FAKULTÄT FÜR !NFORMATIK

# Implementation of an Improved Billboard Cloud Algorithm

DIPLOMARBEIT

zur Erlangung des akademischen Grades

**Diplom-Ingenieur**

im Rahmen des Studiums

**Computergraphik & Digitale Bildverarbeitung**

eingereicht von

**Christian Luksch**
Matrikelnummer 0525392

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung:
Betreuer: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Mitwirkung: Dipl.-Ing. Dr.techn. Robert F. Tobler

Wien, 24.07.2009

(Unterschrift Verfasser) (Unterschrift Betreuer)

**Christian Luksch**

Bahnstraße 27a
2202 Enzersfeld

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 24.07.2009

_____

(Unterschrift)

# Abstract

Real-time rendering is a huge field with many applications. Large scale urban visualization or outdoor game environments set highest demands on the visual complexity of the virtual world. The frame rate on the other hand has to stay above 60 frames per seconds, otherwise movements and animations are not perceived smoothly anymore. Even though hardware gets faster every year, geometry simplification is essential to distribute the resources for optimal quality and performance.

This thesis gives an overview of geometry simplification algorithms based on the billboard cloud principle [DSDD02]. The problematic of the summarized algorithms are discussed and several improvements and possible modifications are described. The focus is on reducing simplification artifacts and improving the performance without affecting the visual quality. Finally, the results will be presented in an extensive evaluation. From the gained insight rules and parameters for an automatic geometry simplification process will be derived.

# Kurzfassung

Das Gebiet der Echtzeitgrafik hat viele Anwendungen. Visualisierungen von großen Städten oder weitflächigen Landschaften stellen höchste Ansprüche an die visuelle Komplexität der virtuellen Welt. Dabei muss die Bildwiederholrate über 60 Bilder pro Sekunde bleiben, da sonst Bewegungen und Animationen nicht mehr fließend wahrgenommen werden. Obwohl die Hardware jedes Jahr immer leistungsfähiger wird, sind Geometrievereinfachungen ein wichtiger Bestandteil, um die Ressourcen für optimale Qualität und Leistung zu verteilen.

Diese Diplomarbeit präsentiert eine Übersicht von Geometrievereinfachungsverfahren basierend auf dem Billboad Cloud Prinzip [DSDD02]. Anschließend wird die Problematik der zusammengefassten Verfahren erörtert und zahlreiche Verbesserungen und mögliche Änderungen beschrieben. Der Schwerpunkt dabei ist auf Reduzierung der Vereinfachungsartefakte und Verbesserung der Performance ohne dabei die visuelle Qualität zu beeinflussen. Abschließend werden die Ergebnisse in einer umfangreichen Auswertung präsentiert. Aus den dabei gewonnenen Erkenntnissen werden Regeln und Parameter für einen automatischen Geometrievereinfachungsprozess abgeleitet.

# Contents

## Chapter 1

# Introduction

In computer graphics, a challenging problem is to visualize large environments in real time with the highest possible level of detail. Geometry simplification is an important process to get suitable versions of objects allowing to balance performance and quality.

The displayable level of detail of an object is proportional to its size on the screen, which is proportional to the distance to the viewer in the virtual world. So it is obvious to use different versions for certain distances. Another reason to use simplifications is to reduce aliasing artifacts. They occur when highly detailed objects are rendered in low resolution whereby details smaller than a pixel would be lost. The simplifications should be as accurate as possible and give the same visual expression as the original model viewed in miniature.

This is achieved by the traditional *level of detail* (LOD) approach. It is important that the user does not notice when the representation of an object changes between two levels of detail. Another problem is to decide which level of detail should be used for which distance. Besides that, the levels of detail have to be generated somehow. One possibility is to let the modeler make them by hand. This is very time consuming and later changes of the model are complicated. An automatic process to generate the levels of detail would be ideal.

There have been lots of research in this field that brought up several techniques to automatically generate simplifications. The next section gives an overview of different classes of techniques and point out their properties.

## 1.1 Geometry Simplification Overview

Geometry-based simplification methods work directly on primitives. They usually seek for local optima and do some collapsing or merging based on that, while trying to main-

tain the original form. An extensive survey of such techniques is given by Heckbert and Garland in [HG97]. For certain cases such methods work very well, but on the other hand they have their limits. Extreme simplifications often look very deformed and meshes are losing their shape. Many modeling programs already have such features implemented, so they can support the artist with creating the levels of detail. Another problem comes with textured meshes, they are even harder to simplify with such methods. Textures are usually mapped to the geometry via coordinates stored at the vertices, which then get interpolated over the face. Vertices might also have several coordinates, if they are shared by faces that have different textures. So it is hard to avoid distortions of the texture when vertices are removed or merged during the simplification process. Additional deskewed and combined textures have to be generated. Generally, geometry-based methods have restrictions regarding the geometry. They need to be connected for the algorithms to work, since they only simplify locally by reducing the triangles one by one. This makes them impossible to use with meshes that have very sparse geometry. For example, most plant and vegetation meshes are built of a bunch of faces with foliage textures mapped on them and thin branch geometry.

Vertex clustering methods partition the vertex set into cells and collapse them to reduce the geometry. The advantage is that such methods do not require complete connectivity information of the vertices. A simple algorithm using this method is described by Rossignac and Borrel in [RB92]. Clustering methods reduce the complexity of the geometry simplification and allow a fast and efficient geometry reduction. However, such methods may result in substantial changes of the topology and often generate relatively poor approximations.

Image-based methods use textures to replace complex geometry and therefore are very versatile. Such methods became more and more interesting since graphics hardware is very fast in rendering geometry with textures and has large amounts of dedicated memory. With these methods only a few vertices have to be transformed and the fill-rate cost is proportional to the number of pixels on the screen. Additional hardware texture filtering resolves aliasing artifacts as well and comes almost for free. They also have no restrictions on the input geometry. Unconnected faces as well as points or volumetric effects can be used. Simple techniques like *Billboards* or *Imposters* use a single textured quad for the whole object representation. They are very easy to implement and have been widely used in games. However, because the actual geometry is flat, the three-dimensional look of the substituted object is lost, which will be noticed when the viewer moves his position. This is not sufficient for today's demands on accuracy and detail. A fairly new approach called

*Billboard Clouds* [DSDD02] extends the very naive simplification by billboards with a geometric optimization step that simplifies the original geometry to a set of billboards. Billboard clouds seem to be a very effective and relatively accurate simplification. An optimization process approximates the three-dimensional shape of the original geometry with a set of billboards. The rendering costs of billboards are not very demanding on today's graphics hardware and therefore make billboard clouds very efficient. The extreme geometric simplification is achieved in tradeoff with texture memory, whereby even a low texture resolution is already sufficient, since simplifications tend to be viewed in miniature.

## 1.2 Goal of this Thesis

The main goal of this thesis is to implement a geometry simplification process based on the billboard clouds principle. It should be capable of simplifying a wide range of geometries with different properties. The final implementation should choose most parameters to generate suitable billboard clouds automatically. This allows a fast and easy integration into other applications.

The improvements we are focusing on are reducing the artifacts and enhancing the visual quality of billboard clouds. Since billboard cloud generation is very time consuming, we want to obtain best possible performance. Simultaneously these optimizations should not introduce additional errors.

## 1.3 Structure

In Chapter 2 we describe the principle of billboard clouds and give an overview of the state of the art of billboard cloud simplification algorithms and improvement techniques.

In Chapter 3 the background and suggested improvements are discussed in detail.

In Chapter 4 important details of our implementation are described.

In Chapter 5 we evaluate several aspects of billboard clouds and present the simplifications generated using our implementation.

In Chapter 6 we briefly summarize this thesis and list unanswered questions and ideas to improve billboard clouds.

Chapter 2

# State of the Art

The principle of billboard clouds has been introduced in [DSDD02]. The simplification process uses an optimization algorithm to reduce the input primitives to a set of planes. Each plane is rendered as a textured quad that replaces the geometry mapped by the plane. All the planes together approximate the shape of the original geometry.

The difficulty of this simplification is to find a minimum set of planes, which results in a complex optimization problem. Independent of the algorithm, this is a very computationally expensive and time consuming task and can only be done in a preprocessing step. However, the real difficulty is to preserve the visual accuracy of the simplified representation.

## 2.1 The Original Billboard Cloud Algorithm

Décoret et al. [DSDD02] presented an error-based construction algorithm with the billboard cloud idea and later an improved version in [DDSD03]. The generation is based on an error function and a cost function. The error function is defined in object-space and is view-independent. The cost function involves the number of planes and the compactness of textures. As input parameter a maximum tolerable geometric error $\epsilon$ is set. Then the algorithm builds the billboard cloud respecting this threshold and minimizing the total costs. A major benefit of the billboard cloud simplification is that it can be adapted to any input primitive. The general term *face* will be used for that.

### Density in Plane Space

The plane selection is performed by a greedy optimization, whereby iteratively the plane with the highest contribution is selected. To limit the number of planes, the selection is made in a discretized plane space, that is defined by plane orientation and offset. The plane orientation is parameterized using spherical coordinates $(\theta, \varphi)$ and the offset $\rho$ is the distance to the center of the input geometry, defining a three-dimensional space. A

discretized cell in this space is called *bin*. This is the three-dimensional equivalent of the Hough transform [DH72]. The singularities at the poles and the non-uniform distribution, which results in partly oversampling the plane space, have to be accepted.

A so-called density value is calculated for each bin of the discretized plane space that indicates important planes. The planes are then selected based on this density value. To define density three notions are used: *validity, contribution* and *penalty*. Validity is binary and simply tells if a plane is a valid simplification for a face, enforcing the error threshold $\epsilon$. Contribution includes the quality of the simplification limited to valid faces. Additional, penalty is used to prevent undesirable planes. From these notions density $d$ for an arbitrary plane $P$ is formulated as:

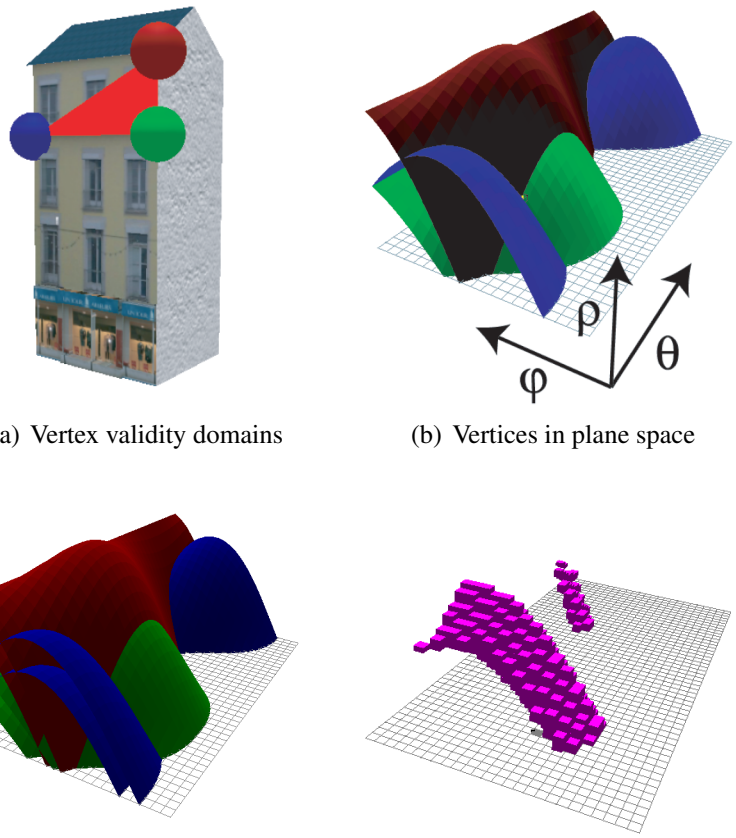
$$d(P) = Contribution(P) - Penalty(P) \tag{2.1}$$

Since a bin actually represents a certain range of planes, this discretization has to be considered when the density is calculated for a bin. An extended validity formulation is required, which will be relevant later when the rasterization is discussed.The next paragraphs discuss the choices of validity, contribution and penalty.

## Validity

The validity ensures that the error threshold is maintained. Therefore, in a view-independent case a point is allowed to move a maximum distance of $\epsilon$. This means a plane that intersects the spherical region with a radius of $\epsilon$ around a point is a valid simplification for that point. That also implies a certain so-called validity domain, which covers all valid simplification planes for the point. In object space this validity domain simply is a sphere around the point as illustrated in Figure 2.1 (a). The duality of this validity domain in plane space is the region limited by the duality of the vertices $\pm\epsilon$ visualized in Figure 2.1 (c).

The validity of faces is now given if all its vertices are valid for a certain plane. Hence, the validity domain of a face is the intersection of the validity domains of all its vertices in plane space. In the example of Figure 2.1 this is illustrated in image (d). It can be interchangeably said, that a face is valid for a plane and that a plane is valid for a face.




(a) Vertex validity domains (b) Vertices in plane space

(c) Vertex validity domains (d) Intersection - face validity domain

Figure 2.1: The spheres in image (a) show the validity domain of the three vertices of a triangle. In image (b) the duality of the points in plane space is visualized, whereby the surfaces represent all possible planes going though the points. Image (c) illustrates the corresponding validity domains of the vertices in plane space. The visualized bins in image (d) show where the validity domains intersect and therefore are valid simplification planes for the triangle. [DDSD03]

## Contribution

The contribution of a plane can be formulated as sum of all faces that can be validly simplified by it. It would not be a good estimate to take just the number of faces, since this would prefer a plane that maps severals small faces instead of one with only a few large valid faces. A more accurate estimation of the plane contribution is the area of the orthographic projection of all valid faces. Thereby, large faces have more influence on the plane selection and it also favors planes parallel to the faces. The contribution in equation 2.1 for a plane is then:

$$Contribution(P) = \sum_{valid(P)} area_P(f) \tag{2.2}$$

## Penalty

The notion of contribution would be sufficient to find a single plane that is the best simplification for the input geometry. However, a greedy selection may result in a poor overall performance. Therefore, an additional penalty prevents planes that miss primitives in their neighborhood.

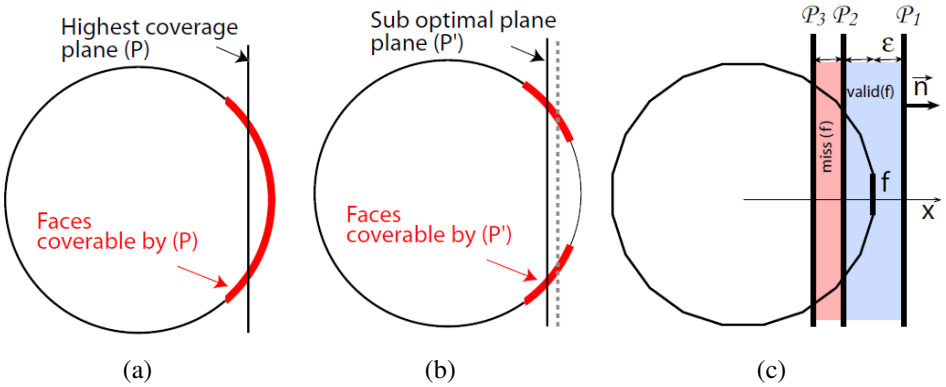



Figure 2.2: Problem of near-tangency countered by penalty. [DDSD03]

In the case of a sphere, as illustrated in Figure 2.2, the plane with the highest contribution would exactly cover a spherical lens (a). Due to limited tessellation of the geometry and discretization of the plane space the selection may end up with a suboptimal plane in practice (b) and therefore miss a small region that is hard to simplify afterwards. To prevent this, each face also penalize planes that would miss it. Considering the set of planes with a certain orientation, we get a certain validity domain $[\rho_{min}, \rho_{max}]$ for a face. Planes slightly missing that interval will be penalized. A suitable width for the penalty region is $\epsilon$. It is important that the penalty is only applied to the lower $\rho$-side, otherwise it would be canceled out when applied symmetrically. The one sided penalty also favors planes more distant to the origin, which helps to preserve the outline of the input geometry. Figure 2.2 (c) illustrates how a face $f$ affects planes with a certain orientation $\vec{n}$ in its neighborhood.

Therefore, the penalty for a plane is formulated similar as the contribution, but with higher weight $w_{penalty}$:

$$Penalty(P) = w_{penalty} \sum_{miss(P)} area_P(f) \tag{2.3}$$

In practice, a weight of $w_{penalty} = 10$ should be large enough to strongly prevent the choice of planes that miss faces.

## Rasterization

The first part of the original billboard could algorithm is to compute the densities on which the greedy selection is based on. The plane space has been discretized, which results in a three dimensional grid defined by the plane orientation $(\theta, \varphi)$ and the offset $\rho$. In other words, the process of calculating the densities can be described as rasterizing the input geometry in plane space. A naive approach would just sample the density at the center of each bin, but thereby not all contribution is gathered and some details may be overlooked. Therefore, the validity between bin and face has to be extended. The new formulation is called *simple validity* and says that a face is valid in a bin if it contains a plane that is a valid simplification for that. Thereby, a bin represents the contribution of its entire planes, but it has to be considered that not all faces have a shared valid plane in the bin. Hence, this density value only guides the selection to a range of planes with a large number of valid faces, where usually a plane with a large subset of valid faces should be found.

For a certain plane orientation $(\theta, \varphi)$ a face has a certain interval of $[\rho_{min}, \rho_{max}]$ where it is valid. This allows to iterate over all orientations, whereby the validity range in $\rho$ is calculated and accumulated in the grid for each face. Bins with the same orientation are bound by four orientations $(\theta_i, \varphi_i)$. To compute this validity interval for this subset of bins, a simple approximation of the validity domain is used. First the validity interval at each corner $i$ is calculated by intersecting the validity domains of all the face vertices. Then the union of the corner validity intervals is the validity interval $[\rho_{min}, \rho_{max}]$ over the bins.

Now the face penalty and contribution, which are constant for a given direction and therefore have to be calculated only once, are added to the bins covering the interval $[\rho_{min}, \rho_{max}]$ for the contribution and $[\rho_{min} - \epsilon, \rho_{min}]$ for the penalty. To account for aliasing, they are weighted by the percentage the bins are covered. Figure 2.3 illustrates how the contribution and penalty are accumulated in the density grid.

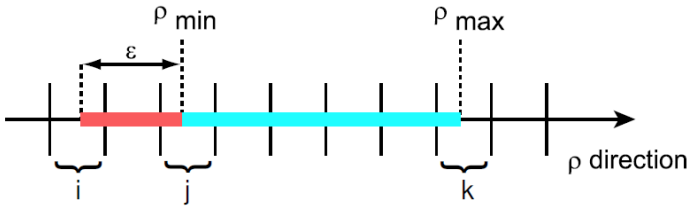



Figure 2.3: Rasterization of contribution (cyan) and penalty (red) between the interval $[\rho_{min}, \rho_{max}]$ for a certain orientation in plane space. [DDSD03]

## Greedy Optimization and Refinement

After the density of all bins in plane space is computed, regions with high density indicate highly contributing planes. Figure 2.4 shows a visualization of such a density grid. A greedy optimization algorithm is used to find a set of planes to approximate the input geometry. Iteratively the bin with the highest density is selected. Then a plane that approximates the corresponding valid faces is calculated. Since the *simple validity* formulation has been used within the density grid and therefore the validity of bins is not equivalent to the validity of planes, the plane needs to be searched in an adaptive refinement of the bin.

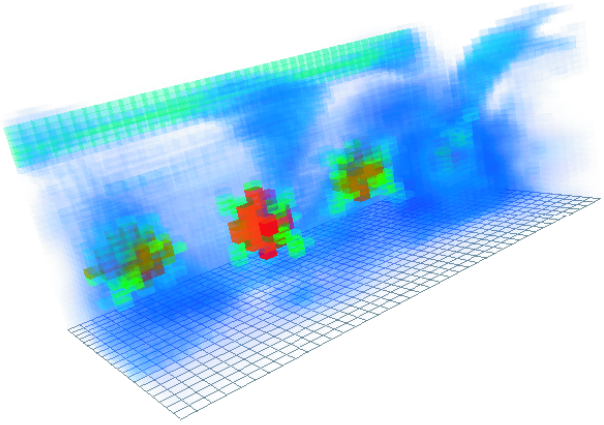


Figure 2.4: Visualization of the initialized density grid. Red areas indicate planes with high contribution. [DDSD03]

An illustrative example where the densest plane is not in the original selected densest bin is shown in Figure 2.5.

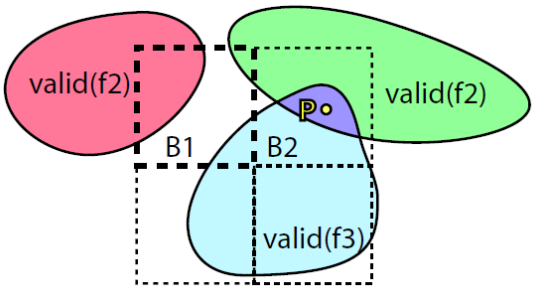



Figure 2.5: The simple density formulation leads to the densest bin $B_1$, although the densest plane $P$ can be found in $B_2$. [DDSD03]

When a bin is selected the refinement is limited to the subset of faces valid for the bin. The refinement continues until the plane going through the center of the refined bin also is a valid simplification for all the faces. If it is necessary to refine the bin, the bin is subdivided into eight sub-bins. Because the densest plane may be slightly outside, the refinement also considers the 26 neighbor bins. Then for each sub-bin the density is computed for the given subset of faces. The sub-bin with the highest density is selected and the refinement continues.

A problem not mentioned by the authors is that refinement can lead to an empty bin, whereby the algorithm ends up in an infinite loop. Fuhrmann et al. [FMU05] suggest a fail-safe extension to this process. In case the greedy algorithm fails to find a suitable plane, the original densest bin is reviewed. Two planes, the plane represented by the bin center and a best fit plane of the bin faces, are evaluated. Then the plane with the highest contribution is selected.

Once a plane is found the contribution and penalty of the mapped faces is subtracted from the density grid. The whole process, starting by selecting the densest bin from the density grid, is continued until all input faces are mapped to a plane.

### Texture Generation

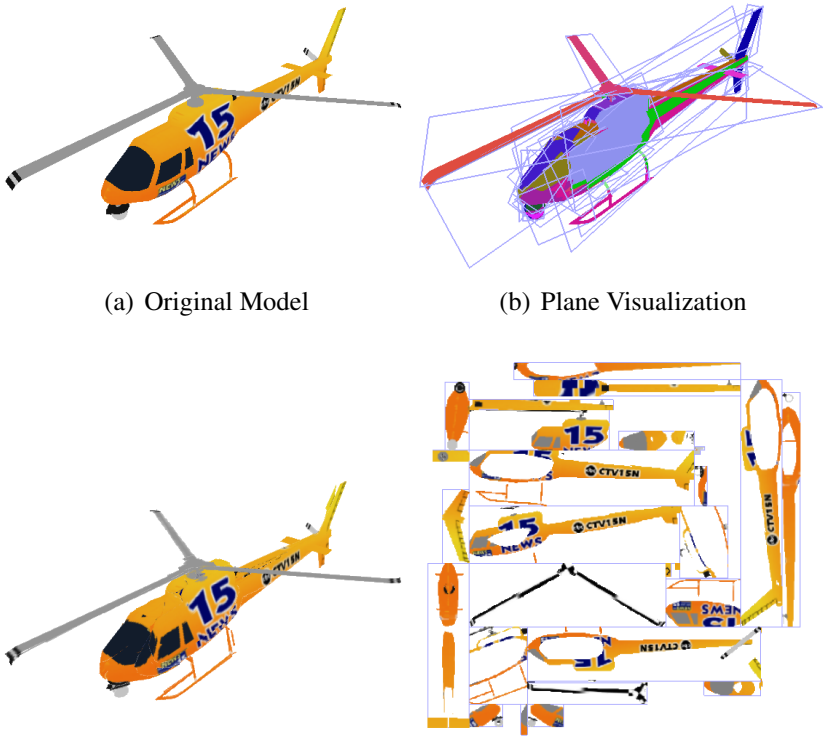
Having the planes and its mapped faces, their textures can be generated. First the plane extent has to be calculated, which is given by the minimum bounding rectangle of the projected faces. The size of the bounding rectangle also gives the texture resolution. To generate the plane texture, the input geometry is simply rendered into a render target texture, by focusing on the bounding rectangle and limiting the valid region with clipping planes. Thereby, automatically adjacent faces are partly rendered into the texture, which is a desired side effect to minimize cracks between the billboards. However, in most cases this is not sufficient and still lots of disturbing cracks remain. Improved approaches of reducing such artifacts are discussed in Section 2.4.

Furthermore, textures may contain large empty regions, because the density formulation does not account for widely separated faces. The authors suggest a modification of the greedy algorithm, which changes the definition of validity to the faces of the maximal compact subset [DDSD03]. Thereby, the original validity set of a bin is clustered and the highest contributing cluster is selected to start the bin refinement. For this task a bucket-like partitioning algorithm, like described by Cazals et al. [CDP95], is used. Faces which

are not simplified because they have been excluded during the clustering will be handled by subsequent iteration of the greedy selection. The algorithm may select the same plane several times to map different clusters.

## 2.2 Billboard Cloud Demonstration

All required tasks to generate a billboard cloud according the authors of the original algorithm have been discussed. Figure 2.6 gives an example of a billboard cloud simplification. Image (a) shows the original geometry with 5,138 faces. Apparently the viewed resolution is much to low too see all the details. The plane search algorithm returned 32 planes that map all faces of the input geometry visualized in separate colors for each plane in (b). Together the textured billboards approximate the original geometry accurate enough to replace the original with the simplification without noticing major differences (c). Image (d) shows the rendered billboard textures side-by-side.



(a) Original Model (b) Plane Visualization

(c) Billboard Cloud (d) Plane Textures

Figure 2.6: Example of the billboard cloud simplification: (a) Original model with 5138 faces; (b) Faces visualized color-coded by their simplification plane; (c) Billboard cloud simplification with the 32 found planes; (d) Generated billboard textures. [DDSD03]

## 2.3 Alternative Generation Algorithms

The problem with the suggested greedy algorithm is that the result is hard to predict and parameters giving a good simplification for one model do not imply to work as well in general. Another problem is that although it results in a small set of planes within the allowed error margin and a high contribution of each plane, it does not guarantee good visual properties of the generated billboard cloud. Furthermore, numerical issues may influence the result, because of the discretization.

Some of the related work is on improving the original algorithm. Besides that there are also completely alternative approaches to generate billboard clouds, which address certain drawbacks of the proposed greedy optimization.

### Stochastic Billboard Clouds

Lacewell et al. [LEST06] presented a completely different and simple algorithm to generate billboard clouds. Its intended usage is mainly simplification of plant foliage. This algorithm treats the input geometry as unordered list of faces and performs a stochastic search to find a set of billboards as approximation.

It starts by randomly selecting a seed face from the working set. Then $n$ random planes going through the perturbed face vertices are generated, whereby the maximum allowed displacement $\epsilon$ is kept. Then $n$ planes are generated going through the seed triangle. This is achieved by randomly perturbing the face vertices within a maximum allowed displacement $\epsilon$ and building the supporting plane through them.

For each plane the remaining faces are iterated and if they lie inside the allowed error margin of $\pm\epsilon$, their projected area is accumulated. From the set of evaluated planes the plane with the largest projected area is selected and added to the set of billboards and its faces are removed from the working set. The whole process is continued as long as the working set contains faces. The following tasks, such as calculating the plane extent and the texture generation, are performed similar to the original algorithm.

The algorithm proceeds very similar to the random sample consensus (RANSAC) approach [FB81]. A similar technique has been used to simplify point clouds to planar subsets for efficient hybrid point rendering with billboards [WGK05].

The random-based plane search produces improved results especially for unconnected

and loose foliage faces in comparison to the original. However, the authors also point out that it is not as well suited for continuous geometry. For parts of the plants such as trunks, stems or branches, they suggest conventional geometry-based simplification methods.

## K-Means Clustering

The problem of finding a minimum set of planes that approximate some geometry can also be formulated as clustering problem. Huang et al. [HNW04] presented a fast billboard cloud generation algorithm based on $k$-means clustering [HW79] to find planes to approximate the input geometry. Furthermore, it offers better user control and allows directly defining the complexity of the simplified geometry by varying the parameter $k$.

The basic method of $k$-means clustering starts by randomly building $k$ sets from the data points. Then calculate the centroid of each cluster and reassign the data points to the nearest centroid according to a distance metric. Calculating the centroids and reassigning is repeated until there is no further change in the data points.

In case of finding billboard planes, clusters are planes and data points are faces. The faces are simply assigned by minimizing a certain error metric. Taking the sum of the distances of the face vertices to a plane is sufficient for this purpose. Calculating the centroid of a cluster means finding a plane that approximates all faces as best as possible. To find such a best fit plane a Singular Value Decomposition (SVD) and the distance metric is used.

In contrast to the usual clustering, the proposed algorithm gradually stops when a cluster approaches a local minimum solution. The minimum is defined by the total distance of all plane faces. This avoids flipping of triangles between clusters and makes the algorithm more stable.
When a cluster is empty after all faces have been assigned to the best cluster, the empty ones are filled by the worst faces of all other clusters. To decide the quality of a face in relation to a cluster the deviation of the face normal and the cluster plane is used.

A very sensible step is the initial clustering, which has huge influences on the optimization result. With a random initialization finding an optimal solution is more difficult for the algorithm and the result varies drastically when it is run repeatedly. Therefore, three steps are described to find well distributed clusters which map similar sized areas of the input geometry. It starts by equally distributed tangent planes on the bounding sphere of the input geometry, which are then optimized by a different clustering process to reduce the variances in cluster coverage.

Overall, the $k$-means clustering algorithm performs significantly faster than the original algorithm. Also the choice of planes is not restricted to the resolution of the discretized plane space. However, there is no guarantee that the simplification is within a certain error threshold. On the other hand it allows precisely choosing the complexity of the simplifications.

Colditz et al. [BCF+05] also experimented with the $k$-means clustering approach in order to build billboard cloud simplifications for trees. The results were suboptimal and lots of artifacts could be seen. They decided to use a more sophisticated clustering algorithm called isodata [GB65]. It dynamically adjusts the number of clusters during the generation by merging and splitting clusters based on the standard deviation. Instead of clustering the faces, they modified that approach and directly clustered the face vertices. Additionally, the hierarchical information of the trees is used to build the clusters.

Another billboard cloud-based tree rendering method presented by Garcia et al. [GSSK05] also used $k$-means clustering to find the planes. In addition they use a hierarchical representation of the clusters to generate of a series of levels of detail. To smoothly blend between the levels the plane parameters are interpolated between the two enclosing levels.

### Hierarchical Face Clustering

The original billboard cloud algorithm has no control over texture memory usage, since it is insensitive to distances between faces simplified by a plane and therefore often results in excessive texture memory requirements. The approach to optimize texture memory usage discussed in Section 2.1 suggested by the authors of the original billboard cloud algorithm cannot resolve all cases. Therefore, a new generation algorithm focusing on efficient texture memory usage has been presented by Meseth et al. [MMK03].

Their new approach to find the billboard planes is based on ideas from hierarchical face clustering [GWH01], which iteratively merges adjacent faces based on a cost function. Since arbitrary meshes usually have many unconnected parts, the definition of adjacency has been generalized to spatial closeness to remove the limitation to connected meshes. To efficiently compute spatial closeness for large meshes a grid or octree can be used, whereby finding an adequate number of spatial subdivisions is important. The number of evaluated possible merges is also limited to a subset of faces. During their experiments they found a number of 50 to be a suitable threshold, but it still depends on the specific input geometry.

The generation process starts by building a proximity graph. Then the leaves are successively merged based on the cost function whose sum is to be minimized. The process continues until either a certain number of clusters is reached or the costs of the next merge operation exceeds a certain threshold. The mentioned cost function is composed of three factors. A geometric error $\epsilon_g$, a maximal normal deviation $\epsilon_n$ and a texture waste factor $\epsilon_t$, which is an adapted shape-error introduced by Garland et al. [GWH01].

Using their new approach reduces the texture memory requirements up to a tenth of that from the original algorithm. This is especially important when advanced texture mapping techniques that require lots of memory should be used. An outline of such techniques can be found in Section 2.4. On the other hand their comparison also showed a significant increase of geometry, since the clustering approach tends to numerous small billboards.

### Maximal Tiles

Andujar et al. [ABC+04] presented an efficient approach to find the largest planar region of geometries based on previous research in [ABE+03]. Unlike the original billboard cloud algorithm, it does not directly work with the faces of the geometry, instead a voxel discretization of the geometry is used and a new ingredient called *sticks* is introduced. The sticks are placed at the orthogonal edges connecting the inside and the outside of the black-white voxelization of the input geometry. The length of these sticks gives the desired error tolerance. The largest planar tile that approximates the most surface area can now be formulated as the plane that stabs the most sticks. The second important part of their algorithm is a linear parameterization of possible planes, in contrary to the non-linear Hough transform used by the original algorithm.

The first phase of the plane search algorithm is computed at a $24 \times 24 \times 24$ resolution of the voxelization and performs a voting for the best plane. For each stick, the local $4 \times 4 \times 3$ neighborhood is inspected using a pre-computed lookup table to detect planar regions. When such a planar region is detected, the also pre-computed list of cells of the discretized plane parameterization is used to quickly increase the counter of possible planes. In the second phase the cell of the plane parameterization with the most votes is refined in an octree-like subdivision and a candidate plane in the deepest level is computed. Finally, in the last phase, which does the actual plane search, the candidate plane is used as start to discard most of the possible planes and the actual sub-cell stabbing most sticks is calculated.

The runtime of their plane search algorithm is only dependent on the resolution of the voxelization instead of the face count of the input geometry. Another benefit of the voxelization is that the plane search is not restricted to the original discretization of faces of the input geometry. This allows the algorithm to find planes mapping only parts of faces and still guarantees the complete mapping of the geometry.

## 2.4 Additional Improvements

Finding the planes is only one task of the whole billboard cloud generation and alone it is not enough to get truthful simplifications. Most papers also mention various modifications they implemented to improve the visual impression. The following sections give an overview of suggested improvements.

### Post Plane Tweaking

A problem of the original algorithm is that found planes often do not look visually optimal, as illustrated in Figure 2.7 (a). The reason is that the recursive refinement stops immediately when a sufficient plane has been found. However, planes often barely fulfill the error threshold and are quite out of orientation compared to their faces. Fuhrmann et al. [FMU05] suggest a simple approach to correct the plane position and orientation afterwards in order to reduce the geometric error.

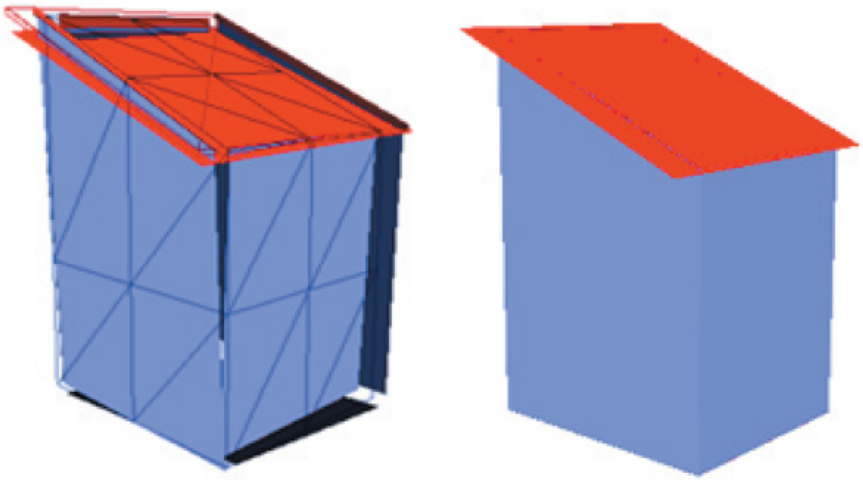


Figure 2.7: Post plane tweaking: (a) Generated billboard cloud from the original algorithm; (b) Optimized planes that reduce the geometric error. [FMU05]

First the corrected plane orientation is calculated by averaging the normal vectors of all

faces. To allow double-sided faces, face normals are flipped if they point into the opposite direction than the original found plane normal. Then the corrected plane offset is calculated by averaging the face center points. Additionally, in both steps each normal and center point is weighted by the geometric area covered by its respective face, whereby more relevance is given to the dominant faces. Afterwards, the corrected plane is tested for valid simplification of the concerned faces and if the area of the projected faces is equal or greater than the area of the original plane. If both conditions are met, the corrected plane is used instead.

## Crack Reduction

The billboard cloud simplification often creates planes with clearly visible gaps, since the topology is ignored. These artifacts mostly occur when faces that share a vertex are projected onto different planes. Figure 2.8 (a) illustrates such a case. The authors of the original algorithm already addressed that problem and reduced it as side effect of their texture rendering approach, which seems to work fine for foliage simplification of plants, where the input faces are mostly disconnected. However, especially continuous geometry still exhibits irritating gaps. To further reduce these distinct artifacts more accurate procedures are required.

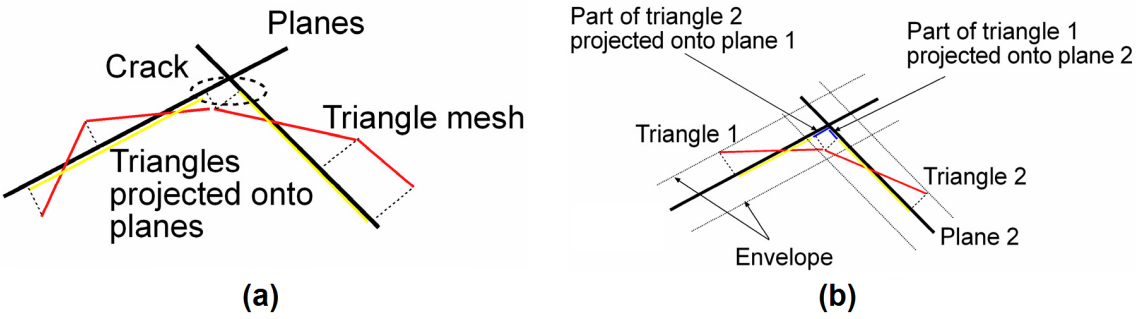



Figure 2.8: Illustration of the crack (a) resulting by projecting the shared vertex of two adjacent faces onto different planes and (b) the proposed envelope projection to reduce this artifact [HNW04]

### Envelope Projection

Huang et al. [HNW04] reduce cracks by projecting the concerning faces onto multiple planes. First each plane envelope containing all the plane faces is intersected with each other. Then for each pair of intersecting planes the required faces are determined by testing if at least one of their vertices is inside the other envelope. These faces are then

projected onto both planes. The process is illustrated in Figure 2.8 (b). When only a part of a face needs to be projected, the stencil buffer is used during the rendering. Their comparison shows that this method drastically reduces the number of cracks at the expense of creating erroneous pixels in the simplified model.

### Vertex Welding

A slightly simpler approach is suggested by Fuhrmann et al. [FMU05], whereby they reduce crack artifacts by permanently displacing ("welding") the vertices of the original model. After a plane has been found, its vertices are moved to their projected position before the next plane is computed. This moves the vertices closer to the plane intersection of later found planes. While this does not violate the notion of validity, the contribution of the effected faces has to be updated accordingly after the welding operation.

(a) (b) (c)

Figure 2.9: Vertex welding: (a) Original mesh; Billboard clouds without (b) and with (c) vertex welding. [FMU05]

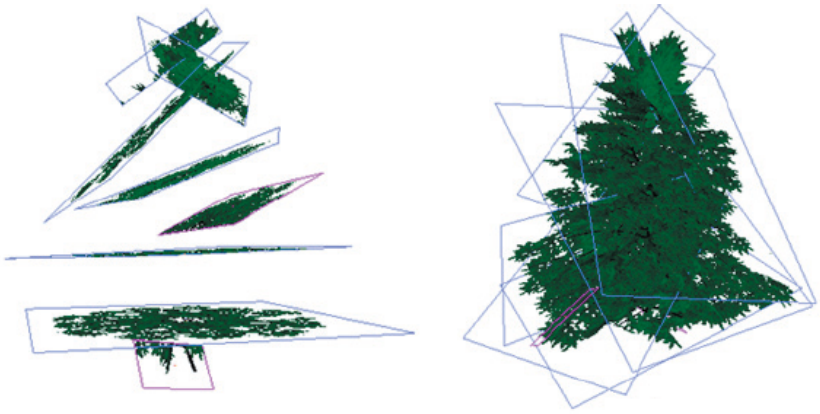
Of course there are cases where this extension does not work, but in general cracks can be reduced for most viewing directions. Figure 2.9 shows the original geometry (a) and a comparison without (b) and with the new extension (c).

## View-Dependency

The generation algorithms described so far all simplify in a view-independent way. However, with a large error threshold such an approach cannot faithfully capture the appearance of an object suitable for all view directions. Eisemann et al. [ED07] describe an analytical expression to ensure exact view-dependent error bounds for general simplification methods. It allows to extend the validity formulation used by the original billboard

cloud algorithm so that it is restricted to a certain tolerable view angle for an arbitrary view cell. When the simplification is viewed from inside this cell the planes will be guaranteed to face to the viewer and discontinuities are significantly reduced. This allows even higher levels of simplification, but when viewed from outside the cell the discontinuities of extreme billboard cloud simplifications are clearly visible as for example shown in Figure 2.10 (a).

Fuhrmann et al. [FMU05] focused their research of billboard clouds on tree simplification and also presented a practical view-dependent extension to the original billboard cloud generation algorithm. Especially with plant foliage the algorithm often decides that the best simplification can be achieved by collapsing multiple layers of geometry whereby it is literally sliced and disturbing visual problem occurs. Figure 2.10 (a) shows a tree simplification using the original algorithm with a high error threshold where these artifacts are very distinctive.



(a) (b)

Figure 2.10: View dependency: (a) Billboard clouds of a tree with a high error threshold, resulting in horizontal slices; (b) Improved result by introducing a view-dependent penalty [FMU05]

Since in most applications the viewing-angle is rather small, they implemented a view-dependent penalty to prevent certain planes. In most scenarios the viewer is on the same height as the trees, therefore, purely horizontal planes should be avoided. In the case of a flyover, completely vertical planes are not convenient either.

Their solution is to substract a certain penalty from the values in the density grid when they are accessed during the algorithm. The penalty weight is given by a slope function taking two parameters, the maximum penalty and a cutoff angle. Recalling the plane

space, regions with $\varphi = \pm 90^\circ$ represent horizontal planes. The slope function simply returns a linear interpolated value from maximum penalty to zero between $\varphi = \pm 90^\circ$ and the cutoff angle. The described extension works extremely well for most tree models, as shown with the result in Figure 2.10 (b).

## Advanced Texture Mapping

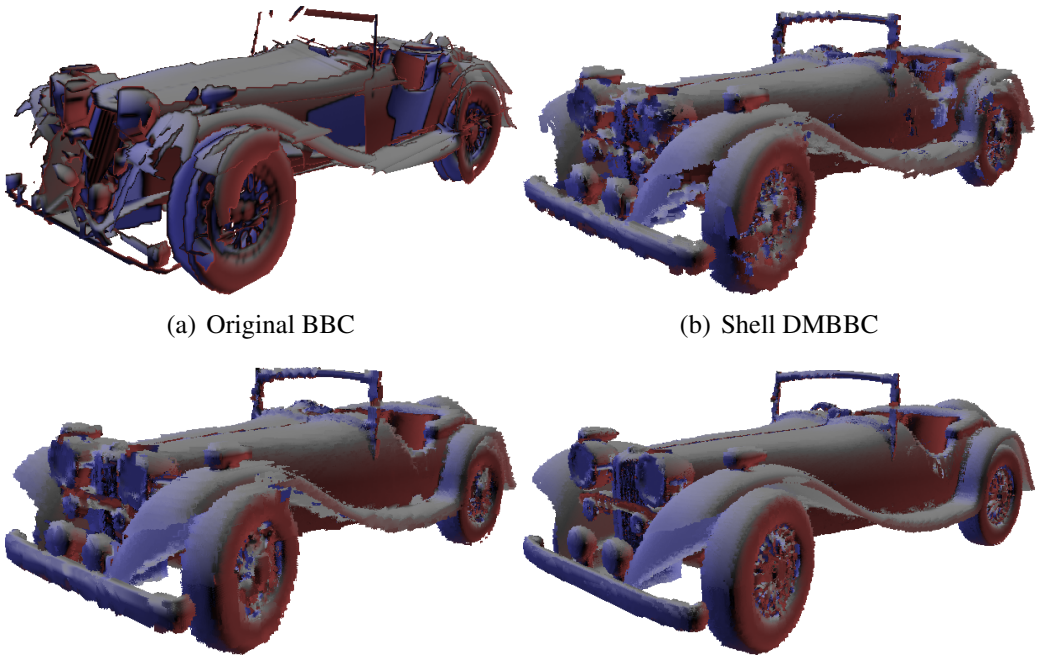
A typical real-time application requires dynamic lighting of the scene objects. If only a pre-lighted diffuse color is stored in the textures and the plane normals are used to re-light the billboard cloud geometry, the actual flatness of the planes will be noticeable. For better resemblance of the original geometry, a normal map needs to be generated besides the diffuse texture to encode the normal vectors along the plane. This approach is usually sufficient for a distance representation of an object. However, it should be possible to switch to a simplification as near as possible to gain the best performance improvement, but only if no visual difference can be perceived. Advanced texture mapping techniques can help to improve the visual accuracy and make the rendering more efficient.

### Displacement Mapped Billboard Clouds

Mantler et al. [MJW07] presented an improved billboard cloud representation called *displacement mapped billboard clouds* (DMBBC) that reduces the visual error by replacing the planes with boxes. The geometric details are represented by a so called *volumetric displacement function*, which can be a 2.5-dimensional height map or a full three-dimensional volume texture. This new approach allows for efficient rendering of complex geometry closer to the viewer than other simplification methods. Additionally, parallax as well as visual depth are preserved, while the representation remains image based. For farther objects it also allows smooth blending with traditional billboard clouds.

To generate DMBBC it is assumed that the billboard quads are already computed by any algorithm. For the highest quality DMBBC, the geometry inside the boxes, extruded from the quads by a certain $\epsilon$ distance, is each rasterized into a volume texture, storing the diffuse color and the normal vector per voxel. The achieved image quality is relatively high as shown in Figure 2.11 (d), but the memory consumption is also very high. Therefore, simpler approximations of the displacement function, called *shell* and *thick shell*, can be used instead.

With the *shell* representation it is assumed that the displacement function only occupies a single region (voxel) per $(u, v)$ texel, so that it can be efficiently stored by a single dis-



(a) Original BBC (b) Shell DMBBC

(c) Thick Shell DMBBC (d) Volumetric DMBBC

Figure 2.11: Comparison of different DMBBC techniques: (a) Shows the billboard clouds rendered with normal mapping; (b) The simplification rendered using the simple shell representation; (c) Result using the thick shell rendering; (d) Best possible quality whereby the geometry information has been stored as volume texture. [MJW07]

tance value to the billboard plane per texel. A user-defined thickness controls the size of the region, because in contrast to pure displacement mapping the geometry inside the billboard box is not continuous. Figure 2.11 (b) shows DMBBC using this technique.

The *thick shell* representation stores two distance values per texel, defining the interval where geometry occupies the billboard box. In comparison to the simple shell representation is has slightly less artifacts as showed in Figure 2.11 (c).

Both approximations are only correct as long as the geometry is continuous, otherwise false positive intersections with the displacement function are created.

The billboard boxes are rendered using a ray-casting algorithm on the GPU to determine the intersection of the viewing ray with the displacement function stored as two- or three-dimensional texture. If no intersection is found, the pixel is discarded. Instead of linearly sampling the texture, an accelerated process adapted from Donnelly [Don05] that is based on distance functions to find the intersection with less texture reads is used.

## Bidirectional Texture Functions

Since billboard clouds preserve the three-dimensional shape of the original geometry, they can be used to efficiently cast approximate shadows. However, shadow cast onto themselves will be incorrect because of the flatness of the planes. Additionally, changing material properties, occlusion and interreflections cannot be preserved with the current representation. Therefore, Meseth et al. [MK04] use view- and light-direction dependent textures, also called bidirectional texture functions (BTFs), to efficiently store this information.

A BTF represents a 6-dimensional function BTF$(x, y, \theta_v, \varphi_v, \theta_l, \varphi_l)$ of the point $(x, y)$, the view-direction $(\theta_v, \varphi_v)$ and light-direction $(\theta_l, \varphi_l)$ in spherical coordinates. In contrary to traditional BTFs, where only the hemisphere over a point on a flat surface is sampled, it is required to sample the complete sphere around a point, since the actual geometry is not flat.

The construction of such BTFs requires lots of memory and is very time consuming. It can be either done by using a raytracer or using rasterization of graphics hardware in combination with a shadowing algorithm.

(a) Normal Mapping (b) BTF Textured (c) Original Geometry

Figure 2.12: Billboard clouds rendered using normal mapping (a) and with BTF textured (b) compared to the original geometry (c). [MMK03]

Since the amount of data is unfeasible to be used directly during the rendering, a reasonable compression algorithm is required. Therefore, the approach by Müller et al. [MMK03] is used which uses a local PCA-algorithm to compress a BTF for real-time rendering. Additional, a suitable billboard cloud generation algorithm that creates memory efficient

texture quads should be used.

A comparison between normal mapping and BTF textured billboards is shown in Figure 2.12. Especially when comparing the small images an increased quality though more detailed shading can be noticed.

## Indirect Texturing

Billboard cloud simplifications are conceived to be viewed in miniature and therefore usually have low texture resolutions. For a closeup view this is not sufficient or would require high resolution textures consuming huge amounts of memory. Indirect texturing is a technique that uses the characteristics of the input geometry to efficiently simulate high details.

Garcia et al. [GSSK05] presented the concept of indirect texturing used to render high detail billboard leaf clusters. To build the billboard cloud, the leaves are handled as primitives. The billboard texture is an indirect texture that only stores the rectangles of the projected leaves, where a low resolution texture can be used. The rotated leaves are separately packed to a high resolution texture which is used to lookup the high detail texture information indexed from the indirect texture. Figure 2.13 illustrates the polygonal leaves of the billboard plane and how the final texture is rendered using the indirect texture to lookup the packed leaves texture.

In cases where the leaves are overlapping, artifacts can be seen when only a single indi-

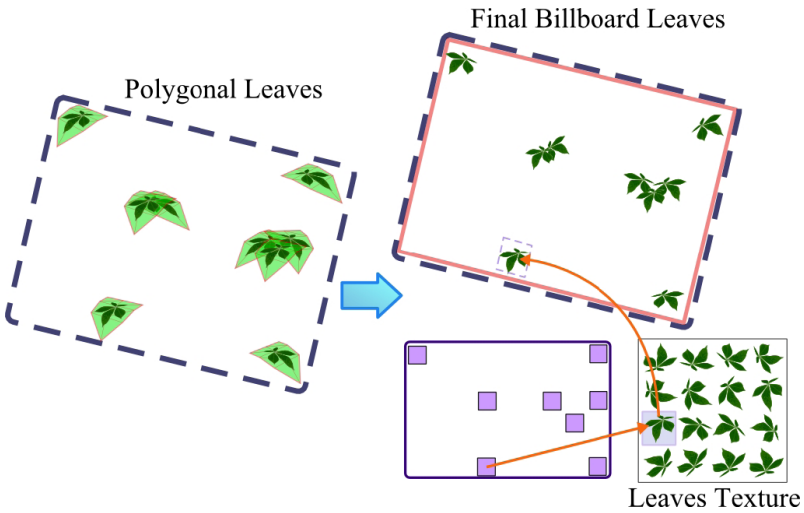



Figure 2.13: Indirect texturing: The final billboard texture is rendered using the indirect texture to lookup the high detail packed leaves texture. [GPSSK07]

rect texture is used. Therefore, a double layer indirect texture is suggested, one where the leaves are ordered from to back and one in the inverse order. A fragment shader can easily decide which texture to use, which should remove the artifacts in most cases.
Additional layers could be used to completely eliminate these artifacts, but this would require significantly more texture memory and most regions of the texture would be empty. An improved version of indirect texturing supporting arbitrarily overlapping leaves has been presented in [GPSSK07].

## 2.5 Summary

Billboard clouds are capable of extreme geometry reduction. However, the original algorithm seems to have numerical problems finding suitable planes for low tolerance values, mentioned multiple times by other authors. Alternative algorithms have been presented to overcome certain drawbacks of the original greedy optimization, but partly remain not evaluated in detail. Usually the most disturbing artifacts are caused by cracks between the billboard planes. Several approaches to reduce these artifacts have been discussed, but it seems hard to remove them entirely.

Lots of related work is on simplification of plants and vegetation because the usually sparse and jumbled properties of foliage can be handled well by the algorithm and the typical artifacts are not noticeable that well. For most applications several specific heuristics have been used to adapt the generation algorithm to get feasible simplifications. Connected geometry appears to be more difficult to simplify without artifacts, but all the improvements give lots of possibilities to enhance the visual quality.

## Chapter 3

# Theory

In the last chapter we gave an overview of several billboard cloud generation algorithms and improvement techniques. Now we will discuss the steps required to build billboard clouds according to the original algorithm in detail. Thereby, important properties will be analyzed in comparison to other generation algorithms. Additionally, we present possible modifications and alternative approaches to reduce artifacts of the simplifications and improve the generation performance.

Based on that theory our desired billboard cloud generation process, which should be capable of simplifying general geometry, will be implemented. Finally, we will describe the error metrics we used to evaluate our improvements and to make an algorithm comparison.

## 3.1 Generation Overview

The billboard cloud generation is structured into separate tasks, which will be discussed successively in the following sections. Such a modularization has also been useful for our implementation and makes the process extensible. The flowchart in Figure 3.1 gives an overview of the whole generation process.

The first module abstracts the plane search including different algorithms. A more detailed discussion of every algorithm will follow in the next section.

The plane search provides a set of planes with assigned faces which then can be improved with optional methods. They are illustrated in the second stage in the flowchart. The corresponding tasks will be discussed in Section 3.3.

The final task of the billboard cloud generation is the texture generation.

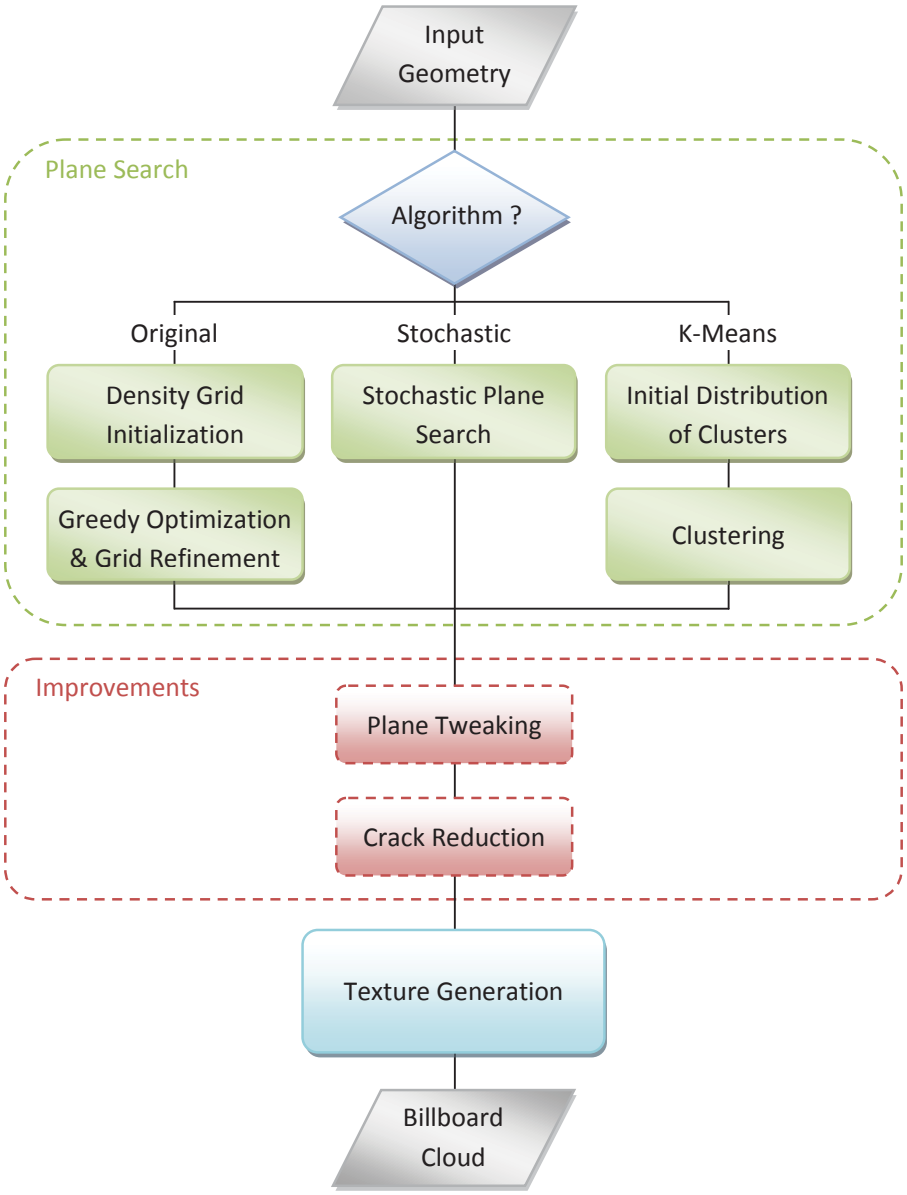



Figure 3.1: Overview of the billboard cloud generation process: The first module encapsulates the plane search. Afterwards, optional improvement tasks can be performed. Finally, the billboard textures are generated.

## 3.2 Plane Search

The main difficulties of the billboard cloud generation are to find the planes to approximate the original geometry and minimize the geometric error. Since this is an NP-hard problem, an optimal solution is unpractical. We have already discussed several generation algorithms, which all have different properties that will be discussed in this section.

## The Plane Space

The key part of the original algorithm is the plane space. It tries to evaluate all possible planes that can be put through the geometry. Therefore, a three-dimensional equivalent of the Hough-Transform [DH72], here called plane space, is used. Thereby, planes are parameterized by their orientation in spherical coordinates $(\theta, \varphi)$ and the offset $\rho$ from the origin.

A plane can be described as normalized plane equation, built from a point $p$ and a normal vector $\vec{n}$:

$$n_x x + n_y y + n_z z + \rho = 0, \text{ where } \rho = -n \cdot p \tag{3.1}$$

The following conversions are required for the reparameterization in spherical coordinates:

$$\begin{aligned} \theta &= \arctan 2(n_x, n_y) \\ \varphi &= \arccos(n_z) \end{aligned} \tag{3.2}$$

$$\begin{aligned} n_x &= \sin(\theta)\cos(\varphi) \\ n_y &= \sin(\theta)\sin(\varphi) \\ n_z &= \cos(\theta) \end{aligned} \tag{3.3}$$

Equation 3.2 is used to encode normal vectors to spherical coordinates and Equation 3.3 is the corresponding inverse transformation.

Spherical coordinates span from $0 \leq \theta \leq 2\pi$ and $0 \leq \varphi \leq \pi$, whereby all plane orientations can be mapped to a two-dimensional grid. Figure 3.2 illustrates this parameterization.

On top of that the plane offset $\rho$ defines the third dimension. To evaluate all possible planes the plane space is discretized. Recall that a cell in this space is referred to as bin and the discretized plane space as density grid.

## Singularities

On closer inspection of Figure 3.2, immediately the singularities at the poles stand out. Actually this has no influence on the plane evaluation and selection, because it only results in non-uniform sampling of the plane orientations. It makes the whole process less

[1] http://en.wikipedia.org/wiki/Spherical_coordinates

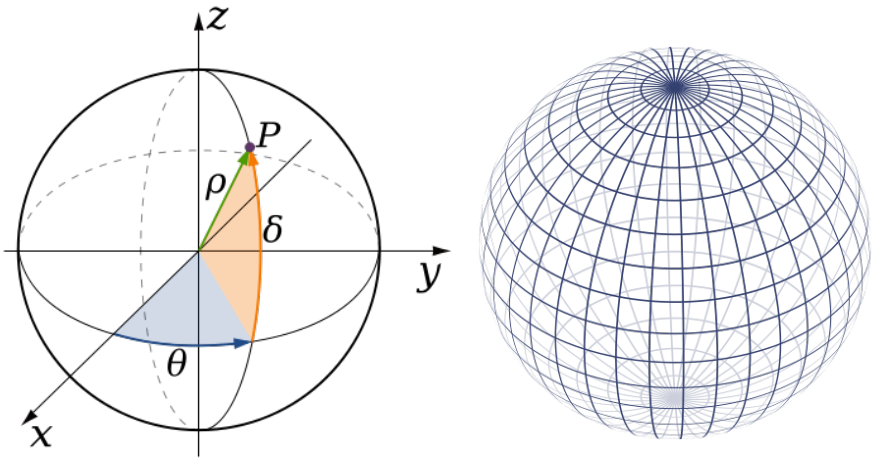



Figure 3.2: Parameterization of plane orientations with spherical coordinates[1].

efficient, because much more bins need to be evaluated. With a uniform distribution of orientations, obtaining the sampling rate at the equator over the whole sphere, the number of orientaions could be reduced by the factor $\pi$. However, since a reference to every orientation neighbor is required for the rasterization described in Section 2.1, a parameterization supporting both properties will require some compromises.

To solve this problem, we represent all possible orientations by projecting the normal direction on a cube, giving six faces as two-dimensional grids and the possibility to index all neighbors of an orientation. More details of our implementation can be found in Section 4.4.

A further singularity of the plane space that the authors do not mention occurs in conjunction with the face position and orientation. Figure 3.3 gives an example with two vertices

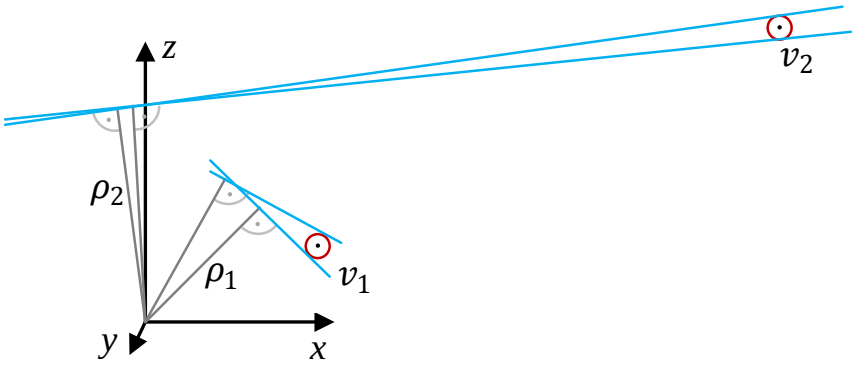



Figure 3.3: Sensitivity of validity when sampling the plane orientations depending on the distance of the vertices.

$v_1$ and $v_2$. Considering planes with a certain offset $\rho_1$ and $\rho_2$ through the vertices, the allowed change of orientation so that the plane still intersects with the validity domain of the vertices (red) depend on the distance to the point on the plane nearest to the origin. This dependency makes the validity of faces which are far away from the origin and have normals nearly tangent to the bounding sphere surface very sensitive to the variation of the orientation.

This problem is unavoidable and the best thing that can be done is to move the geometry in its center of the bounding volume. Thereby, the maximum distance is minimized to the radius of the bounding volume and the properties of this singularity can be calculated. The consequences during the rasterization will be discussed in the Section 3.2.

## Plane Evaluation

The original billboard cloud generation algorithm uses a certain metric to evaluate the quality of simplification planes called *plane density*. It consists of face *contribution* and *penalty*. Additionally *validity* is used for the compliance of the maximum tolerable error $\epsilon$. Contribution described in Section 2.1 is defined as projected area of the face on a plane. However, *face* is an ambiguous term in computer graphics, but we assume that it has an area $a$ and a normal vector $f_n$ which are used in the following computations.
The projected area depends on the angle between the plane and the face surface which can be calculated by a simple dot-product:

$$area_P(f) = (p_n \cdot f_n)a \tag{3.4}$$

Using the projected area, a face will have different contribution depending on the orientation and peak at the plane that mostly coincides with the face surface. We noticed that this often leads to planes barely approximating the faces, because the face distance has no effect on its contribution. The reason is that the plane refinement stops as soon as a suitable plane fulfilling the $\epsilon$-constraint has been found. This is one of the reasons why *post plane tweaking* discussed in Section 2.4 has been suggested.

We want to incorporate the distance between face and plane into the formulation of contribution as well. This should additionally favor planes with less geometric error. Knowing the validity range $[\rho_{min}, \rho_{max}]$ of a face, a fall-off function with a peak at the center $c$ and a symmetric fall-off reducing the weight $w$ to $0.5$ at a distance of $\pm\epsilon$ from the center is used. The function looks like the following:

$$c = \frac{\rho_{max} + \rho_{min}}{2} \tag{3.5}$$

$$w = \frac{1}{\frac{|x-c|}{\epsilon} + 1} \tag{3.6}$$

where $x$ is the sampling position representing a plane with a certain offset for which the weight should be calculated.

Furthermore, the effect caused on the rasterization has to be considered. There are two possibilities to calculate a weight for the coverage in a bin. Either taking the partial integral of the falloff function over the interval of the bin, or sampling the function at the bin center. We find sampling sufficient as long as the grid resolution is high enough. The graph in Figure 3.4 gives an example, assuming the sampling rate of $\rho$ to be $\frac{\epsilon}{2}$ as discussed in Section 3.2. The blue line is the falloff function, clipped by the validity interval $[\rho_{min}, \rho_{max}]$, which is slightly less than $2\epsilon$ in this case.

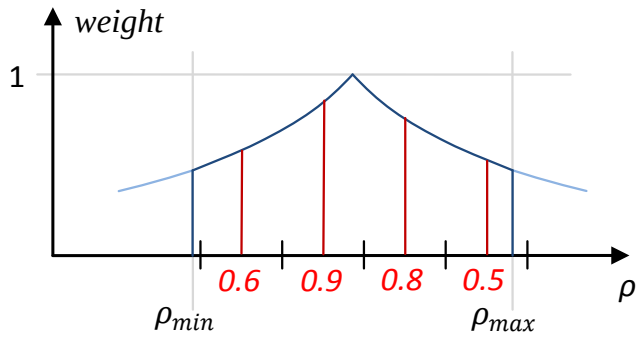



Figure 3.4: Illustration of the falloff function between the interval $[\rho_{min}, \rho_{max}]$. It is sampled at the bin centers to get the weight (red) during the rasterization.

This example shows that even though the function is only sampled, the bin containing the best plane will have the highest weight. However, in total the coverage sum of all faces valid for a bin may be biased, because the falloff function of each face will be sampled differently. To counteract this behavior, the falloff function can be modified to return a weight of 1 for a range $\pm\frac{binwidth}{2}$ around the center:

$$w = \min\left(1, \frac{1}{\frac{|x-c|}{\epsilon} + 1 - \frac{binwidth}{2}}\right), \tag{3.7}$$

The centering bin will always get the full coverage of the face. Alternatively, this falloff function could be omitted for initializing the density grid and only be applied during the refinement, since only there the resolution will be high enough so that the distance falloff has a significant effect. Usually refinement is required nearly every time anyway.

No matter how such a distance falloff is implemented, it will modify the choice of planes in relation to the maximum geometric error $\epsilon$. Figure 3.5 illustrates a case where the plane exactly in between the faces is not selected, because a plane closer to the dominant face $f_1$ will have the most contribution.

Calculating the planes densities gives the following results $d_1$ and $d_2$:

$$\begin{aligned} a_1 &= 8 \\ a_2 &= 1 \\ a_3 &= 1 \end{aligned} \qquad \begin{aligned} d_1 &= 1a_1 + 0a_2 + 0a_3 = 8 \\ d_2 &= 0.5a_1 + 0.5a_2 + 0.5a_3 = 5 \end{aligned} \tag{3.8}$$

where $a_n$ is the corresponding face area and $d_n$ the final density of the planes. Because of the used distance falloff plane $p_1$ has the most contribution $d_1$, although plane $p_2$ maps more total face area.

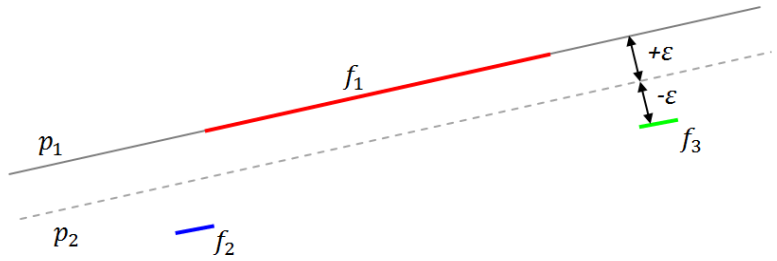



Figure 3.5: The valid plane $p_2$ with most projected face area will not be selected when distance falloff is used. Plane $p_1$ perfectly approximating face $f_1$ has the most contribution.

Another possibility to reduce the effect that faces are missed would be to make the falloff function weaker. For example, instead of reducing to $0.5$ at a distance of $\epsilon$ it would be possible to reduce to $0.8$. However, in general *penalty* will avoid such cases if the difference between the face areas is not extreme. Then the above described distance falloff should improve the plane selection and reduce the levels of refinement. The produced results using this falloff will be analyzed in Section 5.2.

## Rasterization in Detail

The first step of the original billboard cloud algorithm is to initialize the density grid as described in Section 2.1. Thereby, the validity domains of all faces weighted by their contribution are rasterized into the grid. Since a bin represents a range of planes, the grid

is only used as guidance to highly contributing planes, otherwise the maximum tolerable geometric error $\epsilon$ is not guaranteed.

The initialization of the density grid has a runtime complexity of $O(n * g)$ depending on the number of faces $n$ and considering the grid resolution $g$. In detail for each face all possible plane orientations have to be iterated. For each orientation the interval on $\rho$ where the face is valid is calculated and the face contribution rasterized in between. Therefore, the resolution of the grid has a high influence on the runtime and has to be chosen carefully. The choice of grid resolution will be separately discussed in Section 3.2.

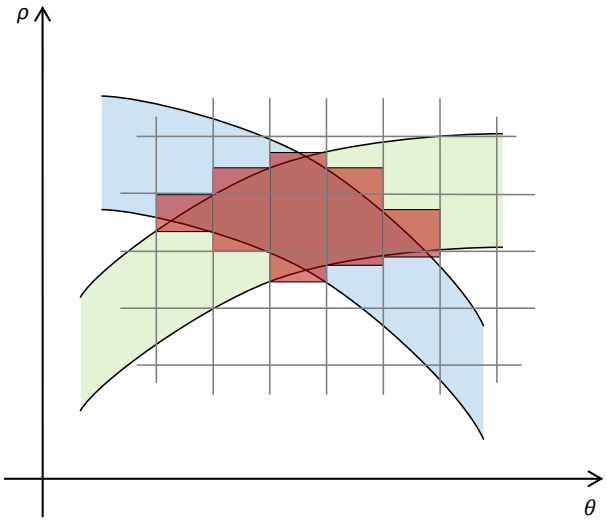



Figure 3.6: Example rasterization in a two-dimensional case of a line segment between two vertices considering a single orientation angle $\theta$ and the offset $\rho$. The blue and green areas are the validity domains of the vertices in plane space. The intersection region is rasterized (red) using *simple validity* in the grid.

In Section 2.1 we also discussed the extended validity formulation *simple valid*, which is required so that the contribution of all planes inside a bin is included. Summarized shortly, it is calculated for a bin with a certain plane orientation by first calculating the validity range on each border orientation by intersecting all vertex validity domains and then building the union of them.

Figure 3.6 illustrates how the validity domain is approximated by simple validity. Remembering that a vertex validity domain in plane space is a surface with a thickness of exactly $2\epsilon$. The figure shows the validity domains of two vertices (blue and green) and their intersection is the validity domain of the face in this simple case. The exact intersection is always $2\epsilon$ at its thickest point. The red boxes illustrate the calculated validity range

of $\rho$ where the contribution of the face is rasterized exactly by the coverage in the bins. In a case like in the figure the validity domain is approximated sufficiently. The same applies for planes that approximate a face quite well, since the vertex validity domains will all be roughly similar and therefore intersect in a shallow angle. However, in the case a face lies disadvantageous regarding to the singularity illustrated in Figure 3.3, the intersection can become a very short interval in $\theta$-direction, because the validity domains will intersect very steeply. Then the real contribution will be either significantly less as with the *simple validity* approximation or the face will be missed completely.

A more precise calculation of the validity domain than with *simple validity* would be required to approximate the coverage of the bins. One possibility would be to assume a linear approximation of the validity domain inside a bin after the intersection of the vertex validity domains for each border orientation has been calculated. With that a more precise volume of the intersecting validity domain can be calculated instead of when simply the union is built.

The computation gets more complex when the cases of the mentioned singularity, where faces may be missed, should be included. In the case of the two-dimensional example in Figure 3.6 the diamond shaped intersected validity domain would be completely inside a single bin and the intersections at the borders would have no validity range. This could be detected by comparing the order of the vertex validity domains between the different borders, but this makes the process more complex and has more specific cases. Then a volume calculation to reconstruct the intersecting validity domain, for example by intersecting the linear approximations of the vertex validity domains, can be used to get the exact coverage.

However, all this effort would cost more computation time and will only pay off if these cases are problematic. For our implementation we use *simple validity* as described, because when the grid resolution is chosen properly the approximation should be accurate enough. Only when a very small allowed tolerance $\epsilon$ is used *simple validity* is not sufficient any more.

Furthermore, we also suggest modifying the way how the valid range $[\rho_{min}, \rho_{max}]$ is rasterized. The authors suggest using the coverage of the range in the bin to weight the contribution. This has the effect that bins containing planes that barely simplify faces will have less density. Since we additionally use a falloff function as described in Section 3.2, we already have this behavior. Therefore, for each bin covering the valid range the falloff

function is sampled to get the weight for the contribution instead of using the bin coverage. We also mentioned that penalty is important to counteract bad properties using the falloff function. Penalty is still accumulated using the bin coverage, but a falloff function could be theoretically used too.

Figure 3.7 uses the example of Figure 3.4 to illustrate the weights for the face area for the bins. The blue values are the samples from the falloff function and the red ones the anti-aliased coverage scaled by $w_{penalty} = 10$. Note that the bin with the crossover is covered $0.8$ by penalty and the weight of the falloff function is $0.4$. Therefore, it has a final weight of $-w_{penalty} \cdot 0.8 + 0.4 = -7.6$.

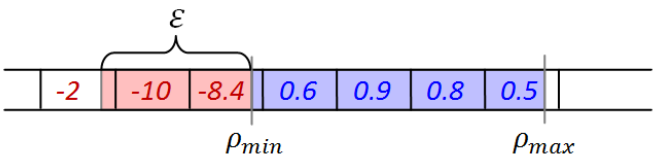



Figure 3.7: Weights given by the distance falloff and penalty used when rasterizing the contribution in the density grid.

## Density-Grid Resolution

What has not been discussed so far and is not mentioned by literature in detail is the correct choice of the density grid resolution. This concerns the choice of a suitable sampling rate for the orientation in spherical coordinates $(\theta, \varphi)$ and for the plane distance $\rho$.

During the density grid initialization, the geometry is rasterized in plane space and the contribution of all faces is accumulated to bins where they are valid. This process has already been discussed in Section 3.2 in detail including the hereby important extended validity formulation *simple valid*. It gives an interval on $\rho$ where the face is valid and the contribution is accumulated to the bins according their coverage of the interval. Remember that we noted that the validity domain of a face is exactly $2\epsilon$ in its thickest point, so a sample rate of $\frac{0.5}{\epsilon}$ would be sufficient. However, neighboring orientations will have much narrower validity domains and the contribution would only be accumulated partially to a bin and potentially highly contributing planes maybe be missed. Even worse, if penalty is used, the whole contribution maybe be canceled out within the same bin when rasterized, because it is weighted very high. Therefore, we recommend using a sampling spacing of at least $\epsilon$.

Since the maximum tolerable error threshold $\epsilon$ is set in percentage of the bounding volume diameter, the sample rate directly gives the resolution to discretize $\rho$ when double sided planes are used and faces are valid for planes independent of their normal orientation. In case the orientation matters and therefore negative values of $\rho$ are allowed too it has the double resolution. Furthermore, increasing the resolution of $\rho$ does not drastically effect the runtime, since rasterizing the contribution is relatively fast after the valid interval of $\rho$ has been calculated.

The sampling rate of the plane orientations is also dependent on the error threshold $\epsilon$ and has most influence on the runtime of the density initialization. In Section 3.2, where we discussed the singularities, the relation to $\epsilon$ is illustrated in Figure 3.3. It shows that the sampling of the validity domain also depends on the distance to the point nearest to the origin on the plane.

In order to include all possible structures of the input geometry, the sampling rate of the orientations has to be adapted to the worst case. This regards cases where faces are at the border of the bounding sphere and perpendicular to the bounding sphere surface. The required sampling rate can be calculated as illustrated in Figure 3.8 and applied in Equation 3.9.

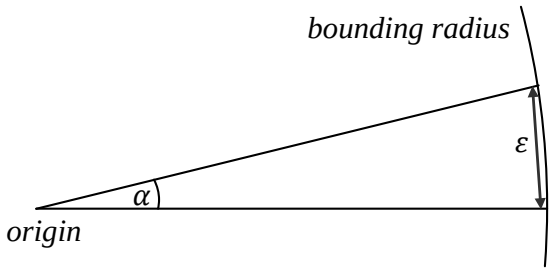



Figure 3.8: Illustration of the maximum angle $\alpha$ the plane orientation is allowed to change when a structure with size of $\epsilon$ should be included.

Therefore, the maximum allowed angle $\alpha$ and the sampling rate are calculated as follows:

$$\alpha = \arctan \epsilon \quad \Longrightarrow \quad samplerate = \frac{1}{\alpha} \tag{3.9}$$

Then the sample rate of $\theta$ and $\varphi$ can be discretized. Considered the parameterization of the orientation in spherical coordinates, where $\theta$ spans 360° and $\varphi$ 180°, $\varphi$ will have only half the resolution. Looking at the required resolution in dependence of $\epsilon$ plotted in Figure 3.9, shows that the resolution increases drastically for small $\epsilon$.

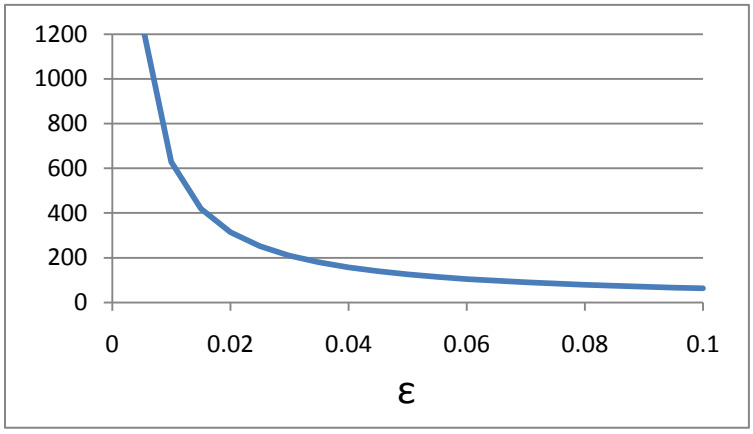



Figure 3.9: The required resolution for $\varphi$ dependent on $\epsilon$ when all possible structures should be included.

The total number of orientations is then given by $res_\theta * res_\varphi$, which increases quadratically when the resolution is changed. Therefore, using the correct required resolutions is unfeasible for small $\epsilon$. In practice it is better to use a fixed resolution and accept that some detail may be missed by the rasterization. In our evaluation in Section 5.2 we will experiment with different resolutions and find a suitable one.

## Plane Search and Grid Refinement

The density grid only gives a direction where highly contributing planes can be found. The focus during the rasterization was that no contribution is missed because of the discretization, which is approximated with the described *simple validity*. Hence, after a bin has been selected it still has to be searched for a plane that approximates most faces of the bin and respects the maximum tolerable error $\epsilon$.

In Section 2.1 we already described the greedy optimization and the refinement process according to the original algorithm. To summarize, after the densest bin has been selected, first the valid faces using *simple validity* for the bin are calculated and then tested for validity for the plane represented by the bin center. If not, the grid is refined and the density is evaluated for subbins in that region. Then the highest contribution subbin is chosen and the process is repeated. Afterwards, the density grid is updated by subtracting the contribution and penalty given by the faces valid for the plane. Then the search is continued until all faces are mapped.

Early tests showed that the two tasks consuming most runtime are the refinement itself and the updating of the density grid. The latter may be underestimated but usually it is

the most expensive, since the rasterization involves millions of bins. The operations performed during the rasterization are already fairly simple. Hence, the performance can only be improved by modifying grid resolution, for which choosing the correct resolution is very important. It has already been discussed in Section 3.2.

Furthermore, because *simple validity* is only a rough approximation, refinement will be required nearly every time and take a few recursions. Therefore, 216 subbins need to be evaluated if all subbins of the neighbors are included like suggested by the authors. However, we find it is sufficient to only evaluate the subbins closest to the old bin, reducing the number to 64. Even in the very unusual case illustrated in Figure 2.5, where the densest plane is significantly outside, the plane should be found successfully.

It has also been mentioned that the refinement is limited to the faces valid for the initial selected bin. This is an important step to reduce runtime, because a refined plane will usually map a subset of these faces. However, since also neighboring bins are evaluated new faces may become valid, but the limitation is definitely recommendable. When evaluating all valid faces, the refinement will not only take significantly longer, because this is much more expensive, it will also cause the refinement to drift more likely and require more iterations.

An addition we suggest is to include faces that became valid by reevaluating the valid faces for the plane after the refinement. Even though these faces have not been explicitly taken into account during the computation, it would be disadvantageous for the generated billboard cloud if they would have to be mapped by additional planes. This evaluation has barely any influence on the runtime, since testing a face on validity for a plane is very fast and simple.

## Possible Modification

During the implementation and study of the behavior of the different generation algorithms we also experimented with modifications of the original algorithm. The general problem in our opinion is that it tries to strictly enforce the maximum geometric error $\epsilon$. Although it produces theoretically optimal simplifications and guarantees a certain error threshold, this often degenerates to unpractical results. Generally we are looking for a practical simplification technique that is preferably simple too, but the original algorithm is very complex. We present two possible modifications that relax the formulation of the maximum tolerable geometric error $\epsilon$ but still use it as guide to generate the simplification.

The original generation algorithm spends a lot of time refining the grid during the plane search. Our first suggested modification is to skip the refinement and immediately select the plane at the center of the densest bin. That is exactly the opposite of what the authors of the original algorithm suggest. The error of this method should be accepted and *post plane tweaking* should be made to handle the further optimization of the plane.

Furthermore, knowing the grid resolution, the worst case error by selecting the plane at the center of a bin can be calculated. So if it is really necessary to maintain a certain maximum error, a lower value for $\epsilon$ can be taken in the first step when the density grid is initialized. This results in a shift of computation time from the grid refinement to the initialization caused by a higher grid resolution. However, the initialization of the density grid is less complex and can be optimized easier and therefore the generation should take less overall time.

The second suggested modification comes from the observation that the last planes found by the greedy selection are often very suboptimal, because they have to map the last remaining faces. One possibility would be to simply skip the last faces and do not simplify them at all. While the whole idea may work in some special cases where missing faces will not be noticed, it is not very well suited for continuous geometry, because holes would remain.

A better solution we suggest is to assign the final faces to the nearest plane. It is a process that is also done by the k-means clustering algorithm whereby with each iteration the faces are assigned to its nearest cluster. However, faces with high distance even to the nearest plane should be completely skipped anyway to avoid distorting the plane when additional *post plane tweaking* is used.

What has not been discussed so far is how to exactly determine the faces that should be skipped. It turned out finding a suitable heuristic that has a similar effect independent of the input geometry is not easy. Setting a threshold for the minimum plane contribution in relation to the surface of the bounding sphere turned out to behave very differently in our test cases. A better result could be achieved by a percentage threshold in relation to the total area of all faces.

The consequences of these modifications will be evaluated in Section 5.2 and 5.2.

## Other Generation Algorithms

In Section 2.3 we gave an overview of other billboard cloud generation algorithms. They all solve the problem of finding a set of planes to approximate the input geometry in a completely different way. In this section we will sum up their properties and discuss their behavior in comparison to the original algorithm.

### Stochastic Plane Search

The stochastic generation algorithm seems surprisingly simple. Even though it is a completely different approach than the original algorithm, it can be said that the result has very similar properties. It guarantees a maximum geometric error. The planes are selected one after the other, whereby the first planes will simplify lots of dense faces and the final ones have to map the remaining ones. It often results in suboptimal planes. This therefore also has the typical drawbacks of greedy selection based algorithms.

The optimality of the generated result should be very similar to the original algorithm. Both basically work on evaluating planes and successively selecting the best one. However, our evaluation showed that the result is strongly dependent on which seed faces are selected. Since starting by selecting a seed triangle and then searching the best plane, a good plane can be found very fast. The evaluation of all these planes is independent which is important when the process should be parallelized. Therefore, finding the planes with the stochastic generation algorithms is much faster in comparison to the original.

The authors of the stochastic billboard clouds mentioned that their algorithm does not work very well for continuous geometry. Since in theory it has similar properties than the original algorithms this statement has not been verified yet. In the paper is no reference whether penalty has been used, which however can and should be applied similar to the original algorithm.

A problem of the original algorithm is that it sometimes ends up with parallel planes which are suboptimal for certain view directions. To solve that problem an extension has been discussed in Section 2.4. In contrary to the original algorithm the random-based selection of planes may result in planes with better distributed orientations. This and the other questions will be picked up in the results section at 5.3.

### Plane Clustering

In Section 2.3 two clustering algorithms have been discussed. $K$-means clustering algorithms use a fixed number of clusters and try to assign the best faces to them. By contrast, the hierarchical face clustering algorithm grows as many clusters as required based on an cost function.

The argument for $k$-means clustering is that it allows choosing the geometric complexity of the simplification precisely, which is practical when a set of certain level-of-detail should be generated. On the other hand it is not possible to predict the average error and it absolutely does not guarantee any accuracy of the generated simplification. The same choice of $k$ will generate different results depending on the input geometry. So even if it is easy to configure, the user will always have to adapt and verify the result of every input geometry.

The second approach is based on hierarchical face clustering. Section 2.3 describes the process in detail. The goal is to minimize texture memory usage. This also has its price and results in significantly more planes. However, billboard cloud rendering will most likely be limited by fill rate instead of geometry, therefore this should not be the crucial performance factor. But another problem that comes with a higher plane count is that there definitely will be more artifacts caused by cracks between the planes. These artifacts are difficult to remove afterwards and can be seen clearly. Overall, the face clustering algorithm is more complex to implement, therefore we will only evaluate the $k$-means clustering algorithm in Section 5.3.

## 3.3 Improving Billboard Clouds

The planes given by the generation algorithms approximate the input geometry regarding the given constraints, which are either a maximum tolerable geometric error or clustering the geometry in $k$ groups of most similar faces. These properties are a good basis of the billboard cloud simplification, but additional improvements are required to get accurate simplifications.

In Section 2.4 we summarized the suggested improvements. The two most crucial are additionally optimizing the planes to improve the geometric accuracy and to reduce the unavoidable cracks between the planes. They will be discussed in detail in the following sections. Our additional thoughts on view-dependency and advanced texture mapping will be discussed as well. All these improvements can be used independently of the gen-

eration algorithm. Their reasonable applications will be pointed out.

## Plane Optimization

The original and stochastic algorithms only guarantee that the planes approximate the faces maintaining the maximum tolerable error $\epsilon$. In Section 2.4 we already summarized *post plane tweaking* which reduces the geometric error. It simply calculates a corrected plane orientation by averaging the face normals and then averaging the face distances to get the new plane distance. Additionally, the normals and distances are weighted with the face area.

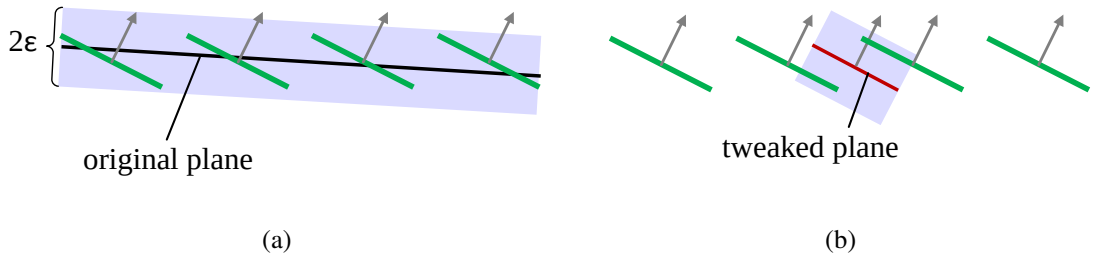



Figure 3.10: Illustration of a case where *post plane tweaking* fails: (a) Selected plane for independent faces with similar normals; (b) Tweaked plane by averaging the normals.

Assuming that the geometry is continuous, the averaged normals will give a good estimate of a well aligned plane. However, this method is not very sophisticated and a case where it completely fails can be found easily as illustrated in Figure 3.10. For example, when independent faces, which may be leaves of a tree, are selected, their average normal can vary significantly from their relative alignment leading to an unsuitable plane. Therefore, the addition that the tweaked plane is only used if it still simplifies all faces regarding the maximum error $\epsilon$ is used to reject these planes. This definitely fails very often and only brings an improvement in a limited number of cases.

A general method to find a best fit plane is reasonable. For example a least squares plane $ax + by + cz + d = 0$ that minimizes the sum of squared distances to the data points, which are in our case either the face vertices or the face center points:

$$f(a, b, c, d) := \sum_i |ax_i + by_i + cz_i + d|^2 \rightarrow \min \tag{3.10}$$

Setting the partial derivative with respect to $d$ equal to zero shows that the plane fulfilling this equation goes through the centroid of the data points. Substituted back into the

equation, the problem can be reformulated and solved using singular value decomposition, giving the normal as singular vector corresponding to its smallest singular value. It should be noted that such a method is also required to calculate the cluster centroids for the $k$-means clustering billboard cloud generation algorithm.

Since a nearly optimal plane is already given, we actually only have to search for a plane minimizing the error in $z$, called a regression plane. This problem can be formulated as equation in the form of:

$$f(z) := \sum_i |(Az)_i - b_i|^2 \rightarrow \min, \tag{3.11}$$

whereby A is the matrix containing the positions of the projected vertices as homogenous coordinates $(x_i, y_i, 1)$, $z$ the regression coefficients and $b$ the distances of the vertices to the original plane.
Then $z$ can be solved using the Pseudoinverse by:

$$z = (A^T A)^{-1} A^T b \tag{3.12}$$

The plane normal direction $n$ and the plane offset $d$ in local coordinates is then given by:

$$n = (z_1, z_2, -1) \tag{3.13}$$

$$d = z_3 \tag{3.14}$$

Additionally, the face vertices can be weighted by the face area, as done before. However, this will increase the maximum error, since a least squares method only minimizes the average distances. This would only worsen the result.

Furthermore, using least squares planes does not guarantee that all faces are inside the maximum tolerable error either, but we find this method a good compromise to approximate the geometry. The fact that faces may be outside the plane validity domain after the optimization should be ignored. If obtaining the maximum error $\epsilon$ is more important than an average good approximation, a plane reducing the maximum distance can be used instead. The calculation of such a plane involves finding a minimum bounding box of the faces.

## Crack Reduction

In Section 2.4 we pointed out that the billboard cloud simplification often results in planes with clearly visible gaps in between. This is an unavoidable problem that cannot be solved by adapting the generation algorithms. There will always remain faces with shared vertices that are projected onto different planes and which potentially result in a noticeable discontinuity.

The envelope projection by Huang et al. [HNW04] approaches this problem very accurately. Their method is able to remove the cracks nearly completely. Gaps are filled with the corresponding projected parts of adjacent faces. However, there are cases where gaps are almost unavoidable as illustrated in Figure 3.11. It shows two planes with intersecting envelopes each containing one face mapped by the other plane. In this case extending the planes and projecting the corresponding parts on both planes does not fill the gap. Such a method will only work when the relative angle of the oriented planes in less than 90°.

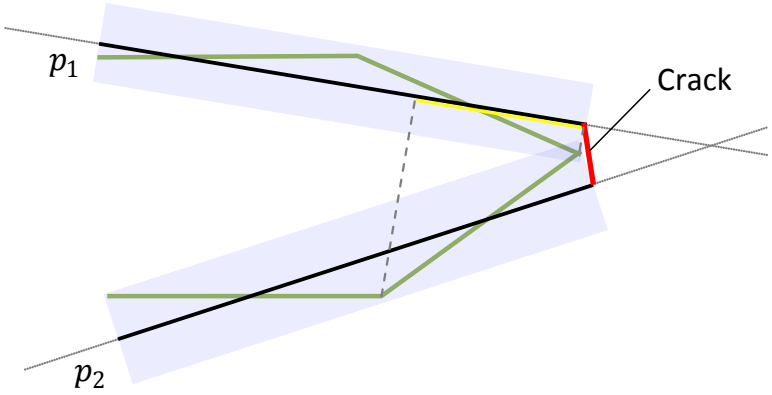



Figure 3.11: Projection of faces inside the intersecting plane envelope on both planes does not fill the gap.

The second method called *vertex welding* described in Section 2.4 tries to reduce the cracks by directly manipulating the geometry. Thereby, the faces are immediately projected onto the plane after it has been found. Hence, the density grid needs to be updated accordingly to modified faces before further planes can be searched. This method only reduces the artifacts but gives no guarantee of any improvement.

We like to present a different approach of *vertex welding*, which does not influence the generation process. The current method is unpractical in a modular system that implements several generation algorithms.

Given a set of planes with each a list of faces they simplify, the first step is to search the particular vertices that are mapped by different planes. Afterwards, the positions of the vertices are adjusted in order to reduce the gap. These positions are ideally the intersection of all planes on which a vertex will get projected.

Since billboard cloud generation is often controlled by a maximum tolerable error $\epsilon$, we limit the vertex displacement to a maximum distance of $\epsilon$. To avoid far distance intersection points in advance, we simply ignore planes whose orientation difference is less than a certain angle. In case there are still more than three planes that map a vertex, the latter are just ignored. Assuming that the planes are given in the order they are found and since most contributing planes are found first, the significance of remaining gaps should be negligible.

Compared to the method Huang et al. [HNW04] this should work in more cases. In the example of Figure 3.11 the shared vertex can be moved to the intersection of the plane. On the other hand this introduces shading errors caused by wrong face normals and distortions when the geometry is textured. In Section 5.2 our method will be evaluated.

## Plane Distribution

The extension to view-dependent billboard clouds by Fuhrmann et al. [FMU05] described in Section 2.4 is very useful for extreme simplifications in certain use cases. It tries to solve a problem of the original generation algorithm, which often chooses roughly parallel planes as best simplification for plant foliage, where there are lots of unconnected faces. Thereby, the densities of the density grid are globally adapted so that planes with certain orientations are favored. With their modification, planes orthogonal to the typical viewer position, where the camera is to be assumed on slightly above the terrain, are chosen more likely.

However, their approach requires a manually configured favored orientation and therefore actually only improves the representation for a limited number of cases. Since a geometry simplification useful for all view directions is favored we suggest an adaption of their approach. The weights of the density grid should be adapted each time a new plane has been found. Thereby, the weights for bins of planes with similar orientations than the selected one are reduced, so that parallel planes are restrained. This new approach should have better behavior in general applications, since it is independent of a certain predefined view point.

### Texture Clustering

Viewed from the resource aspect, billboard cloud simplifications mainly required texture memory. Resources required for the geometry is negligible. Hence, for huge scenes with lots of different objects memory, efficient billboard clouds are important. However, searching a minimum set of good approximating planes may result in lots of unused texture regions. An optimal solution is difficult and will influence the generation, but at least a mechanism to avoid the worst cases should be definitely used.

In Section 2.3 we already discussed a generation algorithm based on hierarchical face clustering that primarily focuses on efficient texture usage. Other generation algorithms do not consider this aspect directly.

The authors of the original generation algorithm suggest to use a simple clustering method after the densest plane has been found and continue the refinement by only considering the faces of the largest cluster. A more detailed description can be found in Section 2.1.

Since there are different generation algorithms, we will speak of planes although the equivalent term using the original algorithm is bin and cluster using $k$-means clustering. Something that has to be considered when the faces are clustered and finally only a subset will be mapped to a plane is that the original intention of finding the plane with the highest contribution is not followed anymore. To still find the highest contributing plane, all the clusters of the other planes ordered descending by their contribution have to be evaluated until a cluster has a higher contribution than an unclustered plane or bin. This guarantees that even though only a subset of faces has been selected it is the largest cluster of faces. However, it can be very expensive and the gained improvement will probably not pay off. It can be assumed that the densest plane also contains the biggest cluster or at least a large one and even though not the optimal result will be reached such an approach should work well in practice.

## 3.4 Evaluation

A part of this thesis is to evaluate different generation algorithms and improvement methods. Using manual evaluation is only applicable to a limited extend and a extensive comparison is not possible. In order to automate this process, an evaluation tool has been implemented that evaluates certain error metrics. A briefly described automatic evaluation has also been used by Huang et al. [HNW04].

## Metrics

The following paragraphs describe the used metrics to measure the difference between the original geometry and the simplification regarding certain aspects. All except the geometric error are calculated by comparing the rendered images of the billboard cloud with the original geometry.

### Coverage Error

The coverage error depicts the difference in terms of pixels where either the original geometry or the billboard cloud is drawn. This can be calculated relatively for each of the views. The number of pixels where the original geometry is drawn but not the billboard clouds tells how many pixels simplification is smaller including the gaps. Seen in relation to the billboard cloud, the pixel count indicates how much the simplification is miss-covering the original geometry.

Both values alone may be deceptive and are not that representative for a certain class of error. Therefore, we also calculate the absolute number of different pixels and use them as measurement for the general visual coverage accuracy.

### Gap Intensity

The most significant artifacts of billboard cloud simplification are the cracks between the planes. To measure them we use a simple edge detection filter to extract the border pixels between the geometry and the background. Assuming that a simplification with cracks and holes has a higher amount of border pixels, the difference of them gives an estimation of the intensity of crack artifacts.

This value does not allow measuring the size of the gaps. The size can only be deduced using the coverage error. However, because crack artifacts are dominant no matter how big the gaps are exactly, it can be clearly distinguished if and how many of these cracks are visible in the simplification.

### Color Error

Since the billboard cloud simplification is image-based, there will be a difference in the color and shading. Further differences of this kind occur because of re-sampling and a usually low texture resolution, which makes fine details unclear and the whole appearance blurry when the simplification is viewed from short distances.

Comparing the pixels one by one is not practical and will lead to high differences, because the textures on the planes represent projected image information, which is slightly moved from its original position. Measuring the error caused by the projection is not possible with such a method.

The intention of the color error metric is allowing the evaluation of the accuracy of the plane shading. For example to measure artifacts caused by distorted normals after vertex welding or when no texture filtering is used. because of vertex welding or when In order to calculate this error the average color for each view is calculated and the difference gives the overall color error. To get a perceptually representative value, the color difference is measured in the CIE Lab color space. RGB colors are first converted[1] to XYZ and then to Lab using the D65 standard illuminant.

### Geometric Error

The error metrics listed so far only evaluate the visual error by comparing the images. However, in some cases the error cannot be evaluated accurately enough. For example the results of different plane optimization approaches have barely noticeable differences. Therefore, we also use a geometric error for comparison.

The average and maximum displacements of the vertices are used, whereby the average displacement is calculated weighting each vertex by the areas of its faces. Since this error value is in object space, it is additionally scaled by the size of the projection of the model on screen to get a value in relation to the pixel size depending on the view distance.

[1] http://www.brucelindbloom.com/Math.html

Chapter 4

# Implementation

All the implementation has been done using the C# programming language in its latest version 3.0. Using a managed programming language made the implementation and debugging easy and we could focus on more general optimizations. Especially C# and all its latest features make applications more clear, which is a huge benefit when dealing with complex algorithms. The known performance drawbacks of managed compared to native code is in our option negligible when compared to the gained productivity that allows other optimizations.

To access the GPU the DirectX API has been used via the SlimDX[1] library. Hence, the shaders are written in HLSL. A path for DirectX 9 as well as for version 10 has been implemented.

The next sections describe how certain parts of our billboard cloud generation have been implemented in detail.

## 4.1 Generation Setup

The first step of the billboard generation is to prepare the input geometry for processing by the algorithm. Our generation algorithm works on an indexed triangle list. In case a single billboard cloud should be generated of several geometries, they have to be merged, and when there are no indices they are simply created.

Additionally, we chose to pre-compute practically everything that is used more than once during the generation. In detail there are the face normals and areas which are required for all algorithms used each time a plane is evaluated. Since faces are accounted for by multiple planes, this significantly improved the speed of all our tested generation algorithms.

[1] http://slimdx.org

Using the original billboard cloud generation algorithm, two preparation steps need to be done before the simplification can start, which also speeds up the process. On the one hand as described in Section 3.2 the geometry should be centered to have a predictable behavior during the plane finding. Therefore, a bounding sphere needs to be calculated first. Unfortunately, finding the optimal minimum sphere is not trivial. Since this is not the key part of the simplification algorithm, we chose to use a simple but still accurate enough solution. As center of the sphere the center of the axis-aligned bounding box of all vertices is taken. Then the radius, which is also required to define the correct rasterization in $\rho$, is calculated by simply iterating all vertices once again. Having the bounding sphere, the vertices are translated using the center of the sphere. This is something that otherwise would need to be done each time a face is tested for validity on a plane.

Second, the plane orientations, respectively the plane normals, parameterized in spherical coordinates by $\theta$ and $\varphi$ are pre-computed. The benefit of this is not much when this suggested parameterization of the plane orientation is used. However, when we later changed to a different parameterization described in the next section, this step was a significant improvement and additionally made the port easier.

These are the major steps of our setup. In comparison to the time consumed by the whole billboard cloud simplification the time for this setup does not carry weight, since it usually takes only a fraction of second. Detailed time statistics can be found in the evaluations in Chapter 5.

## 4.2 Plane Finding

In our implementation the plane finding is completely separated from the improvements and general tasks, such as plane optimization, crack reduction and the texture generation. To find the planes the original, stochastic and k-means clustering algorithms have been implemented. The procedures of these generation algorithms have already been discussed in detail in Sections 2.1 and 2.3. The implementation is straightforward and we will only point out a few important aspects in this section.

Among the three implemented algorithms the original is the most complex to implement. The rasterization and refinement are also very error-prone and require careful studying. Afterwards, implementing the other two algorithms was much faster. Since finding the planes often requires very similar tasks, they are separated into subroutines and shared by

the different generation algorithms. Additionally it makes the source code much cleaner.

For a few tasks the third party library OpenCV[1] has been used. This regards calculating the center of clusters for the k-means clustering algorithm where a best fit plane needs to be found. That problem is solved using a method to solve linear equation systems. Analog, a function that computes the two-dimensional convex hull of a set of points is used to define the plane extent and its minimum bounding rectangle.

## 4.3 Parameters of the Algorithms

In this section we list all possible parameters for the different generation algorithms. In our final implementation most of them should be chosen automatically when no explicit values are set.

The following parameters are configurable when the original generation algorithm is selected. The influence of these parameters will be extensively evaluated in Section 5.2.

| **Original** | |
|---|---|
| Refinement Type | Defines how the grid is refined.<br>Possible methods are: Original, Fail-Safe, None<br>They have been discussed in Sections 2.1, 3.2 and 3.2. |
| $\epsilon$ | Maximum allowed geometric error |
| Penalty | Weighting factor for missed faces |
| Raster Size | Configures the number of discretized plane orientations; the used data structure is discussed in Section 4.4 |
| $\rho$-Raster | Sampling rate of $\rho$ |

Our implementation of the stochastic generation algorithm has the following parameters.

| **Stochastic** | |
|---|---|
| N | Number of random planes to test per seed face |
| $\epsilon$ | Maximum allowed geometric error |
| Penalty | Weighting factor for missed faces |
| Seed | A seed value to be able to generate reproducible results. |

[1] http://opencv.willowgarage.com/wiki

The following table shows the configurable parameter of the k-means clustering algorithm.

| **K-Means** | |
|---|---|
| K | Number of clusters/planes |
| Max Iterations | The number of iterations when clustering is aborted |

The final table list parameters that are shared by all the generation algorithms. They configure tasks performed after the planes have been found, such as billboard cloud improvement and the texture generation of the planes. Additionally two parameters are used for our extended validity formulation, with the *Minimum Contribution* and *Normal Threshold* in order to skip less contributing and suboptimal faces as discussed in Section 3.2.

| **General** | |
|---|---|
| Minimum Contribution | Experimental parameter that skips unmapped faces with low contribution. It is in relation to the sum of all face areas. |
| Normal Threshold | A threshold that prevents faces with high normal deviation from the plane orientation. |
| Texture Resolution | Sets the texture resolution in relation to map a plane with the size of the bounding radius. |
| Crack Reduction | Sets the crack reduction technique.<br>Possible methods are: None, Projection, Welding.<br>They have been discussed in Section 2.4 and 3.3. |
| Plane Optimization | Possible methods: SVD, Least Squares, Average.<br>They are discussed in Section 2.4 and 3.3. |

## 4.4 Parameterization of Plane Orientations

In Section 3.2 the drawbacks of parameterizing the plane orientations using spherical coordinates have been pointed out and we mentioned that a different representation has been used.

For the generation algorithm to work, the neighborhood of orientations is required. Additionally, it is desired that all possible orientations are sampled as even as possible. Therefore, we use a projection of a cube on a sphere and address the orientation per index on the cube faces. Figure 4.1 illustrates this data structure, which has similar characteristic as cubic environment mapping. It shows the raster of the six cube faces with the normal

directions on the sphere.

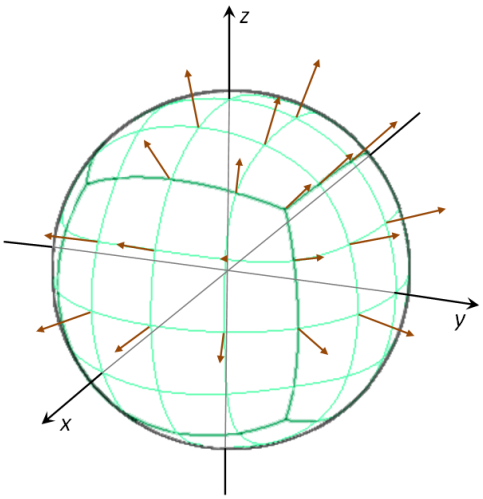



Figure 4.1: Orientations parameterized by grid of a cube projection on a sphere.

The distribution of orientations is quite well and it has significantly less distortion that regularly sampled spherical coordinates. To address the orientation each is assigned a unique index, starting per face as two-dimensional array, then an array for each edge and finally the corners. An encoding and decoding function does the mapping, whereby it can easily replace the original spherical coordinates.

As shown in the illustration there is a singularity at the corners of the cube where the faces meet. At these positions the neighborhood changes. However, it is no problem to still calculate the neighbors, simply the number is variable. This required only minor changes to the function that calculates the bin validity range and the refinement of the grid.

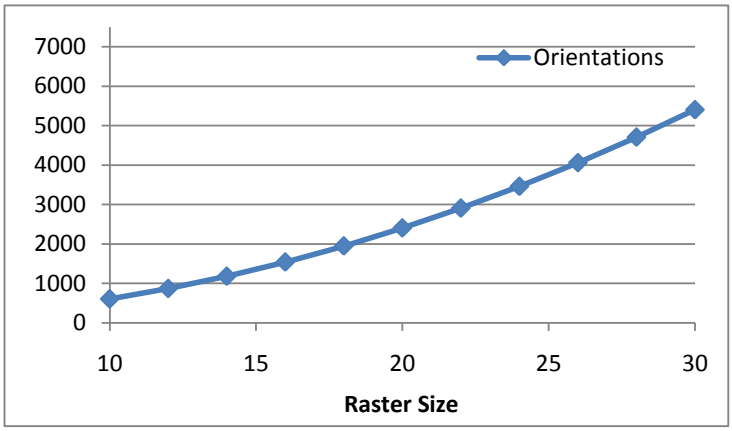



Figure 4.2: Number of total orientations for different raster sizes.

The resolution of this data structure is configured by an even raster size for each cube face. The even number ensures that the positive and negative axis directions (+X, -X, +Y, -Y, +Z, -Z) are exactly represented. Figure 4.4 illustrates the total number of orientations representable with our data structure for different raster sizes.

## 4.5 Texture Generation

The texture generation is the last step before the billboard cloud geometry is generated to complete simplification. In Section 2.4 advanced texture mapping techniques have been discussed. Shortly summarized, normal mapping should be definitely used if re-lighting is required. Bidirectional texture functions seem too excessive, since billboard clouds will be typically used as approximations for far distance geometry. For medium distances displacement mapped billboard clouds result in significant improvement and should additionally increase the performance of the application.

So far we had only time to implement simple shading with normal mapping. To generate the textures, a render-target is setup for each plane. The target resolution is defined by the configured texture resolution $texres$ and the ratio between the bounding sphere diameter $d$ and the extent of the billboard plane $(p_x, p_y)$.

$$\begin{pmatrix} width \\ height \end{pmatrix} = \begin{pmatrix} p_x \\ p_y \end{pmatrix} * \frac{texres}{d} \tag{4.1}$$

Using this equation, all texels will map an area with the same size in object coordinates.

To render the plane faces to the texture, the authors of the original paper simply focus on the plane extent with two clipping planes at $\pm\epsilon$. Because our simplification also supports other generation algorithms that do not necessarily guarantee a maximum geometric error, this process needs to be adapted. We still want to render not only valid faces but also all faces connected to them, which will reduce cracks between the planes as described in Section 2.1. Therefore, the minimum and maximum $z$-value of all valid faces in local plane coordinates are calculated and used to set the clipping planes.

When a texture with the calculated size is rendered and used directly, aliasing artifacts are clearly visible. Such a texture is only usable when the actual size on the screen is much less than its resolution and a higher mip-level is used to sample the color. Therefore, intermediate render targets with a multiple resolution for the diffuse and normal textures are created. A resolution corresponding to $4 \times 4$ supersampling turned out to work well

and is used as default. Then the rendered textures are down-sampled using a specific filter for the diffuse color and another for the normal map. Both only build the average of non-background pixels. In a DirectX 10 implementation multisample anti-aliasing can be used instead of supersampling, because the API allows direct access to the sub-samples required by the filter.

Figure 4.3 illustrates texture aliasing artifacts and the filtered result. Image (a) and (c) show billboard clouds of a tree and a box where the textures have been rendered in the calculated resolution and directly used for the simplification. The scattered pixels in the foliage are very thin branches mostly missed by the rasterization and the simplification of the box shows a distinctive dark stripe at the border of the top cavity. When the described supersampling approach is used, the texture quality can be significantly improved as shown in image (b) and (d). The details are sharper and the borders are smoother.

(a) No Filter (b) Supersampling

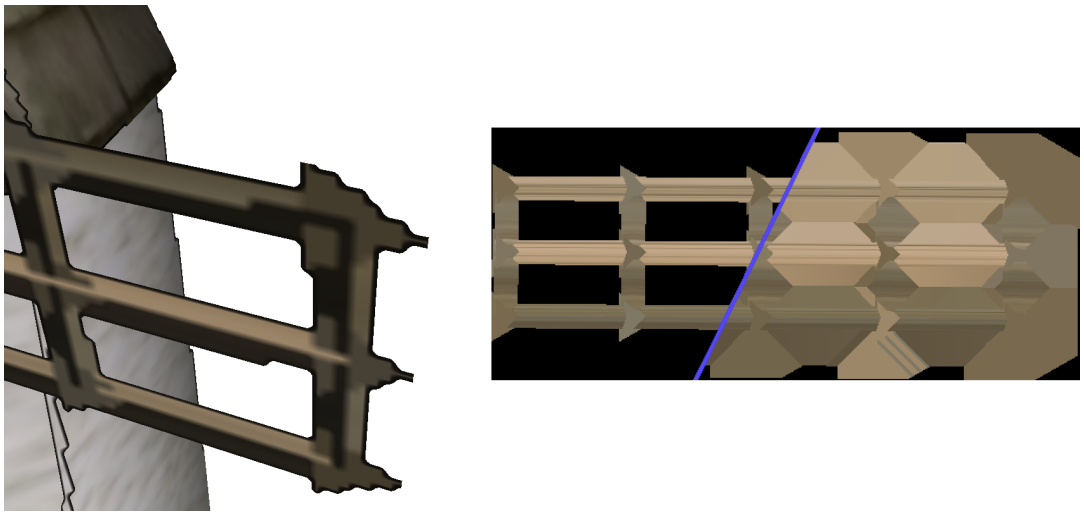
(c) No Filter (d) Supersampling

Figure 4.3: Illustration of texture aliasing artifacts: (a) and (c) show billboard clouds where the textures are generated and used directly in the calculated size; (b) and (d) show the improvement when our described supersampling approach is used.

Afterwards, all textures are packed to a texture atlas using a greedy packing algorithm to allow efficiently rendering the billboard cloud. It is important that the textures are packed using a suitable padding offset depending on the configured texture resolution and desired view distance, so that neighboring textures do not overlap in the used mip levels. Currently a fixed offset of 8 is used, which is relative large. An offset of 4 is probably a suitable general choice.

Since the billboard textures have to be rendered using the alpha information, post processing of the packed texture is required. The intended rendering technique for billboards is to use alpha-testing, which simply rejects pixels below a certain threshold. If the texture would be used directly, texture filtering would blur the surrounding black areas of the texture over the color information, causing unaesthetic artifacts at the borders as illustrated in Figure 4.4 (a). Simply using a high alpha threshold causes the texture to shrink and important details disappear when the billboard cloud is viewed from greater distances. Only a threshold of 0.5 keeps this in balance, but the artifacts are clearly visible. Therefore, we filter the textures and fill each background pixel with the color of the nearest none-background pixel. Figure 4.4 (b) shows the outcome when this filter is applied. When solid geometry has been rendered to textures, the distinction between background and geometry is easy, but in lots of cases such as plant foliage most of that is also alpha-blended. We currently also use an alpha-threshold of 0.5.

The described filters to down-sample the textures and the filter to find the nearest none-background pixel have been implemented as pixel shaders and are executed on the graphics hardware. If implemented on the CPU, additional operations to retrieve the rendered textures from GPU memory would be necessary and even small radii of the border-filter would be very time consuming to compute.



(a) (b)

Figure 4.4: Plane texture filtering: (a) Result when unfiltered textures are used whereby the black background creates a halo around the rendered geometry. (b) Outcome when filtering that fills the background with the color of the nearest foreground pixel is applied. The left and right side illustrate the texture before and after filtering.

## 4.6 Parallelization

Most of the time for the billboard cloud generation is spent in loops that perform complex computations and have loose data dependency between the iterations. This is very well suited for parallel processing. In order to take advantage of multi-core processors, the parallel extensions of the .NET framework have been used. Thereby, the whole processing power of modern CPUs can be utilized with very little additional programming effort and without having to do the thread management explicitly.

The following code sample shows a loop evaluating a plane during the stochastic generation algorithm implemented the traditional way and its parallel version using the parallel extensions.

```
if (mProcInfo.Param.MultiThreading)
{
    Parallel.For(0, N, i =>
    {
        planedensity[i] = CalcPlaneContribution(planes[i], F);
    });
}
else
{
    for (int i = 0; i < N; i++)
    {
        planedensity[i] = CalcPlaneContribution(planes[i], F);
    }
}
```

In this example the data of the operation is independent and therefore this optimization can be done without any concerns. However, this is only guaranteed in very rare cases. Depending on how different iterations share data, it can get quite complicated. Typically, locks are used to synchronize shared data access which, however, often results in blocking for other threads and does not scale very well. An alternative approach to avoid locks is to duplicate the working data and merge the result afterwards. The overhead of merging is usually negligible and results in much better scaling behavior.

An example where we experimented with locks was for initializing the density grid, which is a huge array containing the density values. Thereby, for each face a loop iterates over all orientations and checks if there is a validity range in $\rho$. If so, the contribution is rasterized using our falloff function. The whole density grid was simply locked directly before the

contributions were accumulated. The assumption was that most computation time is spent outside the critical data accessing section. However, the performance gain on a dual core CPU was 60% at maximum and much less in average cases, which was not satisfactory. The reason probably is that the faces are rasterized one after another and two neighboring faces have very likely overlapping validity domains.

To avoid locks, the nested loops have been rearranged at the cost of a 5-10% overhead. The total runtime of the new implementation on the tested dual core CPU was significantly less than the previous one. Additionally, we expect a much better scaling behavior.

Due to the massive computation power of GPUs, general-purpose computing on graphics processing units (GPGPU) is very tempting and can bring a huge performance improvement for selected tasks. We chose a particular example for demonstration and to experiment with this feature. Programming the GPU can be done in two ways. One is using the rendering pipeline and the other accessing the GPU directly by using specific libraries.

In our demonstration we used the traditional approach using the rendering pipeline. The same loop of the example above has been selected for our GPU implementation, because the data is independent for the iterations of the loop and is the significant time consuming part of the stochastic billboard cloud generation. The data required for processing is packed to textures and certain vertex- and pixel shader have been written to process the data. In detail the plane evaluation simply iterates over a list of triangles and aggregates the sum of the coverage of all valid triangles for a plane. Therefore, the vertices and the index-list, as well as all planes to evaluate are accessed over texture lookups. A render-target that will hold the result with the same size than the plane texture is set up and a full-screen quad is drawn over it, whereby each pixel processes a plane.

Even an approach like this which has the overhead of the rendering pipeline and rasterization for GPGPU brings a huge performance improvement of a factor of up to 80 under optimal conditions. Alternatively the GPU can be accessed directly with vendor dependent architectures such as NVIDIAs CUDA[1]. This has less overhead and gives more control over the program flow.

We have not yet optimized all parts of the simplification using parallel processing, but with the chosen parts the potential and scaling behavior can be demonstrated very well. The exact numbers can be taken from the performance evaluation in Section 5.4.

[1] http://www.nvidia.com/cuda

Chapter 5

# Evaluation and Results

In the theory chapter various questions and assumptions of the different generation algorithms and improvements have been made which should be answered here. We will first look at the original generation algorithm and different general aspects of billboard cloud simplification. Finally we make a direct comparison of the three implemented generation algorithms.

All evaluations have been made using a test system equipped with a Core2Duo @ 2.7Ghz, 4GB Ram and a NVIDIA GeForce GTX 285 under Windows Vista 64bit. To compute the image based metrics a resolution of of $2048^2$ has been used. If no distance has been specified, the comparison was made using a standard distance of two times the bounding sphere diameter.

## 5.1 Method

In Section 3.4 our implemented error metrics to measure certain types of artifacts have been described. During the evaluation we will select suitable metrics to measure the studied effect.

One goal of the evaluation is to identify cases where a specific generation algorithm or improvement approach is superior. Therefore, the evaluation is initialized for manually set generation parameters and a view-distance. Then the above described metrics are calculated for a certain number of evenly distributed view-directions and the average and maximum errors are used.

Since suitable default values or heuristics for certain parameters required for the billboard cloud generation should be found, also an extensive automatic evaluation method is used. First the generation parameters or range of values and set of options is specified. After the billboard cloud has been generated for one set of parameters the above listed error metrics

are computed like for the manual evaluation.

## 5.2 Simplification Using the Original Algorithm

The original algorithms is the most complex one. In this section we will handle the choice of its basic parameters for the grid resolution. Then the effects of penalty, our proposed distance falloff and face skipping are measured. Afterwards, the discussed billboard cloud improvement approaches are evaluated. Finally, we show how to automatically find a suitable error threshold and texture resolution for a simplification intended to be used at a certain distance.

### Grid Resolution

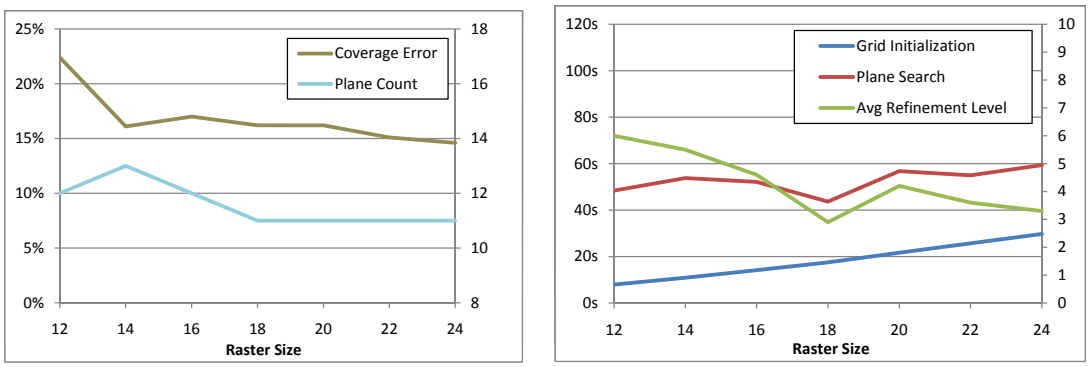
One of the first things that have to be done when implementing the original algorithm is to set up the density grid, which is the key part of this generation algorithm and has most influence on quality and performance. In Section 3.2 we reasoned that using a well chosen fixed resolution for the plane orientations should be used and does not have big influence on the generation result. The same applies for the resolution in which the plane offset is sampled that is controlled by a factor in relation to the error threshold $\epsilon$. The following results should clarify this matter.

We chose a screw driver model with 6,954 faces in order to demonstrate the effect of different resolutions. Because of its oblong form factor, the generation algorithm is more sensitive to small changes of the plane orientation. Figure 5.1 shows the original model and two simplifications with an error threshold of $\epsilon = 5\%$. The graph in Figure 3.9 suggests to use of about 100 samples for $\theta$, which comes to a raster of $24 \times 24$ per cube face in our data structure described in Section 4.4 and gives a total of 3,458 orientations. During this evaluation the plane offset $\rho$ has been sampled at a rate of $\frac{1.5}{\epsilon}$

A clear difference between (c) the highest and (b) the lowest tested resolution can be seen. The following table shows the exact number of the tests with decreasing resolutions. We chose to compare the coverage error which significantly deviates with the lowest tested resolution, although two additional planes have been found. It can be noticed that the algorithm gets instable and has problems to find the optimal planes simplifying faces along the whole model. At a manual visual test all the other results were practically equal except that artifacts where on slightly different locations.

(a) (b) (c)

Figure 5.1: Illustration of different resolutions of the plane orientations. (a) shows the original geometry; (b) and (c) the simplification results of the lowest and highest tested raster of 12 and 24.




(a) (b)

Figure 5.2: Evaluation result using different raster sizes to discretize the plane orientations: (a) Illustrates that the coverage error is decreasing even though the plane count is almost consistent at approximately 11; (b) Shows a linear increase of time for the density grid initialization and illustrates the relation between the time required for the grid refinement and the average refinement level.

The graph in Figure 5.2 (a) visualizes the change of the coverage error and the resulting plane count. It shows that error can be reduced by increasing the grid resolution, even though the plane count is mostly consistent at 11. For resolutions below a raster size of 16 having a total of 1,538 orientations the generation algorithm gets unstable .

The computation time and average refinement level is visualized in (b). The x-axis shows the range of the tested raster size from 12 to 24. It has to be noticed that the number of orientations is not linearly dependent on the raster size as visualized in Figure 4.4 where our used parameterization is discussed. However, the initialization time appears to increase linearly with a factor slightly below one to the number of orientations. The reason is that a face of the input geometry always only covers a few bins and the others can be skipped quickly.

The duration for the plane search appears to vary around the same time and only increases slightly for the highest tested resolutions. This consistency is achieved because the number of required refinement levels are reduced by increasing the resolution. Hence, the total time is dependent on the time for the grid initialization.

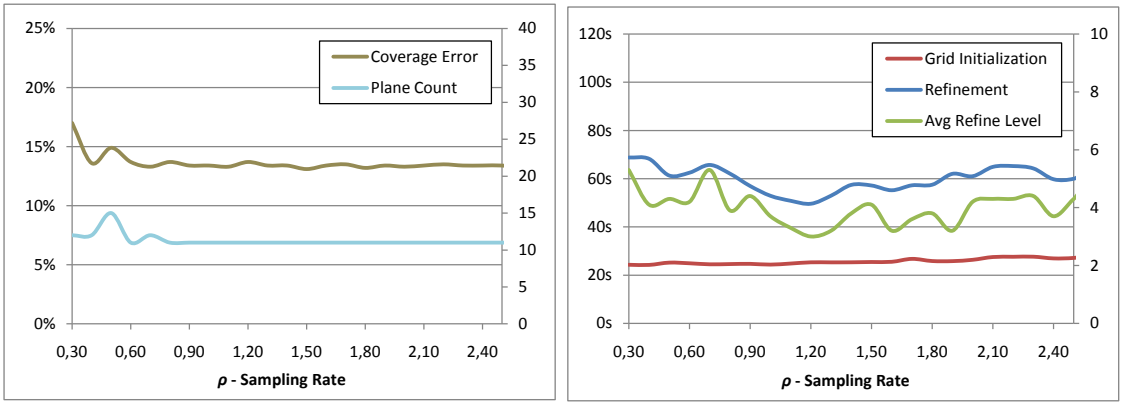
When the refinement is turned off, the results drastically change and a difference in quality can be noticed between every test. The refinement obviously works very well, even though it is limited to the initial selected faces and can only explore bins in the immediate vicinity.

This evaluation showed that varying the resolution in the tested range has only minor effect on the total runtime and a preferably slightly higher resolution can be used without any concerns. We find a raster size of 18 or 20, respectively approximately 2,000 to 2,500 orientations, is a good choice and there is still a buffer until the generation gets unstable.

Next, different sampling rates of the plane offset $\rho$ are evaluated. The theory behind that has been discussed in Section 3.2. Again the screwdriver geometry with the same parameters from the evaluation above has been taken. The results for the coverage error in graph (a) of Figure 5.3 show only very small numerical differences for sampling rates above $\frac{0.75}{\epsilon}$ and the plane count is stable at 11. Visual tests also confirm that there is no difference. For lower sampling rates the result gets unstable and the grid refinement tends to fail to find a refined plane for the selected bin.

As expected Figure 5.3 (b) shows not much difference in computation time when changing the resolution of $\rho$. The time for the density grid initialization only increases at very small factor in relation to the sampling rate. The refinement behaves slightly different. For medium values the refinement time is reduced, but with increasing sampling rates it also increases again. This is coherent to the average refinement level which shows a quite similar curve. It can be said a too high sampling also has slightly negative influence on the generation result.

It has to be noted that our falloff of contribution in relation to the plane distance as described in Section 3.2 has been used. This leads to the searched plane faster during the refinement and therefore also has an effect on the runtime. A more detailed evaluation of the effect of our distance falloff follows in Section 5.2.



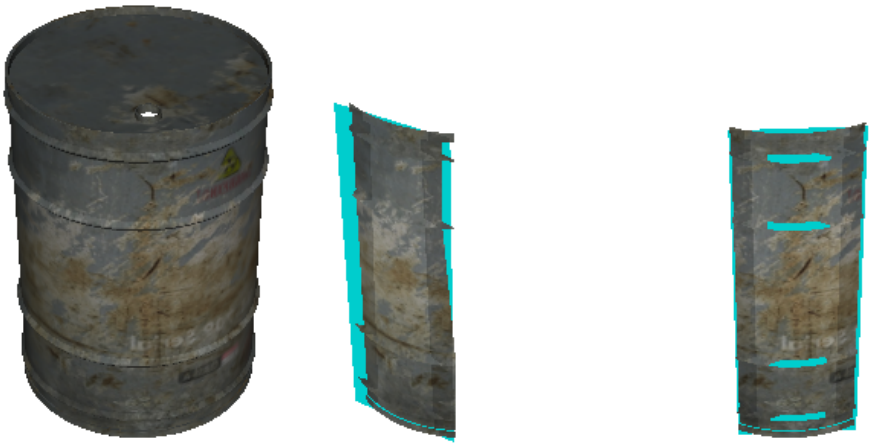

(a) (b)

Figure 5.3: Evaluation result varying the sampling rate for the plane distance $\rho$: (a) Shows a consistent plane count and coverage error for sampling rates above 0.75, below that the algorithm gets unstable; (b) Illustrates that the time for the grid initialization only increases by a very small factor and the correlation between the average refinement level and the time for the plane search.

Concluding these tests, using a higher resolution for $\rho$ does not cost very much, so we decided to use a sampling rate of $\frac{1.5}{\epsilon}$ for all our simplifications.

## Penalty

The authors of the original billboard cloud algorithm already pointed out the importance of penalty which has been discussed in Section 2.1. It is used to prevent planes from missing faces that otherwise would have to be simplified with separate planes and potentially



(a) Original Model (b) Plane with penalty (c) Plane without penalty

Figure 5.4: Illustration of the penalty effect. (a) shows the original geometry; (b) and (c) show a comparison of planes found with and without penalty;

cause additional artifacts. This problem especially occurs when the input geometry has round shapes.

To demonstrate the importance of penalty we chose a model of a barrel with 6,270 faces that has some irregularities along its cylindrical body. All simplifications have been generated with an error threshold of $\epsilon = 4\%$, a raster size of 18 to discretize the plane orientations and a sampling rate for $\rho$ of $1.5\epsilon$.

Figure 5.4 illustrates the problem. Without penalty the algorithm correctly chose a plane slightly missing the tip of the bulges (c). This slightly inwards moved plane can map the biggest area, but the remaining faces have to be simplified separately. This results in additional interleaved planes as clearly visible in Figure 5.6 (a).

To evaluate the effect of penalty, the results of weight factors from 0 to 10 have been compared. Figure 5.5 illustrates the coverage error and the resulting plane count for different penalty weights. It shows that simplification without penalty has a relatively high plane count and coverage error. When the penalty weight is increased, first the plane count and error decreases. For higher weights the plane count increases again, but also further reduces the error.

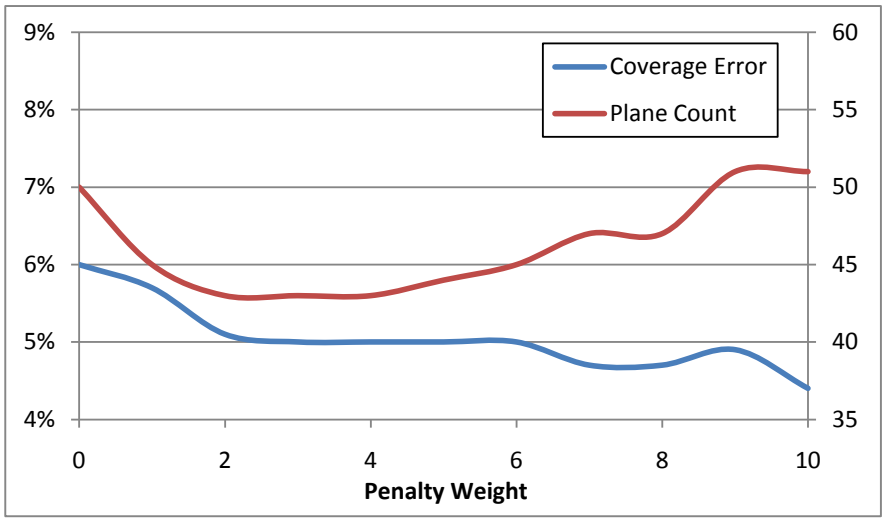



Figure 5.5: Penalty test results with different penalty weight factors.

Originally penalty should be weighted with a factor of 10, which yielded the best result in terms of accuracy. However, it is achieved at the cost of a higher plane count, which may not be the desired result. In our opinion a decent penalty weight of about 2 to 5 is better suited to get practical simplifications. A penalty weight as high as 10 can also cause other

problems which we could find easily. In some cases the penalty cancels out the complete contribution and even the highest densest bin has a negative density. For example a face where a plane slightly closer is penalized by even more faces and therefore is not selected. This phenomenon caused huge problems when we evaluated the algorithm with the Eiffel tower model.

(a) 50 Planes, Penalty = 0 (b) 43 Planes, Penalty = 3 (c) 51 Planes, Penalty = 10

Figure 5.6: Generation results with different penalty weight factors: (a) Simplification without penalty; (b) Result when a decent penalty weight is used; (c) Result of a high penalty weight.

Figure 5.6 compares the generation results of (a) no used penalty, (b) a decent penalty weight of 3 and (c) high weighted penalty. The plane count of no and high penalty is approximately the same, but the result without penalty clearly shows much more artifacts. A high penalty reduces the number of planes to simplify the body of the barrel, but image (c) shows that is has slightly more problems simplifying the small remaining faces on the rim. In visual tests decent penalty weight results in the best simplification and simultaneously has the lowest plane count in this test. We could measure similar results with other geometries and therefore we chose to use a default penalty weight of 3.

In comparison the effect of penalty when using the stochastic billboard cloud generation algorithm are analog. As shown in the evaluations in Section 5.3 the stochastic algorithm has more difficulties finding good planes and penalty helps to miss faces that otherwise would have to be simplified separately.

In addition we could verify that perfect spheres do not cause the algorithm to miss faces at the tip without penalty even tough everything is discretized. We used different trian-

gulizations of the spheres and as in theory, the mathematically optimal planes mapping whole caps have been found. The problem came afterwards when the remaining faces in between where simplified, which resulted in some suboptimal planes.

## Distance Falloff

In Section 3.2 an additional falloff in relation to plane distance of faces for computing the contribution has been presented. It should guide the generation algorithm faster to the final plane during the refinement. This is especially important when there is a high resolution for the plane offset $\rho$ that will be reached sooner or later during the refinement.

(a) Original (b) Original refined (c) Distance falloff

Figure 5.7: Demonstrating the effect of using a distance falloff. (a) shows planes selected from densest bin without refinement and without distance falloff; (b) shows the planes of (a) after the refinement. (c) shows the effect of the distance falloff on planes directly selected from densest bin without refinement;

Depending on the specific parameters, the initial selection of the densest bin can vary from quite good to very bad. We chose a simple box with some small details on the sides that has been additionally rotated around its principal axes so that it is not perfectly aligned in the density grid. Figure 5.7 (b) shows an absolute worst case resulting from a raster size of 20, $\rho$-Raster of 2, $\epsilon = 6\%$ and a penalty weight of 2. The refinement has difficulties in finding properly fitted planes and does not manage to approximate the original shape very well (c). When our described distance falloff is used the planes directly selected from the density grid are already very well aligned (a) and practically no refinement is required any more.

Using the falloff already during the initialization is only practical when the sampling rate of $\rho$ is high enough. A rate of $\frac{1.5}{\epsilon}$ as we decided to use as default is absolutely suitable.

We recommend not only to use the falloff during the refinement, because the refinement has its limits and when the initial planes are as misplaced as in Figure 5.7 (b) the result will still have a high geometric error.

The following table shows the evaluation results of different more complex models with the same parameter from above. Figure 5.8 illustrates the percentage reduction of the average refinement level and the average geometric error. A significant reduction of the geometric error and in almost every case a reduction of the refinement cost can be seen. As mentioned before the distance falloff will also reduce the runtime of the grid refinement, but except one outlier only minor changes could be measured.

| (Original / Falloff) | Teapot | Screwdriver | Gas Can | Barrel |
|---|---|---|---|---|
| Refinement Time (sec) | 4.5 / 4.9 | 56 / 43 | 1.7 / 1.6 | 33 / 32 |
| Avg Refinement Level | 1.4 / 1.8 | 3.1 / 2.4 | 1.8 / 1.6 | 1.9 / 1.6 |
| Plane Count | 22 / 20 | 16 / 14 | 16 / 18 | 44 / 43 |
| Avg Geometric Error (%) | 1.4 / 1.3 | 1.4 / 0.7 | 1.7 / 1.0 | 1.4 / 1.0 |

Table 5.1: Comparison between original algorithm and additional distance falloff.

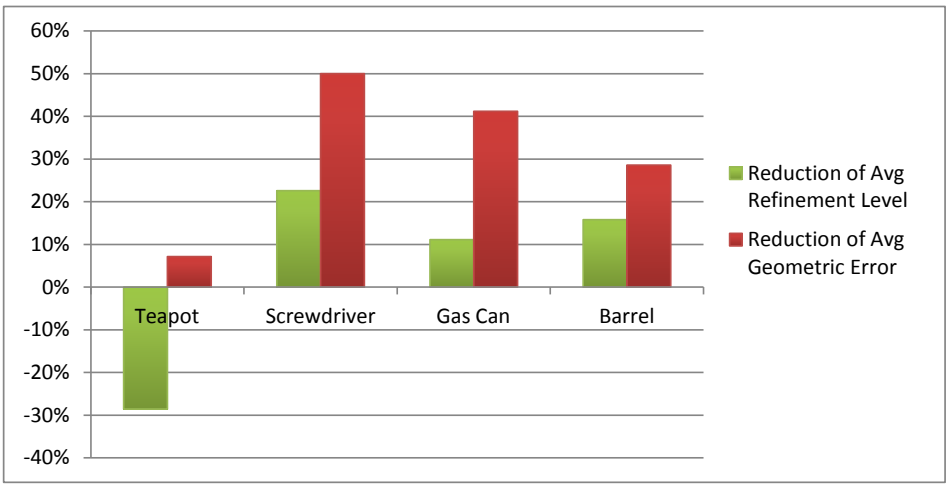



Figure 5.8: Percentage reduction of the average refinement level and the average geometric error of the four test cases of Table 5.1.

The overall impression is very positive and shows similar effects as plane optimization, which will be evaluated in the next subsection.

## Plane Optimization

The billboard cloud generation often ends up with suboptimal planes, since only a certain error threshold is guaranteed. When objects have perfectly aligned structures the planes usually fit them only approximately. Such misplacements will be noticed more likely. In Section 3.3 different approaches to optimize the plane alignment have been discussed.

In the following evaluation we will compare best fit least squares planes and *post plane tweaking* that simply averages the vertices and normals. To evaluate, an error threshold of $\epsilon = 5\%$ has been used and no distance falloff to get clearly misaligned planes.

(a) Skewed (b) Optimized (c) Distance Falloff

Figure 5.9: Plane optimization: (a) Original algorithm with no optimization; (b) Optimized planes using the least square method; (c) Original algorithm with additional distance falloff.

Figure 5.9 (a) shows clearly skewed planes causing a disturbing impression of the geometry. Even the inner structure of the hut intersects with the walls and is visible from the outside. With both plane optimization approaches the original structure is obtained very well and practically no difference can be depicted, so we show only the result from the least squares method.

When the distance falloff has been used during the generation the initial result already follows the overall structure quite well. Additional, optimization mostly causes the plane to shift slightly. Between image (b) and (c) only minor changes can be noticed.

The following table lists the results using plane optimization compared to the original result without using the distance offset. For this, meshes with distinctive geometric shapes have been used. A clear improvement can be read from the data. However, these values do not directly reflect the visual quality, because geometric abnormalities are perceived

more precisely and would cause an even more unaesthetic look.

| (Original / Optimized) | Hut | Box | Gas Can | Windmill |
|---|---|---|---|---|
| Coverage Error (%) | 8.5 / 3.2 | 2.2 / 1.9 | 5.0 / 4.4 | 5.6 / 5.3 |
| Avg Geometric Error (%) | 1.4 / 0.2 | 1.2 / 1.2 | 1.5 / 0.9 | 1.4 / 0.8 |

Table 5.2: Evaluation result of the unmodified original algorithm with $\epsilon = 5$ % to additional plane optimization using least square planes.

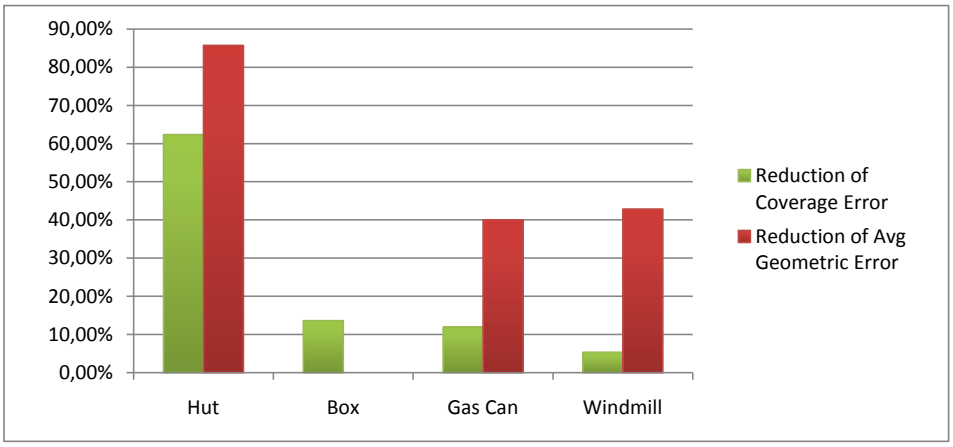



Figure 5.10: Percentage reduction of the coverage and average geometric error when using plane optimization in the test cases of Table 5.2.

Figure 5.10 shows the percentage improvement when using the plane optimization. A consistent reduction of the coverage and the geometric error can be noticed. Especially with the hut mesh the original algorithm has difficulties to find well aligned planes as already shown in the screenshots of Figure 5.9. The other meshes have slightly more extensive structures whereby the error of the original algorithm is in the expected range, but additional plane optimization is still capable of reducing the error and improving the result.

Furthermore, the plane optimization is beneficial for the stochastic generation algorithm and allows using a reduced number of planes that are evaluated per seed face. Thereby, the planes only have to catch the faces in its validity domain and then the plane optimization does the final plane fitting.

## Skipping the Refinement

In Section 3.2 we discussed two modifications of the original billboard cloud algorithm. The questions we will try to answer is if it is worth all the effort to do the refinement

and how large is the total error if the planes are directly selected from the density grid. In Section 5.2 we have already evaluated the distance falloff and showed that the density grid already gives well aligned planes. The plane optimization evaluated in the section above also improves the plane alignment, which both now have to compete against the refinement in the following tests. Thereby, a maximum error of $\epsilon = 6\%$, a penalty weight of 1 and default raster size of 18 has been used.

| (Refinement / Density Grid) | Barrel | Teapot | Screwdriver |
|---|---|---|---|
| Coverage Error (%) | 9.6 / 13.9 | 19 / 23.9 | 19.5 / 21.4 |
| Avg Geometric Error (%) | 0.8 / 0.9 | 1.4 / 1.5 | 0.6 / 0.7 |
| Plane Count | 39 / 30 | 22 / 15 | 11 / 10 |
| Plane Search (sec) | 61 / 42 | 9 / 6 | 83 / 51 |

The result does not surprise, skipping the refinement introduces a clearly measurable error and approximative 30 percent reduction of time consumed by the plane search. However, it has to be considered that these simplifications also have less planes, whereby a higher error is consequential. Reducing $\epsilon$ so that the simplifications have the same plane count the error values come to approximately the same level, but the time reduction benefit remains. When the computation time is important this modification can alternatively be used with the original algorithm, but the choice of $\epsilon$ has to be made more carefully.

## Face Skipping

The second modification we presented in Section 3.2 reduces the plane count by stopping the plane search when the area of the remaining faces drops beneath a certain threshold. Optionally, these faces can then be assigned to suitable planes, which has been used in this evaluation. Ideally, this modification should not significantly influence the generation result. To evaluate the effect we chose to use a low error threshold of $\epsilon = 4\%$ in order to get accurate simplifications with lots of planes. The density grid has been set up with our default resolution. Improvements like penalty, plane optimization and the distance falloff have been used and additional vertex welding has been used with the barrel and the screwdriver model.

| (All Faces / Skipping) | Eiffel Tower | Barrel | Teapot |
|---|---|---|---|
| Coverage Error (%) | 38.1 / 38.3 | 3.9 / 4.9 | 8.9 / 10.8 |
| Avg Geometric Error (%) | 0.6 / 0.6 | 0.4 / 0.3 | 0.5 / 0.6 |
| Plane Count | 54 / 38 | 65 / 50 | 47 / 39 |
| Skipped Face Area (%) | 0 / 1.5 | 0 / 5 | 0 / 5 |

An error increase can definitely be read from the table, but on the other hand the plane count is reduced notably. It has to be considered that a relatively high number of faces has been skipped. For example, the skipped faces at the barrel were lots of the small difficult to simplify faces on the top rim and the closure.

Figure 5.11 illustrates the skipped faces and the difference when using face skipping during the test with the Eiffel tower model. The skipped faces visualized in image (b) do not appear to be important parts of the model, but as listed in the table above it takes 16 additional planes to simplify them. When manually comparing image (c) and (d) the difference can hardly be noticed. There is also hardly any change in geometric error. Because of all the fine structures, the coverage error is not very meaningful.

(a) Original Model (b) Skipped Faces (c) With Skipping (d) All Faces

Figure 5.11: Face skipping: (a) Original geometry of the Eiffel tower mesh; (b) Skipped faces when using a 1.5% area threshold; (c) Simplification result with 38 planes where skipped faces have been added to the nearest planes; (d) Result of the original algorithm giving 54 planes for all faces.

The coverage error turned out to be not very representative of the simplification quality for the Eiffel tower mesh, because the geometry has lots of extremely fine structures. Comparing the plane count and doing manual visual tests clearly show the potential of face skipping.

Without the use of face skipping, the resulting simplifications would be hard to achieve otherwise. However, the effect of face skipping strongly depends on the input geometry. Either only a very small threshold is practical to use in general, or a manually tuned one to obtain the desired optimization result.

## Vertex Welding

Most images of our generated test results show billboard clouds that have lots of clearly visible gaps and cracks between the planes. Such simplifications are not very practical and will cause an unpleasant impression that can be perceived in all distances.

To evaluate these artifacts we implemented a special evaluation metric described in Section 3.4. Since only border pixels are counted and not the area of the cracks, it has to be noticed that it only gives an estimate of how intensive these artifacts are. Additionally, the size of crack artifacts is partly included in the coverage error, but since it generally measures the misplacement of planes, the area of gaps is not evident.

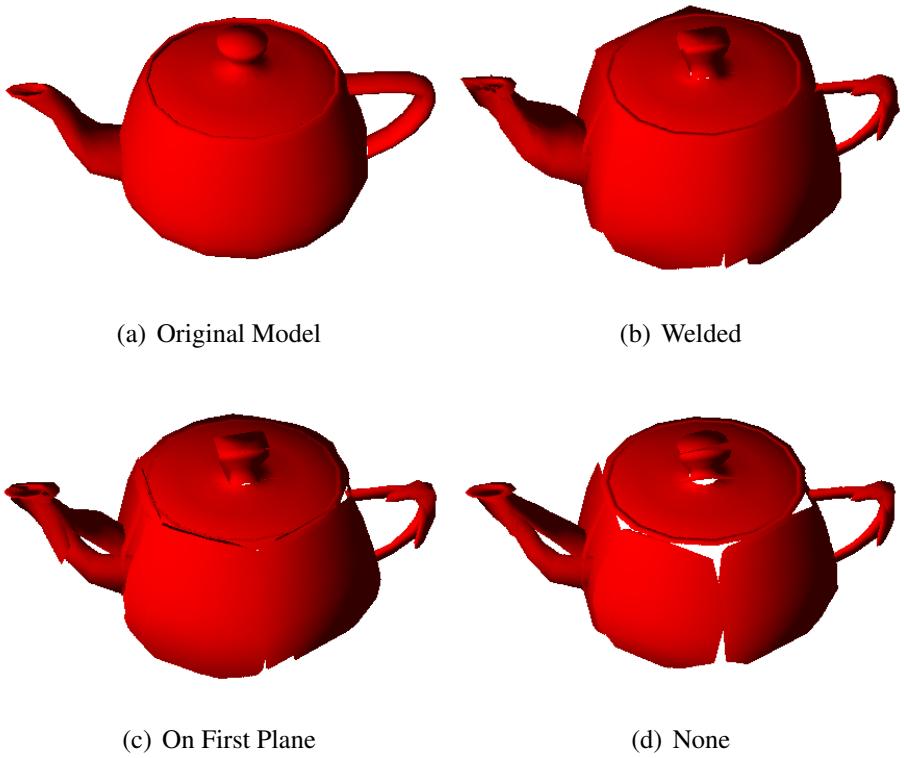
In Section 3.3 we discussed different approaches to reduce crack artifacts and described our implemented vertex welding. Besides our technique, a slightly modified version of the welding technique of Fuhrmann et al [FMU05] will be used as comparison. Their original approach directly moves the vertices on the plane after it has been found and then continues the plane search. In difference our implementation simply projects the vertices on the first plane afterwards, because we did not want to make this step part of the plane search algorithm.

For all simplifications during this evaluation all other optimization techniques have been used with default parameters and an error threshold of $\epsilon = 6\%$ was set. Figure 5.12 illustrates the problem very clearly in image (d) where the closed surface of the teapot has lots of disturbing holes. Welding the torn-apart vertices at the plane intersections nearly completely removes the cracks (b). Simply projecting them on the first plane also brings a distinct improvement, but some artifacts remain (c). On the other hand the welded vertices change the outline of the geometry.

Table 5.3 list the gap and coverage error of the test cases. Additionally the color error is listed, because we suspect an influence of welding measurable with that metric.

| (Original / Welded) | Teapot | Gas Can | Barrel | Windmill |
|---|---|---|---|---|
| Gap Intensity | 0.42 / 0.11 | 0.42 / 0.07 | 0.96 / 0.29 | 0.12 / 0.04 |
| Coverage Error (%) | 10.8 / 11.5 | 6.0 / 5.8 | 9.2 / 5.6 | 8.1 / 6.7 |
| Color Error (%) | 0.47 / 0.53 | 0.82 / 0.87 | 0.36 / 0.35 | 1.03 / 0.90 |

Table 5.3: Evaluation results of testing the influence of our vertex welding method on different geometries.



(a) Original Model (b) Welded

(c) On First Plane (d) None

Figure 5.12: Crack reduction: (a) Original teapot model; (b) Results where shared vertices have been welded on the intersection of their planes; (c) Vertices projected on the first plane; (d) Result of the original algorithm showing clearly visible gaps.

Figure 5.13 illustrates the change of the evaluation result when vertex welding is used. It clearly shows that the gap intensity is significantly reduced and visual tests confirm that the gaps are mostly closed. As already mentioned moving the shared vertices causes

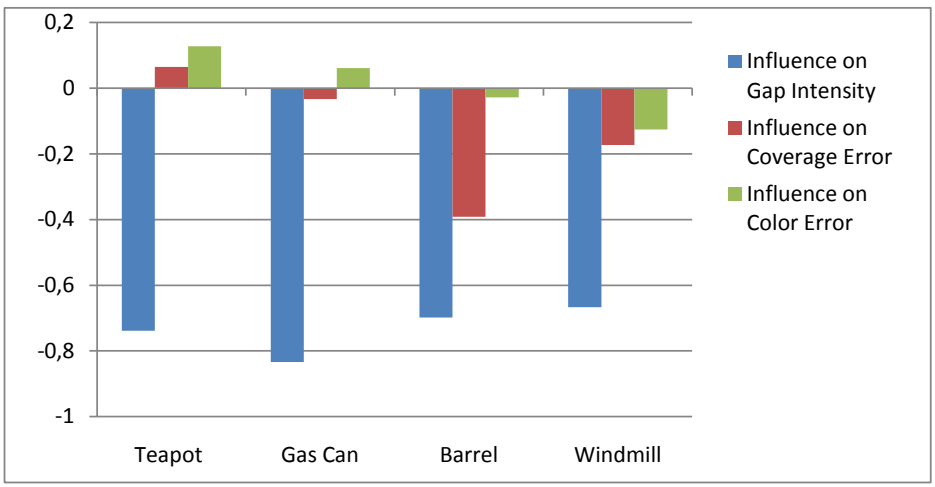



Figure 5.13: Illustration of the influence of using vertex welding from the evaluation results of Table 5.3.

changes of the original shape which can be partly read in the coverage error, but it is either very small or also reduced.

Moved vertices also cause texture distortions and wrong shading, which affects the accuracy of the simplification. However, the evaluation results show no consistent trend and no difference can be seen in a manual visual comparison.

Figure 5.14 shows the effect of vertex welding on a textured object. The welded simplification in (b) exhibits shading errors around the handle and texture distortion where vertices have been moved. However, when objects are viewed from middle and farther distances only an approximate color information is required and such artifacts cannot be noticed anymore.

(a) Original Model (b) Welded (c) None

Figure 5.14: Effect of vertex welding on textured objects: (a) Original model; (b) Welded vertices that cause texture distortions and shading errors; (c) Simplification result of the unmodified original algorithm showing disturbing gaps between the planes.

Overall, the benefits of vertex welding clearly outweigh the drawbacks and improve the overall quality of the simplification. It should be used whenever solid geometries are simplified.

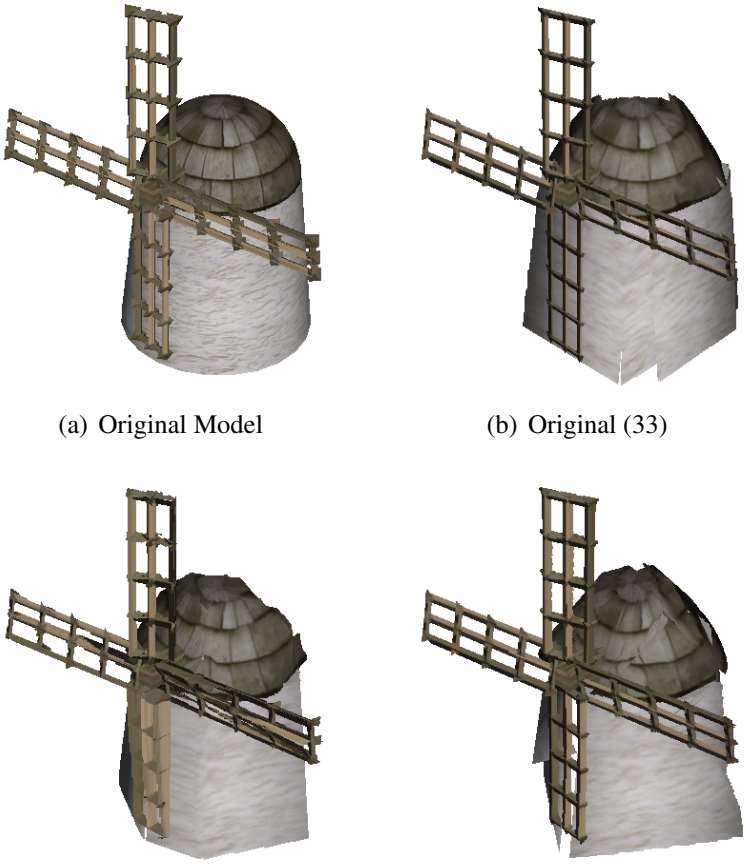
## 5.3 Algorithm Comparison

In the first part of this chapter all general aspects regarding billboard cloud simplification have been evaluated in detail. Now we continue by directly comparing the generation algorithms and evaluating the different results. As first scenario a windmill model has

been selected that is rather hard to simplify, because of its complex structure. Table 5.4 lists the used parameters. All have been chosen in best interest to get a best possible result. The number of planes for the k-means clustering algorithm have been set to approximate the result of the stochastic generation algorithm. Additionally, the distance falloff has been used. The images in Figure 5.15 and Table 5.3 show the different results.

| | |
|---|---|
| Original | $\epsilon = 10\%$, Penalty = 3, Raster = 18, RhoSample = $\frac{1.5}{\epsilon}$ |
| Stochastic | $\epsilon = 10\%$, Penalty = 3, N = 500 |
| K-Means | K = 75, MaxIter = 20 |
| General | Optimization, Vertex Welding, FaceSkipping = 2% |

Table 5.4: Parameters used for the algorithm comparison.



(a) Original Model (b) Original (33)

(c) Stochastic (76) (d) K-Means (75)

Figure 5.15: Algorithm comparison showing plane count in brackets: (a) Original model with 3,082 faces; (b) Result of original algorithm yielding expected plane count; (c) Generation result of stochastic algorithm requiring much more planes, but is also more accurate; (d) Result of our implemented version of the k-means clustering algorithm when trying to achieve a similar qualitative result than the stochastic algorithm.

| | Original | Stochastic | K-Means |
|---|---|---|---|
| Plane Count | 33 | 76 | 75 |
| Plane Search (sec) | 4.5 | 1.5 | 0.1 |
| Coverage Error (%) | 10.5 | 10.6 | 15 |
| Geometric Error (%) | 1.2 | 0.5 | 1.8 |
| Color Error (%) | 1.0 | 1.02 | 1.53 |

Table 5.5: Evaluation result comparing the output of the different generation algorithms using the windmill model.

The original algorithm works well and returns a simplification with a decent plane count according to the configured maximum tolerable error $\epsilon$. Thereby, the average geometric error is quite low and the coverage error acceptable.

## Stochastic Algorithm in Comparison

Our assumption in the discussion in Section 3.2 was that the stochastic algorithm should generate similar simplifications than the original algorithm, but the evaluation results show differently. Comprehending how the algorithm works the reason becomes obvious. For example the original algorithm managed to find the two optimal planes for the given $\epsilon$ to simplify the rotor sails. Since the stochastic algorithm randomly selects seed faces and evaluates random planes through them, it is very unlikely that when one of the numerous cross bar faces is selected a perpendicular random plane is generated. Therefore, the optimal planes are not found and it takes the algorithm many more planes to approximate this complex geometry. On the other hand the simplification is more accurate. However, the detailed comparison in the next section shows that this improvement is not in the same relation than the increased plane count.

The crucial parameter of the stochastic generation algorithm is the number of planes per seed face $N$. A value of $N = 500$ as selected for the comparison gives a very stable result in a reasonable time. By increasing the value even higher it is not possible to significantly reduce the plane count and get a similar result than the original algorithm. As illustrated in the graph of Figure 5.16 the parameter $N$ only has a noticeable effect on the plane count when a small value is selected. With values of $N < 200$ the results have much more deviation and the simplifications required significantly more planes. On the other side of the scale we actually evaluated up to $N = 5000$, but a further decrease of the plane count could not be measured.

If a simplification with as few planes than with the original algorithm is desired, $\epsilon$ has to

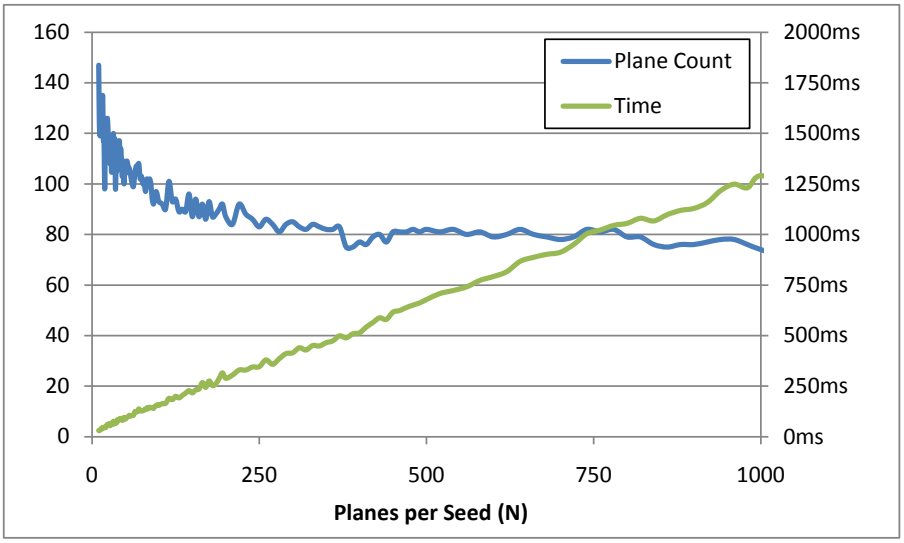



Figure 5.16: Variation of the N-parameter: Graph of the resulting plane count and the generation time for various N.

| | $\epsilon$ | Plane Count | Coverage Error | Geometric Error | Color Error |
|---|---|---|---|---|---|
| Original | 10 | 33 | 10.5 | 1.2 | 1.0 |
| Stochastic | 16 | 33 | 14.8 | 1.2 | 1.54 |

Table 5.6: Comparison between original algorithm and stochastic of results with the same plane count.

be adapted. However, in comparison the result then is mostly worse than that of the original algorithm. Tuning $\epsilon$ to result in the same plane count as the original in the comparison the stochastic algorithm generates a simplification with the following evaluation results. The real visual perceived difference is actually even higher than the values in the table indicate.

The first results in the evaluation above were only to demonstrate the behavior of the different generation algorithms. Now follows a more extensive comparison using a set of geometries with different properties. The generation parameters from Table 5.4 have been used again, except $\epsilon$=5% and vertex welding was used only for the teapot and the screwdriver model.

Table 5.7 shows the result of evaluations with different geometries. The results in miniature are compared in Figure 5.17. Similar to the first comparison the plane count using the stochastic algorithm is significantly higher in all tests. With the teapot geometry the simplification of the stochastic algorithm has a surprisingly high accuracy and shows less artifacts than expected from the higher plane count. With the other geometries the behavior of the stochastic algorithm is similar than in our first evaluation. The coverage error

| (Original / Stochastic) | Teapot (992 Faces) | | Screwdriver (6954 Faces) | | Eiffel (26216 Faces) | | Aspen (113016 Faces) | |
|---|---|---|---|---|---|---|---|---|
| Plane Search (sec) | 6.2 | 0.7 | 58 | 1 | 292 | 8 | 1514 | 51 |
| Plane Count | 23 | 33 | 10 | 12 | 24 | 31 | 122 | 159 |
| Coverage Error (%) | 19.7 | 10.2 | 14.5 | 16.2 | 42.5 | 41.6 | 18.4 | 17.7 |
| Geometric Error (%) | 0.9 | 0.5 | 0.5 | 0.4 | 1 | 0.9 | 1 | 1 |
| Color Error (%) | 0.55 | 0.36 | 0.64 | 0.7 | 1.19 | 1.4 | 2.38 | 2.97 |
| Gap Intensity | 0.62 | 0.23 | -0.01 | -0.03 | -0.05 | -0.2 | -0.1 | 0.12 |

Table 5.7: Algorithm comparison with different geometries.

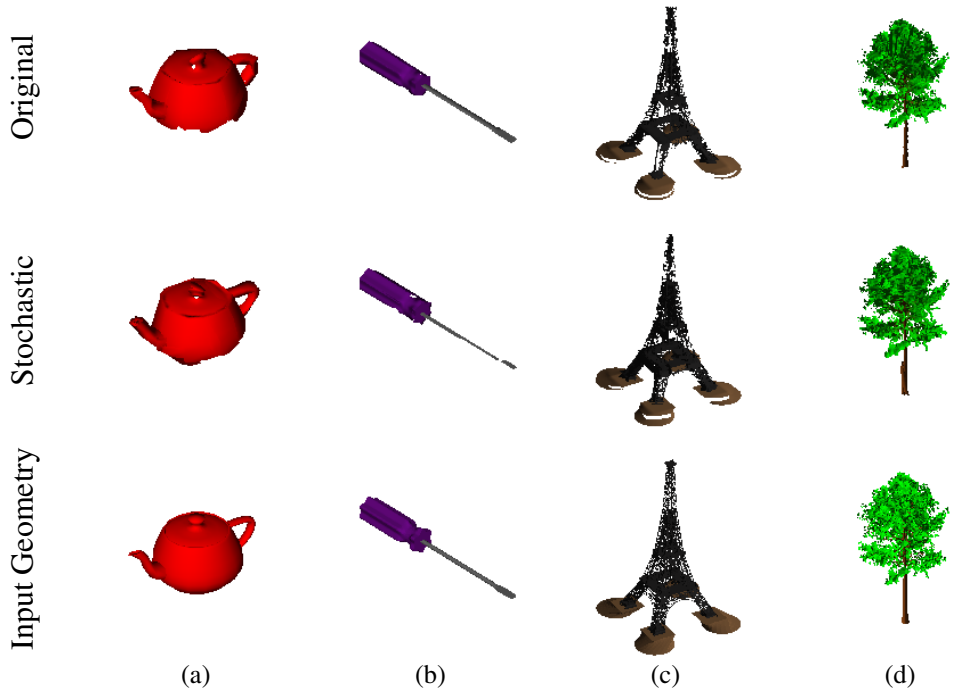



Figure 5.17: Comparison of the generation result between the original and the stochastic algorithm.

and gap intensity are slightly lower than with the original algorithm, but is not in relation to the increased plane count. The graph in Figure 5.18, where the evaluation results are illustrated in percentage change to the result of the original algorithm, gives a good overview of the different characteristics.

## The K-Means Clustering Algorithm

The generated simplification of the k-means clustering algorithms does not seem very optimal. It has to be admitted that our implementation is not very advanced. It turned out that it has to be well tuned to get practical results. We are convinced that if everything is implemented according to the paper, summarized in Section 2.3, much better simpli-

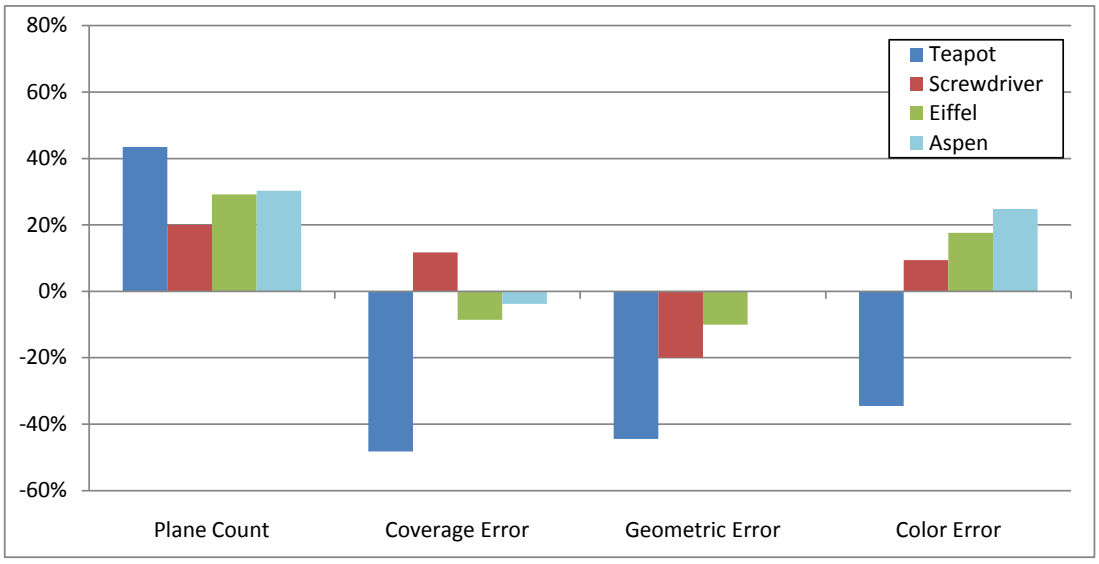



Figure 5.18: Illustrates the difference of the evaluation results in percentage to the result of the original algorithm for the tested models in Table 5.7.

fications could be generated. Because of limited time we decided to focus on the other two generation algorithms, therefore the evaluation results of our implemented k-means clustering algorithm should not be used as comparison.

Anyway, it is possible to deduce some properties of the algorithm. The selected plane count was clearly too high in relation to the final result. This resulted in lots of degenerated planes that did not manage to get suitable faces assigned anymore. The used distance-based error metric is too weak to build well grouped clusters of faces. Our implementation seems to be only capable of producing simplifications with a certain accuracy that can even get worse when setting too many planes.
Besides choosing the number of planes, nothing else can be configured, so it is hard to make an assumption on how accurate the result would be and how it would look like. In our opinion it is generally more practical to set an error tolerance as done with other generation algorithms.

We will not make any further evaluation of the k-means clustering algorithms for now, since our implementation is not fully developed yet.

## 5.4 Parallel Processing Benchmark

So far the listed times have only been used to compare the effects of certain parameters or extensions. These measurements have been taken without any of the parallel processing optimizations we described in Section 4.6. In this section we will make an extensive performance evaluation and show how parallel processing scales with different face counts and order of parallelization.

For all evaluations in this section the parameters from Table 5.4 have been used. The following table lists the used geometries and their face count. Additionally, the time the original algorithm requires to initialize the density grid and search the planes including the grid refinement and the time for the stochastic plane search using the single threaded implementations are listed.

| | Gas Can | Teapot | Screwdriver | Eiffel | Aspen |
|---|---|---|---|---|---|
| Face Count | 304 | 992 | 6954 | 26216 | 113016 |
| Original Search (sec) | 0.9 | 3.4 | 25 | 143 | 885 |
| Stochastic Search (sec) | 0.16 | 0.51 | 0.82 | 10 | 48 |

### Multithreading

Some parts of our generation algorithms have been extended to use multithreading. First we will take a closer look at the original algorithm, where the rasterization of the contribution in the density grid and the evaluation of the subbins have been parallelized. It has to be noted that the implementation of the rasterization has a small 5-10% overhead, as mentioned in the discussion in Section 4.6, which means that the single-threaded implementation could be optimized.

The bars in Figure 5.19 show the scaling factor of the total plane search time on our dual-core CPU using multithreading. It is not consistent, but approximately at about 1.8. Since only the major parts are parallelized the total time depends on specific mesh properties and how many planes it takes to simplify. The current implementation shows no sign of any limitation and we except a similarly good result on systems with more CPU cores.

The plane search of the stochastic generation algorithm is data independent when the seed planes are evaluated. This is an optimal condition for parallelization and each plane can be evaluated separately. The number of planes to evaluate are configured with the parameter $N$ and the complexity of a plane evaluation is linearly dependent on the number of faces

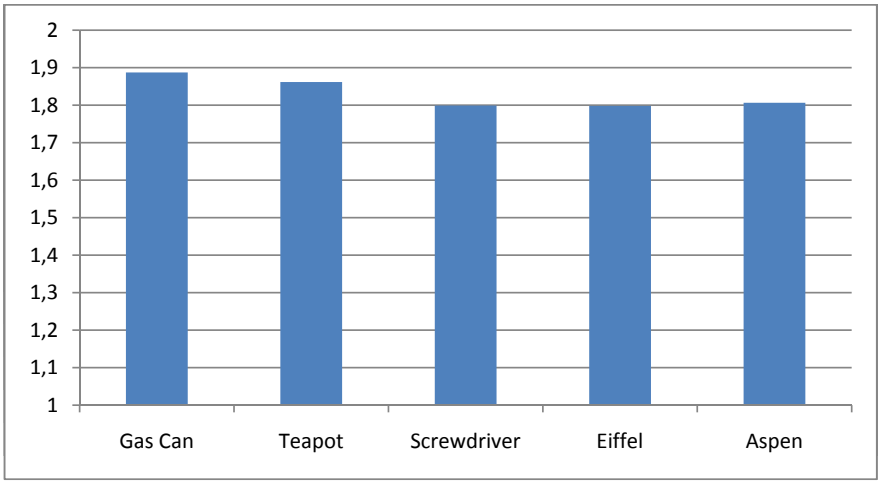



Figure 5.19: Scaling factor for the plane search of the original algorithm using our multi-threaded implementation on a dual-core CPU.

left in the working set. To evaluate the scaling factor, geometries with different face count have been used and the parameter $N$ was varied.

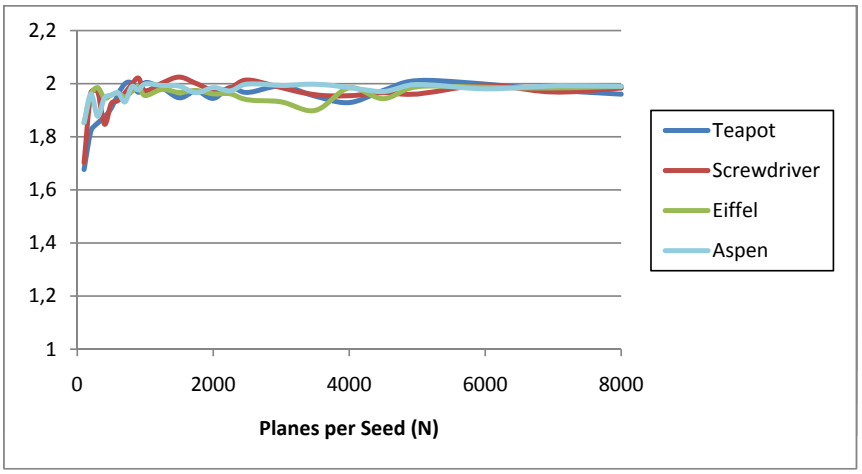



Figure 5.20: Scaling factor of the plane search time when multithreading is used for the stochastic plane search on the test system dual-core CPU.

Figure 5.20 shows the scaling factor when using multithreading on our dual-core CPU. With almost any seed plane count $N$ and all evaluated geometries the scaling factor is $> 1.9$. This is an optimal result when considering that background processes where also running.

### GPU-Processing

A GPU implementation for the stochastic plane search has also been implemented. This implementation has much more overhead, thus a high dependency on the parameters is expected. As in the evaluation before, the same geometries and range of $N$ are used.

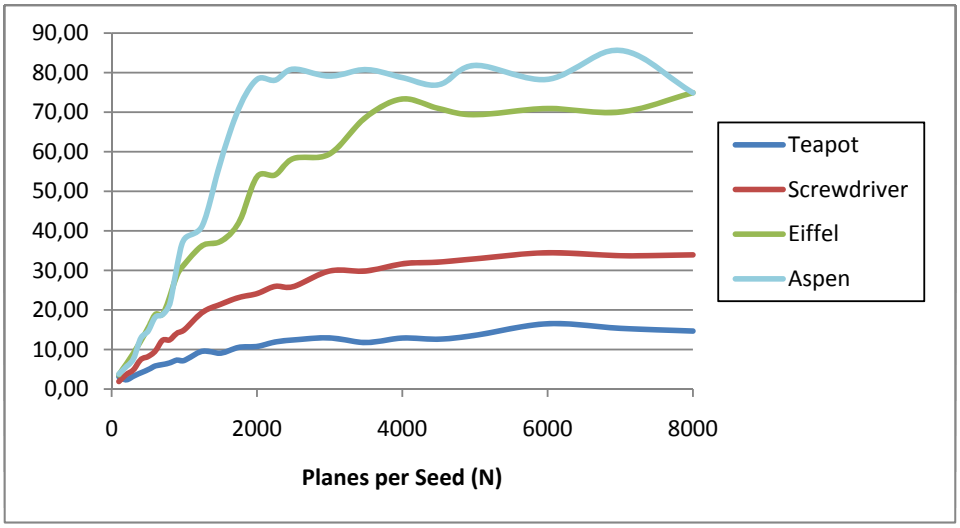



Figure 5.21: Scaling factor of our GPU implementation of the stochastic plane search.

The scaling factor when using the GPU implementation over the single-threaded CPU implementation is illustrated in Figure 5.21. It mostly shows an increase over the whole range of evaluated parameters, which means that the implementation does not manage to utilize the GPU very efficiently and has much overhead. Only when very large meshes are used the scaling seems to be limited at a factor of about 80. We also still have possibilities to optimize our implementation.

In the evaluations in Section 5.3 we have shown that a value of $N > 500$ barely brings any improvement. This gives more options for improving the algorithm. For example evaluating 100 planes each through 100 seed faces and then selecting the best of them. Thereby, we expect a result closer to the original algorithm.

## 5.5 Summary

In this section we will summarize the properties of the evaluated algorithms and make a final comment on the evaluations. The following table gives a brief overview of the behavior of the different generation algorithms regarding selected aspects:

| | Original | Stochastic | K-Means |
|---|---|---|---|
| Generation Speed | Slow | Medium | Fast |
| Implementation | Complex | Easy | Tricky |
| Quality | Good | Variable | n/a |
| Stability | $\epsilon$-dependent | High | n/a |

The original is in our opinion the most complex and difficult to implement, but the effort brings good simplifications and a well predictable behavior. For very small $\epsilon$ it can get unstable, but when higher quality simplifications are required billboard clouds are generally less suited.

The stochastic generation algorithm clearly has its strength with simplicity and good performance, which is important when extreme complex geometries need to be simplified.

We could not properly implement the k-means clustering algorithm in the scope of this thesis. Therefore, we cannot make any estimates on quality and stability. However, clustering should allow good simplifications, but it seems tricky to implement.

The overall billboard cloud generation time is mostly depended on the plane search. Therefore, we have not made any comments on times regarding the other tasks, such as texture generation, vertex welding or plane optimization.

Our automatic evaluation system was very useful to improve the implemented algorithms and to make extensive analyses. Although image based error metrics have their limits, the results were as expected and consistent with visual tests.

## Chapter 6

# Conclusion and Future Work

First, we will summarize the research on billboard cloud simplification of this thesis. Afterwards, we give an outlook on future work to improve the implemented algorithms.

## 6.1 Conclusion

In the first part of this thesis we gave an overview of the state of the art of billboard cloud simplification. It showed that even though the principle of replacing groups of faces, which lie approximately on the same plane by billboards, is very simple, the process of finding and generating these billboards is quite difficult. Various extension and improvement methods have been discussed. All together makes billboard cloud generation a complex task with numerous parameters. We tried to make our implementation as modular as possible and introduced the capability to choose most parameters automatically.

As main part of this thesis we presented several improvements to the original algorithm. The introduced distance falloff to calculate the contribution of faces for a plane significantly improves the alignment of planes. Additionally, a simply but effective heuristic to skip remaining face and assigning them to suitable planes can be used to remove suboptimal planes. The runtime and effectiveness of the plane search algorithm was improved by using a new parameterization of the plane space and adapting the algorithm to allow parallel processing.

The challenge of generating billboard cloud simplifications is to reduce artifacts such as cracks and shading errors. Our vertex welding approach turned out to preserve solid shapes very well and almost completely removes crack artifacts. The first important task to reduce shading errors is to chose the texture resolution precisely for the desired range of view distance. Then simple normal mapping is capable of simulating the non-planar geometry represented by the billboards sufficiently.
A good simplification in our opinion has a low plane count and guarantees that all faces

have been mapped respecting a certain error threshold. We have shown that the original algorithm results in nearly optimal simplifications in that respect, but it is very time consuming. With this characteristic well predictable results required for an automatic level of detail generation can be achieved. In theory the stochastic generation algorithm should behave similarly, but comparable results could only be achieved in a few cases and the algorithm generally tends to generate significantly more planes, but does not reduce the error in the same relation. On the other hand it finds the set of billboards in a very short time.

Overall, billboard clouds with its extreme geometry simplification have lots of potential applications. Implemented in a level-of-detail system this technique allows visualizing huge outdoor environments in high detail. Thereby, an automated process as our implementation is very practical.

## 6.2 Future Work

The generation of billboard clouds does not work right out of the box and requires lots of careful tuning and time intensive evaluations. In the scope of this thesis we focused on the generation of billboard clouds and even here could not evaluate all aspects. Some questions from the theory chapter still remain unanswered, such as the automatic choice of the optimal error threshold and texture resolution for a given distance. The implementation still requires some expert knowledge for proper simplifications, but will be improved soon.

Most urgently the texture usage efficiency needs to be improved. As summarized in the state of the art there are already a few approaches. A solution that works for at least the original and the stochastic algorithm is desired. We would start with a similar method as suggested by the authors of the original algorithm, but in order to get optimal results it should be also included into the formulation of *density*.

Furthermore, crack artifacts can still be noticed occasionally, since vertex welding has its limits. A possible solution that could be experimented with is one that builds additional geometry to fill the gaps. However, this would already diverge from the billboard cloud principle.

As we have shown, stochastic billboard clouds are not capable to compete with the results

of the original algorithm, but in our opinion this approach has potential with additional improvements. Its simplicity and fast generation are promising properties and make it interesting to experiment with. A better seed plane heuristic would definitely improve the quality. For example instead of evaluating $n$ planes going though one seed face, $n$ planes though $k$ faces could be used.

Our experiment with parallel processing showed great potential. Upcoming CPU generations, allowing eight or more parallel processing threads, have huge potential of reducing the generation time when the algorithms are implemented properly. This gives new possibilities for alternatives approaches and modifications of the algorithms.

Currently we have not made any rendering performance analyses. This concern is very important when billboard clouds should be used at a huge scale. Even though some evaluations are listed in a few papers, the development of graphics hardware makes them outdated and current bottlenecks of billboard cloud rendering remain to be evaluated.

# Bibliography


[ABC+04] C. Andujar, P. Brunet, A. Chica, I. Navazo, J. Rossignac, and A. Vinacua. Computing maximal tiles and application to impostor-based simplification. *Computer Graphics Forum*, 23:401–410, 2004.

[ABE+03] Carlos Andújar, Pere Brunet, Jordi Esteve, Eva Monclús, Isabel Navazo, and Alvar Vinacua. Robust face recovery for hybrid surface visualization. In *VMV*, pages 223–231, 2003.

[BCF+05] Stephan Behrendt, Carsten Colditz, Oliver Franzke, Johannes Kopf, and Oliver Deussen. Realistic real-time rendering of landscapes using billboard clouds. *First publ. in: Eurographics, 24 (2005), 3*, 2005.

[CDP95] Frédéric Cazals, George Drettakis, and Claude Puech. Filtering, clustering and hierarchy construction: a new solution for ray tracing very complex environments. In *Proceedings of the Eurographics conference*, September 1995. Maastricht, the Netherlands.

[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.

[DH72] Richard O. Duda and Peter E. Hart. Use of the Hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15, January 1972.

[Don05] William Donnelly. *GPU Gems 2*, chapter Per-Pixel Displacement Mapping With Distance Functions, pages 123—-136. Addison-Wesley, 2005.

[DSDD02] Xavier Décoret, François Sillion, Frédo Durand, and Julie Dorsey. Billboard clouds. Technical Report RR-4485, INRIA, Grenoble, June 2002.

[ED07] Elmar Eisemann and Xavier Décoret. On exact error bounds for view-dependent simplification. *Computer Graphics Forum*, 26(2):202–213, 2007.

[FB81] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, 1981.

[FMU05] A Fuhrmann, S Mantler, and E Umlauf. Extreme model simplification for forest rendering. *in Proceedings of EGWNP - Eurographics Workshop on Natural Phenomena*, 2005.

[GB65] David Hall Geoffrey Ball. Isodata, a novel method of data analysis and classification. Technical Report AD-699616, Stanford University, Stanford, CA., 1965.

[GPSSK07] Ismael García, Gustavo Patow, Mateu Sbert, and László Szirmay-Kalos. Multi-layered indirect texturing for tree rendering. In S. Mérillou D. Ebert, editor, *Eurographics Workshop on Natural Phenomena*, 2007.

[GSSK05] Ismael Garcia, Mateu Sbert, , and Laszlo Szirmay-Kalos. Tree rendering with billboard clouds. In *In Proceedings of Third Hungarian Conference on Computer Graphics and Geometry*, pages 9–15, 2005.

[GWH01] Michael Garland, Andrew Willmott, and Paul S. Heckbert. Hierarchical face clustering on polygonal surfaces. In *I3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 49–58, New York, NY, USA, 2001. ACM.

[HG97] Paul S. Heckbert and Michael Garland. Survey of polygonal surface simplification algorithms, 1997.

[HNW04] I.-T. Huang, K. L. Novins, and B. C. Wünsche. Improved billboard clouds for extreme model simplification. In *In Proceedings of Image and Vision Computing*, pages 255–260, 2004.

[HW79] J. A. Hartigan and M. A. Wong. A K-means clustering algorithm. *Applied Statistics*, 28:100–108, 1979.

[LEST06] J. Dylan Lacewell, Dave Edwards, Peter Shirley, and William B. Thompson. Stochastic billboard clouds for interactive foliage rendering. *journal of graphics tools*, 11(1):1–12, 2006.

[MJW07] Stephan Mantler, Stefan Jeschke, and Michael Wimmer. Displacement mapped billboard clouds, January 2007. human contact: technical-report@cg.tuwien.ac.at.

[MK04] Jan Meseth and Reinhard Klein. Memory efficient billboard clouds for btf textured objects. In B. Girod, M. Magnor, and H.-P. Seidel, editors, *Vision, Modeling, and Visualization 2004*, pages 167–174. Akademische Verlagsgesellschaft Aka GmbH, Berlin, November 2004.

[MMK03] Gero Müller, Jan Meseth, and Reinhard Klein. Compression and real-time rendering of measured btfs using local pca. 2003.

[RB92] Jarek Rossignac and Paul Borrel. Multi-resolution 3D approximations for rendering complex scenes. Technical report, Yorktown Heights, NY 10598, Feb. 1992. IBM Research Report RC 17697. Also appeared in *Modeling in Computer Graphics*, Springer, 1993.

[WGK05] Roland Wahl, Michael Guthe, and Reinhard Klein. Identifying planes in point-clouds for efficient hybrid rendering. In *The 13th Pacific Conference on Computer Graphics and Applications*, October 2005.