m_2} - cot\Theta = 0 \\ \Leftrightarrow \quad & \alpha = -\frac{\sin\Theta \cos\gamma m_2}{2\sin(\Theta + \gamma)} cot\Theta \end{aligned}$$

So we are able to detect the tangent.

Now that we have completely discussed the function, the next step has to be to determine the validity of the function. Remembering that it contains false intersection, we have to assure, that for a given direction and an angle Θ we are only taking into account the restrictions which are not due to false intersections. An exhaustive classification can be found in the appendix C.

Chapter 7

View-dependent Billboard Clouds

The view-dependent Billboard Clouds algorithm shows several similarities to the original Billboard Cloud algorithm. The main difference is the way the validity regions are calculated. In the original approach simple spheres where sufficient. In the view-dependent case it is much more complicated, but all the theory needed to calculate the validity regions for view-dependent billboard clouds had been described in chapters 5 and 6, we will apply the results for the 2 dimensional case to simplify 2.5 dimensional scenes.

As for the Billboard Clouds algorithm we will use the Hough parametrization [HOU] to represent a plane (ϕ, ρ). Given a face, we will find for a given angle ϕ two values ρ_{min} and ρ_{max} such that any plane (ϕ, ρ) is a valid simplification plane if and only if $\rho_{min} \leq \rho \leq \rho_{max}$. This will allow for a speed up in the final algorithm, too. We will refer to the interval $[\rho_{min}, \rho_{max}]$ as the validity interval.

To obtain these values for a polygonal view cell we will calculate all the appropriate intervals for all view cell segments independently. As pointed out before the formulas assumed that the face has to be completely in front or behind the view cell segment. We have also seen that if this is not the case, we can cut the mesh face into two parts, where one does not have to be taken into account, as there is another view cell segment for which the validity of those points will be smaller (compare the remark on page 30). The part which could be discarded is the one lying on the same side as the interior of the view cell at this view cell segment. This is actually a very effective acceleration of the algorithm, as lots of faces can be culled for a given view cell segment. The situation is depicted in figure 7.1.
Therefore the algorithm for a polygonal view cell can be described as follows.

```
GetValidityInterval(VIEWCELL v, FACE f, ANGLE phi) {
 rhoMin=-infinity;
 rhoMax=infinity;
 FACE f2;
  for each SEGMENT s of v
   f2=CutFaceAtViewcell(f,s);
   (rhoMinT,rhoMaxT)=GetValidityIntervalForSegmentVC(s,f2,phi);
```

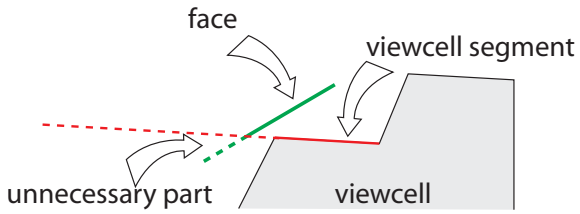


Figure 7.1: *Only the part of a face, which lies in front of the view cell segment has to be considered.*

```
   if (rhoMinT>rhoMin) rhoMin=rhoMinT;
   if (rhoMaxT<rhoMax) rhoMax=rhoMaxT;
  endfor;
 return (rhoMin,rhoMax);
}
```

In general, data types are noted in uppercase, variables start with a lowercase letter and function names with an uppercase letter.

The algorithm can be explained as follows. A polygon view cell is dealt with separately for all its faces, as we have shown that this is sufficient. The algorithm therefore iterates over all the view cell segments. CutFaceAtViewcell will return only the part of the face which lies in front of the view cell segment because the other part will be more restricted by a different view cell segment. GetValidityIntervalForSegmentVC calculates the validity region for a face given a viewcell segment, which will be described in more detail further on. The final validity region for the face is obtained by intersecting the solutions for all view cell segments.

To simplify the explication of the function GetValidityIntervalForSegmentVC we will describe a less optimized, but easier to follow algorithm for the calculation of the validity for a segment view cell, that will not involve all the optimizations already described in chapter 6.

The first step to determine the validity is to detect the linear part (see chapter 6.2). We know that for this region it is sufficient to ensure that the simplification plane $\mathcal{P}$ is valid for the two extremities. If one or both extremities of the mesh face $\mathcal{F}$ fall inside the linear part, it is sufficient that $\mathcal{P}$ passes via the validity extent of the extremity/extremities inside of the linear part. The remaining segment(s) without linear part has (have) then to be dealt with separately (see chapter 6.3). To deal with the non-linear part we calculate the extrema. To find the corresponding point on $\mathcal{F}$ it suffices to project the minimum on $\mathcal{F}$ in the projection direction $\vec{d}(\phi)$. This is because the graph was obtained by calculating the intersection between two lines, one of them in direction $\vec{d}(\phi)$ and passing through the examined point on $\mathcal{F}$. Now if the minimum is actually on $\mathcal{F}$ it is sufficient to calculate the validity segment at this point, otherwise it is sufficient to test the extremities, as we know that the function is monotone in the projection direction $\vec{d}(\phi)$. It is also important to realize, that this non-linear part of $\mathcal{F}$ will be related to one of the extremities of $\mathcal{V}$, therefore calculating the validity region in the non-linear part corresponds to one evaluation of the formula from theorem 5.1. Unfortunately the whole calculation has still to be done for both 'sides' (of the isotorus), as pointed out

in chapters 5 and 6. Still we will end up with a finite number of validity extents from which we chose the smallest one with respect to $\vec{d}(\phi)$, which gives us the value ρ_{max}. Identically to obtain ρ_{min} the same steps are performed only for the opposite projection direction $-\vec{d}(\phi)$. For optimization purposes one should realize that the function for the non-linear part is the same for $\vec{d}(\phi)$ and $-\vec{d}(\phi)$. Therefore both extrema could be recovered at once. It is correct to take the min of the ρ_{max} with respect to $\vec{d}(\phi)$. Given a simplification plane of angle ϕ and distance ρ, it follows that if ρ exceeds any of the ρ_{max} we calculated, then the plane cannot be a valid simplification plane. In the same way one can explain why the maximum of all ρ_{min} has to be used to determine the final validity interval. A simplification plane that lies between both of these values is therefore a valid simplification plane for all the points on the face we have chosen. As we selected them in a way to bound the error for the whole face, we deduce, that the simplification plane is also valid for the face.
In pseudocode a simplified version of the algorithm could be described as follows:

```
GetValidityIntervalForSegmentVC (VIEWCELL v, FACE f, ANGLE phi) {
 rhoMax= GetValidityForDirection(v,f,phi);
 rhoMin= GetValidityForDirection(v,f,phi+PI);
 return [rhoMin,rhoMax];
}

GetValidityForDirection (VIEWCELL v, FACE f, ANGLE phi) {
 FACE nonLinLeft,linear,nonLinRight;
 rho[2];
 index=0;
 for BOTH SIDES OF THE ISOTORUS s do
  (NonLinLeft,Linear,NonLinRight)=CutOutLinearSegment(v,f,phi,s);
  rhoNLLeft=GetValidityForDirNonLinear(nonLinLeft,Left(v),phi,s);
  rhoNLRight=GetValidityForDirNonLinear(nonLinRight,Right(v),phi,s);
  rhoLinear =GetValidityForDirLinear(linear,v,phi,side);
  rho[index++]=min(rhoNLLeft,rhoNLRight,rhoLinear);
 endfor;
 return min(rho[0],rho[1]);
}
```

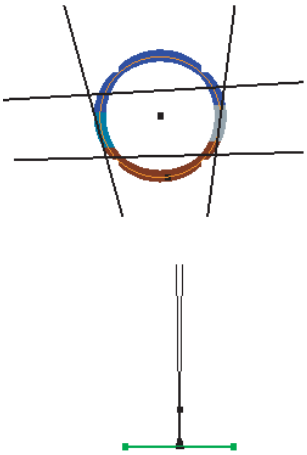
To summarize, we calculate a validity region (ρ_{min}, ρ_{max}) such that for all planes with normal $(\cos(\phi), \sin(\phi))$ they are valid if their ρ-value lies inside of this interval. Both values are calculated by the same function GetValidityForDir. In this function the face is partitioned into a linear region, where it is sufficient to test the extremities and two non-linear regions. The corresponding functions are called for each face segment and for both sides of the iso-torus. The non-linear part only needs the single viewpoint, which corresponds to the maximum viewpoint (the functions Left, Right return the corresponding extremity of the view cell segment). In the function GetValidityForDirNonLinear, the extrema of the curve are computed, then if they fall onto the face, the value is evaluated, otherwise, due to monotonicity, it only has to be evaluated at the extremities.

In the last step, the intersection of all obtained validity regions is performed. It has to be pointed out, that the extremities of some subdivision segments coincide, therefore one does not have to calculate the validity region for those vertices twice.

It is now possible to use these functions directly in the Billboard Clouds algorithm.

Nevertheless, guiding the plane selection using the density is slightly more complicated. We remember that the Billboard Cloud algorithm uses a discretization of the plane space into bins. For each bin densities are calculated, which represent the amount of surface area that could be simplified on any of the planes represented by the bin. This leads to better results than simply testing for the center plane. In the original Billboard Cloud approach it was sufficient to calculate the valid region of each face vertex to each limit plane of the bin and intersect the valid regions of all the face vertices. In the view-dependent case this step is a not so easy to perform. Creating the union as before would be a bad approximation, although it remains conservative. Intuitively speaking, this is because rho values are not only influenced by the plane normal, but also by the direction of the face. To get a better estimate for the planes we decided to test, whether the face is at least valid for one of the borders, if not, then the face does not contribute, except for the case, when the face's normal is included in the bin's plane normals. This leads to a much more efficient plane search guiding, if the discretization is coarse.

The other question that remains is on what value the density will be based. The penalty introduced in the original Billboard Clouds algorithm to assure better placement for curved surfaces cannot easily be transferred. First of all the application of the penalty is only valid if the origin is inside of the mesh. The second assumption is, that the object has a rather spherical shape. For a single object this might still hold, in our view-dependent approach it is more likely that there are several objects in which case the assumption does no longer hold. It is also questionable whether this penalty would be of benefit in the view-dependent case for another reasons; the shape of the validity regions. The validity regions look like drops and are a little longer away from the view cell, therefore the first simplification plane at a sphere will not be placed very close to the front, but more likely towards the center. Leaving a larger piece to the front. Therefore even without penalty the billboard representation might be optimal. See the figure in the margin showing an example simplification, where the black cone represents the error. Concerning the calculated density we tried to remain as general as possible. Therefore our density is based on two terms, a *view cell contribution* $ViewC$ and a *projection contribution* $ProjC$. For a given plane $\mathcal{P}$ and a face $\mathcal{F}$ the density $C_{\mathcal{P}}(F)$ is given by:



$$C_{\mathcal{P}}(F) = ProjC_{\mathcal{P}}(F)ViewC_{\mathcal{P}}(F)$$

Both terms can be explained as follows. $ProjC_{\mathcal{P}}(F)$ corresponds to a factor taking into account the projected area of the surface onto the plane. This projected area is solely based on the segment face. To take the height into account (for 2.5 dimensional scenes) $ViewC_{\mathcal{P}}(F)$ is used. It could be observed, that simply defining the value $ViewC_{\mathcal{P}}(F)$ as the height of the object, would lead to a density value, that corresponds exactly to the projected area in 3D. On the other hand one could also include 3D visibility information in this term. It is very common to decouple the visibility information as a multiplicative term. One could also simply define $ViewC_{\mathcal{P}}(F)$ to be a constant making the density become a completely 2 dimensionally based value. We experimented with several approaches including size as seen from the view cell of both, the

face and its simplification, preferring planes aligned with edges of the view cell values based on the height, pure 2D approaches and their combinations. One has to remember that the density is only used to guide the plane selection. In each case the simplification will be valid. None of the aforementioned formulas resulted in a constantly more convincing result. Nevertheless the projected size onto the billboard seems to be a good density value.

Another interesting aspect is that the sampling of the plane space via Hough transformation is not uniform. Planes closer to the origin are sampled more densely than planes farther away. On the other hand our view-dependent approach allows far away faces to be simplified more aggressively. These two properties could be exploited. It is a good idea to place the origin in the center of the view cell, where we define the center as the simple average of all view cell vertices. This choice lead to better results of the heuristic plane selection.

Concerning the plane search guidance we realized another problem of the original Billboard Clouds approach. The subdivision scheme always selects the bin of highest density, but there might be several. Therefore the question arises, whether it is acceptable to take the first bin of highest density. We discovered, that this might lead to an algorithm that does not necessarily terminate. To give an example, one might consider a single triangle aligned with the x-axis. The highest density will therefore be encountered for planes orthogonal to the x-axis. There is a region of $[\rho-\varepsilon, \rho+\varepsilon]$ of valid distances for the planes. Taking the first bin encountered, would lead to a selection which contains the plane with distance $\rho - \varepsilon$. Which remains true for all further subdivision. So the plane that is actually approximated will be the one at distance $\rho - \varepsilon$, this could lead to numerical problems because the center plane in this bin could be at $\rho - \varepsilon - \delta$ for a δ arbitrarily small and is therefore not valid. Several solutions exist. One, which works well in practice is not to chose the first maximum encountered, but to select the one which is the most centered one, meaning that if there are several consecutive bins in ρ-direction having the same density we will select the one in the middle. This improved convergence and assures termination of the program, even in the special case described before.

Another step to further optimize the result is to calculate a least square plane $\mathcal{P}_{\in}$ for the faces that have been selected to be simplified on a plane $\mathcal{P}_{\infty}$. If $\mathcal{P}_{\in}$ is also a valid simplification plane we will replace $\mathcal{P}_{\infty}$ by $\mathcal{P}_{\in}$. This leads to a better result. As in the aforementioned point the original algorithm, for a single triangle would not be guided to the best approximation plane, which would be the plane containing the triangle, but a plane, at a certain distance. Therefore it is a very good idea to optimize the plane position in a final step. It is highly possible, that even better plane positions could be found involving e.g. the work by Alliez et al. [CSAD04].

Finally the texture storage, although this is seems no longer a real problem (see [LN04]), can be optimized by taking the view cell into account. This part will be explained in the chapter D, where we will derive a minimum texel size to be sure, that for each viewpoint the pixel size remains below a given size. Further on in chapter D.2 we will describe a technique to optimize texture storage in the particular case of view-dependent billboard clouds. We will derive an algorithm to find a "best texturing" viewpoint for a given billboard, which leads to smaller textures, as perspective deformation is exploited. Further on, we will describe, how view-dependent textures can be used to obtain better results.

Chapter 8

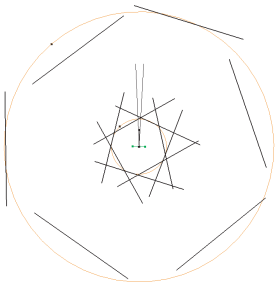
Discussion and Results

8.1 Results

The resulting algorithm allows for the first time to perform view-dependent simplification with an exact error bound.

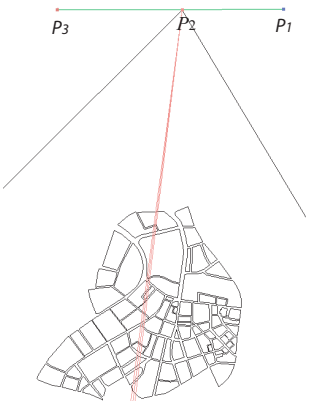
We implemented the algorithm described in this document in C++. The implementation seems to be very effective, as scenes containing several thousands of polygons are simplified in a couple of minutes, depending on the chosen quality. (One has to remember that without heuristic we still have to deal with a NP complete problem.)

The view-dependent Billboard Clouds algorithm proves the usefulness of our theory. The results are very promising. We tried for example to simplify the faces of a tesselated circle around the view cell. The number of billboards used for the simplification remained more or less constant with respect to the radius. This would have been impossible without taking view-dependence into account.



The view cell corresponds
to the green line,
the cone represents the error

We applied the algorithm also on a city model of the Boston Financial District. Unfortunately it is not the best model to show the strengths of our approach as the streets are very narrow. Still it was interesting to examine the algorithms behavior. With an error of approximately 3◦ all 4480 triangles have been simplified on less than 50 billboards. Where 460 faces had been simplified on the first billboard($> 10\%$). Figure 8.1 shows the resulting decomposition. One problem remaining is the greedy plane selection approach. As some isolated faces remain, which have to be simplified on very small billboards. This is a general drawback from applying heuristics on NP-hard problems.



Another example can be seen on the image 8.2. Here 115 billboards represent the whole scene with an approximate pixel error of 10. The margin image shows the situation as seen in 2D.

Figure 8.3 shows the scene as seen from above. Revealing the stronger simplification for faces farther away. In particular, one realizes that faces far away project even on planes that are almost perpendicular.

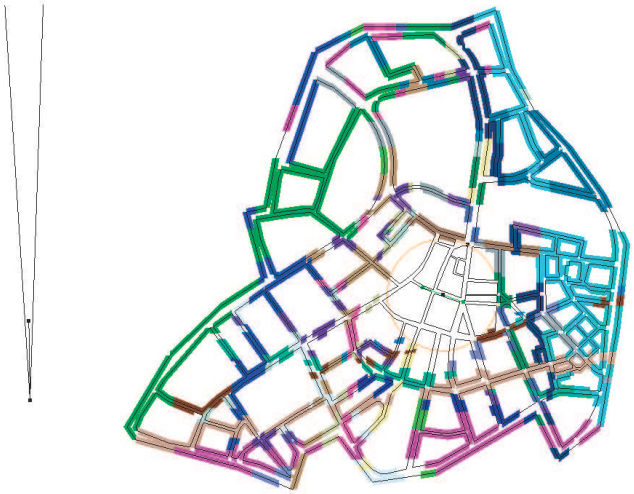


Figure 8.1*: A city decomposition on less than 50 Billboards. The view cell corresponds to the green line, the faces inside the red circle where culled. Shown is the decomposition for the first 30 billboards.*

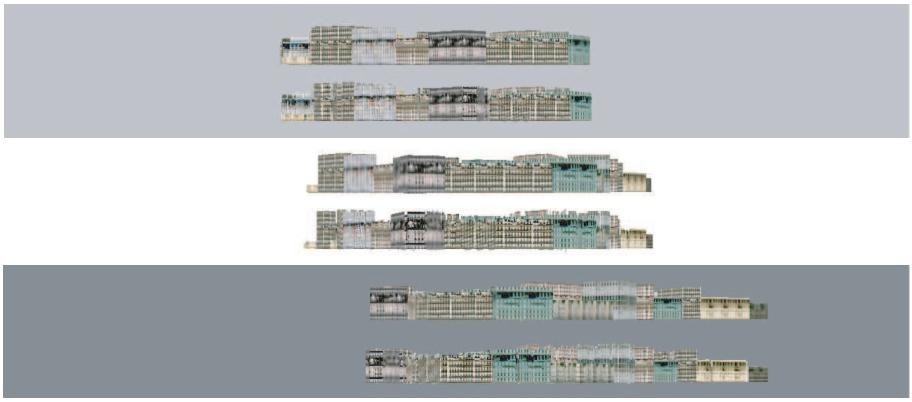


Figure 8.2*: 115 Billboards have been used to simplify the more than 4000 triangles of the initial mesh. The error is at 0.3 degrees. The view frustum approximately 40 degrees, leading to a final approximate error of 10 pixels. The upper two images correspond to the P_1 extremity of the view cell, the middle part to P_2, the lower part to P_3, as shown on the 2D overview.*

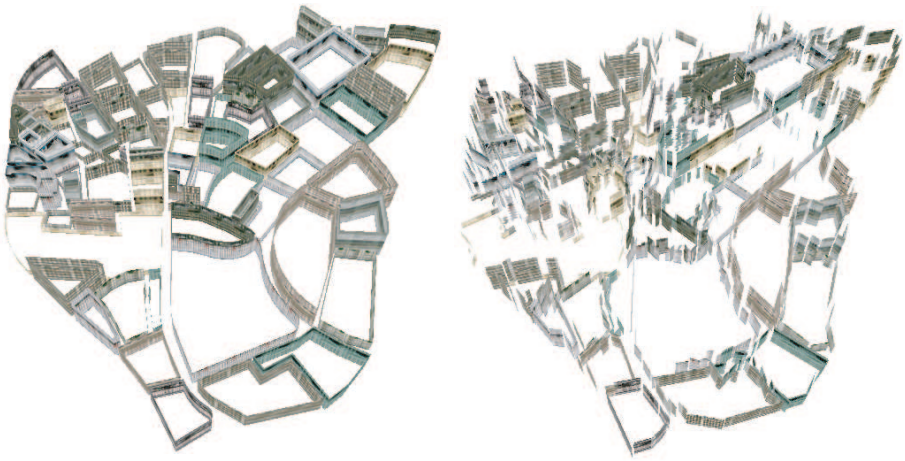
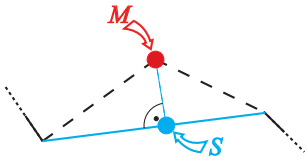


Figure 8.3*: Comparison, original scene to the left, simplified scene on 115 billboards right. Realize the extreme simplification for far away faces. It becomes hard to estimate the original geometry for those parts, although from the view cell both representations are close.*

8.2 Discussion

We developed a new error norm, which allows to bound the reprojection error after a simplification has been performed. This result could easily be transferred to many other algorithms. In particular we want to mention the work by Cohen et al. [CVM+96]. Their main idea was to create a hull around the object, which assures the simplified model to remain in an assured distance to the original model. A simplification step would then be valid, if the simplified model remains inside the hull. In the original approach the hull was based on an approximation of the Hausdorff distance. Our case is more precise, our Hull is defined by a validity region around all and each of the points therefore we also take texture distortion into account. The application is very simple. The algorithm [CVM+96] tries to delete vertices one after the other. A deletion is accepted if the simplified mesh remains inside the hull. In our solution it is tested, whether the orthogonal projection of the deleted faces onto the new resulting faces results in a valid simplification. The image in the margin shows the two dimensional case. The point M is simplified on a point S, which lies in the orthogonal direction.



The huge generality can also be observed by considering to apply our approach on subdivision -, iso- or spline surfaces. One could apply the simplification on a sufficiently refined model, which is fine enough, so that the distance to the exact surface becomes invisible. This is given for a finite number of triangles, as the screen resolution is discrete. One could also imagine to start with an algorithm, like e.g. the one by Alliez et al. [ALSS] and use our metric to further improve the result.

Our calculus also justifies a very common approach to represent the background in computer games by a single image. As for very far away objects, the validity regions will become very long and therefore a simplification on a single billboard becomes possible.

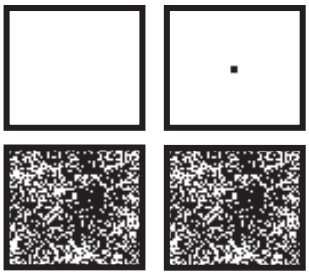
Another important property of our algorithm is the preservation of silhouettes. The validity region has the form of a drop, therefore allowing much more movement in direction back and forth. Until now, most algorithms added a test on the normal direction to stop subdividing at silhouettes. In no case these ad-hoc solutions had a mathematically sound relationship to the other simplification criteria. Our approach merges the silhouette preservation with the simplification in a comprehensive mathematical based way. The properties remain no longer independent. Usually the algorithms up to now calculate test a silhouette criterion based on the normal direction and if the normal is orthogonal to the viewing direction a subdivision is performed. It is independent of the actual error, that will be observed. In our case silhouettes are preserved, only if otherwise it would lead to a perceptual difference. The length of edges no longer plays a role in our approach. For example it is possible that the validity region or a point is unbounded, in which case even in the simplified model edges might appear that are longer as those of the original model, showing that edge lengths are not necessarily useful to measure the quality of an approximation.

One important point has been neglected until now. The validity region for a point M has always been calculated with respect to all the viewpoints in the view cell. Still it is not sure, that all of them will see M, therefore they should usually not contribute to restricting the validity region. In two dimensions algorithms exist, that calculate the exact polygonal part of the view cell that corresponds to the viewpoints that can actually see a given point, but unfortunately it is not easy to code, nor very effective concerning computation time. (An example would be the use of a radial decomposition [Ola02].) Therefore it would be possible to include it in the algorithm, but as there is no convenient visibility calculation in three dimensions, we decided, to exclude it, in order to keep hoping for a complete three dimensional solution. The first step of the algorithm would be to delete triangles which cannot be seen from the view cell. Simple solutions involve a sampling of viewpoints and reading out the frame buffer. Other more advanced techniques exist ([NBG02]). Nevertheless, this cannot be considered as future work, as we deliberately decided not to involve this step, to better show the quality of our theory.

Concerning view-dependent Billboard Clouds, we achieved a highly efficient approach to simplify models and still ensuring closeness to the original model. Interestingly, the original Billboard Clouds approach can be seen as a subcase of the view-dependent Billboard Clouds, if the view cell is chosen to be at infinity. Nevertheless a complete three dimensional solution, for which we already explored some optimizations concerning the representation, remains to be found. Nevertheless we already pointed out several properties and results as well as numerical solutions and directions in this document, which might be the basis for future research.

In the case of city simplification, the main subject of impostor strategies, the algorithm performs very well and creates a view cell based alternative representation with respect to the geometry of the original scene, which lots of common techniques do not (compare 2).

8.3 Future Works



A whole variety of interesting avenues for future research have been proposed in this work.

Our approach does not take into account any human visual system based perception measures. This is important to mention, as a small geometric error might still make a huge perceptual distance. In the margin we see four images, the distance in the norm L_∞ between the upper two images is exactly the same as the distance between the lower two images. Still it is much easier to distinguish the upper two images.

The view-dependent Billboard Clouds have the same problem. A Billboard Cloud artifact are cracks, that can appear between different billboards. Cracks are created when connected faces in the model are projected onto different billboards. Although the size is bounded by 2Θ, a distance of Θ for each side, they might be displeasing and evident depending on the surrounding. A remedy might be found in approaches like the so-called fat borders ([BMK]). Originally the idea is to add line primitives that are rendered at the borders of Bézier patches, to hide cracks, that could appear when two patches are differently subdivided. One could imagine to add parts of faces to the texture, that have not been simplified on a given billboard, but that are connected to a face, which has been simplified onto this billboard. Unfortunately high tessellation will make cracks even more probable, as we do not exploit any neighboring information, in order to make our algorithm work for polygon soups, too. Future work could include the use of neighboring information to perform a per object instead of per face simplification.

Another problem is the limited possibility to perform relighting. Billboard Clouds already used normal maps to allow for an approximation of diffuse lighting. Specular effects, reflections and also transparent objects cannot be handled.

Shadows for Billboard Clouds were calculated by projecting all the billboards on the ground, using the transformation matrix by Herf et al. [Her]. View-dependent Billboard Clouds do not assure to correspond in any way to the shape of the original object, when seen from the light source. (As an example one might think of the extreme case, where the validity regions are unbounded and parts could be simplified to infinity. In this case the cast shadow of the simplification might be completely incoherent with respect to the original shape.) To diminish this problem, if really necessary, one could add the light, or even a region around it, as a possible viewpoint. Therefore the shadow will remain closer to the original. Unfortunately the error could still be arbitrarily large, a small difference at the object might result in a huge difference in the shadow (compare the ray theorem). On the other hand we developed a new approach to calculate 'nice' shadows, inspired by view-dependent billboards, but which we decided not to include in this thesis.

It could be interesting to see whether a different parametrization instead of the Hough space could be used. Which is more important for the original Billboard Clouds, than for the view-dependent Billboard Clouds.

Other interesting avenues could be to reuse screen information. Briere et al. [BP96] described a technique, that is usable in ray-tracers. Moving objects are detected and only the parts that become visible from one frame to the other are updated. Our situation would be more complicated, as the camera is moving.

Finally, as for all image based simplifications, the texture amount cannot be

neglected, but the memory usage of our algorithm remains reasonable, due to the storage optimization described before. Concerning the loading of textures, one could imagine the use of an estimator for the movement of the observer to pre-fetch textures, that might be used soon.

Theoretical work, that remains, is finding a bound for the error when sampling a three dimensional surface. It could also be interesting to perform a classification of the shape of the linear part of a plane and to gain further inside in the shape of the non-linear part, maybe by exploring the ruled surface given in theorem 5.1. The analytical solution for the three dimensional case for points also remains an open problem. The proposed avenues in chapter 5 merit further exploration. We also started exploring to describe the whole problem based on projective geometry, which we abandoned in the first place, as expressions concerning angles can become very complicated.

Appendix A

Validating the circle growing approach

The problem that has been mentioned before is that we have not yet proven, that the approach of growing a circle is correct, in the sense that the intersection point we obtain is on the part of the circle, belonging to the iso-torus. In practice, this is not a severe problem, as both sides are taken into account and therefore the 'wrong' part of the circle is hidden inside of the correct part for the other side. Still the computation can be accelerated by excluding these wrong viewpoints.

To prove this step we will first examine more closely what the choice of a side actually represents. In the following chapter the idea, which will now only be stated informally will be explained in detail. The choice of the side is equivalent to the choice of a ray of the view frustum. This can be seen by imagining a direction $\vec{d}$ and a point M. Let's assume two viewpoints V_1 and V_2 on two different sides of the line through M in direction $\vec{d}$. The view cones of V_i will be represented by the two rays $r_i^+ = V_i + R_\Theta(M - V_i)$ and $r_i^- = V_i + R_{-\Theta}(M - V_i)$, where R_Θ denotes a rotation matrix of angle Θ. For one of the view points the intersection will be on r_i^+ for the other on r_i^-, if there is one. This idea of choosing a ray instead of a side will reappear in the next chapter. The situation is depicted in figure A.1.

Assuming that a point M lies in the halfspace $y > 0$ and the view cell is given by $y = 0$. Given a direction $\vec{d}$ we will define $S(t) = M + 2t\vec{d}$ the parameterized simplification point as in the chapter 5.6 on page 28 on. The tangent at the point $S(t)$ will allow us to determine when we have an intersection on the side of the circle which is not part of the Thales theorem. We will perform all reasonings for a viewpoint on the side given by $\vec{d}^\perp$. The tangent at a circle can be calculated, as its direction is always normal to the vector connecting the center of the circle $C(t)$ with the tangent point. To clarify the situation one can refer to the image in the margin. For the point S_1 the tangent's direction implies that parts of the circle are lower than the part between S_1 and M. S_2 represents an example where we would encounter a false intersection.

A first observation shows that if we had a false intersection, then the first real maximum viewpoint that would be achieved by growing the circle further until the correct part intersects the view cell is the intersection of the ray from M in direction $\vec{d}$ with the plane. This maximum viewpoint could be considered

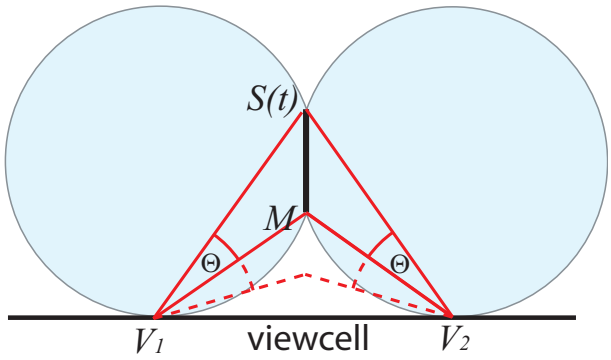
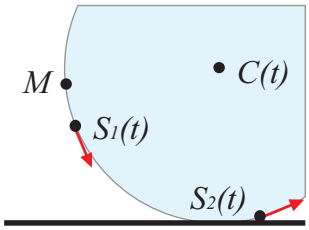


Figure A.1: *The choice of the 'side' corresponds to a choice of a ray. Viewpoint V_1 has been chosen to be on the 'left', and therefore the ray with an angle Θ corresponds to this choice, whereas V_2 is the opposite.*



as a viewpoint for the other 'side' because the intersection is in the viewpoint itself and therefore on both rays. Having a maximum viewpoint for the other 'side' would therefore be sufficient. The question is now, for which direction $\vec{d}$ this happens. This is where we will use the tangent. The moment we are looking for is when the tangent at $S(t)$ becomes horizontal. Fortunately we will see that the slope of the tangent at the point $S(t)$ does only depend on the direction $\vec{d} = (\cos\gamma, \sin\gamma)^\top$.

Remembering the equations from section 5.6, we have

$$C(t) = M + t\vec{d} + t\cot\Theta\vec{d}^{\perp}$$

$$S(t) = M + 2t\vec{d}$$

As the tangent is direction is normal to the vector from $S(t)$ to $C(t)$, we are interested in:

$$\begin{aligned} & < S(t) - C(t) | (1,0)^\top >= 0 \\ \Leftrightarrow \quad & < t\vec{d} - t\cot\Theta\vec{d}^{\perp}, (1,0)^\top >= 0 \\ \Leftrightarrow \quad & \cos\gamma + \cot\Theta\sin\gamma = 0 \\ \Leftrightarrow \quad & sin(\Theta + \gamma) = 0 \\ \Leftrightarrow \quad & \gamma = -\Theta(+\pi) \end{aligned}$$

For case $\gamma = \pi - \Theta$ the point S(t) will in this case the wrong side always remains above M therefore an intersection can only be valid. The other case $\gamma = -\Theta$ (figure A.2) is the one which is interesting. For all the maximum viewpoints we detect between the angle $-\Theta$ and Θ the intersection we calculate will lie on the wrong part of the circle. Nevertheless a maximum viewpoint is useless in this range. As mentioned before, if the angle γ lies between $-\Theta$ and 0 the maximum viewpoint would be at the intersection between the ray through M in direction $\vec{d}$ and the view cell. Therefore the intersection lies on both rays and therefore the maximum viewpoint for this ray is given by the other maximum viewpoint for the other ray. Between 0 and Θ we cannot encounter any intersection between the corresponding frustum ray and the ray from M in direction $\vec{d}$ (see figure 5.6).

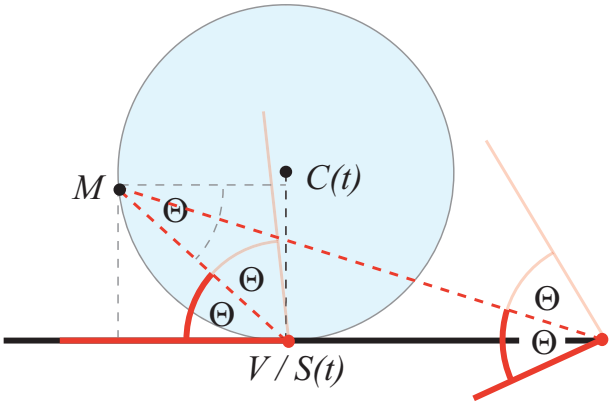


Figure A.2*: The limit case is depicted to the left. On the right a second viewpoint is shows that when it will become maximum viewpoint, the intersection will lie on both rays of the view cone.*

To conclude the detection of the maximum viewpoint for a given side/ray is given by our equation in the case that the angular constraint is satisfied, otherwise there is no maximum viewpoint for this side/ray.

Appendix B

Solving the problem in 3D

As mentioned before, it is Unfortunately not true that the maximum viewpoint corresponds to one of the extremities of the boundary, if the max viewpoint for the hyperplane view cell falls outside. Nevertheless we have shown, that they lie on the border, which leaves us with a one dimensional view cell.

The figure B.1 is an approximation of what the iso-viewpoints on the view cell plane look like. ('approximation' because the values are scaled and clamped, to better show the shape)

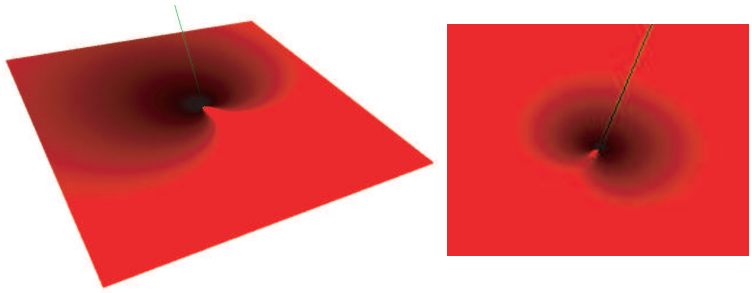


Figure B.1*: The iso-viewpoints on the plane are obtained by intersecting with the growing isotorus. Therefore the resulting shapes seem to form half moons creating ellipsoidal shapes. The brighter the red is the bigger is the abcisse. The maximum viewpoint is therefore the darkest one.*

The problem in the 3D case is, that it is still quite complicated to derive an exact solution. Still it is possible to obtain a numerical solution. From what we have seen, the function on a 3D segment has either one or two minima (two in the case where we cut the half moon shape). The poles are can be classified using lemma 5.2. It would therefore be possible to subdivide the segment, in a way to approach the minima. In our case this would be possible due to the 'simple' shape. There are lots of numerical approaches to achieve this. One example would be the 'golden cut', which would only have to be slightly modified. It has the advantage, that during subdivision the calculated borders could be reused. For more details the interested reader could refer to [Fai01].

For an analytical solution we would have to discuss the function resulting from theorem 5.1. Unfortunately the derivative becomes already quite complex,

still it is possible to transform it into the form of a polynomial, whose roots we would have to find. Unfortunately the degree exceeds four, which had been proven in 1820 by Abel to be the highest degree allowing for a general solution formula and for which Cardano had already found closed form solutions in the 16th century. In 1830 Galois was able to classify the different polynomials which could actually being solved using radicals.[1] Unfortunately we were unable to succeed in solving this final equation and even a last attempt using Maple did not lead to a result. Still it might be possible that a geometric solution could be found. Another approach we tried and which might be an interesting basis for future research is based on the fact that we know how to solve the problem in 2D and there is actually a way to bring the 3D case back to the 2D case. As we know by lemma 5.3, the function is rotationally invariant. Therefore we could imagine to rotate a plane around this axis and describe the resulting graph of the intersection of the segment with the rotating plane. The resulting curve would then be the representation of the 3D segment. Solving the 2D case for this curve would solve the 3D case. Several researchers already worked on the curves resulting from such an intersection, but the time for this master thesis was not sufficient to undertake an exhaustive research in this direction. Other approaches involving distances to a sphere that is then rotated around the invariant axis (The iso-torus could also be seen as the result of a rotating sphere instead of a circle), as well as trying to inverse the approach by first finding angle maximizing points, which are then restricted to form an angle Θ did lead to no success. Either the polynomial had a degree that exceeded four as well, or trigonometric expressions mixed with polynomials, leading to very complicated expressions.

Nevertheless, even the result for planes in the 3D case is useful and applicable. The view cell could be approximated via hyperplanes, culling the geometry accordingly. An approximate, conservative solution could then be obtained applying the developed equations.

One could also imagine to approximate a model instead of a whole scene. The observer moves around the object in this case. Therefore one view cells could be created corresponding to faces of a discretized englobing sphere. Each of these faces could be approximated by a plane which would allow to calculate conservative validity regions for all vertices of the object. Due to the approximation via planes the transition between two cells will also become smoother, as they share common viewpoints.

[1] More information concerning algebra could be found in [vdW]

Appendix C

Restricting the non linear function

This section is very technical and only describes how to classify the extrema obtained in the last section. It can be skipped during the first reading, especially, as the result is not necessary for the algorithm and is more of theoretical and optimization interest.

We arise the question which parts of the function X (equation 6.3) contain the valid information we need to determine whether a simplification plane is valid. First of all we are going to detect the false intersections. So the question is, what part of the segment does not lead to false intersections.

Remembering, that we constructed our function by calculating the intersection between the two lines:

$$l_1(\beta) := M + \beta\vec{d}$$

and

$$l_2(\alpha) := V + \alpha R_\Theta(M - V)$$

We assumed that $\vec{d} = (0,1)^\top$ and $V = (0,0)^\top$. The intersection between the two lines was given for:

$$\frac{m1}{\cos\Theta m_1 - \sin\Theta m_2}$$

A false intersection occurs, when the intersection point does not lie in the direction of M. In other words, when $\alpha < 0$. For the continuous parts of the function we know that it would have to pass via 0 to change the sign. This is only the case when m_1 equals zero. But the function also changes its sign, if the denominator passes via zero. Therefore we will as before replace the point M via a line. With a slight abuse of the notation we will assume that $m_1 = x\cos\gamma$ and $m_2 = m_2 + x\sin\gamma$. The denominator can therefore be transformed:

$$\begin{aligned} & (x\cos\gamma)cos\Theta - (m_2 + x\sin\gamma)\sin\Theta = 0 \\ \Leftrightarrow \quad & \cos(\gamma+\Theta)x = m_2\sin\Theta \\ \Leftrightarrow \quad & x = \frac{m_2\sin\Theta}{\cos(\gamma+\Theta)} \end{aligned}$$

As usual the case where $\cos\Theta+\gamma = 0, \gamma = -\Theta+\pi/2$ has to be treated differently. Here the denominator, as it is constant has no influence on the sign of α. Having these two values the space is cut into three parts, in each of which we have to determine whether it is a part of valid intersection or not. This implies only two possibilities, the space is partitioned into (valid, invalid, valid) or (invalid, valid, invalid). The test which of these two cases we encounter can be deduced from the abcisses.

$$\lim_{x->\infty} \frac{x\cos\gamma}{x\cos(\Theta+\gamma) - m_2\sin\Theta} = \frac{\cos\gamma}{\cos(\Theta+\gamma)}$$

Therefore the sign of the limit implies the order. The two points on the segment where the sign might change are given by $(0, m_2)$ and $(\frac{m_2\sin\Theta\sin\gamma}{\cos(\gamma+\Theta)}, m_2 + \frac{m_2\sin\Theta\cos\gamma}{\cos(\gamma+\Theta)})$.

What remains is the question when the intersection actually lies in the direction we have chosen. Remembering the two lines we intersected:

$$l_1(\beta) := M + \beta\vec{d}$$

and

$$l_2(\alpha) := V + \alpha R_\Theta(M - V)$$

Applying the discussed simplifications ($V = (0,0)^\top$ and $\vec{d} = (0,1)^\top$) we could consider the lines:

$$l_1(\beta) := M + \beta(0,1)^\top$$

and

$$l_2(\alpha) := \alpha R_\Theta(M)$$

We can see that all we are looking for is when the value of β for the intersection is positive. As before, for the value α we will be interested in the points where β becomes zero. Therefore we know that β might change its sign at the points where we encounter zero and at the definition gap. One could imagine this situation like a camera that is following a point on a line. Intuitively there can only be one point where the orientation changes. The situation could be visualized by a camera watching an airplane take off (figure C). The airplane passes right above the camera and the question is, for which moment the airplane will be turned over on the image of the camera. Intuitively there is one point from where on the camera will always see the plane upside down. The moment this happens is when the plane is right above us, therefore we cannot see any of the upper parts of the plane.

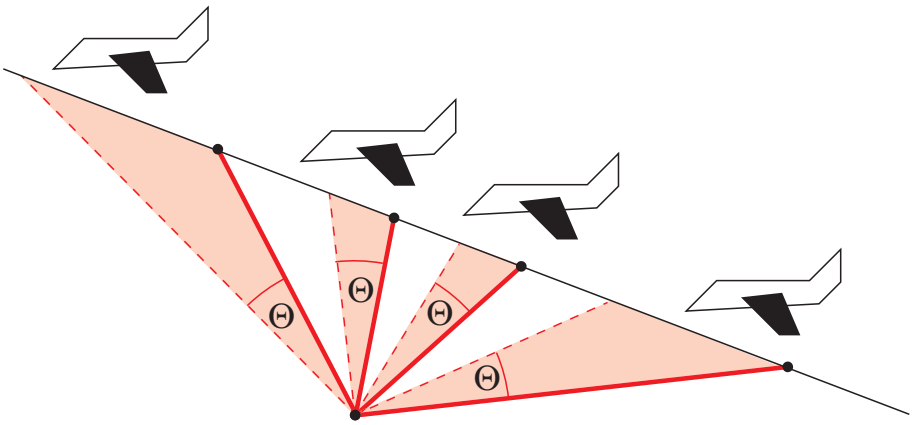
Although it seems intuitive, the proof has to be done. The intersection point between the two lines is given for:

$$\beta = m_1\frac{\sin\Theta m_1 + \cos\Theta m_2}{\cos\Theta m_1 - \sin\Theta m_2} - m_2$$

replacing the point M by a line equation $x(\cos\gamma, \sin\gamma)^\top + (0, m)$ we obtain:

$$\beta(x) = \cos\gamma\frac{\sin\Theta(\cos\gamma) + \cos\Theta(x\sin\Theta + m)}{\cos\Theta(x\cos\gamma) - \sin\Theta(x\sin\gamma + m)} - (m + x\sin\gamma)$$

If this function is positive, we know that it is oriented in the projection direction (β was the coefficient of the projection direction in the second line equation).



One sign change is given at the definition gap, which we already classified. A second sign change can now only occur if $\beta(x)$ has a root.

Transforming the system $\beta(x) = 0$, we will end up with

$$x^2 + 2m \sin\gamma x + m^2 = 0$$

The roots of this polynomial are all complex. The roots are given by:

$$x_{1,2} = m(\sin\gamma \pm \sqrt{\sin\gamma - 1})$$

As $\cos\gamma \neq 0$ (vertical line) we have $\sin\gamma - 1 < 0$ and therefore no real roots.

This finishes the classification. Only one side of the definition gap will be of interest to us. To find the side which is of interest, we will use the same reasoning as before

$$\lim_{x\to\infty} \frac{\beta(x)}{x} = \cos\gamma \frac{\sin\Theta + \gamma}{\cos(\Theta + \gamma)} - \sin\gamma$$

¿From this value one could conclude which part we are interested in. As it still involves several cases (it is not independent of the line orientation), in practice it is more convenient to simply calculate a value of β for each side of the definition gap and compare. This is correct, as we have seen before that the sign only changes at the definition gap. Or one assures the angle of γ to lie in $[0, \pi)$.

Appendix D

Texturing

This section will explain two major things. First we will explain, why a single viewpoint is not sufficient. Secondly we want to explain how the texture size in the view-dependent Billboard Clouds could be further optimized, which is very, seen that lots of objects will be far away from the view cell.

Starting with the first problem, texturing via a texturing viewpoint. This is actually the case for several mesh impostor representations (see chapter 2). First of all, a lot of the geometry will be hidden. Using appropriate clipping planes this problem could only be solved partially. All faces of the model, which align more or less with the view rays of the texturing viewpoint, will not be appropriately represented in the texture.

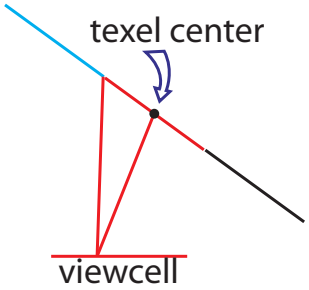
Therefore one the viewpoint should be placed 'perpendicular' to the face from which the texture is taken. A very close texturing viewpoint with perspective projection would lead to a huge distortion in the texture. To minimize this unwanted effect, the camera should be placed as far away as possible. Placing the camera at infinity leads to an orthogonal projection. So one could say that for a face the orthogonal projection creates uniform information. On the other hand we would like to benefit from the fact, that being restricted to a view cell the resolution of far away objects could be smaller than for closer ones.

The selection of a best texturing point in a view cell shows many similarities with the art-gallery problem. Unfortunately the classic version of the problem has been shown to be NP-complete(see [O'R]). It is therefore not very likely that the problem can be solved easily.

We considered two solutions two optimize textures. The first will measure the size of the texels that is necessary to remain below an angular threshold for all viewpoints in the view cell. We further point out how the face could be locally subdivided to benefit from perspective texture transformation that could be used to further improve the storage space. Then we will explain the probably best approach, which is a tradeoff between optimized complexity and texture quality, which for practical applications seems the best solution.

D.1 Minimum Texel Sizes

To assure that the texel size of a texture remains below a certain angular error, we realize, that what we are looking is very similar to the calculations we



encountered in chapter 5 and 6. As before, we will be interested in the angular distance between two points as seen from the view cell. These two points will correspond to the center and the border of a texel. Therefore finding the maximum viewpoint in this situation will lead to a minimum texture resolution, nevertheless the approach described will mostly be of theoretical interest, as a better solution will be described, based on choosing an optimal 'texturing' viewpoint.

To derive the maximum size for a texel, such that for all viewpoints the texel size of the texture remains below a certain threshold the same framework as for the simplification can be used. As seen in the margin the texel border could be considered as a simplification of the texels center. The exact algorithm as before can be applied for a simplification direction inside the face. There remains one problem because we excluded this situation when considering the non-linear part, as it is not a reasonable projection direction. On the other hand it is not very difficult to treat this case separately.

We are interested in the distance between the center of the texel and the ray intersection. Using the equations of chapter 6.3, this leads to the function:

$$Tex(\alpha) = |m1\frac{\sin\Theta m_1 + \cos\Theta\alpha}{\cos\Theta m_1 - \sin\Theta\alpha} - \alpha| = \sin\Theta|\frac{\alpha^2 + m_1^2}{\cos\Theta m_1 - \sin\Theta\alpha}|$$

Therefore the function is positive if the denominator is positive. The denominator is zero for $\alpha = \cot\Theta m_1$. This is the moment where the ray of angle Θ aligns with the vertical direction. As before we will concentrate on the case $\Theta \in (0, \pi/2)$. Nevertheless the same reasoning should be done for $-\Theta$, but it is completely analog. In the case our case we have therefore $\alpha < \cot\Theta m_1$, otherwise, there is no limitation on the texel size. Calculating the derivative leads to:

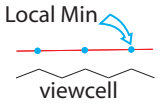
$$\frac{dTex}{d\alpha} = \frac{\alpha^2\sin\Theta + 2\alpha\cos\Theta m_1 + m_1^2\sin\Theta}{(\cos\Theta m_1 - \sin\Theta\alpha)^2}$$

Calculating the extrema we obtain:

$$\alpha_{1,2} = m_1(\cot\Theta m_1 \pm \frac{1}{|\sin\Theta|})$$

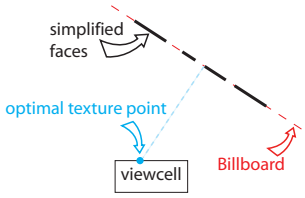
One solution lies outside of the area of interest ($\alpha < \cot\Theta m_1$). Therefore leaving us with a single test to perform whether the minimum lies on the face, if not the minimum must be one of the extremities due to monotony.

D.2 Optimizing the Texture Storage



In theory we would be able to calculate the exact texel size for each point on the face. One might think, that it could be a good idea to perform subdivisions into regions on the face, just as we did for the validity region detection. For each of these regions, we could then approximate the variation via a linear function. Having a texel variation that is linear, we could exploit the texture matrix, to perform a linear transformation of the texture and therefore optimize the storage. Unfortunately the number of subdivisions correspond directly to the number of view cell segments, increasing the number of needed primitives drastically. One has to keep in mind, that using a texture will always be an

approximation in some way, therefore we propose a much easier method which allows to exploit the benefit of perspective transformation of the texture and keeps the simple geometry. We will shoot the texture for the billboard from one single point inside of the view cell. As mentioned before, having a fixed point leads to several artifacts, therefore we should select the texture viewpoint in dependance of the billboard. We mentioned before that it is a good choice to place the camera in front of the billboard. To generalize this idea, one might want to place the camera such that the angle between the billboard normal and the vector between camera and billboard center becomes minimal, as shown in the margin. The billboard center will be the center of the rectangle created by the simplified faces on the billboard.



In general there might not be a viewpoint which creates a zero angle. In this case the texture viewpoint minimizing the angle will correspond to a viewpoint, such that a tangent cone at the view cell can be found with apex in the center of the billboard. Due to the tangency, there cannot be a viewpoint creating a smaller angle. One problem remains, as it is not clear how to project the faces onto the billboard. Projecting the geometry would lead to artifacts, as overlapping faces will result in random pixel values because the depth values coincide. The solution is quite simple. In a first step a temporary billboard texture is created, with a very high resolution. This texture is obtained via a usual orthogonal projection. The perspective texture is then deduced by shooting a texture of the billboard textured with the temporary texture.

The calculation of the optimal texturing viewpoint can even be done in three dimensions. Let's recall, that the optimal position is defined by

$$argmin_{x \in viewcell} angle(x - center, normal(plane))$$

If the line passing through the center in the direction of the plane normal intersects the view cell the optimal positions, correspond to the intersection of the view cell and the line. The more difficult cases are those where the line passes aside. In this case the optimal texturing viewpoint can always be found on the boundary of the faces of the view cell (the idea behind is to grow the cone, until it touches the view cell, either the cone becomes tangent, then it also intersects a boundary, or it touches a boundary already). It has to be pointed out, that finding the closest point to this line in the L_2 norm would be insufficient, as the intersection between a plane and a cone is a quadric and in general not a circle. Let P, Q be the two extremities of one of the front face's segments. We will denote $\vec{n}$ the normal of the billboard plane. Without restriction, we suppose that the scene's center is the coordinate system's origin. As we look for the smallest angle, we actually have a maximization of the cosine between the two vectors. Therefore our problem can be written as:

$$max_{0 \leq \alpha \leq 1} \frac{< (\vec{n} | (P + \alpha(Q - P))) >}{||P + \alpha(Q - P)||}$$

To simplify the calculation we will first derive the denominator:

$$\begin{aligned}\frac{d}{d\alpha}||P+\alpha(Q-P)|| &= \frac{d}{d\alpha}\sqrt{||P+\alpha(Q-P)||} \\ &= \frac{\frac{d}{d\alpha}||P+\alpha(Q-P)||^2}{2*\sqrt{||P+\alpha(Q-P)||^2}} \\ &= \frac{<P+\alpha(Q-P)|(Q-P)>}{||P+\alpha(Q-P)||}\end{aligned}$$

Now we start deriving the real function:

$$\begin{aligned}&\frac{d}{d\alpha}\frac{<\vec{n}|(P+\alpha(Q-P))>}{||P+\alpha(Q-P)||} \\ =\ &\frac{<\vec{n}|(Q-P)>||P+\alpha(Q-P)|| - \frac{<\vec{n}|(P+\alpha(Q-P))><(P+\alpha(Q-P))|(Q-P)>}{||P+\alpha(Q-P)||}}{||P-\alpha(Q-P)||^2}\end{aligned}$$

At a maximum we will encounter a zero derivative, therefore setting this equation equal to zero and multiplying with $||P+\alpha(Q-p)||^3$:

$$<\vec{n}|(Q-P)>||P+\alpha(Q-P)||^2 - <\vec{n}|(P+\alpha(Q-P))><(Q-P)|(P+\alpha(Q-P))>=0$$

Transforming the equation leads to:

$$\begin{aligned}&(<P|(Q-P)><\vec{n}|(Q-P)> - <\vec{n}|P>||Q-P||^2)\alpha \\ &+<\vec{n}|(Q-P)>||P||^2 - <\vec{n}|P><P|(Q-P)>=0\end{aligned} \tag{D.1}$$

It is now interesting to see when we have a solutions. Therefore we examine the coefficient of α. We want to see when

$$<P|(Q-P)><\vec{n}|(Q-P)> - <\vec{n}|P>||Q-P||^2 = 0 \tag{D.2}$$

One case is $Q-P=0$ which is impossible as it represents a segment of the view cell. If P equals zero the whole system becomes trivial. As this means that the segment is actually passing through the billboard. (Remember the assumption that the billboard center is at the origin) Dividing equation D.2 by $||Q-P||^2||P||$ leads to:

$$<\frac{P}{||P||}|\frac{(Q-P)}{||Q-P||}><\vec{n}|\frac{(Q-P)}{||Q-P||}> - <\vec{n}|\frac{P}{||P||}>=0$$

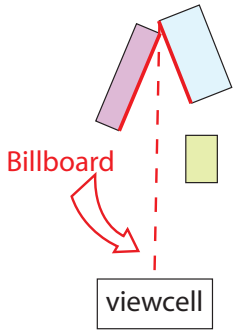
We realize that we now have only normalized vectors in this equation. Rewriting the expression (where the subscript n refers to the fact, that the vectors are normalized), leads to:

$$<\vec{n}|<P_n|(Q-P)_n>(Q-P)_n - P_n>=0 \tag{D.3}$$

, the right hand vector could geometrically be interpreted as a the inverse of the projection of P_n onto the orthogonal space to $(Q-P)_n$. Interestingly the expression for the part without α in the equation D.1 could be written as a projection of $(Q-P)_n$ onto P_n. Supposing that the coefficient before α vanishes, that is the equation D.3is true, we can write $n_n = \gamma(Q-P)_n + \delta(P_n \times (Q-P)_n)$, as the two vectors are also orthogonal and therefore form a basis. Examining the

part without α from the main equation D.1, we deduce that either γ equals zero, or the equation has no solution. The case $||(Q-P)_n||^2 - ((Q-P)_n * P_n)^2 = 0$ means that the segment aligns with the billboard center, in the closest point to the billboard would do. The case where γ equals zero corresponds to a billboard which passes through the segment, therefore all points on the segment are identically bad chosen because from no viewpoint of this segment, anything on the billboard will be visible. On the other hand being aligned with a segment means that there is at least one other segment, for which the angle is smaller and through which the plane does not pass. Comparing the angles obtained for all the segments of the view cell leads to the final optimal texturing point.

Other possibilities to improve the size for walkthroughs with several view cells, would be an on-the-fly texture creation like the one described in [DSSD99]. A first approximation could be given by keeping first of all the textures of the last view cell projected on the new billboards. A low-res ground texture could have been saved to fill up eventual holes. Then one by one the textures are calculated and replaced, an order could also be fixed off-line to replace first of all the most important textures, or the ones, that could be calculated rapidly etc.



D.3 Improving Quality

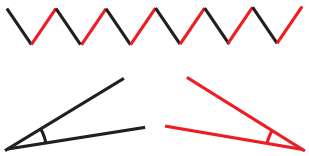
It is possible to further enhance the quality of the billboard textures by taking the orientation of the billboard to the observer into account. A typical example where this could become useful is depicted in the margin. Both faces with different colors are simplified on the same region of the plane because they remain at a distance smaller than Θ. Therefore the error is measurable, but on the other hand it is impossible to choose a color which would be convenient for the pixels on the billboard. To solve this problem, we could associate two textures for each side of the billboard. Depending on the current viewpoint the according texture is chosen. Like for other on-the-fly simplifications, billboard clouds could be extended in this way to adapt in a similar way.

A different idea concerning billboard clouds in general would be to save a height field on the texture, too. It would then be possible to create an output sensitive relief of the original geometry ([OBM00]). The error bound would remain valid, as points are actually just approaching its original representation. But this remains a future work to explore.

Other possible avenues include the view-dependent texture mapping by Debevec et al. ([DTM96]).

Appendix E

Appearance vs. Geometry



This section underlines the difference between closeness to geometry and actual closeness to appearance.

The image in the margin shows a typical example. For different viewpoints the appearance of the scene might completely change. Projecting all the points on a central plane would make, for certain viewpoints, appear parts, which were hidden in the original model. Therefore the perceived difference might be arbitrarily huge.

To overcome this general problem, lots of articles can be found. They all parameterize the so-called plenoptic function. It depends on the position of the observer and the viewing direction[1]. It encodes exactly what a ray shot from the observers position in the viewing direction would encounter. Therefore the approaches become completely independent of the geometry of the scene. (see [LH96]).

Point-based impostors [WWS01] is an approach which combines geometry and plenoptic function. The far away objects are represented by point clouds, where each point is associated a color function that varies depending on the observers position. Unfortunately the function is obtained using a numerical approach, based on Monte Carlo ray-tracing. Therefore the absolute error cannot be bounded. Nevertheless for reasonable scenes the method will succeed and also solve the problem of aliasing, that appears when several triangles of different colors fall into the same pixel.

[1] In the original formulation it also takes time into account.

Bibliography

[ACSD+03] Pierre Alliez, David Cohen-Steiner, Olivier Devillers, Bruno Levy, and Mathieu Desbrun. Anisotropic polygonal remeshing. pages 485–493, 2003.

[AL99] Daniel G. Aliaga and Anselmo Lastra. Automatic image placement to provide a guaranteed frame rate. In Alyn Rockwood, editor, *Siggraph 1999, Computer Graphics Proceedings*, pages 307–316, Los Angeles, 1999. Addison Wesley Longman.

[ALSS] Pierre Alliez, Nathalie Laurent, Henri Sanson, and Francis Schmitt. Efficient view-dependent refinement of 3d meshes using sqrt(3)-subdivision. volume 19(4.

[BMK] Á. Balázs, M.Guthe, and R. Klein.

[BP96] Normand Brière and Pierre Poulin. Hierarchical view-dependent structures for interactive scene manipulation. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 83 – 90. ACM Press, 1996.

[CK01] Jatin Chhugani and Subodh Kumar. View-dependent adaptive tessellation of spline surfaces. In *Symposium on Interactive 3D Graphics archive, Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 59 – 62, 2001.

[Cla76] J. Clark. Hierarchical geometric models for visible surface algorithms. In *Communications of the ACM 19(10)*, 1976.

[CRL01] Derek Cornish, Andrea Rowan, and David Luebke. View-dependent particles for interactive non-photorealistic rendering. In B. Watson and J. W. Buchanan, editors, *Proceedings of Graphics Interface 2001*, pages 151–158, 2001.

[CSAD04] David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun. Variational shape approximation. In *Proceedings of the ACM Siggraph*. ACM Press, 2004.

[CVM+96] Jonathan Cohen, Amitabh Varshney, Dinesh Manocha, Greg Turk, Hans Weber, Pankaj Agarwal, Frederick Brooks, and William Wright. Simplification envelopes. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. ACM Press, 1996.

[DDSD03] Xavier Décoret, Frédo Durand, François Sillion, and Julie Dorsey. Billboard clouds for extreme model simplification. In *Proceedings of the ACM Siggraph*. ACM Press, 2003.

[Deu03] Oliver Deussen. Fast modeling and rendering of plants. In *Course notes Siggraph 2003: Simulating Nature: Realistic and Interactive Techniques*, 2003.

[DSSD99] Xavier Décoret, François Sillion, Gernot Schaufler, and Julie Dorsey. Multi-layered impostors for accelerated rendering. *Computer Graphics Forum*, 18(3):61–73, September 1999. ISSN 1067-7055.

[DTM96] Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. Modeling and rendering architecture from photographs: a hybrid geometry- and image-based approach. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 11–20. ACM Press, 1996.

[ESSS01] Jihad El-Sana, Neta Sokolovsky, and Claudio T. Silva. Integrating occlusion culling with view-dependent rendering. In *Proceedings of the conference on Visualization '01*, pages 371 – 378. IEEE Computer Society, 2001.

[ESV99] J. El-Sana and A. Varshney. Generalized view-dependent simplification. In *Computer Graphics Forum*, 1999.

[Fai01] Ulf Faigle. *Skript der Vorlesung Numerik*. University of Cologne, 2001.

[Fre00] Pascal J. Frey. About surface remeshing. In *Proceedings, 9th International Meshing Roundtable*, pages 123–136, October 2000.

[GDL+02] Benjamin Gregorski, Mark Duchaineau, Peter Lindstrom, Valerio Pascucci, and Kenneth I. Joy. Interactive view-dependent rendering of large isosurfaces. In *Proceedings of the conference on Visualization '02*, pages 475 – 484. ACM Press/Addison-Wesley Publishing Co., 2002.

[GGSC96] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 43–54. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[Gol97] Dan B Goldman. Fake fur rendering. In *Proceedings of the ACM Siggraph*. ACM Press, 1997.

[GwHC] Steven Gortler, Li wei He, and Michael F. Cohen. Rendering layered depth images. Tech Report MSTR-TR-97-09.

[Her] M. Herf. Efficient generation of soft-shadow textures. In *Technical Report CMU-CS-97-138*.

[Heu] Harro Heuser. *Lehrbuch der Analysis I.* Teubner.

[Hop96] Hugues Hoppe. Progressive meshes. *Computer Graphics*, 30(Annual Conference Series):99–108, 1996.

[Hop97a] H. Hoppe. View-dependent refinement of progressive meshes. In *Computer Graphics (SIGGRAPH'97 Proceedings)*, 1997.

[Hop97b] Hugues Hoppe. View-dependent refinement of progressive meshes. In *Proceedings of SIGGRAPH '97*, *Computer Graphics* Proceedings, Annual Conference Series, pages 189–198. ACM SIGGRAPH, August 1997.

[HOU] HOUGH.

[JW02] Stefan Jeschke and Michael Wimmer. Textured depth meshes for real-time rendering of arbitrary scenes. In *Proceedings of the 13th Eurographics workshop on Rendering*, pages 181 – 190. Eurographics Association, 2002.

[JWS02] Stefan Jeschke, Michael Wimmer, and Heidrun Schuman. Layered Environment-Map Impostors for Arbitrary Scenes. In *Proc. Graphics Interface*, pages 1–8, May 2002.

[LE97] David Luebke and Carl Erikson. View-dependent simplification of arbitrary polygonal environments. *Computer Graphics*, 31(Annual Conference Series):199–208, 1997.

[LH96] Marc Levoy and Pat Hanrahan. Light field rendering. *Computer Graphics*, 30(Annual Conference Series):31–42, 1996.

[LKR+96] P. Lindstrom, D. Koller, W. Ribarsky, L. Hodges, N. Faust, and G. Turner. Real-time continuous level of detail rendering of height fields. pages 109–118, 1996.

[LN04] Sylvain Lefebvre and Fabrice Neyret. Streaming textures. 2004.

[LT00] Peter Lindstrom and Greg Turk. Image-driven simplification. *ACM Transactions on Graphics*, 19(3):204–241, 2000.

[MN00] Alexandre Meyer and Fabrice Neyret. Multiscale shaders for the efficient realistic rendering of pine-trees. In *Graphics Interface*, pages 137–144. Canadian Information Processing Society, Canadian Human-Computer Communications Society, 2000.

[NBG02] Shaun Nirenstein, Edwim Blake, and James Gain. Exact from-region visibility culling. In *Rendering Techniques '02 (Proceedings of the 13th EG Workshop on Rendering*, Springer Computer Science, pages 191–202. Eurographics, Eurographics Association, 2002.

[Ney00] Fabrice Neyret. A phenomenological shader for the rendering of cumulus clouds. Technical Report RR-3947, INRIA, 2000.

[OBM00] Manuel M. Oliveira, Gary Bishop, and David McAllister. Relief texture mapping. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 359–368. ACM Press/Addison-Wesley Publishing Co., 2000.

[Ola02] Olaf Hall-Holt. Kinetic visibility. 2002.

[O'R] J. O'Rourke. Handbook of discrete and computational geometry.

[RCRB] I. Remolar, M. Chover, J. Ribelles, and O. Belmonte. View-dependent multiresolution model for foliage.

[RMD04] Alex Reche, Ignacio Martin, and George Drettakis. Volumetric reconstruction and interactive rendering of trees from photographs. *ACM Transactions on Graphics (SIGGRAPH Conference Proceedings)*, 23(3), July 2004.

[SDB97] François Sillion, George Drettakis, and Benoit Bodelet. Efficient impostor manipulation for real-time visualization of urban scenery. In D. Fellner and L. Szirmay-Kalos, editors, *Computer Graphics Forum (Proc. of Eurographics '97)*, volume 16, Budapest, Hungary, September 1997.

[Sim94] R.B. Simpson. Anisotropic Mesh Transformations and Optimal Error Control. *Appl. Num. Math. 14(1-3), 183.198.*, 1994.

[SP] François Sillion and Claude Puech. *Radiosity and Global Illumination*. Morgan Kaufmann Publishers, San Francisco, 1994.

[Sta04] Florian Stangl. Doom 3. In *Gamestar 1/04*, 2004.

[Tur92] Greg Turk. Re-tiling polygonal surfaces. In *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 55–64. ACM Press, jul 1992.

[Tur01] Greg Turk. Texture synthesis on surfaces. In *Proceedings of the ACM Siggraph*. ACM Press, 2001.

[vdW] Bartel L. van der Waerden. *Algebra*. Springer.

[WLC+03] Nathaniel Williams, David Luebke, Jonathan D. Cohen, Michael Kelley, and Brenden Schubert. Perceptually guided simplification of lit, textured meshes. In *Proceedings of the 2003 Symposium on Interactive 3D Graphics*, 2003.

[WM03] Andrew Wilson and Dinesh Manocha. Simplifying complex environments using incremental textured depth meshes. In *Proceedings of the ACM Siggraph*. ACM Press, 2003.

[WWS01] Michael Wimmer, Peter Wonka, and François Sillion. Point-based impostors for real-time visualization. In *EuroGraphics Workshop on Rendering*, 2001.

[WWT+03] Lifeng Wang, Xi Wang, Xin Tong, Ligang Liu, Baining Guo, and Heung-Yeung Shum. View-dependent displacement mapping. In *Proceedings of the ACM Siggraph*. ACM Press, 2003.

[XV96] Julie C. Xia and Amitabh Varshney. Dynamic view-dependent simplification for polygonal models. In *Proceedings of the conference on Visualization '96*, pages 327–334. IEEE-CS Press, October 1996.