Animated Billboard Clouds for Foliage Simplification

June 2007

Aalborg University

AALBORG UNIVERSITET
DEPARTMENT OF COMPUTER SCIENCE, FREDERIK BAJERS VEJ 7E, DK 9220 AALBORG ØST

Title:

Animated Billboard Clouds for Foliage Simplification.

Project period:
DAT6, 2007-02-01 - 2007-06-11

Project group:
d629a

Group members:
Bennett-Therkildsen, Anders
Carlsen, Jeppe
Schmid, Jakob

Supervisors:
Claus B. Madsen and Olav Bangsø

Copies: 3

Report – page count: 158

Appendices – page count: 12

Total page count: 170

Synopsis:

A recognized weakness of existing billboard cloud algorithms is their lacking ability to simplify animated models. In this thesis we focus on the simplification of animated polygonal tree models, and our goal is to improve the billboard cloud technique with our own solutions for simplification of the animation of polygonal tree models; an extension we refer to as animated billboard clouds.
We present two solutions for simplification of animated tree models. The solutions use our own implementation of a billboard cloud algorithm developed by Lacewell et al., specifically aimed at the simplification of the foliage of static polygonal tree models. In order to evaluate the visual fidelity of an animated billboard cloud model we define a set of error metrics that enable us to objectively measure the quality of the simplification.
A prototype implementation of one of the solutions has been created as part of the project. This implementation, along with our implementation of the chosen billboard cloud algorithm, will be tested and evaluated.

Abstract

Billboard clouds, originally presented by Décoret et al. [7], is a well-established image-based technique used for extreme model simplification. It simplifies polygonal models of arbitrary complexity to a greatly reduced set of textured polygons that replace the original geometry. However, a recognized weakness of existing billboard cloud algorithms is their lacking ability to simplify animated models.

In this thesis we focus on the simplification of animated polygonal tree models. Our goal is to improve the billboard cloud technique with our own solutions for simplification of the animation of polygonal tree models; an extension we refer to as animated billboard clouds. Specifically, we combine the billboard cloud simplification technique with our own developed animation simplification techniques in order to create discrete animated billboard cloud LOD models of polygonal tree models rigged with animation skeletons.

We present two solutions for simplification of animated tree models. The solutions use our own implementation of a billboard cloud algorithm developed by Lacewell et al. [17], specifically aimed at the simplification of the foliage of static polygonal models. In order to evaluate the visual fidelity of an animated billboard cloud model we define a set of error metrics that enable us to objectively measure the quality of the simplification.

A prototype implementation of one of the solutions has been created as part of our project. This implementation, along with our implementation of the chosen billboard cloud algorithm, will be tested and evaluated.

For a more detailed summary we refer to Appendix D.

Contents

Introduction

For centuries, trees have been the study of artists and scientists alike. The remarkably simple underlying structure of trees coupled with the overwhelming complexity of their visual appearance captures the imagination of great thinkers such as Leonardo da Vinci.

Any forest is inhabited by a number of different tree species, each having their own unique structure. A single tree may consist of thousands upon thousands of branches and leaves. The seemingly complex structure of a fully grown tree has emerged from simple internal processes, common to all members of the same species. Thus, two trees of the same species will have the underlying system in common, yet the trees will not have the placement of a single branch or leaf in common. And although a single tree exhibits self-similarity, every single leaf or twig is unique. Furthermore, each tree has been influenced by its surroundings during its lifespan and has adapted its shape to the environment.

Computer science may be the latest scientific branch to concern itself with tree modelling. The object of realistic real-time computer graphics applications with outdoor scenes, most of them computer games, is to create a convincing illusion of nature for the viewer without compromising rendering performance. Trees are necessary for most of such scenes, and the addition of trees to a simple landscape without much detail may prove invaluable for the illusion. However, almost every single aspect of rendering trees has problems for a practical real-time application.

First of all, the tree models need to be created. If the models should be realistically modelled, they will consist of an incomparably large number of polygons. Besides manually performing the modelling, algorithms that emulate the internal processes of the growth of real trees can be employed. Examples of such algorithms are L-systems, which are formal grammars used for recursively generating self-similar tree models [23], and the Weber and Penn model [31] based on geometrical observations.

To accurately capture the light phenomena in a forest, advanced illumination models may be employed. However, such techniques are currently intractable for real-time graphics applications, so a local illumination model such as Phong [2] is usually opted for. Local illumination models yield no shadows by themselves, so a separate step involving a shadow rendering technique must be used to render shadows. A commonly used shadow technique for outdoor scenes is shadow maps [2].

The modelling and shading aspects aside, a defining visual property of a tree is its movement in reaction to wind. The movement may be described by

relatively simple physical models, yet the resulting movement of a tree seems complex and chaotic.

Needless to say, a method for simplifying a tree model for rendering whilst retaining visual fidelity to the original tree model is very valuable. Such techniques already exist, yet they often ignore the animation of the tree model, resulting in unrealistic static models in the final scene.

The focus of this thesis is simplification of animated tree models. We want to improve an already existing method for simplification of static models with our own solutions for the simplification of animated models. The method, which is called billboard clouds, simplifies textured polygonal models of arbitrary complexity to a greatly reduced set of textured polygons, that replace the original geometry. It yields impressive simplification results of the foliage part of static tree models, and our goal is to develop billboard cloud simplification solutions specifically for animated foliage.

Chapter 1

Tree Models in Real-Time Computer Graphics

This chapter will provide the setting for our work and place it into the context of real-time computer graphics. We will introduce an array of techniques used for polygonal model simplification and animation as a pre-analysis necessary for specifying our project goals in proper terms in the next chapter.

In Section 1.1 we define the type of models we will be using throughout the report. Different model simplification techniques are discussed in Section 1.2, both geometry- and image-based techniques, including billboard clouds, and the notion of an LOD scheme is explained. In Section 1.3 we discuss how the billboard cloud technique can be used to simplify the foliage part of polygonal tree models, and briefly review related work. In this section we also discuss the performance issues involved in using the billboard cloud technique. Model animation is introduced in Section 1.4, with focus on skeletal animation. In this regard a definition of a skeletal model is introduced. We conclude the chapter by listing a number of observations of the movement of real-life trees. This happens in Section 1.5.

1.1 Polygonal Tree Models

In this section we define the type of tree models we will be using throughout the report.

We define a *polygonal model* as one consisting of a set of triangles with fixed positions and orientations. In this report we limit our discussion of tree models to polygonal models, which we refer to as *polygonal tree models*.

Figure 1.1 and Figures 1.2(a) on the facing page and 1.2(b) on the next page are screenshots of trees in two state-of-the-art computer games, namely Call of Duty 3 [1] and Gears of War [20]. The trees in these two example games are polygonal tree models according to our definition. We can observe that a tree model consists of a single closed mesh for the trunk and large branches, while the foliage is built from few but large textured polygons, such that each polygon represents several leaves.

Figure 1.1: Screenshots of trees in Call of Duty 3 that conform to our definition of a polygonal tree model.

(a) Screenshot from Gears of War.

(b) Another screenshot from Gears of War.

Figure 1.2: Two screenshots of trees in Gears of War that confom to our definition of a polygonal tree model.

Several state-of-the-art games use SpeedTree [13] for modelling and rendering tree models, e.g. aforementioned Call of Duty 3 and The Elder Scrolls IV: Oblivion [26]. Figure 1.3 is a screenshot of trees in the latter. The trees in this example are not polygonal tree models, as the textured polygons used for chunks of leaves are rotated to face the observer at all times.

Figure 1.3: Screenshot of trees in Oblivion. These trees do not conform to our definition of a polygonal tree model.

A number of closely positioned leaves and small branches can be modelled by a single textured polygon, which is normally applied to reduce rendering complexity. This can be observed in all the above screenshots.

The fact that our definition of polygonal tree models excludes the modelling techniques used to create the trees in some popular games is a limitation. However, the definition is reasonable, first of all because polygonal tree models are in fact used in different applications (as exemplified). Second, since we focus on the simplification of animated tree models, we need to know about the actual positions of the polygons in the model in order to be able to perform simplification. If view-aligned billboards (or the like) are used, this knowledge cannot be aquired.

In the remaining chapters of this report we will assume that the trunk and large branches of a tree are modelled as a single closed mesh, while the individual smaller branches and leaves are modelled as isolated polygons. The mesh representing the trunk and large branches will be referred to as the *trunk* of the tree model, and the smaller branches and leaves will be referred to as the *foliage*.

1.2 Model Simplification

Starting from a high-polygonal model, several methods exist for retrieving suitable simplified models. In pursuit of our project goals, that we have hinted at in the report introduction, we will investigate techniques used for model simplification in this section. In the following we discuss geometry and image-based simplification techniques, as well as the notion of an LOD scheme.

1.2.1 Levels of Detail

Using *levels of detail* (LODs) for displaying an object is introduced in order to obtain faster rendering without compromising visual quality. We define an LOD scheme to be a technique that reduces the rendering complexity of an object at run-time from certain simplification criteria. The observation is that, in perspective projections, when a model is positioned far from the observer, its smaller details are no longer visible at a specific screen resolution. Consequently, a carefully simplified model can replace the original one, and equal or similar visual quality is obtained with less rendering complexity. Other metrics than the distance to the observer can be used to determine when a simplified model should be used, e.g. whether the transformed model resides in the center region of the screen, or some estimate of available rendering resources.

The rendering of a detailed polygonal model positioned far from the observer can lead to aliasing artifacts, as small polygons transformed to screen space might cover less than a single pixel [7]. LODs can assist in reducing this problem.

Discrete LODs

We define a *discrete LOD scheme* as a series of models for displaying an object, each less detailed than the previous, and rules defining when to render which model. The models in such a scheme are all constructed before the actual rendering is started. In practice, even though a change from one LOD to another should not lower visual quality, a slight noticeable change is almost impossible to avoid. To conceal this, a blending operation between models of different LODs is often applied.

Blending from one LOD to another can be performed by drawing the new LOD model on top of the old model using alpha transparency, and gradually make the new model more visible, while making the old model less visible, as the observer approaches the object, and vice versa if the observer retreats from the object. Using such a blending scheme adds to the rendering complexity, since two objects are effectively rendered instead of one when blending. Furthermore, the use of semi-transparency requires sorted rendering of the semi-transparent objects, which is expensive if the number of such objects is large [32].

To avoid using semi-transparency, a dissolve effect can be used [32], which is used for blending between different LODs in SpeedTree. The idea is to modulate the alpha channel of an image with a noise texture, and only render pixels

with alpha values larger than an alpha test value. By sliding the alpha test value from 0 to 1, the object will appear to be dissolving.

Neither of the blending methods are perfect and should be used with care, as blending tends to result in graphical artifacts.

Continuous LODs

Discrete LODs have several drawbacks, e.g. the additional memory consumed by storing the simplified versions of a model, and the need for blending between different LODs. A *continuous LOD scheme* can be opted for instead. We define a continuous LOD scheme for an object to be a data-structure from which a model representing the object at a desired LOD can be constructed.

An example of a continuous LOD scheme is the *N-patches scheme*, details on which can be found in [2]. Given a polygonal model constructed as a mesh of triangles with normals specified for each vertex, the N-patches scheme tessellates the surface that each triangle approximates to an arbitrary number of triangles. In effect, this improves the visual quality of the model, as the silhouette becomes smoother and, if vertex colour values are interpolated across each triangle surface, shading will be more precise. The N-patches scheme has been implemented in hardware, and can be used in real-time applications.

1.2.2 Geometry-based Simplification

Geometry-based simplification, also referred to as mesh simplification, is the process of reducing the number of polygons in a polygonal model (a mesh) while still retaining the overall shape and structure of the model. Many different automated techniques to perform geometry-based simplification exist, and a few of these are briefly mentioned here. The descriptions are based on [28].

Face Merging : Search the polygonal model for coplanar or almost coplanar adjacent faces. Replace such with a larger single face and thereby reduce the polygonal complexity of the model.

Vertex Clustering : Group the vertices of the model into clusters. For each cluster, find a single vertex that is representative for the whole group.

Edge Collapsing : Remove an edge by collapsing its vertices into one. In effect, this removes the two polygons containing the edge, thus reducing polygonal complexity.

When removing polygons from a textured model, problems arise with the texturing. The vertices of a textured model are mapped to texture coordinates, and when such vertices are removed, the texturing of the model might be distorted.

Different geometry-based methods exist for simplifying animated closed meshes. Decoro and Rusinkiewicz present a method for automatic simplification of articulated meshes in [8].

1.2.3 Image-based Simplification

The term *image-based simplification* refers to techniques that simplify a polygonal model by replacing geometry with images obtained by rendering the geometry. An image in this context is usually a 2D texture. This will normally result in rendering speed becoming proportional to the amount of pixels rendered, and not the number of polygons transformed. Image-based techniques do not suffer from texture distortion.

Billboards

A polygonal model can be simplified by a *billboard*, which is an image-based simplification. A billboard is simply a rectangle with a texture and a transparency map. We distinguish between *static* and *view-aligned* billboards. A static billboard has a fixed orientation in world space, whereas a view-aligned billboard is always oriented towards the view point. View-aligning the billboard helps hiding its two-dimensional nature, if the camera is allowed to rotate around the object it simplifies, but will make most otherwise static objects appear as if they are rotating.

A billboard representation of a polygonal model is simply retrieved by rendering the model and storing the result in a texture. As such, billboard simplification can be considered a pre-rendering of the model, i.e. we do the rendering in advance and store the result in a texture to be used at runtime. Naturally the billboard is only visually faithful to the original polygonal model, when the viewing direction is similar to the direction used when rendering the object onto the billboard.

Using a single billboard to simplify a model is a very simple approach, and the two dimensional nature of the billboard yields no parallax effect within the object. Nonetheless, the method is still used in state-of-the-art LOD schemes (e.g. in SpeedTree), typically as the lowest level of detail to be displayed when the object is very far from the observer.

Other image-based simplification techniques exist, e.g. *impostors*, which are very similar to billboards, but attempt to remedy the problem that a billboard only provides a faithful representation of the object from a certain viewing direction. An impostor is a view-aligned billboard whose texture is dynamically updated during run-time depending on view direction.

Billboard Clouds

A *billboard cloud* is a set of static billboards used to simplify a polygonal model, as opposed to using just a single static billboard. The concept and an algorithm for automated model simplification is presented in 2002 by Décoret et al. in [7].

The observation is that if two polygons lie in the same plane, they can be replaced by a larger polygon (a billboard), onto which they are both rendered. This should not be confused with face merging, as the polygons do not even have to be adjacent. If a model contains lots of coplanar polygons, the result

will be a visually identical model consisting of fewer polygons. Unfortunately, polygonal most models, including polygonal tree models, do not contain a significant amount of coplanar polygons, if any.

The concept behind billboard clouds is to move sets of almost coplanar polygons such that they become coplanar and can be simplified by a billboard. In general, the shorter distance the polygons are moved to perform the simplification, the better the billboard cloud resembles the original geometry. Moving polygons from their original positions introduces visual artifacts such as cracks between adjacent surfaces. As opposed to using a single billboard to simplify a model, a billboard cloud can provide a faithful representation of the model from all angles and preserve parallax. More billboards generally yield higher fidelity to the polygonal model. Figure 1.4 illustrates a car model consisting of 12,000 polygons and the billboard cloud simplification consisting of 46 billboards. Artifacts can be spotted, especially around the tires, but the simplication is a good approximation of the original model, considering the reduction in polygonal complexity.

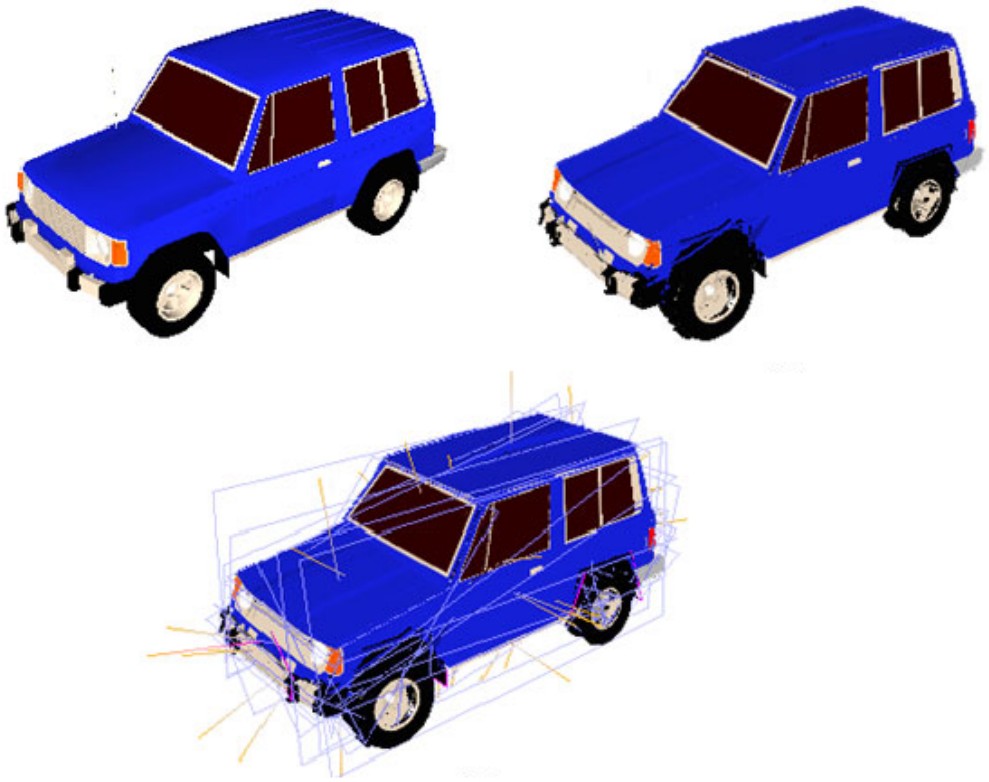


Figure 1.4: Billboard cloud simplification of a car model. Left: the original model consisting of 12,000 polygons. Right: A billboard cloud simplification of the car consisting of 46 billboards. Bottom: The contours of the billboards simplifying the original model. The illustration is from [28].

The billboard cloud algorithm by Décoret attempts to find and choose as few

billboards as possible, such that all polygons can be simplified without moving them far away from their original positions. As one could expect, finding such a set of billboards is a non-trivial task. In Section 3.2 we will discuss a selection of billboard cloud algorithms, including the original algorithm by Décoret et al.

1.2.4 Simplification of Polygonal Tree Models

Assume a polygonal tree model with a closed mesh representing the trunk and a set of unconnected triangles representing the foliage. When considering simplification of such a model, it can be beneficial to consider the trunk as one entity and the foliage as another. The mesh representing the trunk is very suitable for geometry-based simplification, whereas the foliage is not. This is argued by Remolar et al. in [24], where a new geometry-based simplification technique specific for foliage is introduced. The problem with traditional geometry-based simplification techniques is that they tend to simplify foliage by removing leaves from the tree, resulting in a tree model with sparser foliage. The basic idea of the technique proposed for foliage is to replace two closely positioned leaves with a single larger polygon representing both, such that the total leaf area is maintained. It is questionable how this technique handles the case where several leaves are modelled by a single textured polygon.

Image-based simplification handles foliage well and has been used manually by artists to reduce the number of polygons when modelling foliage. This can be observed in the example games in Section 1.1 on page 10. Our research in billboard cloud simplification has convinced us that automated simplification using billboard clouds is a very suitable approach for creating the LOD versions of the foliage of a polygonal tree model, as described in [28] and [10]. This is further discussed in Section 1.3.

As proposed by Umlauf in [28], combining a geometry-based simplification for the trunk and an image-based simplification for the foliage seems to be a good solution.

1.3 Billboard Cloud Simplification of Tree Models

Using billboard cloud simplification, complex foliage can be simplified to a very small number of billboards, while still retaining parallax. The typical problem of billboard clouds introducing lack of visual connectivity is not an issue for foliage, as foliage appears unconnected in the first place.

In the sections to follow we will discuss how the billboard cloud technique previously has been applied to simplify foliage of trees as well as the performance gain involved in using this technique. First we discuss LOD schemes for polygonal tree models using billboard clouds.

1.3.1 Level of Detail Scheme Using Billboard Clouds

Recall that we differentiate between discrete and continuous LOD schemes. We suspect the billboard cloud concept unsuitable for continuous LOD schemes as

construction of billboard clouds is a complex process suitable only for preprocessing.

A discrete LOD scheme has been applied for tree models in SpeedTree, which uses the dissolve effect for blending between different LOD versions. Lacewell et al. propose an alternative blending scheme specifically for billboard clouds in [17], along with an algorithm for recursively producing billboard clouds of different levels of detail. The idea is to simplify the original model to a billboard cloud to produce one level of detail, simplify this billboard cloud to produce the next level of detail, and so forth. When the LODs of a model have been constructed using this algorithm, a linear interpolation of vertex positions can be applied to blend from one LOD to another, such that the vertices of a triangle smoothly move into their positions on the billboard simplifying the triangle. This avoids the visual artifacts related to usual blending techniques, but could potentially introduce new visual artifacts. The blending technique can only be applied when the billboard clouds in the LOD scheme have been constructed recursively. An immediate drawback of recursively producing a billboard LOD from another billboard LOD is that errors accumulate through the LODs, as a triangle might get farther away from its original position in each LOD.

In a discrete LOD scheme simplified models are stored in addition to the original model, so the additional memory consumption can be a problem. Instead of storing separate LOD versions of a tree Weber and Penn [31] propose to store one geometric description of each tree and then re-interpret the tree geometry when increasing range to it. In this manner stem meshes could be interpreted and rendered as lines and leaves as points. Memory consumption is reduced, because only a single representation of the tree model is stored. We will discuss the memory performance involved in using the billboard cloud technique in a discrete LOD scheme in Section 1.3.3.

1.3.2 Related Work

Application of billboard cloud simplification to foliage is demonstrated by Lacewell et al. [17], Fuhrmann et al. [10], and the related master's thesis by Umlauf [28]. Figure 1.5 on the next page illustrates two examples of polygonal tree models simplified to a small number of large billboards. Note that both trunk and foliage are simplified in these examples.

How well a billboard cloud can simplify a foliage model depends on the amount of nearly coplanar leaves in the foliage, which in turn depends on the tree species. Consequently, some tree species might not be very suitable for billboard cloud simplification. Despite of this, tree models of the Aspen, Chestnut, Oak, Palm, and Spruce species have all been simplified with success in [28].

The real-time results in [28] and [10] using billboard cloud simplification of tree models are very promising. Scenes with more than 100,000 trees are rendered with impressive visual quality. A polygonal tree model and a billboard cloud algorithm is used to automatically generate the different LOD versions of the tree, the complexity of which ranges from 6 to 22 billboards. These are used

Figure 1.5: (Left) An Aspen tree model simplified to 12 billboards. (Right) A Chestnut tree model simplified to 16 billboards. The original polygonal models both consist of more than 100,000 polygons. The illustration is from [28].

in a discrete LOD scheme, and alpha blending is performed when changing LOD model [28].

Figure 1.6 and Figure 1.7 on the next page are screenshots from Umlauf's implementation. They illustrate the capability of rendering large scale forests. The frame-rate is approximately 10 fps and the scene contains three different tree models.

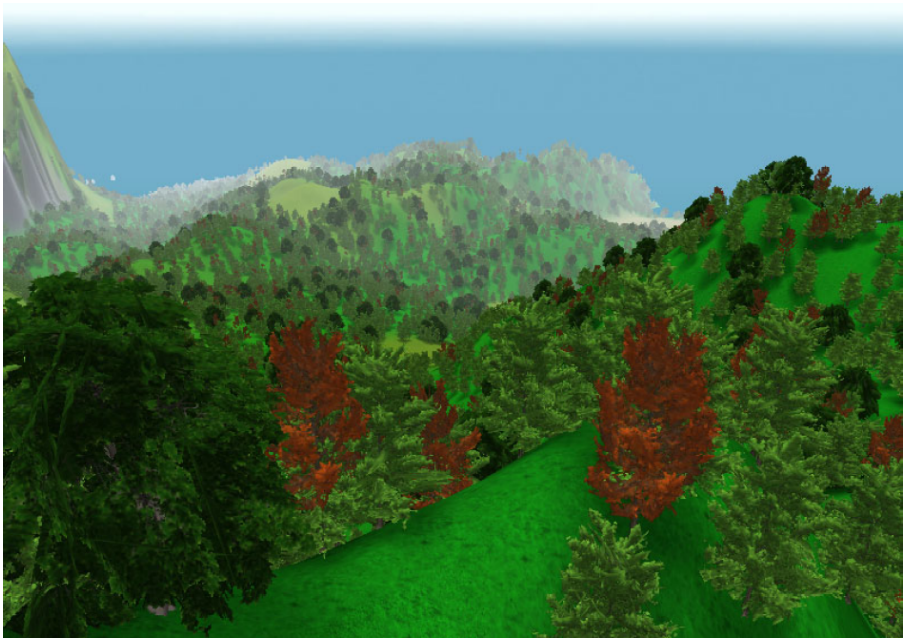


Figure 1.6: A screenshot of a real-time rendered forest from [28].

The trees in Umlauf's implementation do resemble real trees to some degree, and a large amount of trees is being rendered to represent a dense forest realistically. Furthermore, the trees retain visual fidelity when the camera rotates around them. However, the scene is limited since only three different tree

Figure 1.7: Another screenshot of a real-time rendered forest from [28].

models are used to render the entire forest, one model for each tree species. This reduces the amount of memory required for billboard textures and increases fps performance, since texture switching is reduced. Texture switches are expensive, even if all the textures of the scene reside in memory, because performance of texture caching is reduced.

As can be observed in the screenshots, the trees lose visual quality when observed closely, as the unrealistic amount of coplanar geometry becomes apparent, and likewise does the limited texture resolution. The reason is that the most detailed version of a tree is a billboard cloud containing only 22 billboards, not the original polygonal model which the billboards simplify. Among other limitations (e.g. dynamic shadows cast from trees, and of realistic lighting of trees), is the lack of animation.

Umlauf's focus in [28] is to "render huge forests at interactive rates", hence the aforementioned limitations. As stated in our report introduction, we, however, focus on the simplification of *animated* polygonal tree models. As mentioned, our research in the billboard cloud technique has shown us that there is a clear potential in this technique for simplifying polygonal tree models, especially the foliage part.

The fact that Umlauf's implementation of the billboard cloud technique does not cope with animation exists for a reason. As it is, existing billboard cloud algorithms cannot simplify animated models. It is thus very interesting to explore the possibilities in strengthening the billboard cloud technique to include animated polygonal models, particularly animated polygonal tree models. Model animation is the topic of Section 1.4.

We have conducted a series of experiments in order to analyze the per-

formance of billboard cloud simplification. In Section 1.3.3 we summarize these results, which we have categorized in *fps performance* and *memory performance*.

1.3.3 Performance Analysis of Billboard Cloud Simplification

Performance of a rendering scheme has two basic aspects, *frames per second (fps) performance*, i.e. rendering speed, and *memory performance*. In this section the motivation regarding performance for using billboard clouds is discussed, and rendering speed as well as memory usage is analyzed. Finally, we evaluate the use of billboard clouds to render a large amount of different tree models. First, an overview of the rendering pipeline is provided.

Rendering Pipeline Bottleneck

State-of-the-art real-time rendering is based on a pipelined architecture that is divided into three conceptual stages: application, geometry, and rasterizer [2].

Rendering billboard clouds basically only involves rendering textured polygons, which can be performed exclusively by the geometry and rasterizer stages. The application stage is therefore ignored in the following discussion.

The geometry stage handles transforming vertices and normals of a polygonal model from its local model vector space to screen coordinates using various intermediate transformations. Any models that are outside the visible view volume are culled, i.e. not considered further during the rendering. Models whose appearances are to be affected by light sources are shaded. A shading technique often used in real-time computer graphics is Gouraud shading, in which a colour is calculated using some local illumination model for each vertex of a polygon and subsequently, during the rasterization stage, interpolated across the polygon surface. The time spent in the geometry stage for a given model is roughly linearly dependent on the number of polygons in the model.

The rasterization stage line-wise interpolates vertex screen coordinates to approximate polygons by a set of pixels. For each pixel, a colour is calculated by interpolating vertex colours. Each vertex may have an associated texture coordinate pair, which makes it possible for the rasterizer to look up a pixel in a texture. A depth buffer can be employed to perform hidden surface removal, in which case the depth (z) value of a pixel is calculated by interpolating the depth values of the vertices of the polygon being rasterized. The calculated colour and depth values are only stored in the colour and depth buffers, if the depth value is less than the depth value already present in the associated entry in the depth buffer. The time spent on the rasterization stage for a given polygon is roughly linearly dependent on the number of pixels rendered on the screen to draw the polygon.

The rendering architecture is a pipeline, and thus each polygon must undergo the three stages in turn. At a given instant in a rendering application, one of these stages is the slowest, which in the simplest case means that the other stages must wait for it to complete. The slowest stage at a given time is called the *bottleneck* of the pipeline.

Figure 1.8: On the left, a polygonal Spruce tree model with 20,547 foliage triangles and approximately 2000 trunk and branch triangles. On the right, a billboard cloud model of the tree with 13 billboards [10].

Example Data

To evaluate the memory requirements of the billboard cloud technique we have evaluated a specific example. We have used a polygonal Spruce tree model and a billboard cloud simplification, as shown in Figure 1.8.

The models are arranged in a distance-based discrete LOD scheme with three levels:

1. Polygonal tree model consisting of 20,547 foliage triangles and approximately 2000 trunk and branch triangles.

2. Billboard cloud representation of the tree with 13 billboards. The billboard cloud representation of the tree model is used by the LOD scheme, when the tree takes up $\frac{1}{32}$ of the screen. Each billboard has an associated texture with a resolution reasonable on this distance.

3. Single billboard representation of the tree. The single billboard representation is used by the LOD scheme at the distance where the tree covers 64 $\times$ 64 px on the screen.

Fps Performance

Using billboard clouds to simplify complex polygonal models is motivated by the desire to gain performance by replacing many polygons with fewer larger polygons. To understand why performance can be gained by doing so we investigate fps performance of billboard clouds and the impact on the performance when using textures on the billboards.

Consider the Spruce polygonal tree model consisting of more than 20,000 polygons shown in Figure 1.8. At some depth, d, not far from the observer, the fps performance of rendering this model will be bounded by the number of polygons transformed and not the number of pixels rendered, i.e. the bottleneck is located in the geometry stage. For all depths larger than d the fps performance

of rendering the model will remain constant, as a constant number of polygons has to be transformed. Consequently, the model is not faster to render, when it covers a smaller area on the screen.

Consider now the billboard cloud simplifying the Spruce tree consisting of 13 billboards (hence 26 polygons). The number of polygons in the billboard cloud is very small, so it is reasonable to assume that the fps performance of rendering the billboard cloud at depth d is bounded by the number of pixels rendered, i.e. the bottleneck is now located in the rasterizer stage.

At the depth d, the fps performance of the billboard cloud might be better or worse than the fps performance of rendering the original model, as the billboard cloud is not guaranteed to provide faster rendering close to the observer. But the billboard cloud does have a nice performance property that the original model is missing, namely the fact that performance is gained, if the model is moved farther away from the observer. The reason is that the fps performance of the billboard cloud is bounded by the amount of pixels rendered, and the fact that a model farther away covers less pixels on the screen. If the billboard cloud covers very few pixels on the screen, the performance of rendering it will be bounded by the number of polygons it contains, as close to no pixels are rendered.

The rendering of the billboard cloud is thus trivial (the performance of transforming a very little amount of polygons), if the billboard cloud is moved far away from the observer. Consequently, there must be some depth at which the billboard cloud has better performance than the original model, and the improvement will be more significant for all depths larger than this one.

The conclusion is that replacing a polygonal model with a billboard cloud simplification does gain fps performance for all depths larger than the depth implying an equal amount of rendering time for the two. For models with a large amount of polygons (10,000+), we guesstimate the distance at which a billboard cloud with less than 50 billboards has better performance to be very close to the observer. Using a billboard cloud simplification in a distance-based discrete LOD scheme for this reason makes lots of sense, and we furthermore deem that very few billboard clouds are needed in the LOD scheme, as a billboard cloud automatically becomes faster to render as distance to the observer increases.

An important performance note related to the rendering of billboard clouds, is the rendering state switches introduced due to each billboard having a unique texture associated. As such, each billboard imply a state switch in the rendering pipeline, and consequently a poor fps performance. For this reason the textures are packed into a single large texture, which is kept in memory during the rendering of all billboards of a billboard cloud [28].

Memory Performance

When the rendering pipeline is implemented in hardware, all vertex and texture data must reside in the memory on the graphics card. As the memory on the graphics card is expensive and therefore limited, memory performance of a simplification scheme becomes a concern.

We have investigated the memory performance of the input model and LOD scheme discussed in Section 1.3.3 on page 22. To summarize the results, the polygonal tree model uses approximately 1.5 MB, and the billboard cloud uses approximately 2.0 MB. It is not surprising that an image-based representation of a polygonal model may take up more memory than the original model, as a single polygon easily can represent a large number of pixels. The single billboard LOD uses only approximately 16 kB, which is insignificant compared to the other two LODs.

Assuming that our example tree and LOD scheme is used in a forest scene, and that instances of the example tree are scattered in the scene, then all three LOD versions of the tree may be used simultaneously during the rendering. In this case the total memory requirement of rendering these identical trees is approximately 3.5 MB.

Normal maps may be used for dynamic relighting of billboards. If normal maps are used, additional memory is required. Our analysis has shown that adding normal maps to the tree model of the example nearly doubles the memory requirements. If lack of memory is a more serious concern than visual fidelity, the usage of normal maps should be omitted.

Forest Variation and Memory Performance

If a large number of different tree models are used to create a convincing illusion of a forest, the billboard cloud simplification scheme creates a different set of billboards for each tree model.

Billboard clouds simplifying different tree models normally do not share textures, and as memory is limited, there is a trade-off between the amount of trees that constitute the forest and the variation in the billboard cloud models. Alternatively, simpler tree models and billboard clouds with fewer billboards could reduce memory usage, and hence improve variation, at the cost of compromising the polygonal complexity.

1.4 Model Animation

As stated in Section 1.3 existing billboard cloud algorithms cannot simplify animated models. Since the billboard cloud technique seems very promising to simplify the foliage part of polygonal tree models, we will investigate into the possibilities of improving the technique to include simplification of animated models.

In the following sections we discuss techniques that are used for animating models, including polygonal tree models. We will comment on different types of animation and explain how a model can be animated with a skeleton.

1.4.1 Animation Types

Different types of animation exist, all of which can be divided into two categories, namely *key frame animation* and *procedural animation*. A key frame

animation is pre-defined, while a procedural animation is automatically generated in real-time.

Key Frame Animation

In key frame animation an animation is stored as a number of key frame poses of a mesh. Interpolating between these poses can yield vertex positions of the mesh at any given time. Since linear interpolation can result in non-continuous motion, splines can be used instead.

A key frame animation may be looped in order to create an animation cycle. However, to make the animation appear naturally continuous when looping, the first and the last key frame poses should be very much alike, such that the last key frame can be succeeded by the first key frame without introducing an obvious transition.

Procedural Animation

Using procedural animation an animation is generated in real-time from an algorithm and a set of rules dictating the motion, e.g. the laws of physics.

An example of a way to apply procedural animation to polygonal tree models is to specify wind as vector fields that are updated with time. The wind force applied to the leaves is propagated from the leaves toward the root. Depending on the wind force, the area of the leaves, and the rigidity of branches, the polygonal tree model will stand oscillating with different frequencies and amplitudes.

Such calculations can be expensive. Therefore, a skeleton can be constructed for the animation of the tree model (this is further discussed in Section 1.4.2), and simplification of this skeleton can thus reduce the cost in performance of performing these simulation calculations. This has been done by Beaudoin and Keyser in [5]. In this article the focus is simplification of simulation levels of detail, or SLODs, which are articulated structures (skeletons) that dictates how plants and trees are set into motion, when wind is applied in real-time. The higher detailed SLOD, the more precise the simulation of motion (i.e. the animation).

1.4.2 Skeletal Animation

There are basically two different techniques of animation: *vertex animation* and *skeletal animation*. In vertex animation, if pre-defined animation such as key frame animation is used, the vertex positions of the object are stored in each frame, and interpolating between them yields the animation. In *skeletal animation* a bone tree structure, referred to as a *skeletal model*, is stored for a mesh. If key frame animation is used, an animation of the mesh is then stored as a number of key frame skeleton poses. If procedural animation is used, an animation could be a simulation as described in Section 1.4.1. Regardless of animation type, the mesh is thus animated as dictated by its skeleton.

Each vertex of the mesh is associated with one or several bones in the skeletal model, with a weight that represents how much influence a bone has on a vertex.

Skeletal animation is an efficient animation technique. Instead of storing a mesh in different poses, only a number of skeleton poses are stored, as mentioned. Furthermore, it separates the information about animation from the mesh [3].

Polygonal tree models are well suited to be animated using skeletal animation, because these models are articulated meshes, structurally fit for this kind of animation. In the following we define the notion of what a skeletal model is in our report. It should be noted that our definition of a skeletal model is not aimed at an efficient implementation of skeletal animation. Its purpose is to provide a well-defined model that we can refer to in our report. Our model is partly based on [2] and [11].

Skeletal Model

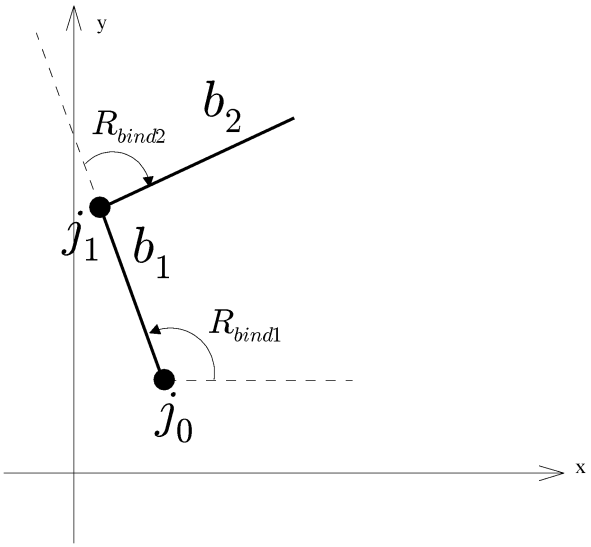
Intuitively, a skeletal model is a set of vertices attached to an animation skeleton. The process of associating vertices to the animation skeleton is called *rigging*.

Formally, a skeletal model is a rooted tree, i.e. a connected acyclic graph with a distinct root node. Each edge has geometric information associated with it, defining its length as well as orientation during the animation. Nodes represent rotation points. Furthermore, a set of vertices that define the visual shape of the model is associated with each edge. The nodes of a skeletal model are called *joints*, and edges are called *bones*. A bone has thus two joints. The joint closest to the root is called the *start joint* and the joint farthest from the root is called the *end joint*.

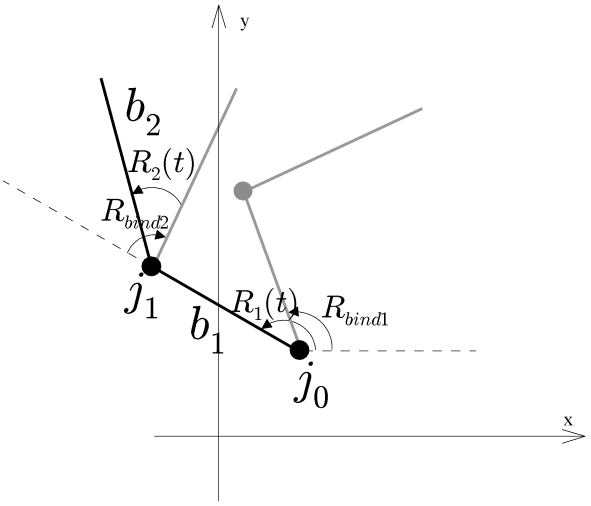
When the skeleton is associated with the geometry, the bones in the skeleton are in a certain pose, which is called the *bind pose*. The rotation $\mathbf{R}_{bind}$ of each bone in this pose is stored, which we refer to as the *bind pose rotation*. An illustration of a skeleton in its bind pose is shown in Figure 1.9(a) on the facing page (in two dimensions for easier comprehension). The reason for storing these rotations is that they are used when transforming vertices from world space to bone space.

During an animation, bones are rotated in space and the vertices associated with a bone are rotated with the bone, hence yielding a new pose of the skeleton. The rotation animation of the bone is specified as a function, $\mathbf{R}(t)$, that yields a rotation around its start joint to time t relatively to the bind pose rotation. Hence, $\mathbf{R}'(t)$ that denotes the total rotation is given by: $\mathbf{R}'(t) = \mathbf{R}(t)\mathbf{R}_{bind}$. Figure 1.9(b) on the next page illustrates a skeleton being rotated from its bind pose. If key frame animation is used, spherical linear interpolation can be used to interpolate between rotations.

At any given time during the animation, a local coordinate system is defined at each joint in the skeleton, and bones contained in the joint specify rotation as rotation of this coordinate system. The bones contained in the root joint always specify rotation as rotation of the world coordinate system. A bone



(a) An illustration of a skeleton in its bind pose. Bone b_1 is rotated with $\mathbf{R}_{bind1}$ around joint j_0 and b_2 with $\mathbf{R}_{bind2}$ around j_1 to yield the skeleton pose.



(b) A skeleton being rotated from its bind pose. $\mathbf{R}'(t)$ for bone b_1, denoted $\mathbf{R}'_1(t)$, is given by: $\mathbf{R}'_1(t) = \mathbf{R}_1(t)\mathbf{R}_{bind1}$. Similarly, $\mathbf{R}'_2(t)$ is given by: $\mathbf{R}'_2(t) = \mathbf{R}_2(t)\mathbf{R}_{bind2}$.

Figure 1.9: Illustrations of a skeleton in its bind pose and of its rotation animation.

contained in any other joint specifies rotation at a given time as rotation of a local coordinate system given by rotating the world coordinate system with the rotations of its parent bones at that time. A bone specifies thus a rotation relatively to its parent bone. How this works is illustrated in Figure 1.10.

Furthermore, at any given time during the animation the position of the end joint of a bone is determined by the length of the bone multiplied with the normalized x-axis of the local coordinate system of the start joint of the bone, followed by the rotation animation of the bone yielded by $\mathbf{R}'(t)$. A translation vector T_0 is given, which specifies where the root joint is positioned. For a bone b_i with $i-1$ ancestors, where $i > 0$, the translation vector $\mathbf{T}_i$ yielding the end joint of b_i is given by

$$\mathbf{T}_i = \mathbf{R'}_i(t)\mathbf{R'}_{i-1}(t) \cdots \mathbf{R'}_1(t)\hat{\mathbf{x}} \cdot length_i, \tag{1.1}$$

where $\hat{\mathbf{x}}$ denotes the normalized x-axis of the world coordinate system, $length_i$ denotes the length of b_i, and $\mathbf{R'}_i(t)$ yields the rotation of b_i at time t.
This is illustrated in Figure 1.11.

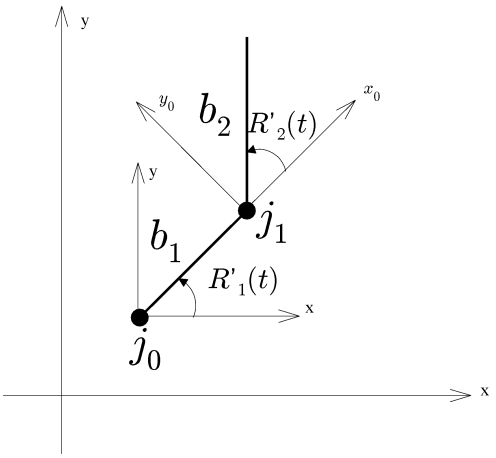


Figure 1.10: An example of how the local coordinate systems are rotated at the root joint j_0 and another joint, j_1. Since the start joint of b_1 is the root joint, j_0, its rotation $\mathbf{R'}_1(t)$ is specified as rotation of the world coordinate system. This yields the local coordinate system with axes x_0 and y_0 in the start joint of b_2, j_1. b_2's rotation, $\mathbf{R'}_2(t)$, is a rotation of this local coordinate system (or, in other words, $\mathbf{R'}_2(t)$ is a rotation of the world coordinate system rotated by $\mathbf{R'}_1(t)$).

A skeletal model is a set of joints and a set of bones, such that the joints and bones represent a rooted tree. The skeletal model contains a reference to the root joint.

Joint : A joint is simply a container for a number of bones. These bones are referred as the children of the joint, and are said to be *siblings* in the skeleton.

- *children*: A non-empty set of bones.

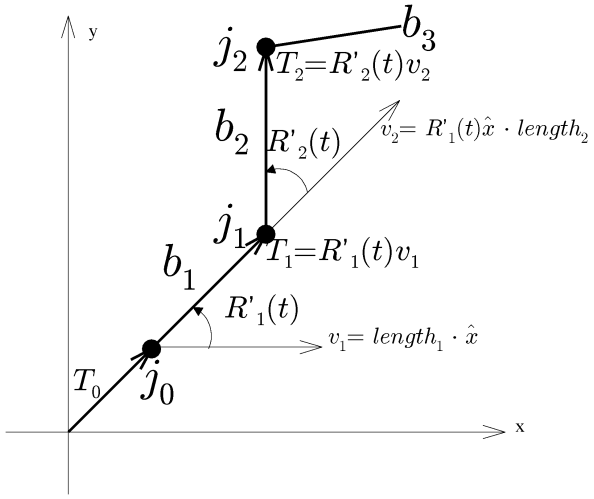


Figure 1.11: How to determine the position of the end joint of a bone in a skeleton. The end joint of b_1, j_1, is determined by scaling the x-axis of the world coordinate system with $length_1$ and rotating this vector with $\mathbf{R}'_1(t)$, yielding $\mathbf{T}_1$. For the end joint of b_2, j_2, the normalized x-axis of the world coordinate system is scaled with the $length_2$, and this vector is rotated by $\mathbf{R}'_2(t) \cdot \mathbf{R}'_1(t)$, hence yielding $\mathbf{T}_2$.

Bone : A bone is represented by the following elements:

- *end joint*: A bone can specify a joint positioned at the end of it. This bone is said to be the *parent* of any bones contained in the children set of this end joint. A bone specifying an end joint is called a *non-terminal bone*, while a bone that does not specify an end joint is called a *terminal bone*.
- *length*: The length of this bone measured in world space units. A length is not defined for terminal bones.
- $\mathbf{R}_{bind}$: The bind pose rotation of this bone.
- $\mathbf{R}(t)$: The rotation animation dictated by this bone, specified relatively to $\mathbf{R}_{bind}$.
- *vertices*: The set of vertices attached to this bone.

In Appendix B), we show how this skeletal model can be used to animate a mesh; more precisely, how to determine the position of each vertex during an animation.

1.5 Observations of Tree Movement

Having discussed model animation and, in particular, skeletal animation, we conclude this chapter by listing a number of observations concerning movement of real-life trees, which we will refer to throughout the remaining chapters. The observations should be fairly general for most tree sorts, but some may only apply to a subset of tree sorts.

- The branch structure of a tree can be adequately described by a connected acyclic graph, each edge of which corresponds to a branch or a part of the trunk, and each node of which corresponds to either a joint between branches, or a leaf. It seems reasonable that such a graph is called 'a tree' in the field of graph theory.
- Wind generates the movement in trees. The wind hits the foliage, and the force enacted upon the foliage is transmitted down through the branch structure of the tree from the leaves towards the trunk, affecting all branches on the way.
- Branches are flexible, and the girth and length of a branch largely determine its rigidity, and thus the amount of force that must be applied make it move a certain distance from its rest point. When a constant wind force is applied to the foliage and thus to a branch that the foliage is connected to, the branch will start to move in a cyclic motion in periods of time. Furthermore, the speed and distance of the movement is largely determined by the rigidity of the branch as well as by the strength of the wind.
- When force is enacted upon a branch, it bends smoothly in the entire length of the branch. However, a reasonable simplified model of a branch may view the branch as a rigid body that may rotate around the point on the trunk or another branch to which the branch is attached.
- The internal structure of a branch makes it resistant against twisting rotations around the axis of the branch itself. The movement of a branch is largely restricted to upwards, downwards and sideways bending around the point to which the branch is attached.
- As gusts of wind in a forest may have varying directions, branches are often affected by wind both from the sides and from the top and bottom. A cyclic movement of a branch will often be elliptical.
- The movement of a single branch is a product of a sequence of bending movements of the parent branches in the branch structure of the tree. A reasonable model of such a movement is a series of rotations of rigid body representations of branches around branch joints.

Furthermore, we list a few observations concerning the appearance of trees:

- When viewed from the ground the foliage of a tree hides the underlying branch structure, and leaves may appear to be placed randomly around the trunk of the tree.
- As the purpose of leaves is to receive energy in the form of light emitted from the Sun, leaves that are in shade of other leaves are defoliated by the tree. In general, this process will result in the foliage being located primarily in the outermost region of the tree, i.e. on the branches furthest from the trunk.

Summary

In this chapter we have introduced and discussed an array of techniques used for simplifying polygonal models and animating them. Along the way we have introduced definitions that we will be using in our work. The purpose of doing this has been to place our work in the context of real-time computer graphics, to identify, discuss, and use some of the well-established approaches to some of the problems we will be addressing in our project, and to introduce the terminology necessary for us to specify our problem definition in precise terms in the next chapter.

Our definition of polygonal tree models has established the type of models we will be working on in our project. As mentioned, the definition delimits from some types of tree models used in different applications, e.g. in some computer games, but, as argued, the limitation is reasonable in regards to our work of simplifying animated tree models.

Different model simplification techniques have been discussed, under the categories of geometry- and image-based simplification techniques, and the notion of an LOD scheme has been presented. The billboard cloud technique was identified as particularly interesting for simplifying the foliage part of polygonal tree models, which was further investigated and justified in the related work by Umlauf (among others) and in our performance analysis of billboard cloud simplification.

The fact that existing billboard cloud algorithms cannot simplify animated polygonal models has encouraged us to explore the possibilities in strengthening the billboard cloud technique to also take animation into account. An introduction to model animation has thus been given. We have presented the key frame and procedural animation types, as well as skeletal animation as an animation technique, and a definition of what a skeletal model is in this project, which will be used throughout the report.

In the final section we listed a number of observations concerning movement of real-life trees. These observations will also be referred to throughout the report.

Chapter 2

Problem Specification

In the last chapter we identified the billboard cloud technique as being particularly applicable to the simplification of the foliage of polygonal tree models. However, we also concluded that the technique cannot be used to simplify animated polygonal models. This is a major limitation, since polygonal tree models in realistic outdoor scenes are animated. If the advantages of this image-based simplification technique could be retained while incorporating the possibility of simplifying animated foliage, the technique would become even more useful.

In this chapter we state our problem goals, based on the research from the last chapter. We begin by defining the input models we will be using and the output models we will be creating.

2.1 Animated Polygonal Tree Models

Having discussed the notions of key frame animation and skeletal animation, as well as defined a skeletal model, we will now expand the definition of a polygonal tree model from Section 1.1 on page 10 to include animation.

Cyclically Animated Polygonal Tree Model: A *cyclically animated polygonal tree model* is a polygonal tree model that is animated by key frame animation, using some animation technique (e.g. vertex animation or skeletal animation). Furthermore, the animation is suitable for being looped.

As with polygonal tree models we distinguish between the foliage part and the trunk part of a cyclically animated polygonal tree model. Throughout the remaining chapters, when we refer to the foliage part of an cyclically animated polygonal tree model, we specifically state it.

Cyclically Animated Skeletal Tree Model (CASTM): A *CASTM* is a cyclically animated polygonal tree model rigged with and animated by a skeleton as defined in Section 1.4.2 on page 26.

The vertices of a triangle in the foliage part of a CASTM are not allowed to be associated with more than a single bone (i.e. each vertex of the

triangle is associated with one and the same bone), while the vertices of a triangle in the trunk part are allowed to be associated with more than one bone, due to the reasons stated in Section B on page 158.

Joints are positioned inside the trunk mesh only, not in the foliage, and each joint in the skeleton corresponds to an articulation in the trunk mesh. Likewise, bones are placed along the branches only. In other words, we will assume that the trunk mesh of a polygonal tree model makes out the ”skin” on top of the skeleton, not the foliage. The foliage triangles will be animated by following their associated bones in the trunk mesh. Consequently, we do not allow a bone per leaf.

We will require that the input models to be simplified are CASTMs.

The fact that we require our input polygonal tree model to be rigged with an animation skeleton is a reasonable requirement, because polygonal tree models are suitable for this kind of animation, as mentioned in Section 1.4.2 on page 25. Skeletal animation is in fact commonly used for articulated articulated meshes [3]. Furthermore, it is an effecient way of storing animation information.

The intuition behind our restrictions of joints in the animation skeleton being positioned in the mesh and bones being placed along the branches is that the skeleton reflects the structure of its CASTM. Hence, a bone in the skeleton placed far away from the root in the hierarchy will represent a branch in the polygonal tree model far away from the trunk in the hierarchy. We judge this assumption reasonable in the sammer manner as we judge it reasonable that the skeleton of a character model is placed inside the mesh, and that it follows the structure of the model.

We will not discuss how a polygonal tree model can be animated using procedural animation, the reason being that when using procedural animation, no a priori knowledge about coplanarity of vertices in the model can be assumed, and hence no billboard cloud simplification can be performed. Instead we will focus on key frame animation in the remainder of our report.

When we refer to the foliage part of a CASTM, we specifically state it.

2.2 Animated Billboard Cloud

Having specified that our input models are CASTMs, we now turn to specifying the output models we will be creating. In the following we define animated billboards and animated billboard clouds.

Animated Billboard: An *animated billboard* is a rectangle with a texture, a transparency map, and an associated animation cycle dictating the position of each rectangle vertex as a function of time. The rectangle defined by the four vertices is not allowed to be altered by the animation. More precisely, the billboard is only allowed to be transformed by rigid transformations (i.e. transformation consisting only of translations and rotations).

An animated billboard is not to be confused with a view-aligned billboard. Contrary to a view-aligned billboard an animated billboard does not change orientation as dictated by the movement of the observer, but solely as dictated by its animation function.

Note that our definition only includes animation of the actual billboard rectangle, and not of the texture associated with a billboard. Although animated textures definitely could be used to simplify animated geometry, we do not consider such an approach.

Animated Billboard Cloud: An *animated billboard cloud* is a set of animated billboards.

Given a CASTM, we will create simplified LOD versions of it consisting of an animated billboard cloud for the foliage part of the simplified CASTM and an animated trunk with a simplified cyclic animation.

The definitions of our input and output models enable us to state our project goals.

2.3 Project Goals

A recognized weakness of existing billboard cloud algorithms is their lacking ability to simplify animated models. In this project we state the following project goals:

- We want to develop simplification solutions specifically for animated polygonal tree models that combine the billboard cloud simplification technique with animation simplification techniques in order to create discrete LOD versions of a CASTM.
- These LODs should consist of an animated billboard cloud for the foliage and a simplified cyclic animation of the trunk. Hence, we want to perform image-based simplification on the foliage part.
- The solutions should be capable of outputting LODs of high visual fidelity to the input CASTM for close range and extreme simplification LODs for far-away distances. We require that the rendering complexity drops with the degree of simplification, such that the fps performance increases the more simplified the LODs are.
- In order to evaluate the visual fidelity of an animated billboard cloud model we want to define a set of error metrics that enable us to objectively measure the quality of the simplification. Since we want to use the animated billboard cloud model in an LOD scheme for the CASTM, quality is measured in terms of visual resemblance to the input CASTM. Furthermore, quality is also measured in terms of the performance requirements involved in using the generated LODs.

- Aside from using the error metrics to measure the quality of the LODs against the input CASTM, we want to use them to compare the quality of the LODs generated by different solutions.

2.4 Project Delimitations

Having stated our project goals we turn to state what is outside the scope of this project.

- Our focus is the simplification of the foliage part of animated polygonal tree models. We therefore limit ourselves from considering geometric simplification of the trunk part of a CASTM. We refer to the methods mentioned in Section 1.2.2 on page 14 for simplifying animated closed meshes.
- We want to create simplified LOD models of CASTMs. However, we do not provide techniques for smooth transitions between the LODs at runtime, even though such techniques should be applied in order to obtain satisfactory results.
- Unlike Umlauf [28], our focus is not to render huge forests at interactive rates. Our sole focus is simplification of a single cyclically animated polygonal tree model and the performance issues of rendering a single cyclically animated polygonal tree model. Specific performance issues involved in rendering our animated billboard cloud LOD models in a large scale forest are not considered, even though these issues must be addressed, e.g. by considering batching of the LODs, in order to obtain fast rendering of large scale forests.
- We do not consider simplification of procedurally animated polygonal tree models.
- The issues involved in rendering our animated billboard cloud LODs with realistic light and shadows are not considered.

2.5 Report Structure Outline

In Chapter 3 we initially discuss data clustering approaches, and the notion of error- and budget-based clustering. These terms reappear in a discussion of strategies for billboard cloud construction. Finally, an overview of three of the known billboard cloud algorithms is provided, including the original algorithm by Décoret et al. The chapter is concluded by discussing pros and cons of the three algorithms and we end up choosing an algorithm for the implementation of our solutions.

In Chapter 4 we discuss how animation can be analysed. Different animation properties of cyclically animated polygonal tree models are identified, which should be taken into consideration when simplifying the animation of the

polygonal tree models, comprising CASTMs. Spectral analysis of animation will be the focus, which is why Fourier transform is introduced, including Discrete Fourier transform. In the remainder of the chapter we discuss suitable ways of analyzing animation using Discrete Fourier transform.

In Chapter 5 a set of error metrics is provided. These error metrics will be used to measure the quality of our LOD simplifications of CASTMs. Equally important, we will discuss different strategies that can be applied to perform the simplification of CASTMs.

In Chapter 7 and Chapter 6 our solutions are presented, each one following a different strategy from Chapter 5.

The results of implementing one of the solutions will be presented in Chapter 8, as well as an evaluation of the results.

We conclude our report in Chapter 9.

Chapter 3

Billboard Cloud Construction

In this chapter we present algorithms for automated model simplification using billboard clouds. Given a polygonal model a billboard cloud algorithm constructs a billboard cloud that simplifies the original model.

Before presenting the algorithms, we briefly introduce the topic of data clustering in Section 3.1. This is succeeded by a discussion of strategies on how to apply data clustering in order to construct billboard clouds in Section 3.2, including how to evaluate a simplification.

In Section 3.3 we present the original billboard cloud construction approach. A billboard cloud algorithm based on k-means clustering is presented in Section 3.4, and finally one based on a stochastic search is presented in Section 3.5.

Some general improvements to the construction of billboard clouds are presented in Section 3.6, which can be used to obtain higher visual fidelity simplification of foliage. The chapter is concluded with us choosing an algorithm for foliage simplification, and a short discussion of an implementation of this algorithm.

3.1 Data Clustering

A billboard cloud algorithm is basically a clustering of coplanar triangles. Therefore, we give a short introduction to data clustering theory in general prior to discussing different billboard cloud construction approaches. This section is based on a review of data clustering by A.K. Jain et al. [14].

Data clustering is the organization of a set of data elements called *patterns* into subsets. The subsets are referred to as *clusters*, and the intuition is that the patterns in one cluster are more similar to each other than they are to the patterns in other clusters. Figure 3.1 illustrates the clustering concept.

Such data clustering has many useful applications, a few examples of which are data mining, image segmentation, and, in our case, billboard cloud construction.

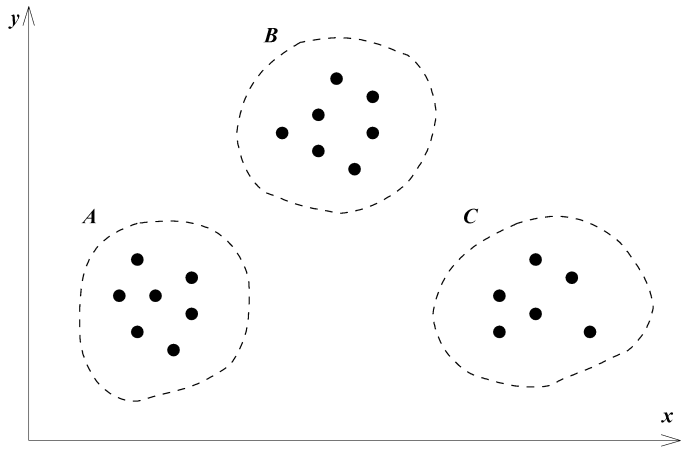


Figure 3.1: An example of data clustering where a set of 2D points are clustered by Euclidean distance. The three clusters are denoted A, B, and C.

3.1.1 Components of the Clustering Task

The task of clustering a set of patterns consists of some basic components, which we will briefly go through.

Pattern representation: One must define a reasonable representation of the data elements to be clustered. A pattern is usually an n-dimensional vector of scalar components, each of which is referred to as a *feature*. A pattern is therefore also referred to as a *feature vector*.

Distance measure: Definition of a reasonable pattern proximity measure for a given problem domain is of key importance. This is usually defined as a function yielding the similarity of two patterns as a scalar value. The most commonly used measure is the Euclidean distance between the two feature vectors, although a drawback of this is that a large-scale feature can dominate the others. The solution is to apply normalization on the features.

Clustering: This refers to the strategy of how to group the patterns based on the distance measure. A clustering can either be *fuzzy* or *hard*. In a fuzzy clustering, each pattern can have a variable membership in several clusters, and in hard clustering each pattern is member of one and only one cluster. A grouping of the dataset in clusters is also referred to as a *partitioning*.

Data abstraction (optional): Sometimes a simple compact representation of a set of patterns is useful. An example is representing the content of a cluster using a single centroid pattern.

Output Assessment (optional): A validation of the clusters yielded by a clustering algorithm. The goal is to ensure that the clusters could not have been chosen by chance or by an artifact of the clustering algorithm.

A simple output assessment is to run the clustering algorithm several times on the data, and verify that the clustering does not vary much.

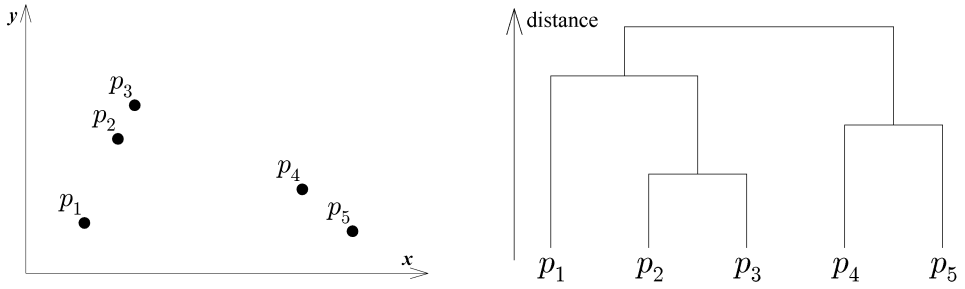
In the following, we focus on the different strategies for doing the actual clustering. Many different approaches to clustering exist, but a distinction can be made between hierarchical and partitional approaches. A hierarchical clustering produces a nested series of partitions, while a partitional clustering only produces one. We briefly discuss each type and present some example algorithms.

3.1.2 Hierarchical Clustering Algorithms

A hierarchical clustering algorithm produces a series of partitions as follows:

1. Begin with the partition in which each pattern defines its own cluster.
2. Compute the distance between each cluster pair using some distance measure.
3. Merge the cluster pair that are closest (most similar).
4. Unless only a single cluster remains, goto step 2.

The output of the algorithm is a tree structure containing all the different partitions during the process, called a dendrogram. An example of a such is illustrated in Figure 3.2(b).



(a) A simple dataset consisting of five patterns.

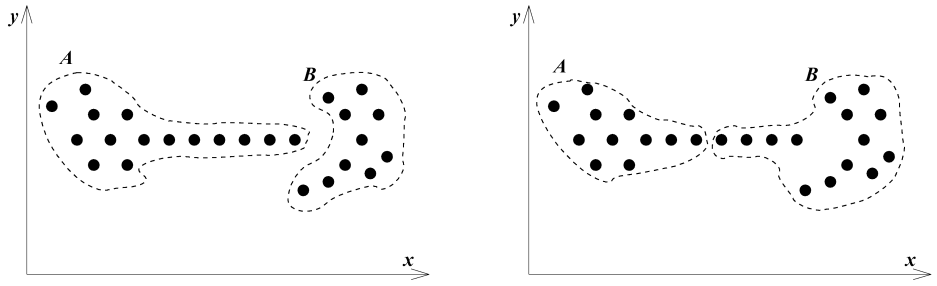
(b) Clustering the dataset by Euclidean distance yields this dendrogram. The distance axis denotes when which clusters are paired.

Figure 3.2: Example of a hierarchical clustering, and the constructed dendrogram.

The two most popular hierarchical clustering algorithms are the *single-link* and *complete-link* algorithms. They both follow the above steps, but differentiate in how the similarity between a cluster pair is computed in Step 3. In the single-link approach, the distance between two clusters equals the *minimum* distance between a pair of patterns from the two clusters, such that the first element of the pair is a pattern from the first cluster, and the second element a pattern from the second cluster. Complete-link defines cluster distance as the

exact opposite, namely as the *maximum* distance between the pairs of patterns that can be constructed from the two clusters.

The single-link clustering algorithm implies a "chaining" effect, because clusters can be constructed that contain patterns being far distanced from each other, by keep expanding a cluster with patterns that have a low minimum distance, but potentially a large maximum distance. This is illustrated in the example in Figure 3.3(a). The complete-link clustering algorithm produces more compact clusters, which is demonstrated by the example in Figure 3.3(b).



(a) A dataset clustered using the single-link algorithm. Notice the chaining effect.

(b) The same dataset clustered by the complete-link algorithm. The clusters are more compact than when using single-link.

Figure 3.3: The partitioning obtained using two different hierarchical clustering algorithms, and Euclidean distance as distance measure.

Depending on the application and the dataset, the chaining effect can be useful, as it can be used to find shapes in the dataset, such as patterns arranged in a circle in feature space.

The main drawback of hierarchical clustering algorithms is their complexity. The running time is $O(n^2)$, where n is the size of the dataset, and they hence do not scale well for large datasets.

3.1.3 Partitional Clustering Algorithms

Partitional clustering algorithms produce only a single partitioning of the dataset. Most such algorithms are faster than the hierarchical algorithms, as they do not compute a dendrogram. They usually attempt to minimize some criterion function, although finding the exact minimum is in general computational too expensive for practical purposes.

We differentiate between *error-based* and *budget-based* algorithms, a separation used by Décoret et al. when discussing billboard cloud construction [7]. We, however, apply the terms to data clustering in general. The intuition is to consider the number of clusters as a *cost*, and the distance between the patterns as an error.

In an error-based algorithm, a maximum allowed distance between any two patterns in the same cluster is specified. The task of the clustering algorithm is then to find a partitioning using the minimum number of clusters that respects this maximum distance. In a budget-based algorithm, a maximum allowed

number of clusters is specified, and the task of the algorithm is then to minimize the criterion function using no more than that amount of clusters.

K-means Clustering Algorithm

The k-means clustering algorithm is a partitional budget-based clustering algorithm employing the frequently used *squared error metric*. This metric works well with isolated and compact clusters. Given a dataset (set of patterns), D, partitioned into k clusters, $P_1, P_2, ..., P_k$, the squared error is:

$$e^2(P,k) = \sum_{j=1}^{|k|} \sum_{i=1}^{|P_j|} distance(x_i^j, c_j)^2 \tag{3.1}$$

where x_i^j is the i'th pattern in cluster P_j, c_j is the centroid pattern of the cluster P_j, and $distance$ is the pattern distance measure.

The k-means clustering algorithm is the most popular squared error clustering algorithm. In simple steps, the algorithm is as follows:

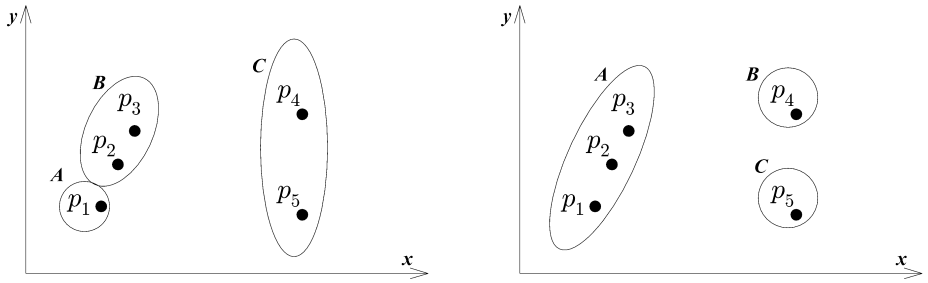
1. Randomly assign the patterns in the dataset to k clusters.
2. Compute the centroid of each cluster.
3. Assign each pattern to the cluster, where the distance to the centroid is least.
4. If a termination criterion is not met, go to step 2.

In each iteration of the pattern reassignment loop, the squared error is guaranteed to either decrease or remain the same. The algorithm is, however, not guaranteed to minimize the squared error, as it can converge to a local minimum. The termination criterion is typically that when no patterns are reassigned, or when the squared error remains constant. The time complexity of the algorithm is $O(n)$, where n is the number of patterns in the dataset. It hence scale well with large amounts of data.

How the clusters are initialized in step 1 is crucial for which local minimum is met, and hence the quality of the clustering. An example of this is illustrated in Figure 3.1.3 on the following page.

There is no perfect solution to the problem of determining a good initialization, so the common workaround is to run the algorithm several times on the same data with random initializations. If the partitions do not vary much in the yielded results, then the algorithm has probably not yielded this partitioning by chance. In general, however, there are no guarantees.

Another problem is how to select the number of clusters, k. If the number of natural groupings in the dataset is unknown in advance, and the algorithm is run with a too large k value, then over-fitting may occur. If the task in some clustering application is to find the minimum set of clusters representing the actual groupings in the input data, then choosing a reasonable k value in advance is not possible. This is in fact a problem with all budget-based clustering



(a) The three clusters obtained using initial clusters $\{p_1\}$, $\{p_2\}$, and $\{p_3, p_4, p_5\}$.

(b) The clusters obtained using the initial clusters $\{p_1, p_2\}$, $\{p_3, p_4\}$, and $\{p_5\}$. This result minimizes the squared error.

Figure 3.4: A dataset clustered using the k-means clustering algorithm with $k = 3$ and Euclidean distance as distance measure. The two figures demonstrate the difference in result with different cluster initializations.

algorithms, and unfortunately there do not exist any theoretical solutions to it [19].

Many other partitional clustering algorithms exist. For further details we refer to the data clustering review in [14].

3.2 Billboard Cloud Construction Strategies

In this section we discuss how clustering can be used to perform billboard cloud simplification of a polygonal model, including the difference between error- and budget- based billboard cloud construction. How to measure the quality of a billboard cloud simplification is also discussed.

3.2.1 Clustering Coplanar Triangles

In Section 1.2.3 on page 15 we argued that a set of coplanar triangles can be replaced by a single billboard, onto which the triangles are rendered. Almost coplanar triangles can also be replaced by a billboard, but will introduce error due to the distances the triangle vertices are translated in order to become coplanar. We refer to this as the *vertex displacement error*.

Constructing a billboard cloud is at its core a data clustering task, where triangles are clustered by coplanarity. The clustering is hard, as a single triangle should only be represented on a single billboard. When this clustering is performed, only mere technical issues remain. These include finding a least bounding rectangle of the triangles in a cluster projected onto a best fitting plane, and then render the triangles into a texture.

In clustering terminology, the patterns to be clustered are triangles, and the distance measure between two triangles is the sum of their vertex displacement error when projecting the triangles onto their best fitting plane. The squared error metric can be applied by considering the sum of squared displacement error the triangles in a cluster have, when projecting each to the best fitting plane

of the triangles in the cluster. Intuitively, minimizing this squared error can be used to yield a billboard cloud simplification with high visual resemblance to the polygonal model being simplified, as the geometry is displaced the least possible. In the next section we discuss the quality of a given billboard cloud simplification.

3.2.2 Quality of Simplification

We examine the quality of a billboard cloud simplification with regards to using it in a discrete LOD scheme. In this context the most important quality criterion is that the produced billboard clouds visually resemble the original model as precisely as possible from all angles, using as many (and as large) billboards deemed acceptable at a particular LOD.

We introduce the terms *cost* and *error* of a billboard cloud simplification. The cost of a billboard cloud is its rendering cost. As discussed in Section 1.3.3 on page 21, the rendering complexity of a billboard cloud is determined by the number of billboards, and the size of these billboards. In every billboard cloud construction approach we have studied, the cost has been reduced to the number of billboards for simplicity. This is acceptable, assuming that increasing the number of billboards also increases the total area. However, in practise this assumption does not always hold.

The error of a billboard cloud simplification refers the visual error introduced by replacing the original model by the billboard cloud. The vertex displacement error can be used as a reasonable measure of this visual fidelity. The billboard cloud LOD is to be used at a certain distance from the camera, and this distance can be transformed to pixel error on the screen.

Alternatively, the visual resemblance can be measured using pure image-based means, such as the colour and error metrics proposed by Huang et al. in [12]:

Colour error : For each pixel, the difference between the rendered colour value of the original model and the rendered colour value of the billboard cloud model is calculated. The colour error is the sum of these difference values.

Depth error : Similarly, a depth difference can be calculated for each pixel, and the depth error is the sum of these difference values (obviously, the transparent areas on the billboard must not count as pixel errors).

When using either of these metrics, the models should be rendered from different angles and the error computed for each angle. The total error can be defined as the average error obtained from the different angles, where precision of the total error depends on the number of angles used.

3.2.3 Error- and Budget- Based Simplification

We differentiate between error- and budget- based data clustering, and having defined error and cost with regards to billboard cloud construction, a similar

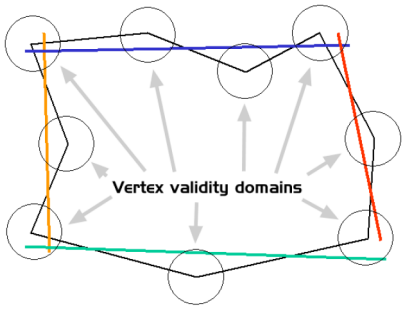


Figure 3.5: Vertex validity spheres and valid planes simplifying the vertices. The figure is from [28].

distinction can be made between different billboard cloud simplification approaches.

In the error-based simplification approach proposed by Décoret et al., a maximum allowed vertex permutation distance, ϵ, is specified. ϵ can geometrically be interpreted as a spherical region around each vertex with radius ϵ, which is referred to as its *validity domain*, or its *validity sphere*. A plane can only simplify a vertex if it intersects the validity sphere of the vertex. Consequently, a plane can only simplify a triangle if it intersects the validity domains of all vertices of the triangle. In this case the plane is *valid* for the triangle and vice versa. This is illustrated in Figure 3.5.

The task is to find the minimum number of billboards that are valid for all triangles in a model to be simplified. This problem is unfortunately NP-complete, as noted by Décoret et al. [7]. Because of this, practical error-based algorithms merely approximate this minimum.

In an error-based simplification approach, a worst-case vertex displacement is specified using ϵ, which in turn is a measure of the visual error introduced by the simplification. The number of billboards in the simplification, and hence a measure of its rendering complexity, is only indirectly specified by ϵ, as the higher value of ϵ, the further the triangles can be displaced, generally resulting in fewer billboards. In order to obtain a suitable rendering cost for a specific LOD model, some trial-and-error with different ϵ values might be done.

Budget-based simplification is on the contrary based on specifying a maximum allowed rendering cost, typically the number of billboards, and then finding a billboard cloud respecting this cost, while minimizing a visual error measure such as the squared displacement error. We believe this problem to be NP-complete, and a budget-based clustering algorithm, such as the k-means clustering algorithm, only guarantees to convergence toward a local minimum of the squared error. The drawback of this approach is that no guarantees for the visual error is given. Therefore, in order to obtain a simplification visually satisfactory for a specific LOD model, some trial-and-error might be done with different budgets.

To sum up, the two billboard cloud simplification approaches are:

Budget-based : Given a maximal budget (total cost), the goal is to build a

billboard cloud respecting this cost while minimizing the error.

Error-based : Given a maximum tolerable error, the goal is to build a billboard cloud respecting this error while minimizing the cost.

Both approaches have pros and cons. A billboard cloud simplification for a specific LOD model should both satisfy some visual error and rendering cost criteria. The approaches differentiate in which of these two criteria sets to hold constant and which one to minimize, and we judge it dependent on the application which approach is the better.

3.3 The Original Billboard Cloud Algorithm

The Original Billboard Cloud Algorithm presented by Décoret et al. in [7] is an error-based billboard cloud construction approach. Given a maximum allowed displacement distance, ϵ, the algorithm attemps to find the minimum set of billboards that are valid for the entire model to be simplified.

The basic idea is to transform the triangles of the input model to a vector space called *dual space*, where nearly coplanar planes are numerically close. This space is then searched for planes that are valid for a large number of triangles, and finally choose a number of such planes that together can simplify all the triangles of the input model. For each of those planes, the triangles simplified by the plane are rendered to a billboard texture placed in the plane.

In the following we explain how the Original Billboard Cloud Algorithm finds planes in dual space, that can simplify the input model. The following is based on [7] and the illustrations are from [28].

3.3.1 Dual Space

A plane is defined by $ax + by + cz - \rho = 0$, where $\mathbf{n} = (a, b, c)$ is its normal vector and ρ is the distance from the origin to the plane, if $\mathbf{n}$ is normalized. Alternatively, a plane can be defined in spherical coordinates as:

$$x \cos\theta \cos\phi + y \sin\theta \cos\phi + z \sin\phi - \rho = 0 \tag{3.2}$$

where (θ, ϕ) are the angles that describe the orientation of the normalized plane normal $\mathbf{n}$ and ρ is the distance from the origin to the plane. The values of θ, ϕ, and ρ unambigously define a plane.

Dual space is a three-dimensional vector space with dimensions θ, ϕ, and ρ, and a point in this space represents a plane in Euclidean space. The point in dual space representing a plane in Euclidean space is referred to as the *dual* of that plane. The transformation of a plane to a point in dual space is called a *Hough Transform*.

A point $p = (x, y, z)$ in Euclidean space can be defined uniquely by an infinite set of planes containing p. Consequently, a point can be described in dual space as a surface. For each pair (θ, ϕ) there exists exactly one plane with normal $\mathbf{n} = (\theta, \phi)$ containing p, namely the plane with ρ-value:

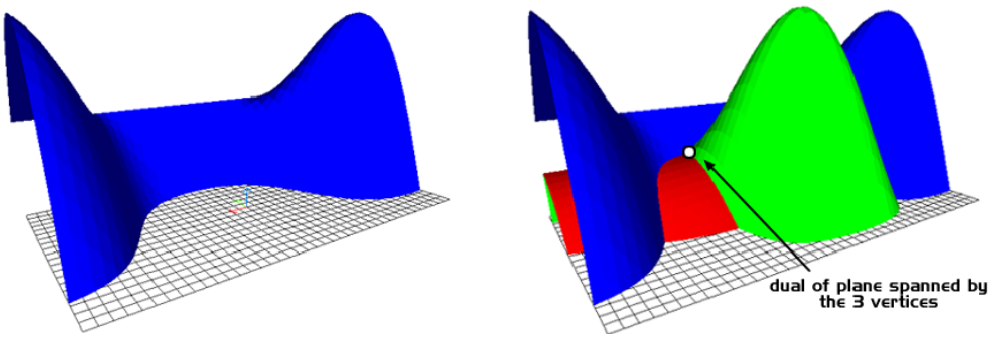


Figure 3.6: The dual of a point and the dual of a triangle.

$$\rho(\theta, \phi) = xcos\theta cos\phi + ysin\theta cos\phi + zsin\phi \tag{3.3}$$

This function yields the surface in dual space, which is the dual of p. An example of such a surface is shown in Figure 3.6 (left).

A triangle in Euclidean space is defined by three points. The dual of a triangle is the duals of its three points, hence three surfaces. These surfaces intersect each other in one point, namely the dual of the plane containing the triangle. This is illustrated in Figure 3.6 (right).

3.3.2 Vertex and Triangle Validity in Dual Space

Recall that vertex validity domain in Euclidean space is a spherical region around a vertex with radius ϵ. The dual of the vertex describes planes (θ, ϕ, ρ) intersecting this validity domain, but only those planes intersecting the center of the validity sphere. Therefore, the dual of the vertex is a subset of the validity domain of the vertex in dual space.

Adding some value to ρ corresponds to translating a plane in its normal direction with this value. Therefore, if ϵ is added to ρ of some plane on the surface that is the dual of a vertex, a plane tangent to the validity sphere of this vertex is obtained. ρ_{min} and ρ_{max} for a given (θ, ϕ) pair are defined as:

$$\rho_{min}(\theta, \phi) = \rho(\theta, \phi) - \epsilon \quad , \quad \rho_{max}(\theta, \phi) = \rho(\theta, \phi) + \epsilon \tag{3.4}$$

The ρ-values in the interval $[\rho_{min}(\theta, \phi); \rho_{max}(\theta, \phi)]$ represent all planes with orientation (θ, ϕ) that intersect the validity sphere of the vertex with dual $\rho(\theta, \phi)$. Computing this interval for all orientations corresponds to translating the dual of a vertex $+\epsilon$ and $-\epsilon$ in the ρ direction. Thus, the validity domain of a vertex in dual space is a validity region defined as:

$$\rho_{min}(\theta, \phi) \leq \rho \leq \rho_{max}(\theta, \phi) \tag{3.5}$$

An example of a vertex validity domain is illustrated in Figure 4.3. Correspondingly, the validity domain of a triangle is the intersection of the validity regions of its three vertices, hence a subspace in dual space.

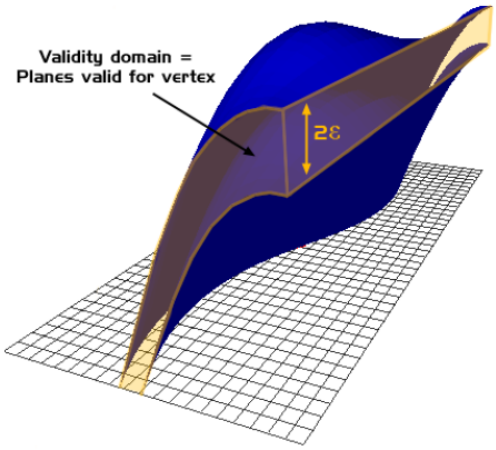


Figure 3.7: The validity domain of a vertex in dual space.

3.3.3 Contribution and Density

An entire set of planes are valid for a triangle (namely the set of planes contained in the subspace defined by the validity domain of the triangle), but the different valid planes cannot represent a given triangle equally well. When rendering billboard textures, an orthogonal projection of triangles onto the billboard plane is performed. If a triangle is orthogonal projected onto a plane with a normal almost perpendicular to the triangle normal, then the triangle will not be faithfully represented on the plane. *Contribution* of a triangle to a plane is defined as the area of the triangle, when it is orthogonal projected onto the plane. A triangle yields thus highest contribution to planes with a normal similar to the triangle normal, which are considered to be the planes that can represent the triangle best.

An even more important reason for introducing the contribution term is to be able to favor large triangles over small triangles when searching for the best billboard planes. The *density* of a plane is the sum of the contributions of all the triangles that the plane is valid for, to the plane. Density is hence a measure of how large an area of geometry a plane can faithfully simplify.

3.3.4 Algorithm

As stated in Section 3.2.3 on page 43 the problem of finding the minumum number of billboards simplifying all triangles in the input model is NP-complete, and so for performance reasons an approximative greedy algorithm is opted for.

To allow searching through dual space, it is discretized into a grid; the ranges of θ and ϕ are divided into a discrete number of steps with equal angle increments. Likewise, the ρ axis is divided into a number of steps with equal distance increments. A triangle validity domain maps to a set of cells in the dual space grid, which are the cells containing planes this triangle is valid for. The triangle is said to be valid for these cells, and vice versa.

The basic idea is simply to let each triangle add contribution to all the cells it

is valid for, and then consider the cell with highest density in the grid. Formally, the density of a cell is defined as the sum of the contributions of all triangles, that are valid for the cell, to the *center* plane of the cell.

Informally speaking, planes inside the highest density cell must be able to faithfully represent a large area of geometry, and are thus suitable for billboard creation. The highest density plane within the cell is searched for, and a billboard is created by considering an orthogonal projection of valid triangles onto the plane found. A least bounding rectangle is found for the projected triangle vertices, and the corners of this rectangle define the billboard quad. The triangles are rendered into a texture, which is associated with the billboard. The contribution of the simplified triangle is removed from the dual space grid, and a new highest density cell is considered, etc.

The algorithm is a simple greedy selection of the highest density planes in the grid, and the steps are as follows:

1. For each triangle, do:
   (a) Compute the set of cells that are valid for the triangle using ϵ.
   (b) Add the contribution of the triangle to each of these valid cells. The contribution to a cell is computed as the contribution to the center plane of the cell.
2. Within the highest density cell, search for the highest density plane and choose the plane found to create a billboard from. Remove the contribution of all the triangles simplified by the billboard from the dual space grid.
3. Unless all triangles have been simplified (i.e. the grid contains no more density), go to step 1.
4. Output the created billboards.

Relating this algorithm to the data clustering theory in Section 3.1 on page 37, it is an error-based hard clustering of coplanar triangles. In order to obtain these hard clusters, however, a fuzzy clustering is iteratively performed. Each cell in the grid is in fact a cluster of (nearly) coplanar triangles, and each triangle (pattern) is contained in several cells (fuzzy clusters). A triangle has a highest degree of membership in the cell to which it contributes with the largest value. The highest density cell is chosen for billboard creation, followed by a new fuzzy clustering of the remaining triangles.

3.3.5 Analysis of the Algorithm

We have left out many details in the algorithm, such as how to compute which cells are valid for a triangle, and how to search for a good plane within the highest density cell. The algorithm in its entirety is actually quite complex and non-trivial to implement, as noted by Umlauf [28].

The algorithm finds only approximately the minimum number of billboards to simplify the model, given an ϵ value, since it greedily picks the cell with highest density in each iteration. The problem is that simplifying the triangles in the highest density cell might have unfortunate consequences for the simplification of the leftover geometry in the following iterations. More precisely, the leftover geometry might not be easily simplified by few billboards. A simplification of the triangles in another cell with less density might result in more easily simplified leftover geometry, and could consequently lead to a simplification using fewer billboards in total. Some improvements are proposed by Décoret et al., in order to generally obtain fewer billboards, but these will not be discussed here.

Experiments with an implementation of the algorithm by Umlauf in [28] has shown that a specific tree model consisting of 200,000 triangles requires between 300MB and 1GB of memory and takes about 20 minutes to simplify, so the algorithm is obviously resource demanding. This is impractical, as obtaining satisfactory results often requires tweaking of ϵ and grid resolution. Furthermore, several LOD versions of the original model is often wanted, which makes the simplification process even more tedious.

3.4 K-means Clustering Billboard Cloud Algorithm

In [12], I.-T. Huang et al. propose a different billboard cloud algorithm based on the K-means Clustering algorithm. The following is a summary of this K-means Clustering Billboard Cloud Algorithm.

As opposed to the Original Billboard Cloud Algorithm, the K-means Clustering Billboard Cloud Algorithm follows a budget-based approach to billboard cloud simplification. The basic idea of the algorithm is to partition triangles into a set of k cluster of triangles each representing a billboard. The centroid of a cluster is the best plane fitting the contained triangles, which can be found using Singular Value Decomposition of the vertices of the triangles in the cluster (details on Singular Value Decomposition can be found in [16]). The distance metric considered is the distance between the vertices of each triangle and the centroid plane of a cluster.

Similar to the general k-means clustering algorithm, triangles could initially be assigned randomly to clusters. This approach, however, yields poor results with the billboard cloud algorithm, and the simplification results tend to vary greatly. Instead, a more intricate initialization procedure is performed.

3.4.1 Initialization

The initialization procedure of the algorithm initially assigns triangles to k clusters based on proximity to k planes evenly distributed on a bounding sphere of the input model. For a closed mesh model this sphere is a rough approximation of the silhouette of the model, and distributing planes evenly on this sphere is usually a reasonable starting point. The initialization procedure is carried out as follows:

1. Compute a bounding sphere of the model. Distribute k points evenly over the surface of the sphere, for example as described in [18].

2. For each of the k points, create a new cluster and associate it with the plane tangent to the sphere at the point.

After this initial assignment, two further steps are taken to reduce the variance in the number of triangles in the different clusters. These steps will not be described further.

3.4.2 Algorithm

When the initialization stage is performed, the algorithm performs the actual k-means clustering:

Repeat until the squared error metric reaches a local minimum:

4. For each triangle: Associate the triangle with the cluster, for which the distance between the triangle and the centroid plane of the cluster is smallest.

5. For each cluster A:

   (a) If A is empty, for each other cluster B:

      i. Find the triangle T of B that has the largest angle between its normal and the normal of the plane associated with B.

      ii. Assign T to A.

   (b) Recompute associated plane using Singular Value Decomposition of the vertices of the triangles in A.

When the clustering is completed, a billboard is created for each cluster as usual. A least bounding rectangle is found for the triangles orthogonal projected onto the centroid plane, and a billboard texture is rendering.

3.4.3 Analysis

The quality of the simplifications produced by the algorithm can be objectively evaluated in comparison with the Original Billboard Cloud Algorithm for a specific input model using, for example, the colour or depth error metrics represented in Section 3.2.2 on page 43. I.-T. Huang et al. present such a comparison in [12], and the K-means Clustering Billboard Cloud Algorithm consistently outperforms the Original Billboard Cloud Algorithm in both the colour and depth error metrics for an example input model simplified to an equal number of billboards. Even though this result indicates that the K-means Clustering Billboard Cloud Algorithm has better simplification quality than the Original Billboard Cloud Algorithm, the result cannot be generalized to other models.

I.-T. Huang demonstrates that an implementation of the K-means Clustering Billboard Cloud Algorithm executes faster than an implementation of the Original Billboard Cloud Algorithm for specific configurations. The analysis is not described adequately enough to draw any general conclusions. However, k-means clustering is known to be fast even for large data sets, as mentioned in Section 3.1.3 on page 41.

The normals of the triangles are not a part of the distance measure of the K-means Clustering Billboard Cloud Algorithm. This is potentially a problem, as a set of parallel triangles may be simplified to an orthogonal plane, which would result in a projected triangle area of zero. To solve this problem, we propose a k-means dual space clustering approach.

3.4.4 K-means Dual Space Proposal

The idea is to utilize the coplanarity proximity features of dual space, and utilize the potential advantages of the k-means clustering, such as simplicity, low time complexity, and the ability to specify a billboard cloud budget. The approach is very simple: Each triangle is mapped to a point in dual space, and these points are clustered by Euclidean distance using the K-means clustering algorithm. In order to favor large triangles over small ones, the triangles can be weighted by area, when computing the centroid of a cluster.

This demonstrates how dual space can be utilized in other ways than originally proposed by Décoret et al, but we will not discuss it further.

3.5 Stochastic Billboard Cloud Algorithm

Lacewell et al. propose a billboard cloud algorithm specifically designed for foliage simplification [17]. The algorithm is an error-based hard clustering of coplanar triangles, and it performs a stochastic search for billboards that are valid for a large amount of geometry. It is simple and fast in comparison to the Original Billboard Cloud Algorithm and the K-means Clustering Billboard Cloud Algorithm.

3.5.1 Algorithm

The Stochastic Billboard Cloud Algorithm operates on a set of triangles, and given a maximum allowed permutation distance, ϵ, it simplifies the input triangles to a number of billboards. We use the notions of validity, density, and contribution from the Original Billboard Cloud Algorithm in the following description of the algorithm.

The basic idea is to randomly search for a high density plane and continue to do so until all triangles have been simplified. In order to guarantee that a plane is always valid for some triangle, a random plane is chosen by considering a random *seed triangle* among the set of input triangles and translate its vertices randomly in the normal direction with a maximum distance of ϵ. The plane

defined by the translated triangle vertices is guaranteed to be valid for at least one triangle, namely the seed triangle.

In addition to the input triangles and ϵ, a parameter, S, is used to specify how many random planes are considered in each iteration before choosing the one with maximum density for simplification. The algorithm in overall steps is as follows:

Repeat until all triangles have been marked simplified:

1. Repeat S times
   (a) Set the highest density found to zero: $D_{max} = 0$.
   (b) Select a random seed triangle t_s among the set of unsimplified input triangles.
   (c) Translate each of the vertices of t_s in the triangle normal direction with a random distance in the interval $[-\epsilon; \epsilon]$. Define the plane, B, containing the three translated triangle vertices.
   (d) Find the set of unsimplified triangles, T, which are valid for B. Calculate the density of plane B, which equals the sum of contribution of all triangles in T to the plane. If B has higher density than D_{max}, then store B as B_{max} and T as T_{max}.
2. Mark all triangles in T_{max} as *simplified*, and add the pair (B_{max}, T_{max}) to a set *result*.

The result of running the above algorithm is the set *result* containing a number of planes, each associated with a set of triangles. To obtain a billboard for each plane, the triangles are orthogonally projected onto the plane, a least bounding rectangle is found to define the billboard quad, and the triangles are rendered into a texture.

3.5.2 Analysis

The biggest advantage of the Stochastic Billboard Cloud Algorithm is its simplicity, and its ability to produce simplifications relatively fast. Given an ϵ value, it might generally produce billboard clouds containing more billboards than the Original Billboard Cloud Algorithm does. That is, unless very high values of S are used, in which case the algorithm, however, is unlikely to outperform the original algorithm in terms of running time.

Lacewell et al. propose the Stochastic Billboard Cloud Algorithm specifically for foliage. They do not, however, argue why the algorithm should be especially efficient for foliage simplification. We judge a stochastic search for high density planes suitable for foliage models, due to the somewhat chaotic positioning and orientation of foliage triangles in many such models.

3.6 Improvements

In this section we present some general improvements that can be applied to billboard cloud algorithms. These improvements apply well to the simplification of polygonal tree models.

3.6.1 View Penalty

In many real-time applications the observer is restricted to certain view angles, in which case a *view penalty scheme* can improve the visual fidelity of the billboard cloud algorithms. An example of such an application is a flight simulation, in which models on the ground, such as tree models, are almost solely viewed from above. We present a very simple view penalty scheme in the following, which applies to error-based construction using the notion of plane density.

A parameter, ϕ_v, in the range $[0; \pi/2]$ specifies an angle between the view vector and the xz-plane (i.e. the ground), and billboards with normals that have similar angles to the xz-plane as the view vector are to be favored. Vertical or horizontal billboards can hence be favored by having $\phi_v = 0$ and $\phi_v = \pi/2$ respectively, for example.

This scheme can be applied in the Original Billboard Cloud Algorithm, or the Stochastic Billboard Cloud Algorithm, by scaling the density value computed for a plane depending on its normal. Let ϕ_n be the angle between the plane normal and the xz-plane. The density of a plane is scaled by: $(\mid \phi_v - \phi_n \mid)/(\pi/2)$.

A potential disadvantage of applying view penalty is that simplifications produced might in general contain more billboards. The reason for this is that it becomes harder to find billboards valid for large amounts of triangles, when certain billboard orientations become penalized.

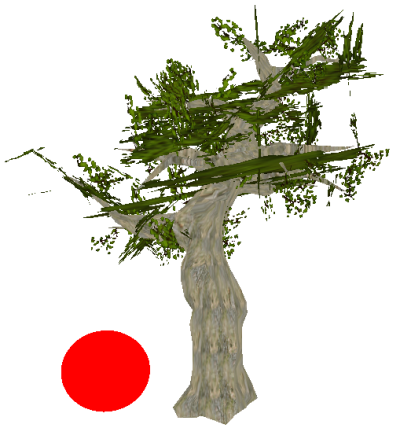
Figure 3.8(b) and Figure 3.8(c) illustrate the effect of applying a view penalty scheme in an implementation of the Stochastic Billboard Cloud Algorithm (the implementation is actually ours, in Section 3.7 we discuss our choice of algorithm).

3.6.2 Increased Foliage Density

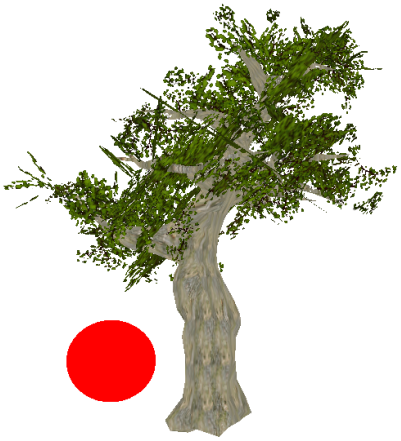
Billboard cloud simplification displaces almost coplanar geometry in order to represent this geometry by a single billboard. Consequently, if a closed mesh model is simplified by a billboard cloud, then a visual artifact in the simplification is often a lack of connectivity, or visual “cracks” as they are often referred to.

Décoret et al. propose a simple technique for crack reduction in [7], which applies to all error-based solutions using the triangle validity term. A triangle might be valid for several of the billboard planes outputted by the Original Billboard Cloud Algorithm. When rendering billboard textures, a triangle can be rendered onto all the billboards for which it is valid, instead of only the billboard chosen for its simplification. This tends to reduce the visual cracks,

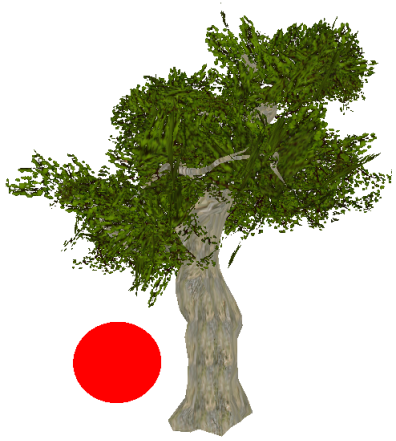
(a) The original tree model.



(b) A billboard cloud simplification created without view penalty. The billboard cloud contains 13 billboards.



(c) A billboard cloud simplification favouring vertical planes. The billboard cloud contains 15 billboards.



(d) A billboard cloud simplification created with view penalty as well as the increased foliage density improvement. The billboard cloud contains 14 billboards.

Figure 3.8: A polygonal tree model and two billboard cloud simplifications created with our own implementation of the Stochastic Billboard Cloud Algorithm. The two simplifications are created with parameters $\epsilon = 6$ and $N = 100$. The different simplifications demonstrate the effect of applying a view penalty scheme and the increased foliage density improvement. Each billboard cloud is illustrated with its associated validity sphere of radius ϵ.

but can unfortunately introduce new visual artifacts, e.g. a model detail, such as an eye in a character model, being represented on several billboards.

Since foliage is modeled using unconnected triangles, crack reduction at first does not seem to be of much relevance when simplifying the foliage part of a polygonal tree model with billboards. However, as noted by Umlauf [28], rendering all valid triangles to each billboard when simplifying foliage tends to improve the visual quality of the simplification.

When performing billboard cloud simplification, the surface area in the simplified model will often appear as if it has been reduced. The reason being that when projecting a triangle onto a plane with normal direction different from the triangle normal direction, the triangle will be represented with a reduced area on the billboard. A further reason for the reduction in surface area is that one triangle might cover another triangle when projecting both triangles onto the same billboard.

This reduction in area tends to make the simplified foliage appear too sparse compared to the original. Rendering all valid triangles to each billboard results in some leaves being represented on several billboards, which in turn makes the foliage appear more dense. We refer to this as the *increased foliage density improvement*. Umlauf et al. present examples in which the visual quality of the simplifications is greatly improved. The results with our own implementation of the increased foliage improvement can be seen in Figure 3.8(d) on the facing page. Notice the increase in foliage density compared to the other simplifications.

It should be noted that the increased foliage density improvement only applies to error-based billboard cloud simplification approaches, because it involves validity. A similar improvement has been made to the budget-based K-means Clustering Billboard Cloud Algorithm. I.-T. Huang et al. present a *crack reduction* scheme, that has the purpose of removing gaps between adjacent billboards, which, in our judgment, may remedy the sparse foliage problem [12]. The crack reduction scheme calculates the convex hull of all the triangles that are simplified by a billboard. If any triangle has a vertex inside the intersection of two such convex hulls, the part of the triangle inside the intersection is rendered to both of the billboard textures.

3.7 Choice of Billboard Cloud Algorithm

To be able to develop and evaluate our solutions for billboard cloud simplification of CASTMs, a billboard cloud algorithm implementation is needed. We have implemented the Stochastic Billboard Cloud Algorithm, as already mentioned when demonstrating the improvements in Section 3.6 on page 53.

We have chosen to implement the Stochastic Billboard Cloud Algorithm instead of one of the other algorithms for several reasons. The algorithm is far more simple to implement than the Original Billboard Cloud Algorithm, and the view penalty and increased foliage improvements are easy to incorporate. Furthermore, the authors claim good results specifically for foliage models, and we do find a stochastic approach suitable for the often seemingly chaotic

distribution of leaves in a foliage.

In Section 8.1.1 we will describe our implementation of the Stochastic Billboard Cloud Algorithm in details.

Summary

The focus in this chapter has been on how to construct billboard clouds. After a survey of data clustering approaches, in which we established the tasks involved in clustering data (e.g. data representation and distance measurement) and discussed hierarchical and partitional clustering algorithms, we discussed billboard cloud construction strategies. Basically, the task of constructing a billboard cloud is a data clustering task, since triangles are clustered by coplanarity.

The terms cost- and error-based data clustering were introduced to billboard cloud simplification, resulting in two different approaches. Following the error-based simplification approach, the problem is to minimize the set of billboards used to simplify the set of triangles from the input model, while respecting a given error threshold, ϵ. Following the budget-based simplification approach, the problem is to minimize the error introduced when simplifying the triangles to billboards, given a budget of k.

Regardless of taking an error- or cost-based billboard cloud simplification approach, it is necessary to measure the quality of the simplification. Two metrics, namely colour error and depth error, were introduced. However, other metrics can be used. We return to simplification metrics in Chapter 5.

Three billboard cloud algorithms have been presented: The Original Billboard Cloud Algorithm, which follows an error-based approach, the K-means Clustering Billboard Cloud Algorithm, which follows a budget-based approach, and the Stochastic Billboard Cloud Algorithm, which also follows an error-based approach. Based on analyses of all three algorithms we have chosen to implement the Stochastic Billboard Cloud Algorithm, among other reasons because it is relatively simple and relatively fast. And because it has been developed specifically for tree foliage simplification.

Some general improvements to billboard cloud algorithms were introduced and discussed. These included a view penalty scheme and increased foliage density. These improvements have been implemented in our implementation of the Stochastic Billboard Cloud Algorithm. Finally, we discussed some of the implementation issues involved in implementing a billboard cloud algorithm.

Chapter 4

Spectral Analysis of Animation

When addressing the problem of simplifying the foliage part of a CASTM with an animated billboard cloud, analysis of foliage triangle animation becomes very important.

Different foliage triangles move independently, and if a set of such triangles are to be simplified by a single billboard, how should this billboard be animated? We propose to analyze the animations of the triangles in order to address this problem. The goal is to define an animation for the billboard that overall represents the average animation of the triangles simplified by it.

Triangles with radically different animations are not suitable for simplification on a single billboard, even if they are somewhat coplanar, as no suitable average animation for such triangles exists. Animation analysis provides information that enables taking animation properly into account, when determining which triangles are suitable for simplification on the same billboard.

The structure of this chapter is as follows. In Section 4.1 we state a number of requirements to an animation analysis technique. This is followed by a discussion of different animation properties of animated foliage in Section 4.2. In Section 4.3 we introduce the notion of a trajectory and define a set of relations between trajectories. Different overall approaches to animation analysis is given in Section 4.4. Spectral analysis and the Fourier Transform is formally introduced in Section 4.5, and the chapter is concluded by a detailed description of how spectral analysis can be used to analyze animation. This happens in Section 4.6.

4.1 Animation Descriptions

We define our animation analysis requirements in terms of *animation descriptions*. An animation description of a triangle can be used to compare its animation with the animation of another triangle, and animation descriptions for a set of triangles can be used to construct an average animation.

Animation Description Type: An *animation description type* is a set of functions which can be used to create and compare descriptions of animation:

- Given an animated triangle, tr, the function $description(tr)$ yields a animation description, d, of t.
- Given two animation descriptions, d_1 and d_2, the function $deviation(d_1, d_2)$ yields a scalar value stating how different the two animations are considered to be.
- Given a set of descriptions, D, the function $average(D)$ yields a new description d_a, which is considered a description of the average animation in D.
- Given a triangle, tr and an animation description, d, the function $apply(tr, d)$ animates tr using d.

The $description$ function can be used to construct animation descriptions for all foliage triangles in the CASTM to be simplified. In order to determine which triangles have similar animations, and can thus agree on a common animation, the $deviation$ function can be used. When a set of triangles are to be simplified by a billboard, a suitable animation for the billboard can be defined by $average$, and applied to it using $apply$.

During this chapter, different animation analysis approaches are considered and evaluated, in the context of using a such to define an animation description type.

4.2 Identification of Properties in Animated Foliage

The animation of a foliage triangle can be considered to have different animation properties, and when simplifying the foliage in a CASTM by an animated billboard cloud, some properties might be preserved while others might not. In this section we identify four essential *animation properties* of the foliage in a CASTM, and discuss which of these we judge important to preserve in a simplification.

Note that when a triangle becomes simplified on a billboard, we compare the animation of the original triangle to the animation of the position on the billboard onto which its center is simplified. This point and its animation will be referred to as the "new" position and animation of the triangle, even though the triangle no longer exists in the simplification.

Animation Time-offset

Triangles simplified on a billboard may become offset in time compared to their original animations. When this is the case, the new positions of the triangles of the foliage may be very different from the original positions of the triangles in individual frames throughout the animation. However, we judge the overall appearance of a time-offset simplified foliage to be visually very similar to the original animated foliage. In other words, we do not judge time-offset an animation property being important to preserve in the simplification.

Generally, for animated models where the movement of parts of the model has an obvious causality with other parts of the model, offsetting subsets of animated triangles in time may be very unfortunate, as this causality can become lost. As an example, if the animation of the upper leg is time-offset compared to the animation of the lower leg of a model of a running human, the skeletal constraints of a human are ignored, and the result will look very unnatural. Furthermore, an animated connected mesh might lose its connectivity if the animation of triangles are time-offset.

We hypothesize that the causality of the movement of foliage in a tree is not obvious to an observer, as the foliage will often hide the underlying branches that cause the movement of the foliage. This observation was already stated in Section 1.5 on page 29. If this observation holds, Time-offsetting a subset of the triangles of a tree model will normally not be detectable by an observer.

Animation Speed and Distance

A triangle moves with a certain speed, and that speed may change when the triangle is simplified with other triangles on a billboard. However, changing the speed of a triangle a small amount may not be detrimental to the appearance of the animated model. The animation speed of the triangles of a tree model suggests the strength of the wind and the rigidity of the branches, as observed in Section 1.5 on page 29. If a lot of triangles increase or decrease their speed, it may look as if the wind strength has changed. Generally, it would be advisable to try to keep the average movement speed in the foliage intact, while changing the speed of individual triangles may not be a problem.

The animation of a foliage triangle in a CASTM is the result of a combination of the movement of several branches with different rigidity and thus different inherent movement frequencies. To be able to handle the movement speed of a foliage triangle correctly it is paramount to be able to identify the dominant frequencies in the movement of a triangle, and ensure that a simplifying billboard moves with similar dominant frequencies.

Similarly to the speed of movement of a triangle, the distance it moves may be changed when the triangle is assigned to a billboard. Also similarly to the speed of movement of a triangle, the distance of movement of a triangle in a tree model suggests wind strength and branch rigidity. The average distance moved by triangles of the foliage model should be somewhat retained when creating billboard clouds, but slight modifications to the distance moved by individual triangles may not be easy to detect by an observer.

Animation speed and distance have been identified as properties of the animation of foliage, and we judge preservation of these properties important to preserve to some degree. Consequently, the simplification should attempt to minimize the alteration in foliage movement speed and distance.

Animation Shape

If the movement of a foliage triangle is largely confined to a plane or a line, or if it moves similarly to a recognizable geometric shape, e.g. an ellipse, main-

taining this shape property when the triangle is simplified may be important. The animation of a foliage triangle in a CASTM is the result of a sequence of rotations around joints in the skeleton. Preserving the underlying structure of the skeleton and thus the causality of the movement of the model when simplifying a set of foliage triangles may be important for the appearance of the simplified model. However, a slight rotation or scaling of the geometric shape in which the triangle moves may be acceptable.

The animation shape property is judged important for the visual fidelity of the foliage simplification, and the general movement shapes of the triangles should be retained if possible.

Animation Orientation

Many animation shapes, such as lines or ellipses, have an obvious orientation in space. For example, a line could be vertical or horizontal. Foliage triangles will often move in elliptical shapes, according to the observations in Section 1.5 on page 29, and ellipses also have recognizable orientation. If the orientation of such an elliptical animation is changed, the animation of the foliage triangle may appear unnatural.

For those animation shapes with an obvious orientation, it is judged important for the visual fidelity of the foliage simplification, that this orientation is maintained. However, slight changes in orientation should not be detectable by an observer.

4.3 Trajectories and Their Relations

To be able to discuss animations unambiguously, we define the notion of a *trajectory*, which describes animation as a position function of time. Furthermore, we will define a set of relations between trajectories. A trajectory can be described by a function $p(t) : \mathbb{R} \rightarrow \mathbb{R}^3$ that for every time t yields a point in space (p_x, p_y, p_z). All trajectories in the following are assumed to be cyclic, which means that for any given cyclic trajectory $p(t)$ there exists a point in time, t_c, such that $p(t \mod t_c) = p(t)$.

A reasonable simplification when analyzing the animation of triangles is to only consider the movement of the triangle center point. Then, the animation of a triangle is represented by a single trajectory, which will be referred to as the *triangle trajectory*.

We define a number of relations between trajectories:

- Two trajectories p and q are *equal*, if and only if (iff) $p(t) = q(t)$ for all t.
- Two trajectories p and q are *parallel*, iff there exists a vector $\mathbf{v}$ such that $p(t) = q(t) + \mathbf{v}$ for all t.
- Two trajectories p and q are *rotated*, iff there exists a rotation matrix R, such that $p(t) = Rq(t)$ for all t.

- Two trajectories p and q are *reflected*, iff there exists a reflection matrix E, such that $p(t) = Eq(t)$ for all t.
- Two trajectories p and q are *congruent*, iff there exists a vector $\mathbf{v}$, a rotation matrix R, and a reflection matrix E such that $p(t) = ERq(t) + \mathbf{v}$ for all t.

Any of the above relations can be relaxed to be *time-offset*, which intuitively means that the trajectories would be part of one of the above listed relations if one of the animations were displaced in time. For example, we say that p and q are *time-offset parallel* iff there exists a vector $\mathbf{v}$ and a real number Δt such that $p(t) = q(t + \Delta t) + \mathbf{v}$ for all t. Of course, the 'time-offset r' relation is always a superset of the r relation, as the r relation is a special case of the 'time-offset r' relation where $\Delta t = 0$.

4.4 Animation Analysis Approaches

Several approaches for analyzing animation exist, but any approach employed to analyze the animation of the foliage part of a CASTM should meet certain requirements.

The properties of animations stated above, time-offset, speed, distance, and shape, describe important aspects of an animation and define to a high degree how the animation appears to an observer. It is hard to predict which properties are the most important for the visual fidelity of the simplified foliage. In general, since we judge the time-offset of triangle animations to be of little visual importance, we want an animation analysis approach to enable us to conclude time-offset equal trajectories to be similar, while minimizing the alteration of the other more visually important animation properties.

We prefer an animation analysis approach that is able to isolate the animation properties, such that changing the animation of a triangle to fit the animation of a billboard is accomplished by changing only a subset of the properties of its animation. If such an approach is found, a simplification procedure can be restricted from violating the properties that are judged essential for visual fidelity.

We discuss two different approaches in the following.

4.4.1 Euclidean Animation Analysis

One approach to performing animation analysis of foliage triangles simplified by the same billboard is based on numerical analysis of trajectory coordinates. Given a sequence of position samples of each triangle taken uniformly over the entire animation cycle, the time values t corresponding to the position samples are denoted frames in the following.

First, the trajectories are mapped to a common origin by subtracting the vector from the origin to the trajectory position at $t = 0$ from all sample positions for each trajectory. Deviation between two trajectories can then be defined as the absolute difference between position coordinates for each sample. To

average a set of trajectories, a naïve approach could be numerically averaging the position coordinates of all trajectories for each frame, hence yielding an average trajectory for the billboard in question.

The deviation measure of this approach is very strict. Trajectories that are not equal or parallel score high deviation values. Even two trajectories that are time-offset equal or time-offset parallel get a high deviation value. An example of the latter is illustrated in Figure 4.1. This is undesirable, as we would like to simplify triangles with, for example, rotated trajectories by the same billboard. Also, when using the naïve approach to compute an average trajectory, trajectories may cancel each other out, e.g. when averaging two reflected trajectories, generally resulting in less movement speed and distance. This is also illustrated by the example trajectories in Figure 4.1.

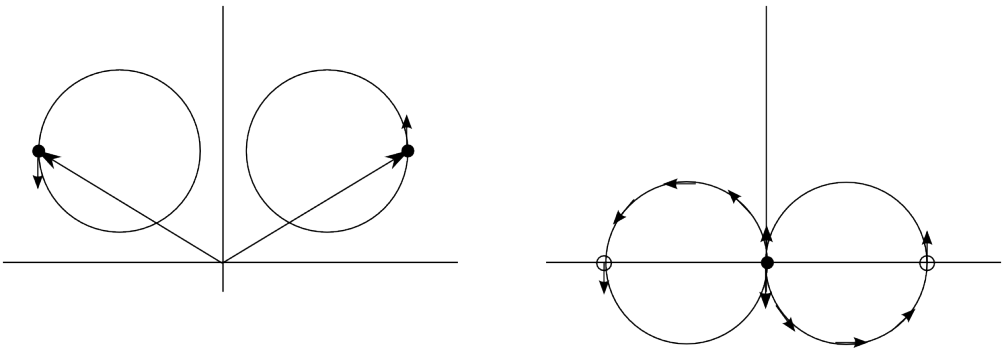


Figure 4.1: An example of two trajectories that are time-offset parallel which yield a high deviation value. To the left the trajectories are mapped to a common origin (the filled dots denote trajectory positions at $t = 0$). To the right it can be seen that the two trajectory positions denoted with non-filled dots yield a high deviation value in that frame. Furthermore, when averaging the trajectories they cancel each other out, hence resulting in a trajectory that yields the same position for every time value. This compromises the distance property.

A suggestion for solving this problem is to handle movement speed and direction separately. Deviation is instead based on two terms, namely difference in direction and difference in speed, and an average trajectory is obtained by averaging each term separately. This strategy also has some problems. For example, it does not guarantee that the average animation is cyclic, i.e. it always starts and ends in the same point.

Other Euclidean animation analysis approaches could be taken. To exemplify, we could represent the movement of an animated foliage triangle by sampling its movement uniformly over its animation cycle, and then approximate the sample points with splines, e.g. cubic Bézier curves [30]. However, we choose to focus on spectral analysis of animation, which is the topic in the following sections.

4.4.2 Spectral Animation Analysis

Another approach to animation analysis is to employ spectral analysis of the animation of foliage triangles. Spectral analysis performs a transformation on

an input signal, i.e. a function of time, resulting in a frequency spectrum that specifies the frequency content of the signal. Spectral analysis can be used for analyzing the animation of triangles, for example by applying it to the three one-dimensional p_x, p_y, and p_z functions of a trajectory. The details on how to apply spectral analysis on animated triangles will be the topic of later sections in this chapter.

When used for animation analysis, spectral analysis effectively isolates the properties of movement speed and movement distance as frequency and amplitude. Furthermore, a spectrum contains phase information that implicitly yields time-offset properties. When two trajectories are time-offset parallel or time-offset reflected, their spectra will be equal, except for the phase information. This separation of properties enables discarding phase information, and hence correctly conclude two time-offset animations to be visually similar and have low deviation. However, problems arise, when discarding phase, as will be discussed in Section 4.6.

Determining the deviation of two animated triangles involves comparing their frequency spectra. The average animation of a set of animated triangles is similarly obtained by averaging their spectral content. Deviation and average definitions based on spectral analysis will be discussed in Section 4.6.2.

Spectral analysis has another nice property when used for analyzing the animation of foliage triangles in a CASTM. As noted in the observations of real trees in Section 1.5 on page 29, the large branches close to the trunk move slowly due to their rigidity, while the small branches far from the trunk are more flexible and hence move faster. It is further noted that leaves in several common tree species tend to be positioned in the outer levels, and their animation must be the result of a sum of several branch motions of different frequencies. Assuming that these features are also present in the given CASTM to be simplified, then the frequencies of the different branches will be explicitly represented in the spectra obtained by analysis of animated foliage triangles.

To be able to explain in details how we apply spectral analysis to animation, the general concept of spectral analysis is summarized in the following section.

4.5 Spectral Analysis

In the following, the Fourier Transform of a continuous function is introduced. However, a discontinuous input function is more practical for our purposes, and to be able to analyze a discontinuous function, the Discrete Fourier Transform is introduced. The following is based on Smith's book [25].

4.5.1 The Fourier Transform

The Fourier theorem states that any periodic function $x(t)$, may be expressed as the sum of a series of sinusoids of different frequencies. A sinusoid is a function of the form:

$$s(t) = A\sin(\omega t + \varphi),$$

where A is amplitude, φ is phase, and ω is frequency. A series of sinusoids may be expressed as a function that for a given frequency ω yields an amplitude A and an initial phase value φ:

$$X(\omega) = \langle A, \varphi \rangle$$

Such a function is called a *frequency spectrum*. In the following, we will simply use the term spectrum meaning frequency spectrum.

Complex sinusoids are like real-valued sinusoids, with an imaginary part in phase-quadrature with the real part, meaning that the phase of the real and imaginary parts always differ by $\frac{\pi}{2}$. A complex sinusoid with amplitude A, frequency ω, and initial phase φ is given by:

$$s(t) = A\cos(\omega t + \varphi) + Ai\sin(\omega t + \varphi),$$

where $i = \sqrt{-1}$. According to Eulers Identity $e^{i\theta} = \cos\theta + i\sin\theta$, this term can be simplified to:

$$s(t) = Ae^{i(\omega t + \varphi)}$$

Generally speaking, the Fourier transform maps an arbitrary function to a function which is composed of a sum of sinusoids. If the input function is $x(t)$, we can for any frequency ω find the amplitude and initial phase of the sinusoid with frequency ω present in $x(t)$. The Fourier transform is given by:

$$X(\omega) = \int_{-\infty}^{\infty} x(t)e^{-i\omega t}, \quad X : \mathbb{R} \rightharpoonup \mathbb{C},$$

where $e^{-i\omega t}$ represents a complex sinusoid with frequency ω. From $X(\omega) = x + iy$ we can calculate the amplitude A and phase φ of the sinusoid with frequency ω present in $x(t)$:

$$\begin{aligned} A(\omega) &= |\, X(\omega)\,| = \sqrt{x^2 + y^2} \\ \varphi(\omega) &= \tan^{-1}\frac{y}{x} \end{aligned}$$

The Fourier transform is invertible and thus the original signal can be reconstructed from the sum of sinusoids. This operation is called the Inverse Fourier transform.

4.5.2 The Discrete Fourier Transform

As the Fourier transform is impossible to evaluate in the general case, practical spectral analysis employs the Discrete Fourier Transform (DFT).

The input to the DFT is a uniformly sampled signal with N samples, represented by a complex vector $\mathbf{x} \in \mathbb{C}^N$ (the imaginary part is zero). When the DFT is computed, N different complex sinusoids of different frequencies are considered. These sinusoids, called the DFT sinusoids s_k, are sampled uniformly N times, yielding the function:

$$s_k(n) \quad , \quad k, n = 0, 1, ..., N-1,$$

where k is the index of the frequency, and n is the sample index.

The output of the DFT is a function representing a discrete frequency spectrum $\mathbf{X}(\omega) : \mathbb{R} \rightharpoonup \mathbb{C}$, from which we can calculate the amplitude and phase of each of the N complex sinusoids $s_k(n)$. The complex sinusoids with the determined amplitudes $\mid \mathbf{X}(\omega) \mid$ can be summed to generate the original signal $\mathbf{x}$. Thus, the result $\mathbf{X}$ determines the frequency content of the original sampled signal.

The sinusoid $s_k(n)$ has the frequency:

$$\omega_k = \frac{k}{N} 2\pi f_s = \frac{k}{N} \frac{2\pi}{T},$$

where f_s is the sampling frequency of the input signal x, i.e. the reciprocal of the time T between each sample of the signal, meaning that $f_s = \frac{1}{T}$. Each sample $x(n)$ is sampled at the time $t_n = nT$. Thus, the frequencies of the sinusoids are evenly distributed from 0 to $\frac{N-1}{N}\frac{2\pi}{T} \approx \frac{2\pi}{T}$.

The amplitude of each complex sinusoid is assumed to be 1 and the initial phase assumed to be 0, which yields

$$s_k(t) = e^{i(\omega_k t)}$$

and for the time t_n it is

$$\begin{aligned} s_k(t_n) &= e^{i(\omega_k nT)} \\ &= e^{i2\pi \frac{k}{N} f_s nT} \\ &= e^{i2\pi \frac{k}{NT} nT} \\ &= e^{i\frac{2\pi kn}{N}} \end{aligned}$$

The Discrete Fourier Transform is a mapping of the signal into the frequency domain, which is an N-dimensional Hilbert space (a vector space that is closed under addition and scalar multiplication and has a well-defined inner product operator) with the sampled sinusoid vectors as basis set. This mapping is performed as the scalar product of the sampled signal $\mathbf{x}$ vector and the conjugate of the sampled complex sinusoids $\overline{\mathbf{s_k}}$ (the conjugate inverts the imaginary part) for each frequency ω_k:

$$X(\omega_k) = \langle \mathbf{x}, \bar{\mathbf{s}} \rangle = \sum_{n=0}^{N-1} x(t_n) \overline{s_k(t_n)} = \sum_{n=0}^{N-1} x(t_n) e^{-i\frac{2\pi kn}{N}}, \quad k = 0, 1, 2, \ldots, N-1$$

The amplitude and phase of the complex sinusoids present in the signal may be computed as the magnitude of the DFT for the given frequency, as shown in Section 4.5.1 on page 63.

If we disregard phase, the intuition behind the DFT may be explained as follows: If a sampled signal represents a sinusoid of a given frequency, the

scalar product with a signal of the same frequency yields a positive value representing the amplitude, in contrast to the scalar product with a signal of a different frequency, which given enough cycles of the two sinusoids will cancel each other out and yield 0.

The DFT is easily implemented, but the unmodified DFT algorithm performs in $O(N^2)$ time. A more efficient variant of the algorithm, the Fast Fourier Transform (FFT), uses a divide-and-conquer approach and performs in $O(N \log N)$ time. Efficient implementations of the FFT exist as open-source libraries that may be employed in an application.

4.6 Spectral Animation Analysis in Detail

The spectral animation analysis approach depends upon the ability to map triangle trajectories to the frequency domain. Let $p(t) = (p_x(t), p_y(t), p_z(t))$ denote a triangle trajectory. A such can be mapped into the frequency domain by computing the DFT on each of the coordinate functions, $p_x(t), p_y(t)$, and $p_z(t)$. As the animation of a foliage triangle in a CASTM is cyclic, these coordinate functions will be periodic. Thus, applying the DFT computes the frequency content and phase information of the movement along each axis. The three spectra computed can be transformed back to the original animation by computing the inverse DFT of each spectrum and applying the resulting motion along the axis corresponding to the spectrum.

4.6.1 Trajectories and Spectra

Figure 4.2 illustrates an example two dimensional trajectory, $p(t)$, given by:

$$\begin{bmatrix} p_x(t) & = & 10 \cdot \sin(t + \pi/2) \\ p_y(t) & = & 10 \cdot \sin(t) + sin(30t) \end{bmatrix}$$

If the DFT is applied on each of the trajectory coordinate functions, it yields the two axis spectra, $X_x(\omega)$ and $X_y(\omega)$ shown in Figure 4.3 on the next page.

In the spectrum for the x-axis movement it can be seen that the animation contains a single frequency in this axis, while the spectrum for the y-axis contains two frequencies. The low frequency present in both spectra defines the overall circular shape of the trajectory, whereas the high frequency with low amplitude present in the y-axis spectrum prevents the trajectory from being a perfect circle. Furthermore, each frequency in the spectra has an initial phase value, which is not illustrated, but the frequency in the x-axis spectrum has an initial phase value of $\pi/2$, while both frequencies in the y-axis spectrum has an initial phase value of zero.

Visually, the trajectory appears to be moving slowly in a circle, while also performing a smaller fast movement along the y-axis. Hence, the animation contains both slow and fast movement, which, as expected can be identified as low and high frequencies in the spectra. Likewise, the relatively small distance property of the fast movement can be identified in the spectrum for the y-axis as a small amplitude. In Section 4.4.2 on page 62 we stated that spectral analysis

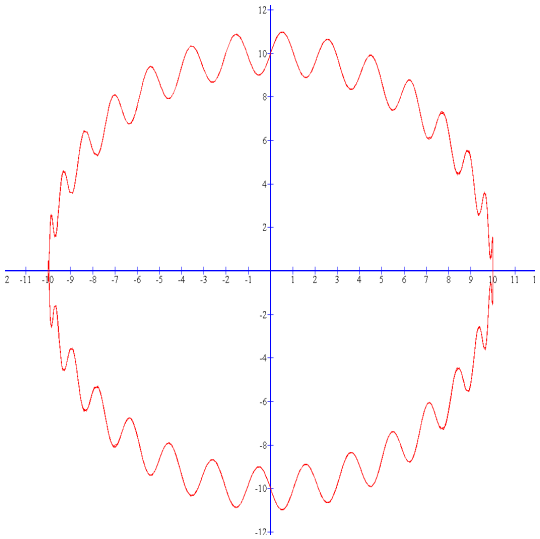
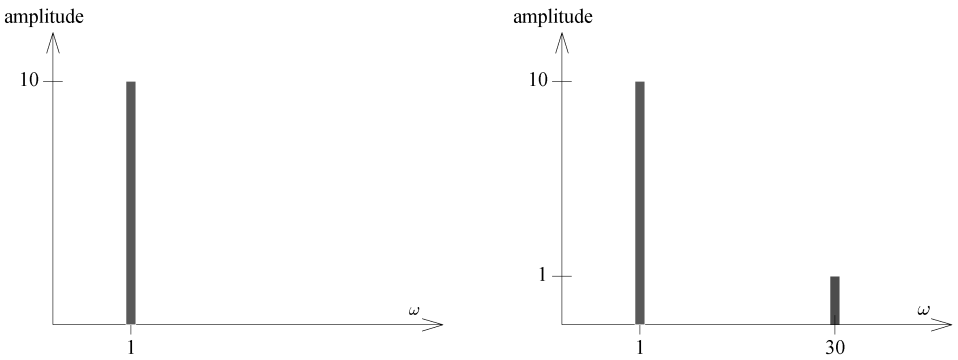


Figure 4.2: Example trajectory.



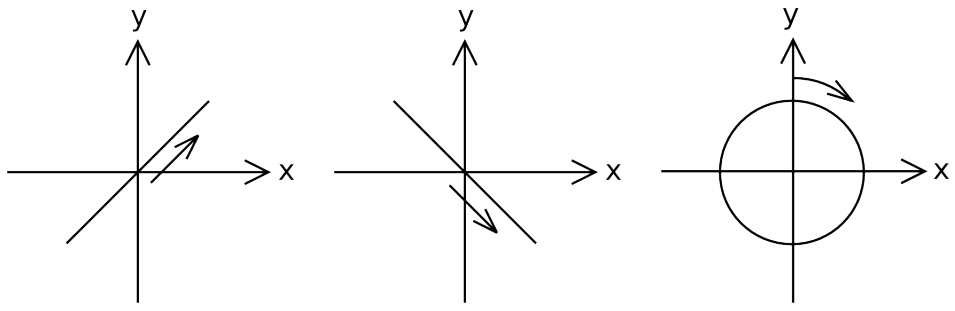
(a) $X_x(\omega)$. (b) $X_y(\omega)$.

Figure 4.3: Spectra for the x- and y-axes of the trajectory shown in Figure 4.2. The frequency ω is measured in periods per signal length.

of animation effectively isolates the animation properties speed and distance as frequency and amplitude, which is illustrated by this example.

The trajectory of a foliage triangle in a CASTM is typically defined by rotation of large branches close to the tree trunk, and rotation of smaller branches further from the trunk. Assuming that the large branches rotate slowly due to their rigidity, and that the smaller flexible branches rotate faster, the different branches will be represented in the axis spectra at different frequencies.

The initial phase content of the axis spectra influence the time-offset of trajectories, and two time-offset equal trajectories will have equal spectra when disregarding phase values. However, initial phase also influences the shape property of an animation. Figure 4.4 illustrates this in two dimensions. The three trajectories illustrated all contain the same amplitude in the x- and y-axes for some frequency. However, the initial y-phase, φ_y, is different in the three trajectories, which results in the different shapes illustrated.



(a) $\varphi_x = \varphi_y = 0$. (b) $\varphi_x = 0; \varphi_y = \pi$. (c) $\varphi_x = 0; \varphi_y = \frac{\pi}{2}$.

Figure 4.4: Three trajectories with the same amplitude in the x- and y-axes for a single frequency, but different initial y-phase values.

It is not trivial to interpret the visual effect of initial phase content in the axis spectra, as it defines both time-offset and shape; properties that should somehow be separated.

4.6.2 Spectral Descriptions

An animation description type can be created based on spectral analysis of animation. An animation description of a triangle created using such an approach is referred to as a *spectral description*. A straightforward spectral description is obtained by applying the DFT on the trajectory coordinate functions for a triangle to yield three axis spectra. These spectra constitute a spectral description of the animation of a triangle.

The $deviation$ function of a description type is used to conclude which triangles have similar animations, and hence is suitable for sharing a common animation. Two spectral descriptions can be compared by comparing the spectra for each coordinate axis. A naïve approach could be computing the sum of the difference between amplitudes for all frequencies and the sum of the difference between initial phase values for all frequencies, and multiplying the two sums.

$average$ defines an average animation of a set of spectral descriptions, and is used to define a suitable animation for an animated billboard simplifying a set of triangles. A naïve approach to averaging spectral descriptions is simply to average the amplitude and phase content at each frequency in the axis spectra.

Finally, a spectral description can be used to define an animation of a triangle by using the inverse DFT on each axis spectrum to obtain a triangle trajectory, which is used by $apply$ to animate a billboard with the average animation of its simplified triangles.

Summary

The concept of animation description types were defined in this chapter, as a formalization of what we need from an animation description approach. Different properties of animation have been identified, and it has been discussed which of these properties are judged important to preserve in a simplification. The Fourier Transform has briefly been introduced, and is the basis of spectral analysis of animation. How the Fourier Transform can be applied to the trajectory of a triangle to yield an animation has been discussed, and it is argued why spectral analysis separates the animation speed and distance properties from the other animation properties.

Our goal is to design solutions to the problem of simplifying the foliage part of a CASTM using an animated billboard cloud. One of our solutions, Spectral Clustering, is based on spectral analysis of animation, and makes direct use of spectral animation description types. The approaches for computing deviation and average presented in this chapter are not necessarily suitable for determining which triangles should share billboards, or determining how to animate billboards. In Chapter 6 the Spectral Clustering solution is designed, and in the process, several more advanced approaches to defining spectral descriptions are discussed. The different spectral animation description types proposed are related to the animation properties in order to evaluate their suitability for our purposes.

Chapter 5

Animated Billboard Cloud Construction

In Chapter 3 we presented billboard cloud construction strategies. We discussed error- and budget-based billboard cloud algorithms, and how the quality of the simplification could be measured in terms of cost and error. Cost was defined as the complexity in rendering the billboard cloud model, and error was defined from different metrics stating where the triangles are positioned; ϵ does it in Euclidean space, while the colour and depth error metrics do it in image space.

In this chapter we present animated billboard cloud construction strategies. Given a CASTM our goal is to output an animated billboard cloud LOD version of its foliage, that retains visual fidelity to the foliage in the input model. But what does ”visual fidelity” mean in this context? We define and discuss a set of objective measures for visual fidelity of an animated billboard cloud simplification in Section 5.1. Cost and error terms for animated billboard cloud construction are then discussed, followed by a presentation of three different strategies for performing animated billboard cloud simplification of the foliage part of CASTMs with high visual fidelity. This happens in Section 5.2. These strategies will serve as the basis for our solutions, which are presented in the chapters to follow.

5.1 Error Metrics for Animated Billboard Clouds

To be able to design algorithms for automated billboard cloud simplification of CASTMs, we need to be able to formally state the goals of such algorithms. We do so in terms of error metrics, which can be used to measure the quality of the animated billboard cloud simplification of a CASTM, and a set of such will be presented in this section.

5.1.1 Euclidean Distance Metrics

When an animated foliage triangle is simplified with an animated billboard, the distance between the actual position of the triangle in the input CASTM and its position on the billboard clearly has impact on the visual fidelity of the

simplification. Note that this distance is variable during the animation cycle and should therefore be analyzed over the key frames, e.g. by finding the average or maximum distance.

When simplifying a static triangle with a static billboard, the distance from the actual triangle position to its simplified position is equal to the distance from the triangle to the billboard. For animated triangles and billboards this is not the case, as the triangle is simplified to a single position on the billboard, even though its projected position on the billboard might vary during the animation. Some animation simplification schemes may handle one aspect of simplified displacement better than others, and to be able to separate the distance to the billboard from the variation in position in the billboard plane, distance is measured in two different values:

Plane distance : The shortest distance from a foliage triangle to its simplifying billboard, D_{plane}.

Projected distance : The distance between the simplified position of a foliage triangle and the projected position of the foliage triangle, $D_{\text{projected}}$.

Figure 5.1 illustrates plane distance and projected distance.

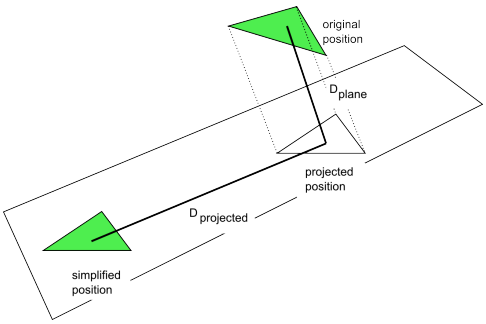


Figure 5.1: Plane distance D_{plane} and projected distance $D_{\text{projected}}$.

The Euclidean vertex displacement, the actual distance between the original position and the simplified position, can be computed by

$$D_{\text{euclidean}} = \sqrt{D^2_{\text{plane}} + D^2_{\text{projected}}}.$$

The distance can be any value larger than 0. The metric is normalized to a value between 0 and 1 using the *normalized error function* as described in Appendix A.1, where 0 signifies no distance and 1 signifies infinite distance. Thus, we define:

Plane distance metric:

$$NE(D_{\text{plane}})$$

Similarly, the projected distance metric is defined as:

Projected distance metric:

$$NE(D_{\text{projected}})$$

5.1.2 Orientation Metric

Difference in orientation has an impact on the visual fidelity when considering billboard simplification of both static and animated polygonal models. If the normal of a static foliage triangle is different from the normal of the static billboard onto which it is simplified, then the foliage triangle is not correctly represented on the static billboard.

Even though the normal of a simplified triangle is aligned with the normal of the original triangle, the orientation may still be incorrect, as the simplified triangle may be rotated around the axis of the normal. To capture this difference in direction, we assume that vertices in the foliage triangles are indexed and define the direction of a foliage triangle to be the vector from the centroid of the triangle to the vertex with index 0. The simplified direction can then be compared to the original direction.

Assuming that the normal of the original triangle $\mathbf{n}$ and the simplified triangle $\mathbf{n}_s$ are normalized, and that the original triangle direction $\mathbf{d}$ and the simplified triangle direction $\mathbf{d}_s$ are normalized, the difference in orientation can be expressed as a number between 0 and 1, where 0 denotes no difference in orientation and 1 denotes maximum difference in orientation (maximum difference in orientation is obtained when $\mathbf{n} \cdot \mathbf{n}_s = -1$ and $\mathbf{d} \cdot \mathbf{d}_s = -1$). In the following definition, recall that the dot product of two normalized vectors yields a number between -1 and 1. We define the orientation metric as:

Orientation metric:

$$1 - \frac{(\mathbf{n} \cdot \mathbf{n}_s + 1)(\mathbf{d} \cdot \mathbf{d}_s + 1)}{4}$$

If the texture of the foliage triangle is symmetric around the axis coincident with $\mathbf{d}$, the normal may be inverted without losing visual fidelity.

Similar to the distance error metrics the difference in orientation varies during the animation cycle, and should therefore be averaged over a number of key frames, for example.

5.1.3 Colour Metric

The colour metric approximates the visual fidelity of the entire simplification of a CASTM, focusing only on rendered pixels.

When rendering each billboard, the billboard is the exact image of the simplified foliage geometry as seen from a particular view direction in a particular frame, perspective and aliasing issues aside. However, when the simplified model is rendered from an arbitrary direction in a frame, the billboards will not be view-aligned in general, and the resulting rendering of the foliage may look “sparser” than the original foliage of the input CASTM. Furthermore, the rendered geometry has been displaced to be rendered on the billboards, and thus the resulting animated billboard cloud foliage may have a different apperance than the original foliage. Although the world space displacement of each triangle already is captured by the Euclidean distance metric, another approach

for evaluating the result is to view the difference in the rendered pixels of the original and simplified models.

An issue that needs attention is that the original and simplified models may be observed from any direction in a frame. To address this issue when computing the colour metric, the original and simplified models must be compared from a finite set of view directions. Certain view directions are often preferred in certain applications, e.g. in a first-person application, the simplified model will normally be viewed from the side and not from the top (or bottom). In such an application, the viewing directions will be limited to those starting from a circle around the tree model and pointing towards the model.

Recall that the animated billboard cloud model will be used in a LOD scheme at a specific distance interval. Given a screen resolution, it is assumed that an average resolution $w \times h$ of the rendered image can be computed, yielding $P = w \cdot h$ pixels.

To avoid aliasing issues influencing the colour metric too much, the colour metric only evaluates to which pixels in the rendered image foliage triangles have been rendered and to which pixels no foliage triangle has been rendered. In the case of the simplified model, the pixels that have been rendered to include all the pixels where not all the rendered billboards have been transparent at that particular pixel. Thus, an image rendered from a given view direction d can be expressed as a vector $\mathbf{i}$, where $\mathbf{i}_p \in 0, 1$ for $p \in 0..P$. $\mathbf{i}_p = 1$ signifies that something has been rendered at pixel p, and $\mathbf{i}_p = 0$ signifies that nothing has been rendered at pixel p. Thus, given a set of D view directions and a set F of uniformly sampled frames, an image vector $\mathbf{i_d}$ can be computed for each view direction, d, and sample frame, f.

To evaluate the colour metric, an image vector for the original model, $\mathbf{o}_d$, and the simplified model, $\mathbf{s}_d$, is computed for each view direction d and sample frame f. A view direction colour error is computed for each view direction as the number of pixel differences divided by the number of pixels with rendered foliage, P_f, in the rendering of the input CASTM. Thus, the colour metric relates the number of pixel differences with the number of pixels actually rendered, thus obtaining the fraction of the pixels rendered, that are different. The colour metric can then be computed as the average colour error for all sample frames, for all view directions. If the metric is 0, it signifies no difference in pixels rendered from any angle. Principally, the metric can yield values larger than 1 (e.g. all other pixels are rendered to when rendering the animated billboard cloud, than those rendered to when rendering the CASTM). If this happens, the animated billboard cloud model is judged to have no low visual fidelity to the CASTM. Hence, all values larger than 1 are discarded.

Colour Metric :

$$\frac{\sum_d^D \sum_f^F \sum_p^P \mid o_{d,f,p} - s_{d,f,p} \mid}{P_f \cdot F \cdot D}$$

5.1.4 Animation Metrics

In this section we define some animation specific metrics, which can be used to compare different animation properties of the foliage part of a CASTM and its simplifying animated billboard cloud (recall the animation properties from Section 4.2 on page 58).

If the plane distance metric and projected distance metric, as well as the orientation metric, yield values close to zero during the entire animation, it would imply that the CASTM is simplified by an animated billboard cloud of high visual fidelity, which preserves all animation properties. These metrics are thus already related to the animation properties, so why introduce further animation specific metrics? For the foliage part of a given CASTM it might be impossible to simplify it to a specific number of billboards, unless relatively large values for plane- and projected distance error and orientation error are introduced. In this case, however, the simplification could still attempt to preserve some animation properties of the CASTM, e.g. the overall animation distance and speed. Hence, we introduce some metrics specific for animation.

The metrics we introduce are based on spectral analysis and attempts to capture the animation orientation property, as well as the properties distance and speed. In order to express these properties for a triangle trajectory, a local axis of its movement is computed. It is important to emphasize that these metrics represent one way of evaluating the visual fidelity in regards to animation from a spectral-based approach. Other, possibly better, spectral-based animation metrics might exist.

Given a trajectory and some axis, the trajectory can be approximated as a one-dimensional animation on this axis, by mapping the trajectory onto it. If the trajectory is sampled, then each sample is mapped onto the closest point on the axis. A centroid for the trajectory is found on the axis, which will be its origin. Also, a direction along the axis is picked as the positive movement direction; which of the two directions is picked does not matter. We can define a scalar function $x(t)$ that describes the movement of the triangle along the axis to time t, and hence the new one-dimensional triangle trajectory.

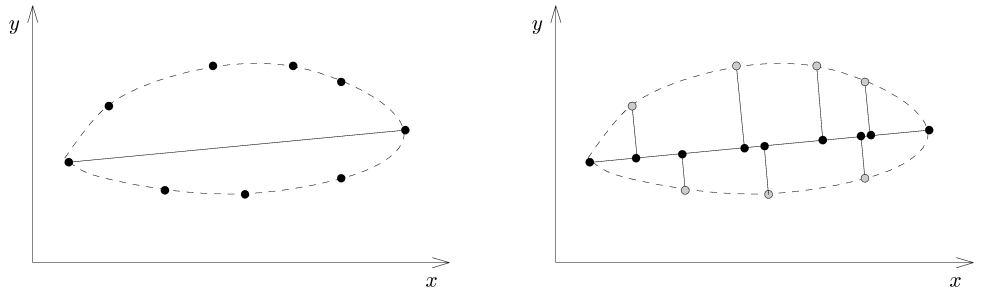
We define the *dominating movement axis* as:

Dominating movement axis : The dominating movement axis of a triangle trajectory is the axis that has the largest distance between two samples of the trajectory, when the trajectory is projected onto this axis.

The dominating movement axis of a trajectory, as well as the new trajectory obtained by mapping the original onto the dominating movement axis, are illustrated in Figure 5.2.

If a trajectory is sampled using relatively few samples, its dominating movement axis may be found by performing a brute-force search of all pairs of samples in the trajectory for the pair of samples with the largest distance between them. The axis through both samples is the dominating movement axis.

The animation metric simplifies animation analysis to only consider a single axis. This is an obvious approximation of the actual animation, and by



(a) The dots represent the position samples of the trajectory represented by the dashed curve. The dominating movement axis is represented by a solid line.

(b) The solid dots represent position samples of the new trajectory obtained by mapping the original trajectory onto the dominating movement axis.

Figure 5.2: The dominating movement axis of a trajectory, and how the trajectory is mapped onto the axis to yield a new trajectory.

choosing the dominating movement axis we seek to minimize the error in this approximation.

By applying DFT on the simplified animation $x(t)$, obtained by mapping a triangle trajectory onto the dominating movement axis, a frequency spectrum is obtained. The intuition behind the animation metric is, to compare the content of this spectrum created for a triangle trajectory in the original animated foliage, and the trajectory of this triangle in the simplification (which, in fact, is a trajectory of the point on a billboard onto which the triangle center is simplified).

The metric based on dominating movement axis animation is as follows. Let $\mathbf{a_t}$ and $\mathbf{a_b}$ be two normalized vectors pointing in the direction of the dominating movement axis of the foliage trajectory and of the trajectory of its simplified position, respectively. If the angle between $\mathbf{a_t}$ and $\mathbf{a_b}$ is greater than 90 degrees, i.e. $\mathbf{a_t} \cdot \mathbf{a_b} < 0$, we invert one of them to obtain the difference in orientation between the axes represented by the vectors. Let $X_t(\omega)$ and $X_b(\omega)$ be the spectra obtained by applying DFT on the dominating movement axis animation of the original triangle, and on the dominating movement axis animation of its simplified position on a billboard, respectively. The dominating axis metric is then:

Dominating Axis Metric :

$$(1 - \mathbf{a_t} \cdot \mathbf{a_b}) NE(amplitudeDistance(X_t(\omega), X_b(\omega))),$$

where *amplitudeDistance* is a function that yields the sum of amplitude difference for each frequency in two spectra, and *NE* is a normalization function, normalizing this sum to the range $[0; 1]$ (see Normalized Error in Appendix A on page 155).

If the metric yields a value of 0, then the dominating movement axes have the same orientation, and the amplitude content in the two dominating axes spectra is likewise equal. The closer the value yielded by the metric is to 1, the

more different the two animations are estimated to be. The metric indicates whether the triangle and its simplified position on a billboard moves in a similar direction, with a similar speed (i.e. frequency), and with similar movement distances (i.e. amplitude) in that direction. In regards to the trajectory relations from Section 4.3 on page 60 the dominating movement axis metric will yield a score value of 0 for the trajectories of the triangle and its simplified position on the billboard in the equal, parallel, and reflected relations (and in the time-offset versions of these relations). On the other hand it will yield high scores in the rotation relation. and in deviation in the amplitude content in the two dominating movement axes spectra.

It might be hard to achieve a low cost billboard cloud that scores a low error value in the dominating axis metric, since only foliage triangles with animation in similar directions can share a billboard. Therefore, we introduce a weaker animation metric, comparing the dominating movement axis spectra only:

Dominating Axis Spectral Metric :

$$NE(amplitudeDistance(X_t(\omega), X_b(\omega)))$$

This metric will yield a score value of zero in the same trajectory relations as the dominating movement axis metric, with the addition of the rotation relation, implying that this metric yields a score value of zero for congruent relations.

5.1.5 Discussion of The Metrics

If the Euclidean distance and orientation metrics are close to 0 in every frame, the simplification should exhibit high visual fidelity throughout the animation. However, achieving this for a low number of billboards may be intractable. Demanding that every foliage triangle is simplified close to its original position may not be paramount, as the triangles are very similar in appearance, and they could exchange positions without any apparent difference in visual fidelity.

The orientation metric may also be less important for visual fidelity, as individual leaves normally will be small and a difference in orientation will be indistinguishable in any but the closest positioned renderings.

Unlike the Euclidean distance metrics, the colour metric ignores triangles that have exchanged positions in the simplified model, based on the intuition that different foliage triangles in foliage cannot be distinguished by the observer. Intuitively, if the colour metric is close to 0 in a given frame of the animation, the simplification should exhibit high visual fidelity in that frame. Evaluating the colour metric alone, however, ignores the relationship between consecutive frames in the animation. Clusters of triangles or billboards that move with a common animation are recognized by an observer and suggest a coherent structure (i.e. a common branch in a tree model).

The animation metrics capture the property of common animations, even though they only consider animation on the dominating movement axis. The distance and colour metrics state where the foliage triangles are positioned (in

Euclidean space and image space, respectively), while the animation metrics state how the triangles are animated. The animation metrics do not take the exact position into account. As an example, the dominating movement axis metric error of a triangle with a trajectory that is parallel to the trajectory of its simplified position is 0. The animation metric that ignores the dominating movement axis direction may be used, if it is deemed important that the tree exhibits the same amount and speed of movement, but not necessarily in the same axes. If a simplified model is viewed from afar, it may only be important to have the same amount and speed of movement, as the movement direction may be indistinguishable.

The colour metric states where foliage triangles are positioned in image space. Similarly, it could be interesting to devise an image-based animation metric. This will not be discussed further, though.

If a simplification yields low values with the colour metric as well as an animation metric, it suggests visual fidelity in every frame and coherent animation structures. Together, this should yield an overall high visual fidelity.

5.2 Animated Billboard Cloud Construction Strategies

We have devised three different strategies for doing animated billboard cloud simplification of CASTMs. We refer to them as strategies, as several different solutions following one of our strategies can be designed. A strategy makes it possible to simplify a model to a variable number of billboards, such that animated billboard clouds suitable for different LODs can be produced.

In this section we will present our strategies and explain the steps involved in each of them. First, however, we discuss cost and error terms in relation to animated billboard clouds, as well as budget- and error-based approaches to the construction of such.

It is important to note that our three strategies do not represent the definitive approaches. Other (possibly better) strategies could exist.

5.2.1 Quality of Simplification

Recall the cost and error terms introduced in Section 3.2.2 on page 43 for static billboard clouds. We define the cost of an animated billboard cloud to be in correspondence with its rendering complexity, which is similar to the rendering complexity of a static billboard cloud, with the addition of the cost of performing the animation of each billboard. Cost can be objectively expressed in terms of number of billboards, size of billboards and textures, as well as the cost associated with animation, such as the number of bones. Error is still defined in terms of the visual error introduced by replacing the foliage part of a CASTM with an animated billboard cloud LOD model. Different metrics have been defined for objectively measuring error in Section 5.1 on page 70.

The budget- and error-based approaches introduced in Section 3.2.3 on page 43 to construct a billboard cloud for a static polygonal model make similar sense when considering simplification of the foliage of a CASTM using an

animated billboard cloud. An example of an error-based approach is to cluster foliage triangles based on a maximum allowed plane and projected distance metric errors, and then seeking to minimize the cost in terms of number of animated billboards, while respecting these error thresholds. An example of a budget-based approach is to specify maximum allowed cost (budget) in terms of number of animated billboards, and then cluster triangles while seeking to minimize the plane and projected distance metric errors, as well as the dominating axis metric error, and still respect the budget.

In the following the devised strategies for simplifying the foliage part of a CASTM using an animated billboard cloud will be presented.

5.2.2 Static Co-planar Clustering

What if we just consider how the foliage geometry of the input CASTM is positioned in one pose? We could then apply a billboard cloud algorithm on this static foliage and subsequently animate each billboard in a fashion that satisfies the animations of the foliage triangles simplified by it. This could seem an intuitive approach to create the animated billboard cloud for the foliage, and it is the idea behind this solution strategy. To put it in steps, we:

1. Create a static pose of the animated foliage in the input CASTM.
2. Create a static billboard cloud for the static foliage geometry using a traditional billboard cloud algorithm.
3. Define the animation of each billboard.

Creating a static pose of the animated foliage can be done in different ways. To exemplify, we can either choose the pose from a keyframe, e.g. the first or the last keyframe, or we can calculate an average position of each triangle over the animation cycle.

Once we have created a static pose of the animated foliage, we create a static billboard cloud of the foliage using a traditional billboard cloud algorithm, e.g. the Original Billboard Cloud Algorithm or the Stochastic Billboard Cloud Algorithm. The number of billboards produced can be adjusted indirectly if resorting to an error-based billboard cloud algorithm is used, or directly if using a budget-based billboard cloud algorithm.

Finally, the animation of each billboard must be defined. From some analysis of how the triangles move, the billboard should be animated in a way that attempts to imitate the animation of as many of the triangles it simplifies as possible. However, it may not be possible to find a common animation with high visual fidelity for a set of triangles simplified by the same billboard. Since triangles have been clustered (i.e. simplified onto billboards) based on co-planarity in one pose alone, it is not guaranteed that triangles in a given cluster have similar animation behaviour. As an example, they could be in counter phase, or move in entirely different shapes. It is thus not trivial to determine a suitable animation of the billboards.

The success of this strategy depends upon the animation definition step, i.e. analyzing the animation of triangles and defining a common animation for all triangles. As hinted, this might prove to be a fundamental problem with this strategy.

Basically, the fact that we disregard the animation of the foliage when clustering the leaf triangles seems to be a bad approach, at least if the animations of the billboards should be faithful to the original animation of the foliage when observed at close range. On the other hand, following this strategy we might be able to produce convincing animation simplification of the foliage in a model placed on a far-away distance from the observer.

If we want to perform animation simplification of high visual fidelity of the foliage part of CASTMs, a much more reasonable approach might be to take the animation of the foliage into consideration when determining which triangles should be simplified by a billboard. This constitutes the basis for the other two strategies.

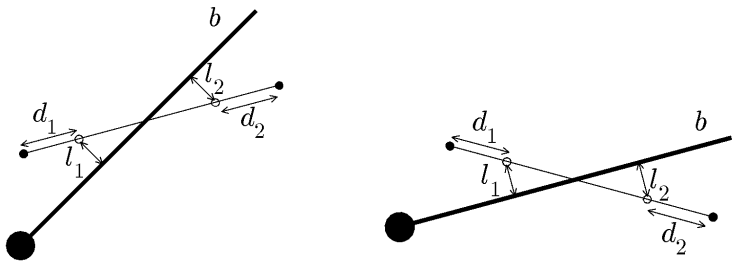
5.2.3 Bone Clustering

The idea behind the Bone Clustering strategy is based on the observation that all geometry attached to a single bone share the same bone rotation animation. Hence, the position of each triangle relatively to the other triangles attached to this bone is static. Therefore, a non-animated billboard cloud can be created for this static geometry, and the billboard cloud is then animated simply by attaching it to the bone. This is illustrated in Figure 5.3(a) and Figure 5.3(b). It should be noted that the more a triangle is displaced from its original position to its simplified position on a billboard, the more will its animation change.

In addition to being co-planar to some degree, triangles should thus be attached to the same animation bone, if they are to share a billboard. If the tree is animated using a non-trivial amount of bones, an individual billboard cloud is created for each bone, which generally results in more billboards, compared to the number of billboards obtained if a single billboard cloud is created for the entire foliage part of a CASTM. Assuming that a maximum allowed permutation distance is used to create the billboard clouds we might have to increase the permutation distance to an unintentionally large value in order to obtain a small number of billboards. This could yield bad visual fidelity. Furthermore, the number of billboard clouds cannot be reduced any further than the number of bones, which is an obvious limitation. For these reasons we introduce common joint rotation simplification, which makes it possible to reduce the number of individually created billboard clouds even further.

Common Joint Rotation Simplification

Consider two sibling bones in the skeleton. Both bones specify a rotation animation around the same joint. If the rotation animations of two siblings are replaced with a single average animation, then the triangles attached to these bones can share billboards, as they perform the same rotation animation around



(a) The non-filled dots represent two foliage triangle centroids and the line represents the billboard simplifying the triangles. d_1, d_2, l_1, and l_2 denote the distances between the billboard vertices and the triangles, and between the triangles and b, respectively. The billboard vertices (the filled dots) are attached to bone b, thus b animates the billboard.

(b) b rotates, thus rotating the billboard and the two foliage triangle centroids. The distances d_1, d_2, l_1, and l_2 remain constant, implying that the triangles remain fixed to their simplified positions on the billboard during the animation, and hence is rotated correctly by the billboard.

Figure 5.3: Inter-distance of triangles attached to the same bone remains constant during animation. Hence, a non-animated billboard can be created for this static geometry and attached to the bone.

the same joint. We refer to this type of simplification as a *common joint rotation simplification*, as the skeleton animation is simplified by defining common rotations for the bones (siblings) contained in a joint. Common joint rotation simplification is exemplified in Figure 5.4(a) on the next page and Figure 5.4(b) on the facing page.

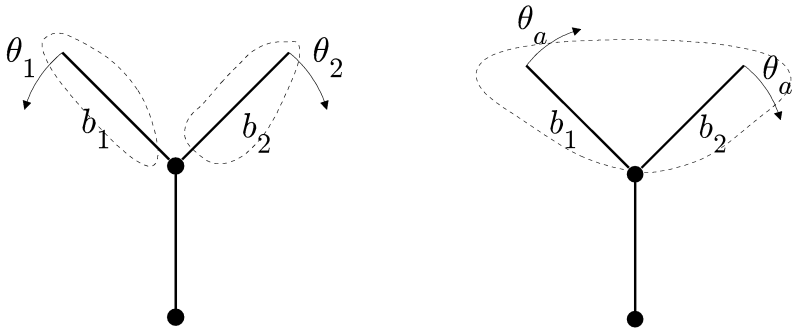
However, common joint rotation simplification cannot reduce the number of individual billboard clouds further than the number of joints in the skeleton. To remedy this shortcoming, we introduce bone reduction.

Bone Reduction

We define bone reduction as follows.

Bone Reduction: *Bone reduction* is an animation simplification procedure, that, given a CASTM, produces a new skeleton with fewer bones and with an animation similar to the original animation. The model geometry is then attached to this new and more simple skeleton.

From this definition no restrictions are put on the skeleton simplification procedure. Hence, using bone reduction we can produce an arbitrary new skeleton; even one that has no structural relations in common with the input skeleton. This is unsuitable, as we could then simply discard the input skeleton. Since we want to use the animation skeleton of the CASTM, we introduce bone collapsing as follows.



(a) Before applying common joint rotation simplification. Foliage triangles attached to b_1 and b_2 cannot be simplified onto the same billboards, because b_1 and b_2 have different rotation animations represented by θ_1 and θ_2. The two individual billboard clouds created are illustrated by the two dashed volumes.

(b) After applying common joint rotation simplification. The foliage triangles attached to b_1 and b_2 can now be simplified onto the same billboards, as both bones now share a single average animation, θ_a. Consequently, a single billboard cloud can be created for the two bones, illustrated by the single dashed volume.

Figure 5.4: The effect of applying common joint rotation simplification illustrated.

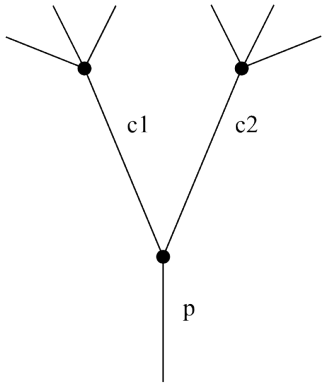
Bone Collapsing: *Bone collapsing* is a bone reduction technique that uses the following two operations:

- **Sibling collapse**: Two bones attached to the same joint in the skeleton (siblings), are replaced with a single bone attached to this joint, which inherits the children of both replaced bones.
- **Parent/child collapse**: A parent bone and its only child bone are collapsed into a new bone, which inherits the children of the replaced child bone and becomes the child of the replaced parent bone's parent.

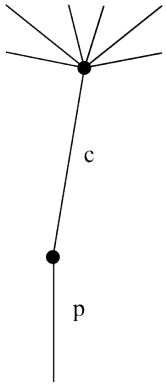
An example of using the sibling collapse simplification operation can be seen in Figure 5.5 and an example of using the parent/child collapse simplification operator can be seen in Figure 5.6 on the next page. Notice that both bone collapse operations can be used to reduce the number of joints in the skeleton.

Strategy

Common joint rotation simplification and bone collapsing can be used to reduce the number of individual billboard clouds needed for simplification, which in general implies fewer billboards. In order to obtain a simplified model containing a certain amount of billboards to be used for some LOD, we are able to favor either precise animation (more bones and more unique bone rotation animations per joint) or precise co-planarity (less permutation distance), when creating the animated billboard cloud.

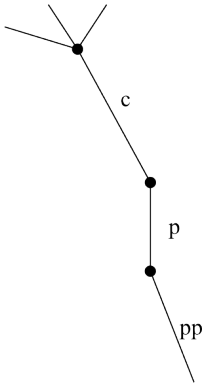


(a) The children $c1$ and $c2$ of bone p are to be collapsed into a new bone.

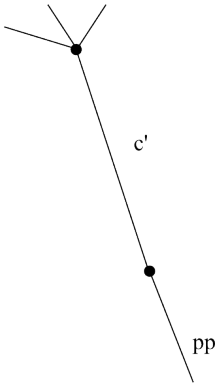


(b) $c1$ and $c2$ are now collapsed into the new bone c, which inherits the children of $c1$ and $c2$ and becomes the child of p.

Figure 5.5: An example of a sibling collapse.



(a) The child bone c and its parent bone p are to be collapsed into a new bone.



(b) c and p are now collapsed into the new bone c', which inherits the children of c and becomes the child of p's parent, pp.

Figure 5.6: An example of a parent/child collapse.

To sum up, the Bone Clustering strategy include these steps:

1. Perform bone collapses and common joint rotation simplifications in order to reduce the number of individual billboard clouds needed for simplification.

2. For each joint in the simplified skeleton, create a static billboard cloud for each set of siblings that have been given identical rotation animation. The billboard cloud can be created using a traditional static billboard cloud algorithm.

3. Attach each billboard cloud to its associated bone, or, in case it was created for a set of siblings with identical animation, attach it to any bone in this sibling set.

The interesting part to design in a solution following this strategy is the common joint rotation simplification and the bone collapsing procedures. If a tree model contains a large amount of bones, a challenge is to select which bones to collapse, and which siblings to give common rotation animations, while still maintaining the overall animation appearance.

As mentioned, triangles should be attached to the same bone, if they are to be simplified by the same billboard. Assume two or more foliage triangles with similar trajectories, each attached to different bones that are placed very far from each other in the animation skeleton. Consequently, only after a number of bone collapses can these triangles be simplified onto the same billboard. This is an important shortcoming of this strategy.

Assuming that billboard clouds are created using an error-based billboard algorithm, where the triangle permutation distance is the error, this strategy generally implies more billboards than a solution following the Static Co-planar Clustering strategy. The intuition is that in order to share a billboard, the geometry has to respect the maximum permutation distance in both strategies, but it should furthermore be attached to the same bone in the Bone Clustering strategy. The more criteria that has to be respected in order to share a billboard, the more billboards will in general have to be used to simplify the model.

5.2.4 Animation Description Clustering

The idea behind this solution strategy is to cluster the animated foliage triangles based on some analyzis of their animations. More precisely, the approach is to cluster triangles based on an animation description type (defined in Section 4.1 on page 57).

An animation description type can be used to cluster animated triangles with similar animations, as it formalizes animation similarity with a $deviation$ function. An animation description is created for each triangle, and the descriptions are clustered based on the deviation function. When a set of descriptions have been assigned to a cluster, a billboard cloud is created for the cluster. The animation of each billboard is given by the average of animation descriptions

of all triangles simpified on the billboard, which is supplied by the $average$ function of the animation description type.

In steps, the Animation Description Clustering strategy is as follows:

1. Cluster the foliage triangles according to some animation description type.
2. Create a pose of the animated foliage.
3. Create a static billboard cloud for each animation description cluster using a traditional billboard cloud algorithm.
4. Define the animation of each billboard by applying the average animation description of the triangles simplified by the billboard.

In order to obtain a reasonable amount of billboards for some LOD we can choose to favor animation precision or a small co-planarity error. Which to favor in order to get a desired number of billboards and the best visual fidelity is obviously subject to experimentation. The essential part in this strategy is defining a reasonable animation description type, which is critical for the visual result.

Compared to the Bone Clustering strategy, a clustering by animation descriptions can more effectively find and cluster triangles with similar animation positioned very far from each other in the animation skeleton. Assume that two triangles, t_1 and t_2, both attached to the outermost level of the skeleton have similar triangle trajectories, but do not share many ancestors in the skeleton. A such scenario is illustrated in Figure 5.7, where the ancestors of the bones the two triangles are attached to are assumed to only rotate very small angles. Given a reasonable animation description type, t_1 and t_2 yield a low deviation, and will hence end up in the same animation cluster using the Animation Description Clustering strategy.

A solution following the Bone Clustering strategy would have to perform several bone collapses in order for these triangles to be able to share a billboard. An example is sibling collapsing the two bones, b_1 and b_2, followed by sibling collapsing the bones b_3 and b_4, which become siblings as a result of the first collapse. The bones which t_1 and t_2 are attached to have now become siblings and can share a billboard, if given a common joint rotation animation. As a consequence of the collapses, all foliage triangles in the tree model will get their animation simplified.

This example illustrates how the Animation Description Clustering strategy is able to benefit from not being restricted to consider the skeleton structure. In general we do not judge the example scenario likely in most tree models, but still we consider the way the Animation Description Clustering strategy handles this scenario a potential advantage.

Summary

In this chapter we have defined a set of error metrics used to objectively measure the quality of simplifying the foliage part of CASTMs. We came to the

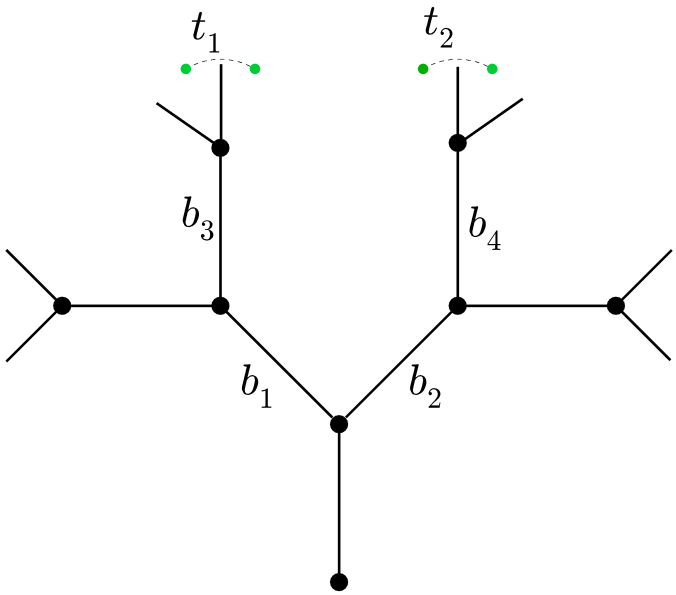


Figure 5.7: Foliage triangles t_1 and t_2 attached to different bones that perform identical rotation animations. Using animation descriptions these triangles can be clustered together. If, instead, following the Bone Clustering strategy, t_1 and t_2 can be simplified onto the same billboard only after a number of bone collapses.

conclusion that if the Euclidean distance and orientation metrics are close to 0 in every frame, the simplification should exhibit high visual fidelity throughout the animation. But, achieving this for a low number of billboards may be intractable. A more suitable combination of error metrics is the colour and animation metrics: If a simplification yields low values with the colour metric as well as an animation metric, it suggests visual fidelity in every frame and coherent animation structures.

Three different solution strategies were then presented. Following the Static Co-planar Clustering strategy a static billboard cloud is created for the foliage from one pose, e.g. the pose from the first frame, and an animation is found for each billboard that imitates the animation of as many of the triangles simplified by the billboard in question as possible. However, finding a common animation for the triangles is not trivial. The essence in the Bone Clustering strategy is the observation that all geometry attached to the same bone share the same animation. Using the bone collapsing operations and common joint rotation simplification, foliage triangles attached to different bones can eventually be simplified onto the same billboard. In the final strategy, the Animation Description Clustering strategy, an animation description type is used to cluster foliage triangles. Having clustered foliage triangles using the $deviation$ function of a chosen animation description type, and applied a billboard cloud algorithm on the foliage in a pose, each billboard is animated using the $average$ function of the animated description type, in order to determine an average animation of the foliage triangles simplified by the billboard in question. Following the Animation Description Clustering strategy, it is not required that the input model

is a CASTM, because the polygonal tree model does not have to be rigged with an animation skeleton.

In the following chapters we present solutions following the Bone Clustering and Animation Description Clustering strategies.

Chapter 6

Spectral Clustering

Spectral Clustering is a solution following the Animation Description Clustering strategy. The foliage to be simplified is not required to be animated using skeleton animation. Assuming skeleton animation, however, makes several improvements possible, as will be discussed in addition to the Spectral Clustering algorithm.

The algorithm creates a spectral description of each foliage triangle in the CASTM to be simplified and clusters these descriptions using k-clustering. Billboard clouds are created using the Stochastic Billboard Cloud Algorithm.

The Spectral Clustering solution is not specifically designed for a certain animation description type, and may in principle use any given type. However, the quality of the simplification is greatly dependent on the soundness of the animation description type applied.

This chapter starts with a thorough discussion of different spectral-based animation description types in Section 6.1, including the shortcomings of each. After this, the Spectral Clustering solution algorithm is presented in Section 6.2. A general spectral animation analysis related improvement is presented in Section 6.3, followed by another improvement in Section 6.4 that utilizes the skeleton of the CASTM. The chapter is concluded with an evaluation of the simplification performed by the solution in Section 7.6.

6.1 Spectral Animation Description Type

Recall that the first step in the Animation Description Clustering strategy is to cluster all triangles based on some animation description type. An animation description type must define a $deviation$ function, which can be used to determine which triangle are suitable for sharing a common animation. When a set of triangles are simplified to a single animated billboard, a reasonable animation of it is yielded by the $average$ function of an animation description type.

6.1.1 Description Type Preferences

Using spectral analysis to define a reasonable animation description type is not trivial. Spectral analysis of triangle trajectories was introduced in Section 4.6 on page 66, but to briefly recap, the idea is to apply the DFT on each of the periodic trajectory coordinate functions $p_x(t), p_y(t)$, and $p_z(t)$. This yields three coordinate axis spectra describing the triangle animation in terms of frequency, amplitude, and phase. Frequency was identified as describing the animation property speed, amplitude as describing distance, while phase describes both relative time-offset and shape.

When simplifying the foliage part of a CASTM, it is judged important to attempt to preserve the speed, distance, and shape animation properties as much as possible as mentioned in Section 4.2 on page 58. On the other hand, time-offset relative to other triangle trajectories was judged insignificant for the visual appearance of a triangle in a foliage model, and we hence encourage simplification of triangles with time-offset equal trajectories on a single animated billboard.

Here, we list three preferences for a spectral animation description type:

- The descriptions of two triangles with time-offset equal trajectories should get no deviation.
- The descriptions of two triangles should yield deviation relative to the amount of difference in both speed, distance, orientation, and shape, i.e. if there is only a minor difference in these properties, a low deviation is expected, while a large difference should imply a high deviation.
- If a set of triangles move with similar speed, distance, shape and orientation, then the average of their descriptions should yield an average description, that when applied to a triangle yields an animation also with a similar speed, distance, and orientation.

In the following, three different analysis approaches will be discussed. The Naïve Spectral Description Type is straightforward, but does not yield low deviation values for time-offset equal triangle trajectories. Two approaches follow, namely the Dominating Axis Description Type, that constricts analysis of trajectories to a single axis, and the Axis-phase Description Type. Both these approaches attempt to remedy the shortcomings of the naïve approach.

6.1.2 Naïve Spectral Description Type

In this section we present a straightforward definition of an animation description type based on spectral analysis of animation.

In the following, the *amplitude-weighted phase average*, φ^a_ω, of the phase values in N spectra for a certain frequency, ω, equals the average of the N phase values weighted with the respective amplitude values for ω:

$$\varphi^a_\omega = \frac{1}{\sum_{i=1}^{N} A^i_\omega} \sum_{j=1}^{N} A^j_\omega \varphi^j_\omega$$

Intuitively, the phase value with the highest amplitude for some frequency will influence the average phase value for that frequency more than the other spectra.

Along the same lines, the *amplitude-weighted phase difference* between two spectra equals the sum of amplitude-weighted phase difference between each spectrum and the amplitude-weighted phase averages of the two spectra:

$$\sum_{\omega} \mathit{difference}(\varphi_{\omega}^{1}, \varphi_{\omega}^{a}) + \mathit{difference}(\varphi_{\omega}^{2}, \varphi_{\omega}^{a})$$

Phase values are specified in the cyclic range $[0; 2\pi]$, and the difference between, or the average of such, values is found as described in Appendix A.4 on page 156.

The description type is an elaboration of the spectral description type briefly introduced in Section 4.6.2 on page 68. The functions constituting the Naïve Animation Description Type are:

$description(tr)$**:** An animation description, d, of a triangle, tr, is obtained by applying the DFT on each of the triangle trajectory coordinate functions, $p_x(t)$, $p_y(t)$, and $p_z(t)$. This yields three spectra, $X_x(\omega)$, $X_y(\omega)$, and $X_z(\omega)$, which constitute the description. We refer to these spectra as the three *coordinate axis spectra* computed from a triangle trajectory.

$deviation(d_1, d_2)$**:** The deviation between two animation descriptions, d_1 and d_2, is defined as follows: let A denote the sum of difference in amplitude for each frequency in the coordinate axis spectra in the two descriptions, and let P denote the sum of amplitude-weighted phase difference for each frequency in the coordinate axis spectra. Deviation is defined as the Normalized Error Product of A and P, $NEP(A, P)$ (see Appendix A). Hence, if either A or P is large, the entire deviation becomes large.

$average(D)$**:** An average animation description is obtained from a set of animation descriptions, D, as follows. Each element in D contains three coordinate axis spectra, and for each coordinate axis (x, y, and z), a new average spectrum is created by averaging the amplitude and computing the amplitude-weighted phase average values for each frequency in the associated axis spectra in D.

$apply(tr, d)$**:** An animation description, d, is applied to a triangle, tr, by applying the inverse DFT on each of the coordinate axis spectra in d, and animating tr with the resulting coordinate functions.

A naïve animation description of a triangle is *lossless*, in that it does not discard information, assuming that the triangle trajectory is sampled with enough samples when applying the DFT. In other words, if an animation description is created for a triangle, then an identical animation is obtained if the description is applied to a triangle.

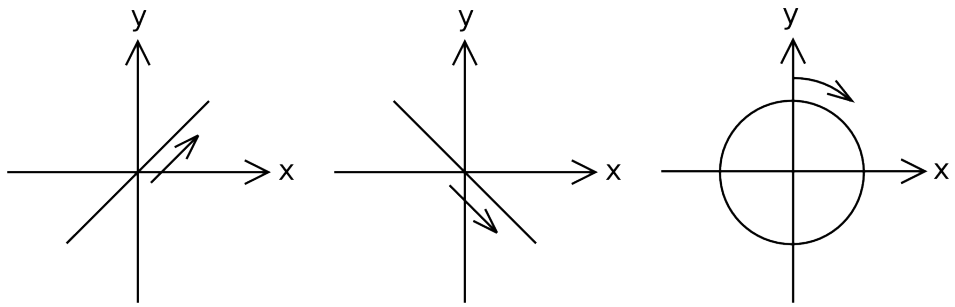
Evaluation

Recall that the $average$ function is to be used to define an animation of billboard that simplifies a set of somewhat co-planar triangles. Using the Naïve Animation Description Type, each triangle is described by three coordinate axis spectra, the content of which is simply averaged to yield an average description.

Frequency and amplitude information represents the animation properties speed and distance, respectively. The amplitude values in each coordinate axis spectrum are averaged for the different triangles, and hence if all triangles have a large amplitude at a certain frequency, ω', in a certain coordinate axis spectrum, $X_i(\omega')$, then the average will also have a large amplitude at $X_i(\omega')$. However, if the triangles have large amplitudes at few but different frequencies, the average will get small amplitudes at many frequencies instead. This may be a problem which is discussed in Section 6.3, in which an improved approach to averaging spectrum amplitudes is also introduced.

Averaging the phase values at each frequency in the coordinate axis spectra, might result in an average animation with a shape that is entirely different from either of the original triangle shapes, and it hence might not represent a meaningful average of the original shapes.

Consider the trajectories in Figure 6.1, which have also been illustrated earlier. The y-axis phase for the two lined shaped trajectories in Figure 6.1(a) and Figure 6.1(b) is $\varphi_y = 0$ and $\varphi_y = \pi$, respectively. If the two trajectories are averaged as dictated by this animation description type, the result would have $\varphi_y = \pi/2$ and equal the trajectory in Figure 6.1(c). As can be seen, the resulting shape is circular, even though both the original trajectories are shaped as a line. This contradicts our animation description preference, that the average of animations with similar shapes should yield an average animation also with a similar shape.



(a) $\varphi_x = \varphi_y = 0$ (b) $\varphi_x = 0; \varphi_y = \pi$ (c) $\varphi_x = 0; \varphi_y = \frac{\pi}{2}$

Figure 6.1: The circular trajectory in Figure 6.1(c) is the obtained by averaging the phase values of the two line shaped trajectories in Figure 6.1(a) and Figure 6.1(b).

Two time-offset equal triangle trajectories will not necessarily obtain a low deviation value using this simple animation analysis approach, as the axis spectra computed for such trajectories may have very different phase content. This contradicts our animation description preferences, and is hence considered a severe shortcoming.

Ignoring Phase

A simple attempt to remedy this shortcoming, is to simply ignore the phase content in the axis spectra when computing deviation. Two time-offset triangle trajectories yield equal spectra apart from phase content. If phase is ignored, the animation descriptions of two such triangles will therefore get a low deviation, and the two triangles concluded suitable for sharing a single billboard animation. However, phase does not only define time-offset, but also the shape and orientation properties of animation. If phase is ignored, two trajectories with very different shapes, or two trajectories that move in entirely different orientations, might also get low or no deviation. This is not to be desired, and contradicts our animation description preferences.

As the Naïve Spectral Description Type does not satisfy the animation description preferences, motivation exists for searching for a better analysis approach.

6.1.3 Dominating Axis Description Type

We want time-offset equal trajectories to get a low deviation, and still be able to differentiate between different shapes. The Dominating Axis Description Type assumes that foliage triangle trajectories can be approximated well by a movement on a single axis.

This animation description type, is based on the Dominating Axis Metric defined in Section 5.1.4 on page 74. Intuitively, the dominating movement axis is a suitable axis for defining a one-dimensional animation approximating the original triangle trajectory well.

The idea behind this animation description type is as follows. If most triangle trajectories in the input foliage are somewhat line-shaped, then by comparing them based on their one-dimensional dominating axis animation, phase and shape no longer presents a problem. Phase can simply be ignored to make two time-offset equal trajectories get equal descriptions, and ignoring phase does not alter the shape or orientation of the dominating axis animation, as it is always line-shaped.

Animation Description Type

The animation of each triangle is described by its dominating movement axis and a spectrum of the one-dimensional animation defined by mapping its trajectory onto this axis. Deviation between two trajectories is yielded by considering the angle between their dominating axes, as well as the difference between their spectra. The average of a set of trajectories is given by averaging their dominating axes, as well as the spectra.

Formally, the description type is as follows:

$description(tr)$**:** The animation description for a triangle, tr, is a tuple $\langle \mathbf{a}, X(\omega) \rangle$, where $\mathbf{a}$ is a normalized vector describing the dominating movement axis of the trajectory for tr, and $X(\omega)$ is the spectrum

obtained by applying the DFT on the animation yielded by mapping the triangle trajectory onto $\mathbf{a}$.

$deviation(d_1, d_2)$**:** The deviation between to animation descriptions, $d_1 = \langle \mathbf{a}_1, X_1(\omega) \rangle$ and $d_2 = \langle \mathbf{a}_2, X_2(\omega) \rangle$, is defined as:

$$(1 - \mid \mathbf{a}_1 \cdot \mathbf{a}_2 \mid) NE(amplitudeDistance(X_1(\omega), X_2(\omega))),$$

where *amplitudeDistance* is a function that yields the sum of amplitude difference for each frequency in two spectra, and *NE* is the Normalized Error function normalizing the distance to the range $[0; 1]$.

In simple terms, deviation equals one minus the cosine to the angle between the dominating movement axes multiplied with the normalized difference of amplitude in the two spectra.

$average(D)$**:** Given a set of animation descriptions, D, an average description is obtained by averaging the axes in D, and likewise averaging the spectra in D. The axes are averaged by considering them as lines containing the origin, and then finding a best average line for this set of lines. The set of spectra is averaged by averaging the amplitude and phase values for each frequency. Note that phase values are averaged taking the cyclic range into account (see Appendix A.4 on page 156).

$apply(tr, d)$**:** An animation description, d, is applied to a triangle, tr, by applying the inverse DFT on the spectrum in d to yield a one-dimensional animation which is applied to the triangle.

In the next section, this single axis approach to animation analysis is evaluated.

Evaluation

Time-offset equal trajectories yield identical dominating movement axes, and applying the DFT on the animations obtained by mapping the trajectories onto the dominating movement axes yields equal spectra aside from phase values. Since phase is not included as part of the deviation function, two time-offset equal triangle trajectories will have no deviation using this description type, which is one of the animation description type preferences.

As discussed in Section 6.1.2 on the preceding page, ignoring phase in the naïve approach would imply no deviation for time-offset equal trajectories, but also potentially imply no or low deviation for trajectories with very different shapes or movement orientations, which contradicts our description type preferences. Ignoring phase is not a problem in this approach, as phase does not influence the shape of a one-dimensional animation. However, this approach has other shortcomings, which will be discussed in the following.

The obvious shortcoming of the approach is that it only analyzes the animation along the dominating movement axis. Hence, the animation descriptions are not lossless, and the original animation of a triangle cannot be computed

from its animation description. When determining the average animation of a set of triangles, this average will also be a one-dimensional axis animation. Consequently, if each billboard in a billboard cloud is animated by the average animation of the triangles it simplifies, all animation in the billboard cloud will be one-dimensional. Although the different billboards will animate on potentially differently oriented axes, this simplification is still likely to be noticeable to an observer.

The success of the analysis approach depends on the nature of the animations in the animated foliage to be simplified. If each foliage triangle moves closely along a single axis in the input CASTM, then the single axis analysis approach does not discard much information. We find it likely that branches move in ellipse like shapes, which is mentioned as a tree observation in Section 1.5 on page 29. How well an ellipse is suitable for a single axis approximation depends on the ratio between its height and width (i.e. the distance between the two ellipse foci points), and a larger ratio implies less approximation introduced by the dominating axis animation. The worst-case shape is a circle, as the dominating movement axis of a circle is ill-defined.

To summarize, the Dominating Axis Description Type is suitable for analyzing triangles moving somewhat along a single axis. However, it is less suitable for circular trajectories or other more complex trajectories. Similar triangle trajectories with circular shapes might in worst case be judged different, as their dominating movement axis is ill-defined, and triangles with different shapes might be concluded similar, if they have similar dominating axis animations. These observations contradict our animation description type preferences.

The Axis-phase Description Type introduced next is similar to the naïve approach, but introduce a new comparison of the phase content in the three axis spectra. This new phase comparison is based upon further observations on how axis phase impact on animation shape and orientation.

6.1.4 Axis-phase Description Type

This approach does not limit the animation analysis to movement on a single axis, but instead apply DFT on each of the three trajectory coordinate functions. This is similar to the Naïve Animation Description Type, the primary shortcoming of which is that two time-offset equal trajectories might get a large deviation.

In order to remedy this problem, the possibility of ignoring phase was discussed in Section 6.1.2. Two time-offset equal trajectories have equal coordinate axis spectra aside from the phase content, hence if phase is ignored in the deviation function, such triangles will get no deviation. However, ignoring phase had the unfortunate side-effect that trajectories with very different shapes or orientations might also get low or no deviation. In the approach described in the following, the phase content of the spectra is not ignored, but will be compared using what we refer to as the *axis-phase relationship*.

Axis-phase Relationship

In Section 4.6.1 on page 66, it was illustrated how the phase content in the axis spectra obtained by applying the DFT on a trajectory represents both the relative time-offset and shape of the trajectory. We identify the axis-phase relationship as a relationship that is equal for two time-offset equal trajectories. For simplicity, the following examples are two-dimensional.

Any trajectory of the form:

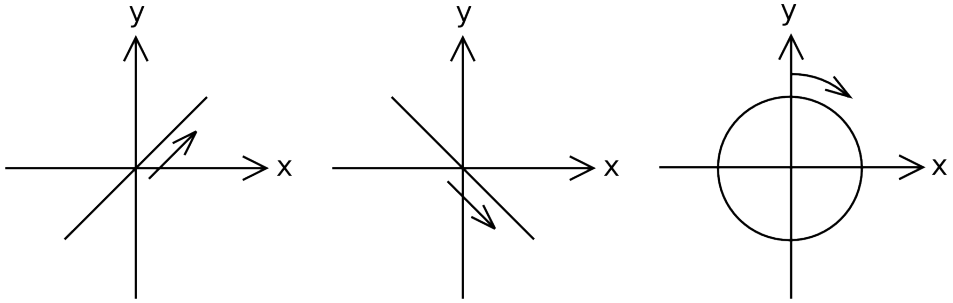
$$p(t) = \begin{bmatrix} Asin(\varphi_x + \omega t) \\ Asin(\varphi_y + \omega t) \end{bmatrix}, \quad \varphi_x = \varphi_y$$

moves in a straight line, whereas any trajectory of the same form, but with the phase difference given by:

$$\varphi_x = \varphi_y + \frac{\pi}{2}$$

moves in a circle.

In general, all trajectories only containing a single frequency in both the x and y- axis, which have same offset between x-phase and the y-phase, are trajectories with identical shape and orientation. Furthermore, any trajectory with another phase offset is one with another shape or orientation. As examples, consider the three trajectory shapes in Figure 6.2, which are all trajectories of the same form $p(t) = \begin{bmatrix} Asin(\varphi_x + \omega t) \\ Asin(\varphi_y + \omega t) \end{bmatrix}$ but with the three different phase offsets indicated in the figure.



(a) No offset: $\varphi_x = \varphi_y$ (b) π offset: $\varphi_x = \varphi_y + \pi$ (c) $\frac{\pi}{2}$ offset: $\varphi_x = \varphi_y + \frac{\pi}{2}$

Figure 6.2: Different shapes generated by different phase offsets.

In three-dimensions, the offset between the x-phase and the y-phase for a given frequency, together with the offset between the x-phase and the z-phase is denoted the axis-phase relationship. The relationship has the form:

$$\langle \varphi_{xy}^{\omega}, \varphi_{xz}^{\omega} \rangle, \qquad \text{where } \varphi_{xy}^{\omega} = \varphi_{x}^{\omega} - \varphi_{y}^{\omega} \text{ and } \varphi_{xz}^{\omega} = \varphi_{x}^{\omega} - \varphi_{z}^{\omega}$$

To describe all axis-phase relationships in the coordinate axis spectra for a trajectory, we will have an axis-phase relationship for each frequency ω. This tuple of axis-phase relationships is denoted the *axis-phase map* of a trajectory.

We observe that even though the coordinate axis spectra obtained for two time-offset equal trajectories have different phase content, the axis-phase maps will be equal. If a trajectory is time-offset with some time value, then for each frequency, ω, in its axis spectra, the three phase values $(\varphi_x^\omega, \varphi_y^\omega, \varphi_z^\omega)$ will have increased with the same value, c. The axis-phase relationship $\langle \varphi_{xy}^\omega, \varphi_{xz}^\omega \rangle$ will remain the same after the time-offset, as we obviously have that:

$$\varphi_x^\omega - \varphi_y^\omega = (\varphi_x^\omega + c) - (\varphi_y^\omega + c) \text{ and } \varphi_x^\omega - \varphi_z^\omega = (\varphi_x^\omega + c) - (\varphi_z^\omega + c)$$

The idea behind the axis-phase description type is to alter the deviation measure of the Naïve Spectral Description Type, such that it examines the difference in axis-phase maps, instead of the difference in the actual phase values. Since time-offset equal trajectories have equal axis-phase maps, they will get no deviation. We postulate that trajectories with different shapes and/or orientation are likely to have different axis-phase maps, and will hence have deviation and be concluded to be visually different. This claim is discussed in Section 6.1.4.

The animation description type using axis-phase maps is presented next.

Animation Description Type

The Axis-phase Description Type is identical to the Naïve Spectral Description type, apart from the comparison of phase by the $deviation$ function, and how phase is averaged by the $average$ function.

The functions constituting the type are:

$description(tr)$**:** An animation description, d, of a triangle, tr, is obtained by applying the DFT on each of the triangle trajectory coordinate functions, $p_x(t)$, $p_y(t)$, and $p_z(t)$. This yields three spectra, $X_x(\omega)$, $X_y(\omega)$, and $X_z(\omega)$, in which phase values are omitted. The axis-phase relationship, $\langle \varphi_{xy}^\omega, \varphi_{xz}^\omega \rangle$, is computed for each frequency ω, yo yield an axis-phase map, which together with the three spectra constitute a description.

$deviation(d_1, d_2)$**:** The deviation between two animation descriptions, d_1 and d_2, is computed as follows. Let A equal the sum of difference in amplitude for each frequency in each of the coordinate axis spectra in the descriptions. Let furthermore P denote the sum of amplitude weighted difference between the axis-phase relationships for each frequency. Deviation equals $NEP(A, P)$, and hence if either A or P is large, then the entire deviation is large.

$average(D)$**:** An average animation description is obtained from a set of animation descriptions, D, as follows. Each element in D contains three coordinate axis spectra, and for each coordinate axis (x, y, and z), a new average spectrum is created by averaging the amplitudes at each frequency in the spectra of the corresponding coordinate axis in D. Furthermore, an amplitude-weighted average axis-phase map is computed.

$apply(tr, d)$**:** An animation description, d, is applied to a triangle, tr, as follows. Since the animation description spectra does not include phase values, random phase values maintaining the axis-phase relationships in d are generated. The inverse DFT is now applied on each of the coordinate axis spectra to yield an animation.

The amplitude-weighted average axis-phase map is computed by computing average axis-phase relationships for each frequency. Each axis-phase relationship component is computed as an average weighted by the sum of the amplitudes of the coordinate axes that the axis-phase relationship is concerned with, e.g. for some frequency in the spectrum we can compute the average of φ^1_{xy} weighted with $(A^1_x + A^1_y)$ and φ^2_{xy} weighted with $(A^2_x + A^2_y)$ and similarly for the x-z axis-phase relationship component.

Shortcomings of the analysis approach are discussed next.

Evaluation

As opposed to the Naïve Spectral Description Type, this animation description type is not lossless, regardless of the sample rate used when applying the DFT on a triangle trajectory. The information discarded are the exact phase values in the coordinate axis spectra. If an axis-phase description is created for some triangle, and this description applied to a triangle, some phase values are needed in order to apply the inverse DFT. As stated in the definition of $apply$, random phase values maintaining the axis-phase relationships are generated. Consequently, the triangle will not get an animation identical to the original one, as these random phase values can time-offset the animation, and even alter its animation shape or movement orientation to some degree, which will be discussed further.

Time-offset equal trajectories get no deviation as desired, but this comes at the cost of discarding phase information, which potentially have side-effects aside from time-offset. We postulated that two triangle trajectories with very different shapes or orientations, will get spectra with different axis-phase relationships. The quality of the Axis-phase Description Type depends on whether this is true. Unfortunately, trajectories with different shapes or orientations and equal axis-phase relationships do exist, a consequence of which is that triangles with different shapes or orientations might be concluded suitable for sharing a common animation. This is undesirable, and does not satisfy our animation description type preferences. This shortcoming is further discussed in the next Section.

Axis-phase Description Type Shortcoming

If trajectories contain only a single frequency, the axis-phase relationship for this frequency uniquely defines the shape, as was the case in the examples in Section 6.1.4 on page 94. However, when a trajectory contains several frequencies, a problem arise, since the Axis-phase Description Type does not include the inter-frequency phase offsets. We demonstrate this using an example.

Recall that a sinusoid is a function of the form $Asin(\varphi + \omega t)$, where A is amplitude, φ is phase, and ω is frequency. Consider the two-dimensional trajectory $p(t)$ defined as follows

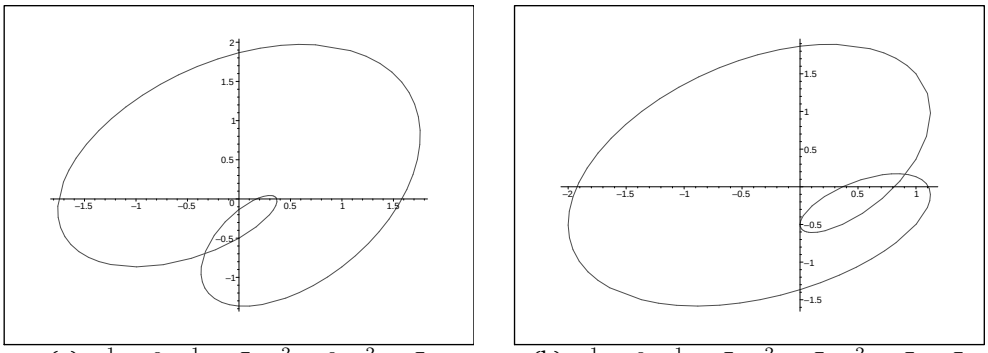
$$p(t) = \begin{bmatrix} sin(\varphi_x^1 + t) + sin(\varphi_x^2 + 2t) \\ sin(\varphi_y^1 + t) + sin(\varphi_y^2 + 2t) \end{bmatrix}$$

The trajectory contains two frequencies in both the x and y- axis movement, namely $\omega_x^1 = \omega_y^1 = 1$ and $\omega_x^2 = \omega_y^2 = 2$. The amplitudes for both frequencies in both axes are the same, and equal to 1. The phase values are given by the four variables φ_x^1, φ_y^1, φ_x^2, φ_y^2.

We consider the following axis-phase relationships between the x and y-axis movement:

$$\varphi_{xy}^1 = \varphi_x^1 - \varphi_y^1 = \pi/2 \qquad \text{and} \qquad \varphi_{xy}^2 = \varphi_x^2 - \varphi_y^2 = \pi/3$$

A trajectory obtained with a set of phase values maintaining this axis-phase relationship is shown in Figure 6.3(a). Another trajectory obtained by adding $\pi/2$ to both of the φ_x^2 and φ_y^2 values of the first example is illustrated in Figure 6.3(b). This second trajectory also maintains the specified axis-phase relationship, but it has a noticeable different shape.



(a) $\varphi_x^1 = 0, \varphi_y^1 = \frac{\pi}{2}, \varphi_x^2 = 0, \varphi_y^2 = \frac{\pi}{3}$ (b) $\varphi_x^1 = 0, \varphi_y^1 = \frac{\pi}{2}, \varphi_x^2 = \frac{\pi}{2}, \varphi_y^2 = \frac{\pi}{2} + \frac{\pi}{3}$

Figure 6.3: Effect of changing the inter-frequency phase offset, without changing the axis-phase relationships.

These examples demonstrate, that two trajectories with different shapes and orientation might have equal axis-phase maps, and would hence have no deviation using the Axis-phase Description Type. This violates the animation description type preferences.

However, the change in shape illustrated is an attempt to create a worst-case example using a inter-frequency phase offset difference of $\frac{\pi}{2}$ on a simple trajectory containing only two frequencies. We judge the change in shape far less drastic than the change in shape and orientation resulting from changing axis-phase maps (see Figure 6.2 on page 94). In other words, we cannot alter the shape of the trajectories in Figure 6.3 to a line, or some other entirely different shape, using inter-frequency phase offsets alone.

In conclusion, we observe a tendency for trajectories with similar axis-phase maps to have similar shapes and orientations, but these observations have not

been generalised to any formal statement. An improvement of the animation description type may be derived by including inter-frequency phase offset information, but such an approach will not be discussed further.

6.1.5 Summary of the Description Types

Three different animation description types based on spectral analysis of animation has been presented. The Spectral Clustering Algorithm can utilize any of these types to cluster the foliage triangles with similar animations in a CASTM. Which spectral description type yields the best results is hard to predict, as each approach has its shortcomings, and for this reason we propose to experiment with all three.

The primary shortcoming of the Naïve Spectral Description Type in relation to the identified animation description type preferences in Section 6.1.1 on page 88 is, that it potentially yields high deviation between time-offset equal triangle trajectories. The Dominating Axis Description Types attempts to remedy this problem, but does so in a way that discards all non-linear shape information. If this description type is used for animating billboard in a animated billboard cloud, each billboard will only be animated along a single axis, which depending on the nature of animation in the input, may be an inadequate simplification. The final approach represented by the Axis-phase Description Type utilizes axis-phase maps in an attempt to interpret phase information in terms of the time-offset, shape and orientation animation properties. Unfortunately, counter examples has been given, for which it fails to separate time-offset from shape and orientation, and hence conclude triangles with different trajectory shapes or orientations suitable for sharing a billboard.

In Section 5.1.4 on page 74, two animation error metrics were introduced, namely the Dominating Axis Metric and the Dominating Axis Spectral Metric, the purpose of which is to measure visual fidelity of a billboard cloud simplification in terms of animation. The deviation measure of the Axis-phase Description Type, for example, could be used to define an alternative animation metric.

In the following section, the Spectral Clustering algorithm is specified.

6.2 Spectral Clustering Algorithm

The input to the Spectral Clustering Algorithm (SCA) is a tree model, $treeModel$, satisfying the definition of a CASTM, and containing a set, T, of animated foliage triangles. The algorithm outputs an animated billboard cloud that simplifies the foliage of the input model.

The foliage triangles are clustered with regards to some animation description type using the budget-based K-means Clustering Algorithm which takes a number of animation clusters, k, as input. The Stochastic Billboard Cloud Algorithm is used for billboard construction, and takes a maximum allowed permutation distance, ϵ, a number of sample planes, S, and any parameters required by the improvements discussed in Section 3.6 on page 53. These billboard cloud construction improvement parameters are omitted in the de-

scription of this algorithm. Finally, the SCA algorithm takes the number of position samples, N, to be used when analyzing the animation of each triangle.

Triangles are first clustered using k as budget in the budget-based animation description clustering. After this initial clustering, a billboard cloud is created for each spectral description cluster using the Stochastic Billboard Cloud Algorithm, which is an error-based clustering, with ϵ as error. Each billboard is finally animated by applying the average animation of the triangles simplified on the billboard. Note that if $k = 1$, then no animation clustering is performed, which corresponds to following the simple Static Co-planar Clustering strategy.

As the algorithm is independent of which one of the presented animation description types is applied, the term *spectral description* is used to denote an description created using any spectral based animation description type.

In steps, the algorithm is as follows:

$SCA(treeModel, k, \epsilon, S, N)$

1. Create a spectral description for all foliage triangles using the *description* function associated with the employed spectral description type. The descriptions are clustered in k clusters by the K-means Clustering Algorithm using the *deviation* function as distance measure, and the *average* function to calculate the centroid of each cluster. The K-means Clustering Algorithm relies on a reasonable initial distribution of descriptions in K clusters. More information about this initialization step can be found in C.1 on page 163.

2. Create an average pose of the foliage part of a CASTM by calculating the average position of each triangle from a set of position samples taken uniformly over the animation cycle. This pose of the geometry is used when creating billboards.

3. For each spectral description cluster of triangles, apply the Stochastic Billboard Cloud Algorithm using ϵ as error value and S as the number of sample planes to evaluate per randomly chosen seed triangle.

4. Each billboard simplifies a set of triangles, and the set of spectral descriptions associated with these triangles, D, can be used to calculate an average spectral description using $average(D)$. The average description obtained, d_a, can be stored, and applied to the billboard at run-time by using $apply(tr_1, d_a)$ and $apply(tr_2, d_a)$, where tr_1 and tr_2 denotes the two triangles defining the billboard quad.

In relation to error and budget-based solution approaches (section 5.2.1 on page 77), the Spectral Clustering solution performs a budget-based clustering of triangles with the number of billboard clouds as budget, that attempts to minimize an animation error represented by deviation. This is followed by an error-based clustering of the triangles in each animation cluster, that use displacement distance ϵ as error, and attempts to minimize cost in terms of number of billboards.

In step 2, a static pose of the animated foliage model is created by placing each triangle at its average position during the animation cycle. This is done in order to be able to apply a traditional static billboard cloud algorithm on the foliage. A billboard represents triangles by a static texture, and the foliage is posed using average positions, as these are judged static positions that are the most representative for the entire animation cycle.

The algorithm only simplifies the foliage triangles in the input CASTM (recall our project delimitations in Section 2.4 on page 35). We propose to simplify the trunk mesh in the CASTM using some mesh simplification technique, that takes animation into account. A consequence of this separation of simplification is, that the foliage billboards will not follow the branches throughout the animation in the produced output. This is may be visually unimpressive, which is a serious limitation of this simplification technique.

The steps in the Spectral Clustering Algorithm involve constructing a spectral description for each triangle in the Cyclic-Skeletal-Animated Polygonal Foliage Model, and using these spectral descriptions to find the average descriptions as well as computing deviations to these average descriptions during the k-clustering. A highly detailed polygonal tree model consists of a large amount of foliage triangles, which implies construction of a large amount of descriptions. A single description is created for each triangle prior to the k-clustering, and the number of average descriptions constructed as well as the number of deviation computations performed depends on the number of k-clustering iterations.

When having a large number of foliage triangles the construction of spectral descriptions, as well as computing average descriptions and deviation measures, might become a performance issue. For this reason we propose a performance optimization in Appendix C.2 on page 164, using which several triangles can share a single spectral description.

6.2.1 Billboard Cloud Performance Details

A performance related issue is that the textures of the billboard cloud should be packed into a single texture, as mentioned in Section 1.3.3 on page 22. This is done in order to reduce the number of render state switches, which generally improves fps performance a great deal.

The billboards in the animated billboard cloud are each associated with a spectral description that describes the average animation of the triangles simplified by the billboard. We suggest to apply this animation at runtime by using a simple vertex program. To reduce the number of render state switches, a single animation vertex program should be shared by all the billboards. The animation descriptions of the billboards are stored in a texture, in which the vertex program can look up the appropriate parameters for the animation of each specific billboard. For example, if the Dominating Axis Description Type is used, this texture could contain the dominating movement axis vector and frequency spectrum for each billboard. In this case, the vertex program animates a billboard by applying the inverse DFT on the spectrum to yield a one-dimensional animation, and then translate the billboard vertices in the dominating axis direction according to this animation.

6.3 Frequency Clustering Improvement

All the presented spectral animation description types in Section 6.1 are based on comparing and averaging the amplitude content of two spectra. In this section we elaborate on this, and identify an improved approach using frequency clustering.

The spectral description types all define deviation of the amplitude content of two spectra as the sum of amplitude difference for each frequency. Similarly is the average of a set of spectra obtained by averaging the amplitudes for each frequency. However, comparing spectra in this fashion may not be favourable as will now be demonstrated.

6.3.1 Deviation and Average Shortcomings

Consider two signals:

$$x(t) = sin(4t) \quad \text{and} \quad y(t) = sin(5t), \quad t = [0; 2\pi].$$

These signals are illustrated in Figure 6.4, as well as the spectra obtained by applying the DFT on these signals.

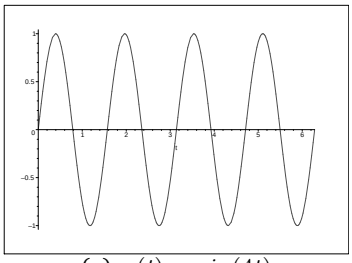
If deviation is obtained simply by considering the amplitude distance between the spectra, as defined by the spectral animation descriptions, then these two signals are concluded to be different, as they do not have any amplitude in the same frequencies. In fact, the deviation between these two signals is just as large as the deviation between any of these signals and the signal $z(t)$, illustrated in Figure 6.5. This contrasts to the intuition that $z(t)$ seems to be very different from the two signals.

If an average spectrum for the signals $x(t)$ and $y(t)$ is obtained by simply averaging the amplitude for each frequency, as is dictated by the spectral description types, then the result would be the spectrum shown in Figure 6.6(a). When this average spectrum is transformed to the time domain by the inverse DFT, we get the signal $avg(t) = \frac{sin(4t)+sin(5t)}{2}$, which is shown in Figure 6.6(b) on page 103. However, this average signal is intuitively not similar to either $x(t)$ or $y(t)$.

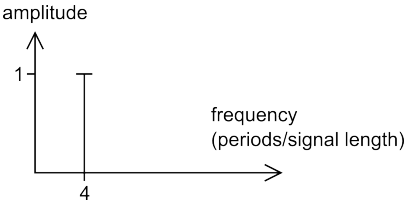
If the animation in a given axis of a foliage triangle is changed from $x(t)$ or $y(t)$ to the presented average $avg(t)$, we believe its animation to appear visually very different.

6.3.2 Amplitude-Weighted Average Frequency

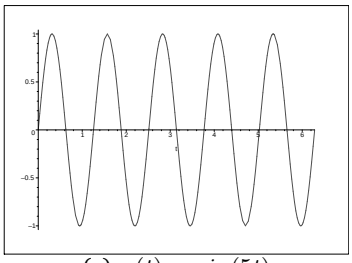
The key observation regarding the signals $x(t)$ and $y(t)$ is that they both consist of a single frequency, while the average signal $avg(t)$ contains two frequencies. A more intuitively correct average signal is obtained by averaging of the frequencies present $x(t)$ and $y(t)$, i.e. $\omega = 4$ and $\omega = 5$, as well as averaging the amplitude present in the spectra for $x(t)$ and $y(t)$. This notion of average yields the spectrum shown in Figure 6.7(a), which transformed to the time domain corresponds to the signal $avg_2(t) = sin(4.5t)$, shown in Figure 6.7(b) on page 103.



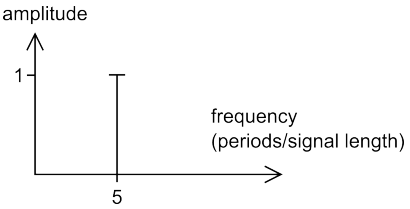
(a) $x(t) = sin(4t)$



(b) Spectrum for $x(t)$.

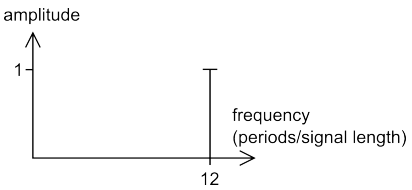


(c) $y(t) = sin(5t)$

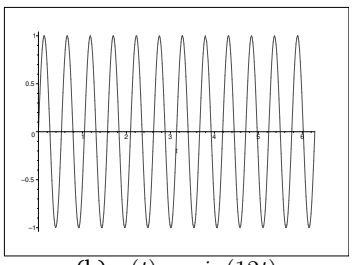


(d) Spectrum for $y(t)$.

Figure 6.4: Two signals and their associated frequency spectra.

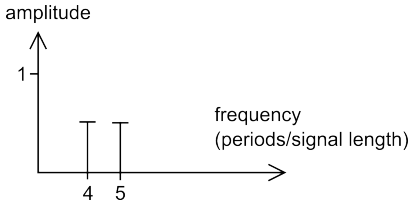


(a) Spectrum for $z(t)$.

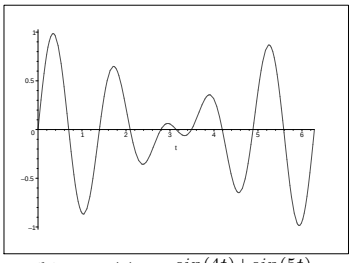


(b) $z(t) = sin(12t)$

Figure 6.5: A signal with high frequency compared to $x(t)$ and $y(t)$, and its associated frequency spectrum.

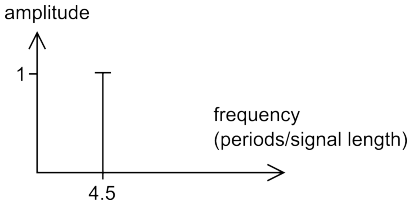


(a) Spectrum for $avg(t)$.

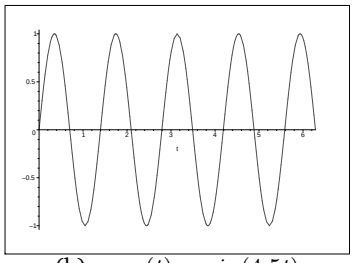


(b) $avg(t) = \frac{sin(4t)+sin(5t)}{2}$.

Figure 6.6: The average spectrum and signal obtained by averaging $x(t)$ and $y(t)$ using the simple average defined in the spectral animation descriptions.



(a) Spectrum for $avg_2(t)$.



(b) $avg_2(t) = sin(4.5t)$

Figure 6.7: The average spectrum and signal obtained by the intuitively correct average of $x(t)$ and $y(t)$.

In the following, we define the *amplitude-weighted average frequency* and a corresponding average amplitude.

Let $X_1(\omega), X_2(\omega), ..., X_M(\omega)$ denote the M spectra to be averaged (omitting phase), and $\omega_1, \omega_2, ..., \omega_N$ denote the N frequencies contained in these spectra.

The amplitude-weighted average frequency in a set of spectra is formally given by:

$$\frac{\sum_{i=1}^{M}\sum_{j=1}^{N} X(\omega_i)\omega_i}{\sum_{i=1}^{M}\sum_{j=1}^{N} X(\omega_i)}$$

while the average amount of total amplitude in a set of spectra is given by:

$$\frac{1}{M}\sum_{i=1}^{M}\sum_{j=1}^{N} X(\omega_i)$$

The amplitude-weighted average frequency of the previously illustrated spectra for the example signals $x(t)$ and $y(t)$ is 4.5 and the average amount of total amplitude is 1. These values correspond with our preferred average of $x(t)$ and $y(t)$ as shown in Figure 6.7(a) on the previous page.

6.3.3 Frequency Bins

Creating an average spectrum using the amplitude-weighted average frequency yields an average spectrum with amplitude in only a single frequency. If the signals to be averaged only contains a single frequency, this is judged a reasonable average. However, if the signals each contain several frequencies, then the average should also contain several frequencies. The solution is to introduce *frequency bins*. A frequency bin is a range of frequencies. When averaging spectra using frequency bins, the idea is to average all amplitudes corresponding to the frequencies within the same frequency bin, as if the bin only contained a single frequency.

When the axis spectra (in either dominating axes or coordinate axes) for a set of triangles are to be averaged, the preferred number of frequencies to be contained in the average spectrum must be determined somehow. According to the tree observations in Section 1.5, Foliage have a tendency of being connected to the highest levels of branches. If the CASTM has a uniform branch depth, then each foliage triangle will typically be animated by an equal amount of bones. These bones represent a combination of large rigid branches, and smaller flexible branches, which are assumed to rotate with different inherent frequencies. Given these assumptions, if a set of triangle axis spectra are to be averaged, the depth of the animation skeleton will represent a suitable number of frequencies in the average spectrum.

Let k denote the number of frequencies judged suitable for the average of M axis spectra, $X_1(\omega), X_2(\omega), ..., X_M(\omega)$, which omit phase values. Let N be the number of frequencies in each spectrum. To compute which frequencies to group together in a frequency bin, a clustering approach is employed.

Initially, the k frequency bins are distributed uniformly, such that the bin, B_1, contains the first $N div\ k$ frequencies, B_2 the next $N div\ k$ frequencies, and so forth. The final bin B_k contains the remaining $N div\ k + N mod\ k$ frequencies. Each bin is considered a cluster, and the K-means Clustering algorithm is used to find a potentially better distribution of frequencies to bins. The patterns to cluster are frequencies, the centroid of a cluster is its amplitude-weighted average frequency, and the distance of a pattern and a centroid is simply the difference between the two frequencies.

When the frequency bins have been computed, they can be used to yield a suitable average of a set of axis spectra. To compute an average spectrum using the frequency bins, we first compute the amplitude-weighted average frequency for each bin, and round it to the nearest of the N original DFT frequencies. At each of these bin centroid frequencies, the amplitude equals the average amount of total amplitude at the frequencies in the bin for the M spectra.

As an example, consider the two axis spectra in Figure 6.8(a) and 6.8(b). Each contains three frequencies, and hence $k = 3$. Figure 6.8(c) illustrates the total amplitude content of the two spectra, as well as the initial frequency bins. The k-means clustering reassigns frequency number 5 (the lowest frequency of bin 3) from B_3 to B_2, as it is closer to the centroid of B_3. The new (and final) clustering is illustrated in Figure 6.8(d). The average spectrum is shown in Figure 6.8(e), in which the centroids are rounded to the nearest DFT frequency, and given the average total amplitude within the associated bin in the original spectra.

The average contains only three frequencies, one for each bin. As both original spectra had high amplitudes at the low frequencies, so does the average. The first spectrum has three large amplitudes at the mid-frequencies, while the second spectrum has amplitude at the high-frequencies. Consequently, the amplitudes in these frequency areas has been scaled down in comparison to the low frequency area.

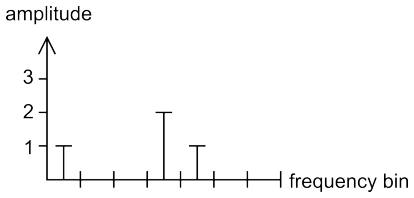
Note that frequency bins could also be employed as a more suitable deviation measure between two spectra. Frequency bins are created by considering the amount of frequencies with amplitude > 0 in the two spectra, and the deviation between the two spectra would equal the difference in amplitude within each bin.

6.3.4 Application of the Improvement

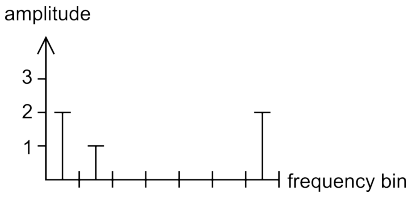
The frequency clustering improvement can be applied to each of the presented animation description types. It should be used in both the $deviation$ and $average$ functions, when the spectra for some axis are compared.

The improvement should furthermore be applied to the animation error metrics presented in Section 5.1.4 on page 74, in order for these metrics to give better comparison of the original and simplified animation.

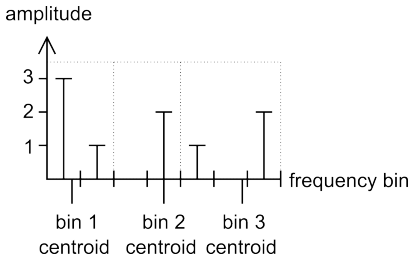
Phase has been ignored in the frequency clustering improvement description, but the description types relying on phase values should define a suitable phase value for each bin. A suitable phase value for a bin could be the amplitude-weighted average phase within the bin.



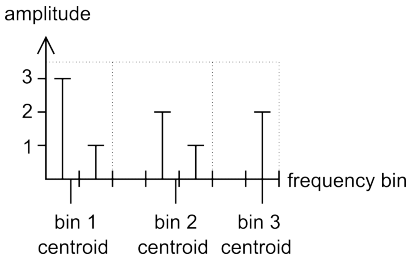
(a) Spectrum A.



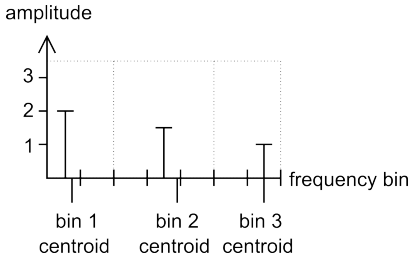
(b) Spectrum B.



(c) Total amplitude for A and B, and initial frequency bin distribution.



(d) Bin distribution after frequency clustering.



(e) Average spectrum of A and B obtained using frequency clustering.

Figure 6.8: Computing an average spectrum using frequency clustering.

6.4 Shared Bone Improvement

The shared bone improvement can provide more precise animation of the billboards in the simplification obtained by the Spectral Clustering solution using the assumed animation skeleton. The improvement can be included as an optional parameter in an implementation.

The animation of a triangle is defined by a sequence of connected bones in the animation skeleton, the bone to which the triangle is assigned, and all ancestors of that bone all the way to the root of the skeleton. The observation is that if all the triangles simplified by a billboard have a set of shared bone ancestors lower in the skeleton, then a more precise animation of the billboard can be achieved by animating the billboard directly using these shared bones. The likelihood of all triangles simplified by a billboard sharing bone ancestors is fairly high, as such triangles are likely to have similar spectral descriptions, due to them being rotated by some common bones.

An animated billboard cloud obtained using the shared bones improvement is animated using both vertex animation and skeletal animation. Vertex animation is used when applying the average animation of the triangle animations defined by bones that are not shared by all triangles, and skeleton animation is used when applying the animation defined by shared bones.

Formally, the improvement is applied by changing the final step in the spectral clustering solution algorithm to the following:

1. For each triangle simplified by a billboard, define a set containing the bone to which the triangle is assigned as well as all the ancestor bones all the way to the root bone.
2. Take the intersection of the defined sets of bones.
3. If the intersection is empty, then the triangles simplified by the billboard have no shared bones and no further actions are taken.
4. If the intersection is not empty, a new spectral description is computed for each triangle. In this new spectral description, the animation of each triangle is generated by all ancestor bones that are *not* in the set of shared bones. The animations of the shared bones are ignored.
5. An average of these descriptions is calculated as usual, and the billboard is vertex-animated by applying the average of the spectral descriptions to the billboard. The bones shared by the triangles define a path of bones starting at the root. The billboard is attached to the shared bone furthest from the root.

In general, this improvement yields an animated billboard cloud that more precisely approximates the original CASTM since parts of the original animation are kept unsimplified. In the worst case, no billboard simplifies triangles that share bones, which consequently means that the improvement has no effect. The cost of the improvement is that the animation skeleton has to be included in the simplification in addition to the cost associated with performing the skeletal animation at run-time.

6.5 Analysis of the Simplification

The spectral description type is crucial for the Spectral Clustering solution, and we expect the visual fidelity to vary a great deal with the different presented spectral description types. Each of the presented types has shortcomings, and the quality of a solution using a given type is not easy predictable.

6.5.1 Spectral Description Types and Error Metrics

We do not expect the Naïve Animation Description Type to yield good results. Visually similar animations, such as two time-offset animations, might have just as much deviation as two animations with entirely different shapes or orientations. Consequently, we expect triangles with significantly different trajectories to end up on the same billboard, and thus yield an unimpressive simplification. Furthermore, we do not expect the animation obtained by simply averaging the amplitude and phase values of each frequency to be representative for the animations being averaged. In terms of error metrics, we expect these shortcomings to result in all-round high error values.

Triangles with visually similar animations are more likely to get low deviation using the The Dominating Axis Description Type. However, this description type is only suitable under the assumption that foliage triangles have line-shaped trajectories. When using this type, triangles are clustered based on the dominating axis animation error metric, and using this type is naturally expected to yield low error according to that metric. If the triangles have non-line-shaped trajectories, we expect an increase in the position and colour metrics. The metrics will not, however, be able to detect artificial-looking animation due to each billboard only moving along a single axis.

The final type, the Axis-phase Description Type, is expected to yield the most impressive simplification, if it is able to approximately separate animations with different shapes and orientations, whilst retaining this shape and orientation when averaging the animations, as claimed above. How this type scores in the error metrics as opposed to the Dominating Axis Description Type is hard to predict, and depends on the nature of the given input.

6.5.2 Inherent Frequencies

As noted in Section 1.5 on page 29, different branches on trees rotate with certain inherent frequencies due to their rigidity. The small branches at the outer levels of the tree are more flexible and perform rotations with higher frequencies than the large branches close to the trunk.

We believe that an advantage of using spectral analysis of animation is that different inherent frequencies are directly present in an axis spectrum (be it the dominating axis or a coordinate axis). The deviation and average measures in the spectral description types compare and average the different frequency ranges independently, and does not discard any inherent frequencies. As a result, the animated billboards simplifying foliage will also contain these inherent frequencies, and appear as if affected by branches of different rigidity.

6.5.3 General Shortcomings

All animation of foliage triangles in the input CASTM is defined by sequences of rotations. None of the defined description types analyze animation in terms of rotation, and does not attempt to preserve the rotational nature of the input animation. As a consequence of this, we expect the simplification to look somewhat artificial. In order to remedy this problem, a rotation-based animation description type should be designed.

As mentioned, Spectral Clustering solely considers the foliage part of the input CASTM. As a consequence, the simplified foliage will be animated separately from the branches in the mesh, which might yield a noticeable visual artifact. This is a severe limitation of this simplification approach. However, as mentioned in the tree observations in Section 1.5 on page 29, foliage may hide the underlying branch structure in certain tree species, which might render this problem slightly less severe. Still, the presented problems may limit the usefulness of the Spectral Clustering algorithm for producing LODs that are to be observed at short range. However, they may not be important for visual fidelity at larger distances, in which case we judge the overall speed and distance of the foliage animation more important. For large distance LODs, the Spectral Clustering algorithm is considered appropriate.

Chapter 7

Skeleton Billboard Cloud Simplification

The Skeleton Billboard Cloud Simplification solution is a solution following the Bone Clustering strategy presented in Section 5.2.3 on page 79, and thus assumes that the tree model to be simplified is animated using skeletal animation. The bones in the skeleton defines the animation of all triangles in the foliage, and this solution only examines the animation of bones.

Bone clustering encompasses performing a simplification of the animation skeleton that reduces its number of bones. A static billboard cloud is then created for each bone and the resulting billboards are faithfully animated by attaching them to their bones. As mentioned, we have implemented the error-based Stochastic Billboard Cloud algorithm for the construction of billboard clouds of the foliage geometry of the input CASTM, due to our argumentation of its effectiveness for foliage simplification from Section 3.7 on page 55.

Simplification of the skeleton is the interesting part of a solution following the Bone Clustering strategy, as the static billboard clouds are created using a traditional billboard cloud algorithm, and animating the billboards is trivial; they are simply attached to the bones. Common joint rotation simplification is discussed in Section 7.1 as our primary simplification strategy. Common joint rotation simplification relies on the definition of a bone animation description type, which is given in Section 7.2. In Section 7.3, a sibling bone collapse strategy is introduced to reduce the number of joints in the skeleton, and thus the number of billboard clouds even further. When performing sibling collapses, the assumed correspondence between the skeleton bones and the trunk mesh is lost. This observation will be discussed in Section 7.4. The algorithm of the Skeleton Billboard Cloud Simplification solution will be presented in Section 7.5. The chapter is concluded with an evaluation of the simplification performed by the solution in Section 7.6 and a relation to the Spectral Clustering solution in Section 7.7.

7.1 Common Joint Rotation

In Section 5.2.3 on page 79 it was argued why a static billboard cloud can be created for each bone in the animation skeleton, and correctly animated by attaching the billboards to their associated bone. The obvious limitation of this approach is that the number of billboard clouds cannot be reduced further than the number of bones. Remember that given some maximum allowed permutation distance, ϵ, it will imply more billboards, if the model is divided into separate parts, and a billboard cloud created for each (discussed in Section 5.2.3 on page 79). Common joint rotation simplification makes it possible to reduce the number of separately constructed billboard clouds, without performing any bone collapses.

Assume that a suitable bone animation description exists. A such includes $deviation(b_1, b_2)$ to compute the deviation in rotation animations between to sibling bones, b_1 and b_2, and $average(B)$ to compute the average rotation animation for a set of sibling bones, B. The procedure of common joint rotation simplification, CJR, is as follows. Let M be the number of bones in the skeletal tree model, $TreeModel$, and let BC be the desired number of billboard clouds:

$CJR(TreeModel, BC)$

1. S defines a set of common rotation sibling bone sets, for each of which a billboard cloud is to be created.

   Each bone b_i initially defines its own common rotation set $C_{b_i} = \{b_i\}$, and we hence initially have $S = \{C_{b_1}, C_{b_2}, ..., C_{b_M}\}$.

2. If $\mid S \mid \leq BC$, then the desired number of billboard clouds can be obtained, and the algorithm terminates.

3. For each pair (C_1, C_2) in S, where all bones in the set $C_1 \cup C_2$ are siblings, compute:

   $$deviation(average(C_1), average(C_2))$$

   If all bones in $C_1 \cup C_2$ are not siblings, then we define deviation to ∞.

4. Let $(C_{1_{min}}, C_{2_{min}})$ denote the pair with smallest deviation, dev_{min}.

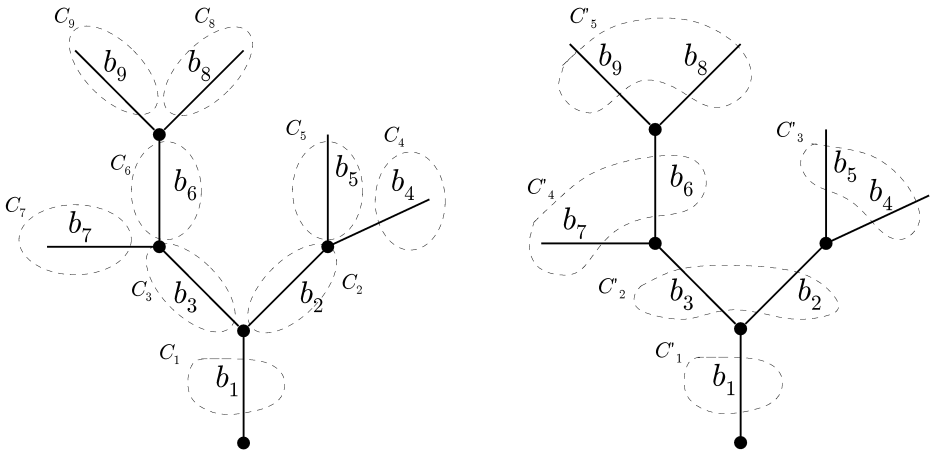
   If $dev_{min} = \infty$, then a single common rotation is already defined for all siblings in the skeleton, and the algorithm terminates, since no further simplification is possible.

5. $C_{1_{min}}$ and $C_{2_{min}}$ are replaced in S with their union $C_{1_{min}} \cup C_{2_{min}}$, which in effect reduces $\mid S \mid$ with one.

6. Go to step 2.

The intuition behind the algorithm is to iteratively find sibling bones with most similar rotation animation, and replace their animation with a single simplified one. The algorithm attempts to reduce the number of billboard clouds needed for simplification from the number of bones to the specified number BC. However, as already mentioned, using common joint rotation simplification, the number of billboard clouds cannot be reduced further than the number of joints in the skeleton. The success of common joint rotation simplification depends on whether a suitable bone animation description can be defined, which is discussed in Section 7.2.

The result of the algorithm is $S = C_1, C_2, ..., C_{BC}$. Each element, C_i, is a set of bones, for which a single average rotation animation is defined by $average(C_i)$. A single static billboard cloud is then created for all the triangles attached to the bones in the set, and each billboard animated using the average animation. Figure 7.1 illustrates the common rotation sets, C_i, before and after running the algorithm on a simple skeleton.



(a) A simple skeleton with nine bones. Each bone b_i initially defines its own common rotation set, C_i.

(b) The common joint rotation sets after running the CJR algorithm with $BC = 5$.

Figure 7.1: Common joint rotation sets.

It is interesting to consider how many triangles get their animation simplified, when simplifying the animations of several bones in a set C_i to a single animation. When a terminal bone replaces its rotation animation with an average sibling animation, this simplifies the animation of all triangles directly attached to it. When a non-terminal bone replaces its animation with an average sibling animation, this simplification influences not only the triangles directly attached to it, but also all triangles attached to its descendants.

An important property of this simplification is that it does not alter the skeleton structure, only its rotation animations. The closed mesh representing the trunk and branches in the tree model can therefore also be animated using the simplified skeleton. This guarantees that the foliage will stay visually connected to the trunk and branches in the simplification using billboard clouds.

We judge it possible to create common rotation sets interactively, and ob-

serve the visual impact of each animation simplification step before creating the billboard clouds. The only processing required is attaching all triangles that are attached to one bone, to another. A manual simplification approach is then to let the user select which common rotation sets to unite, based on his visual judgement, instead of automatically uniting the least deviation sets. Although we do find a such approach interesting, we focus on automated simplification using a deviation measure.

7.1.1 Application of a Common Rotation

The sibling bones in a set C_i all need to apply the common animation, $average(C_i)$, in order for their triangles to share billboards. The straight forward way to do this is simply to change the animation function of each bone to the result yielded by $average(C_i)$. There are, however, other ways to do this, and improvements to be made.

When sibling bones are given identical animation, some bones might become redundant, and can be removed from the skeleton. Different scenarios occur:

- C_i contains terminal bones only: If all bones in C_i are given the $average(C_i)$ animation, then they become identical apart from having different sets of triangles associated. Remember that a terminal bone has no length, nor initial orientation, as it does not need to specify the position of an end joint.

  Having several identical bones is a waste and they can be replaced by a single new bone performing the $average(C_i)$ animation, with the union of their triangle sets associated.

- C_i contains a single non-terminal bone and any number of terminal bones: When the bones are given identical animation, having several identical terminal bones becomes a waste of resources. They can be deleted from the skeleton and their triangles assigned to the non-terminal bone, without any visual impact.

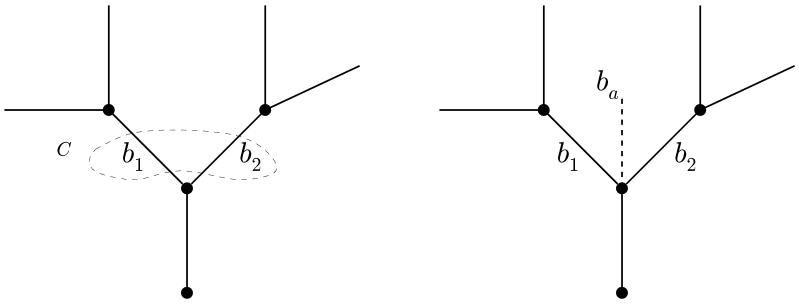
- C_i contains several non-terminal bones: Non-terminal bones cannot be deleted from the skeleton, even though they are given identical rotation animations, as they still specify different end joint positions (i.e. start joints for their respective children).

  Any terminal bones in C_i can be deleted by assigning their triangles to any of the non-terminal bones, but all non-terminal bones must be kept.

7.1.2 Average Animation Bones Improvement

Recall, when the animation of a non-terminal bone is simplified, this simplification influences not only the triangles directly attached to it, but also all triangles attached to its descendants. We propose a method which can be used to reduce the amount of triangles influenced when simplifying the animation of

non-terminal bones. When C_i contains several non-terminal bones, then a new *average animation bone*, b_a, is added to the skeleton, the role of which is to apply the average animation to all the foliage triangles directly attached to bones in C_i. The triangles directly attached to bones in C_i are attached to b_a instead. Any terminal bones in C_i are deleted from the skeleton, while non-terminal bones are kept in the skeleton with their original animation. By keeping the original non-terminal bones, the simplification only influences the foliage triangles directly attached to them, and not those attached to descendants. The average animation bone improvement is illustrated in Figure 7.2.



(a) Two non-terminal bones b_1 and b_2 ended up in the same common rotation set C, as a result of the CJR algorithm.

(b) Instead of replacing their animation with $average(\{b_1, b_2\})$, a new terminal bone b_a is added and given this animation instead. The foliage triangles of b_1 and b_2 are then attached to b_a.

Figure 7.2: An example illustrating the average animation bone improvement.

As animation simplification of descendant foliage triangles is avoided, it is to be expected that this improvement will imply better visual metric scores. If the only simplification done in the example in Figure 7.2 is the one involving b_1 and b_2, then the animation of their descendants will be unchanged, thus receiving perfect metric scores. The immediate judgement is hence that introducing average animation bones will result in a simplification of higher visual fidelity. There are problems, however.

The improvement will result in a visual artifact, as the triangles previously attached to b_1 and b_2 will rotate independently of their descendant triangles. The visual result is that the descendant triangles appear unconnected to their parents. For the unconnected foliage triangles, this might be acceptable in some simplifications, but the trunk mesh will become broken and unconnected. For this reason, we only propose to apply the average bone improvement to the foliage triangles, i.e. the foliage triangles are detached from their respective bones and attached to the new average animation bone, while the trunk mesh triangles are attached to the original bones. The result of this is, however, that the trunk mesh and foliage at this joint is animated independently, which will make the foliage appear unconnected to the branches of the tree model.

To summarize, the effect of using the average bone improvement is:

- When replacing the animation of a non-terminal bone with an average

animation, error does not propagate to their descendants when using the average animation bone improvement. In general, a higher visual metric score is expected.

- The foliage triangles become disconnected from the branches of the tree, which in some cases will be a highly noticeable artifact.

We suspect that the visual result of applying the improvement depends on many factors, such as the given tree model, how much simplification is done, and at how far a distance the simplified tree model is to be observed. When, and if, the improvement actually can improve on the visual result is subject to experimentation.

7.2 Axis-Angle Description Type

The CJR algorithm depends on the ability to evaluate the deviation between the rotation animations of two sibling bones, and likewise to construct an average rotation animation from a set of sibling bones. In other words, we need to define a bone animation description type.

The rotation animation of a bone is specified by a set of key frame rotations during the animation cycle, indicating the rotation of the bone at these key frame time samples. Any of these rotations specify an arbitrary rotation around the joint the bone is contained in, and thus a rotation of a local coordinate system specified at this joint. The key frame rotations are interpolated using a spherical linear interpolation scheme to yield the rotation at any given time [30].

7.2.1 Choosing a Rotation Representation

One approach to comparing rotation animations is to consider the rotations in a fixed angle representation. This refers to describing any rotation as the product of rotating around three fixed axis, e.g. the x, y, and z axes of the world coordinate system. DFT could be applied on each of these rotations separately, and spectrum deviation and average could be used as discussed in Section 6.1 on page 87. A problem arise, however, since more than one set of world axis rotations result in identical rotations [30]. In other words, a fixed angle representation of rotation is not unique, and two identical rotations can be described by two potentially very different fixed angle representations. For example, applying the fixed angles $(\pi/2, \pi/2, \pi/2)$ is equal to applying the fixed angles $(0, \pi/2, 0)$. This makes the representation unsuitable for comparing rotations.

Any rotation in three dimensions can be thought of as a rotation around an axis, as there is a set of points which does not change positions. These points are collinear and the line is called the *axis of rotation*. We propose an animation description type which is based on axis-angle representation of the rotations of bones, since it represents a rotation uniquely.

7.2.2 Axis-angle Rotation Animation

A rotation animation $R(t)$ equals a pair $\langle axis(t), \phi(t) \rangle$, where $axis(t)$ denotes the axis of rotation at time t, and $\phi(t)$ denotes the angle of rotation at time t around this axis. The rotation axis contains the origin, because it is a rotation of a local coordinate system defined at the joint the bone is contained in, and an axis can hence be represented by a vector. The position of a vertex expressed in the local joint coordinate system is at time t found by rotating it from its initial position in the bind pose (see Section B on page 158), with the angle, and around the axis, specified by the rotation animation $R(t)$ of the bone it is attached to.

Sibling bones specify rotations of the same local coordinate system, as they are contained in the same joint, and the axes of rotations specified by the animations hence always contain the origin. We want to analyze and compare the rotation animations specified by two sibling bones in the skeleton. This problem is directly related to the animation analysis of trajectories discussed in Chapter 4 on page 57, and it is interesting to consider the terms time-offset, speed, distance, and shape in the context of rotation animations.

Time offset makes a similar sense as it did with trajectories, i.e. two rotation animations are time-offset equal if there exists some time offset, o, such that:

$$axis_1(t) = axis_2(t + o) \text{ and } \phi_1(t) = \phi_2(t + o)$$

We assume that time-offsetting a rotation animation will have little visual impact. Consider how a rotation animation rotates the axes of the local coordinate system at its joint. Speed and distance refer to *how fast*, and *how large angles*, the axes of the local coordinate system are rotated. Consider the unit vectors aligned with each of the axes in the local coordinate system. During the rotation animation, each unit vector end point moves on a unit sphere centered around the origin, and the *shape* of the rotation can be considered the patterns of the paths travelled by the three unit axes end points. In one rotation animation, the unit axis could move in a circular pattern, for example.

7.2.3 Deviation and Average

Intuitively, we want to define a deviation measure for the rotation animations of sibling bones, from which we can conclude time-offset equal rotation animations to be equal. Furthermore, similar rotation animations, albeit having minor speed, distance, and/or shape dissimilarities, should be concluded similar. In this section we present definitions of simple deviation and average methods that do not take into account time-offset equality, etc. Proposals for how to improve the simple methods using some of the ideas presented in the animation analysis section will be discussed in Section 7.2.6.

A deviation measure between two rotation animations, $R_1(t)$ and $R_2(t)$, can be obtained by considering the angle between the axes of rotations and the difference in angle of rotations.

Deviation

Let the animation cycle length be given by c_t and number of samples by N. The deviation of axes is then given by:

$$d_{axis}(R_1(t), R_2(t)) = \frac{\sum_{i=1}^{N} \mid axis_1((1/i)c_t) \cdot axis_2((1/i)c_t) \mid}{N}$$

Put simply, the deviation in axes equals the average deviation in angle between the two axes during the animation cycle. The functions, $axis(t)$, yield normalized direction vectors of axes, hence the cosine to the angle between the axes is simply obtained by a dot product. Since we do not want to differentiate between an axis and the reverted axis, the absolute value of the dot product is considered. d_{axis} is a number in the range $[0; 1]$, where 0 means identical axes.

The deviation of the angle of rotation functions, $\phi_1(t)$ and $\phi_2(t)$, is found in a similar fashion, by finding the average difference in angle during the animation cycle:

$$d_{phi}(R_1(t), R_2(t)) = \frac{\sum_{i=1}^{N} angleDifference(\phi_1((1/i)c_t), \phi_2((1/i)c_t))/\pi}{N}$$

The $angleDifference$ function yields the difference between two angles, taking the cyclic range $[0; 2\pi]$ into account (cf. Appendix A.4).

The total deviation between two animations is a combination of the axis and the angle deviation:

$$deviation(R_1(t), R_2(t)) = d_{axis}(R_1(t), R_2(t))d_{\phi}(R_1(t), R_2(t)) \tag{7.1}$$

Average

The average rotation animation, $R_a(t)$ of a set of rotation animations, A, is obtained in a similar fashion as deviation, namely by considering N samples uniformly distributed over the animation cycle, and then consider the axis of rotation and the rotation angle separately.

The axis of rotation function of the average rotation animation $axis_a(t)$ is found by averaging the axis of rotations given by the rotations in A at each of the N time samples. Axes of rotation are considered as lines containing the origin, and averaged by finding a best average line for this set of lines. To obtain the axis for any given time, the axis for each time sample is interpolated using a spherical linear interpolation scheme.

The average angle of rotation function, $\phi_a(t)$, of the rotation animations in A, is likewise obtained by averaging the angle of rotation at each of the N time samples. Linear interpolation is applied to obtain the rotation angle for any time. Note that since angles are in the cyclic range $[0; 2\pi]$ they should be averaged as described in Appendix A.4.

The bones to be averaged are likely to animate different amounts of triangles. Remember that a bone animates the triangles attached to itself or any of

its descendants. When determining the average rotation animation, bones that animate few triangles should not have as much influence as bones animating many triangles. For this reason, we weight the axes and the angles with the amount of triangles the respective bones animate. The weighting of the angles, for example, is done by scaling the length of the vectors summed in order to find their average.

Having defined deviation and average, a description type can be defined. The simple means for finding deviation and average presented in this section has several problems, as will be discussed in Section 7.2.5.

7.2.4 Axis-angle Description Type

The four functions constituting the axis-angle description type are:

$description(b)$: An axis-angle description, d, of an animated bone, b, equals the rotation animation $R(t)$ performing its animation.

$deviation(d_1, d_2)$: Deviation between two axis-angle descriptions, $d_1 = R_1(t)$ and $d_2 = R_2(t)$, is given by Equation 7.1 on the previous page. Simply put, the animation cycle is sampled into N uniformly distributed samples, and the deviation is derived from the average angle between the axes of rotation, and the average difference in rotation angles, at these N samples.

Note that this deviation only applies for sibling bones. The deviation between the animation descriptions for two non-sibling bones is defined to be ∞.

$average(D)$: Given axis-angle descriptions for a set of sibling bones, D, the average description is obtained by considering N samples uniformly distributed during the animation cycle. At each sample, the axes of rotation in D are averaged, and likewise are the rotation angles in D, each weighted with how many triangles the respective bones animate. The average axes and angles at each of the N frames are linearly interpolated to yield the actual average description, $R_a(t)$.

Average is *not* defined for non-siblings.

$apply(b, d)$: An axis-angle description of the rotation animation of a bone does not discard any information, and a description, d, can directly be applied to any bone, b.

7.2.5 Description Type Shortcomings

As stated, it is expected that time-offsetting a rotation animation in the skeleton will not impact much on the overall visual appearance of the animated tree model. For this reason, the deviation between two sibling bones performing time-offset equal animations should be small, as a visually satisfying average is obtained by applying their time offset to one of the bones, in effect giving them identical animation.

The simple axis-angle description type presented does not take this into account. By considering the angle between the rotation axes at different time samples, and the difference in angle at these time samples, two time-offset equal animations are potentially concluded to be very different, as they do not have similar axis and angles at the considered time samples. In relation to the animation analysis discussion earlier, the simple approach to comparing rotation animations represented by the axis-angle description type corresponds to the simple Euclidean animation analysis of trajectories presented in Section 4.4.1 on page 61.

In addition to not handling time-offset equality in a reasonable manner, and to make things even worse, the description type does not prevent bones from "out-phasing" each other. The average of two rotation animations might hence become an animation rotation that does not perform any rotation at all. In other words, using the simple presented average will have a tendency to remove animation, which we do not judge satisfying in terms of the visual fidelity of the simplification.

More shortcomings could be discussed, but the above already justifies the discussion of alternative approaches to follow.

7.2.6 Discussion of Advanced Approach

In this section we present ideas for how more suitable means for deviation and averages can be defined. In Section 4.4.2 on page 62 it was argued that spectral analysis of trajectories can be used to conclude time-offset equal trajectories similar, as well as trajectories with small speed and/or distance dissimilarities, by considering their frequency spectra. It is interesting to consider whether a similar spectral approach can be taken when analyzing rotation animations instead of trajectories.

One naïve approach is to consider a rotation animation as a set of three rotations around the world coordinate axes, and then apply DFT to yield a spectrum for each rotation. As mentioned earlier, however, a single rotation can be represented by entirely different world axes rotations, making two rotations hard to compare. For this reason we propose to somehow apply spectral analysis to the axis-angle representation of rotation animation.

The axis of rotation during a rotation animation, $axis(t)$, can be thought of as an animated vector. The length of the vector is of no importance, but its orientation can be expressed in spherical coordinates. We propose to apply DFT on these two angle functions, to yield a frequency spectrum for each. Deviation and average of two axis functions are then given by the distance between, and the average of, their spectra, which is done similarly as presented in Section 6.1 on page 87. Ignoring phase makes two time-offset equal axis rotations obtain zero deviation, although this implies the same problems as it did when analyzing trajectories, namely that the animation might change shape. In the context of rotation, this refers to the shape of the path traveled by the animated vector representing the rotation axis. Changing the shape of the animation of the rotation axis might in worst case yield a rotation animation with little visual resemblance to the original. To remedy this shortcoming of

simply ignoring phase, axis-phase relationships could be employed 6.1.4

In addition to applying DFT on spherical coordinates of the axis, DFT is applied to the angle of rotation function $\phi(t)$. The deviation and average of two rotation animations are thus obtained by considering the deviation and the average of three spectra, two for the axes and one for the angle. All three functions onto which we apply the DFT yield values in a cyclic range. In order for the DFT to provide meaningful spectra, this cyclic behaviour must be somehow taken into account. One attempt is to convert the function values to be continuous and non-cyclic, the details of which will not be discussed further, though. Another issue is present when a rotation axis vector is oriented near a pole in the spherical coordinate representation during the animation. A such rotating around the pole will get unintentionally large spherical coordinate changes over time, and thus large amplitudes in its spectra, when compared to the spectra obtained by analyzing the animation of an axis vector not oriented near a pole during the animation. A simple attempt to minimize this problem, when two rotation animations are compared, is to find the average rotation axes of both animations. The spherical coordinate system is then defined such that average rotation axes are as far from the poles as possible.

We judge a spectral approach for analyzing rotation animations suitable, but the visual artifacts of ignoring phase or using axis-phase relationships are hard to predict. The fact is that ignoring phase will in worst case result in animations clearly rotating around the wrong axes, just as ignoring phase in three world axis spectra for a trajectory can result in animation with completely wrong orientations.

7.3 Non-terminal Sibling Collapse

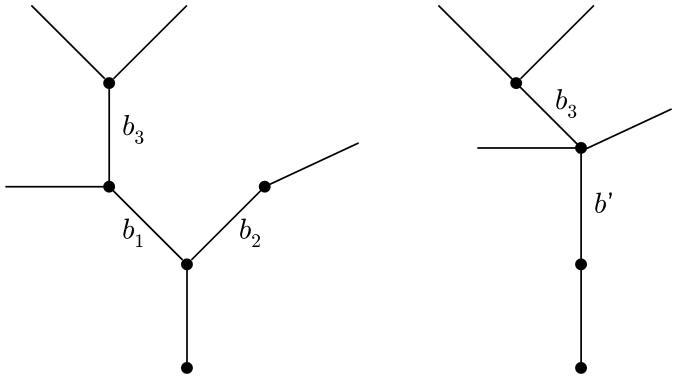
The CJR algorithm reduces the number of billboard clouds needed for simplifying a skeletal tree model using the Bone Clustering strategy. The algorithm cannot, however, reduce the number of billboard clouds further than the number of joints in the skeleton. Recall that an increased number of billboard clouds generally implies more billboards needed for simplification, which is the motivation for reducing it.

In our problem definition, we state that our goal is to derive simplification techniques that can be used to simplify CASTMs to animated billboard cloud LOD models consisting of a reduced amount of triangles. If a polygonal tree model is animated with a high degree of realism, then it might contain a large amount of animation bones and joints, in which case it is not possible to obtain very few triangles using CJR simplification. For this reason, a skeleton simplification procedure is introduced with the purpose of removing joints from the skeleton, and while doing so attempt to alter the animation of the tree the least.

The number of joints in the skeleton is reduced by performing non-terminal sibling collapses. As defined in the strategy, a sibling collapse replaces two siblings with a single bone, which inherits the children of both replaced bones. The single bone is given the average rotation animation of the two non-terminal sibling bones, and their end joints are collapsed into one. The new end joint

contains the bones of the original joints, and these bones are updated to perform new rotations. Terminal bones are not considered, as they do not specify end joints, and hence collapsing such will not reduce the number of joints.

An example of a non-terminal sibling collapse is illustrated in Figure 7.3.



(a) The skeleton before collapsing the two non-terminal siblings b_1 and b_2.

(b) The skeleton after the collapse. b' is the new bone replacing b_1 and b_2.

Figure 7.3: A non-terminal sibling collapse.

Notice that bone b_3 is a non-terminal child bone of b_1, and hence defines an end joint around which its own children rotate. The initial orientation and length of this bone are updated, such that it specifies an end joint at the exact same position in the tree model as before. This guarantees that the rotations of geometry around this end joint remains correct, when the bones b_1 and b_2 are collapsed.

7.3.1 Common End Joint

Recall that an end joint of a bone specifies the start joint of its children bones (cf. the skeletal model presented in Section 1.4.2 on page 26). When collapsing two siblings their end joints will be collapsed into one *common joint*. This single joint specifies a new common joint for the children of both bones. The problem is to determine the best position for this common joint, i.e. determine the joint around which the triangles attached to the children can rotate with as much visual resemblance to the rotations as they did around the original joints.

Finding a reasonable common end joint is not a trivial task. One might expect the midpoint between the two end joints to be a reasonable common joint for all triangles rotating around these joints, but, as Figure 7.4 on the next page shows, this is not the case in general.

Figure 7.4 on the following page illustrates a bad common end joint, since the rotation animations performed by the triangles cannot be expressed well as rotations around the common end joint. A better common end joint is illustrated in Figure 7.5 on the next page.

In the CJR algorithm, only the bones are considered, as all the triangles

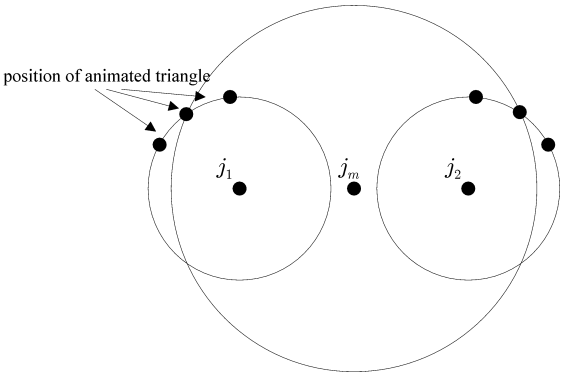


Figure 7.4: The midpoint between the two end joints is not necessarily a reasonable common end joint.

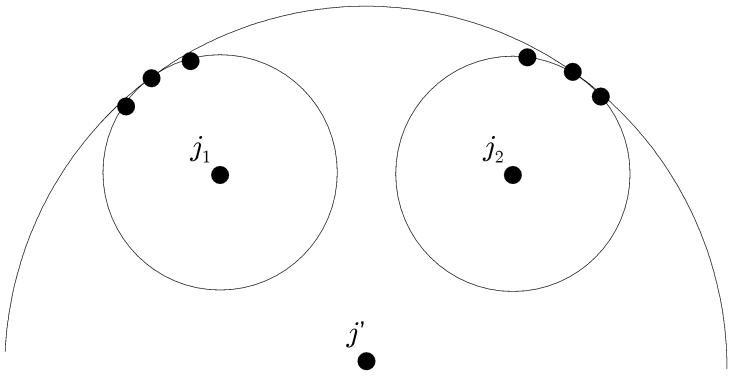


Figure 7.5: A more reasonable common joint.

attached to a bone perform the exact same rotation. The above examples illustrate that when finding a common end joint for two siblings, the positions of the triangles animated by the children bones are essential.

7.3.2 Reasonable Common End Joint Definition

In the following, we will make a definition of a reasonable common end joint for two non-terminal sibling bones being collapsed. All triangles attached to any descendant of the collapsed bones are considered, as these are the triangles performing a rotation around the end joints of the collapsed bones. The triangle orientation is ignored, only the center of a triangle is recorded, and thus a single animated triangle is represented by a trajectory.

First we consider how to convert the rotation animation of a triangle to one around a new joint. The rotation animation of a triangle yields a trajectory, and the centroid (average position) of this trajectory can be computed. From this centroid and a new rotation joint, a *centroid rotation sphere* can be defined with the joint as center, and the distance between the centroid and the joint as radius. The trajectory is mapped onto this sphere to yield its rotation animation around the new joint, which is illustrated in Figure 7.6.

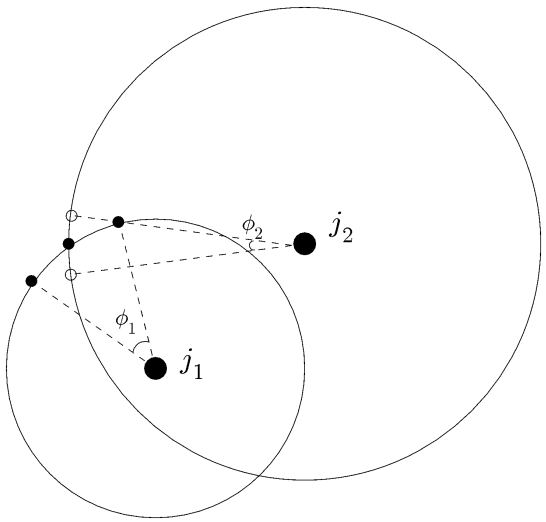


Figure 7.6: A trajectory for a triangle rotating ϕ_1 around joint j_1 is mapped onto a centroid rotation sphere to yield a new rotation, ϕ_2, around a new joint, j_2. The black dots on the circles denote the original positions of the animated triangle, and the white dots denote the positions mapped onto a centroid rotation sphere defined for the triangle centroid and j_2.

We define an error measure for how well a trajectory for an animated triangle, $p(t) = (x(t), y(t), z(t))$, is represented as a rotation around some rotation joint, $j = (j_x, j_y, j_z)$. The centroid rotation sphere has center $c = j$ and radius $r = \mid j - \mathit{centroid}(p(t)) \mid$. The animation cycle is considered at N uniformly distributed samples. The error at a sample, n, can be expressed as the distance between the sample position $p(n)$ and to the sample position mapped onto

the centroid rotation sphere. This error is equal to the difference between the radius r and the distance $\mid j - p(n) \mid$ for each sample n. To avoid the square root in later computations, the squared distance is used.

ε_p denotes the error obtained by animating a triangle with trajectory p around joint j, and equals:

$$\varepsilon_p = \sum_{n=0}^{N-1} \mid r^2 - ((x(n) - j_x)^2 + (y(n) - j_y)^2 + (z(n) - j_z)^2) \mid \tag{7.2}$$

To make ε_p continuous, we square the expression and thus remove the absolute operator:

$$\varepsilon_p = \sum_{n=0}^{N-1} [\ r^2 - ((x(n) - j_x)^2 + (y(n) - j_y)^2 + (z(n) - j_z)^2)\ ]^2 \tag{7.3}$$

For a set of triangles, T, we can also define an error measure:

$$E_T = \sum_{tr \in T} \varepsilon_{trajectory(tr)} \tag{7.4}$$

We can now define the reasonable common end joint for two bones b_1 and b_2. If T is the set of triangles animated by the children of b_1 and b_2, then the reasonable common end joint is the one that minimizes the error measure E_T. This is the joint the triangles in T can rotate around, and each perform the rotation most similar to their original rotation.

Note that E_T is a polynomial. Thus, we can differentiate E_T symbolically and find the roots of the differential. The global minimum is an element in the set of roots, and using this approach we can find the coordinates of the reasonable common joint position. The above approach must be implemented in a software system. The symbolic differentiation can be performed using an algebra system such as GiNaC [4], and the roots of the differential can be found using a numeric computation library such as the GNU Scientific Library (GSL) [15].

7.3.3 Defining the Joint Animation

When sibling collapsing two bones into a single bone, this bone is given their average rotation animation, and set to specify the reasonable common end joint which minimizes the error measure E_T. The created bone inherits the children of both bones, which should thus define their animation around the new end joint.

A naïve approach is to let each children bone keep its original rotation animation, $R(t)$, and simply apply these rotations around the new joint instead. This, however, does not yield satisfactory results. The example in Figure 7.7 on the next page illustrates this.

The example demonstrates that a new animation has to be created for the children of the collapsed bones. Considering a single triangle animated by a bone, its trajectory can be mapped onto the centroid rotation sphere defined

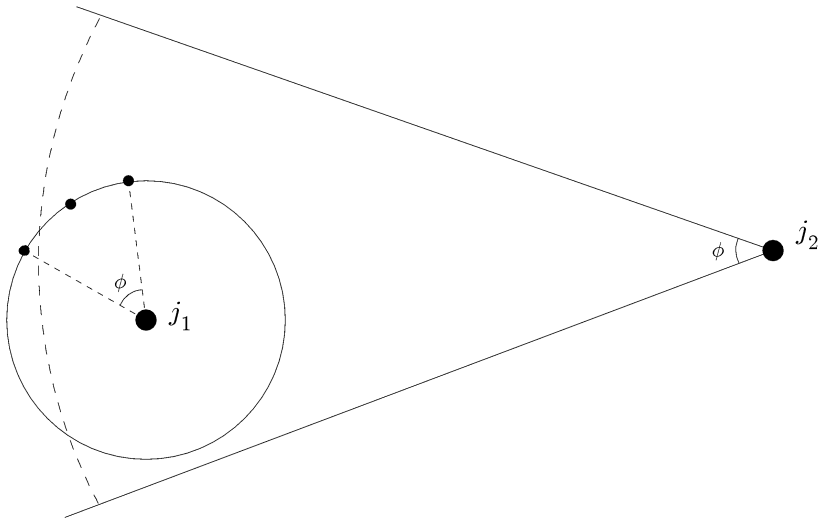


Figure 7.7: A triangle is rotated by an angle ϕ by its bone around joint j_1. If its bone was to rotate around the joint j_2 instead, then its original rotation ϕ would imply the triangle moving the distance illustrated by the dashed arc, which is more than twice the distance originally traveled by it.

for the new joint. This yields a suitable rotation for this triangle around the new joint.

All the triangles animated by the same bone at one of the original joints should still be animated by the same bone at the new joint. These triangles originally performed the same rotation, but they do not necessarily map to similar rotations around the new joint. This is illustrated by the example in Figure 7.8 on the following page.

E_T is minimized to find a joint for which the sum of rotation error, when mapping each individual triangle onto a sphere centered at the joint, is smallest. If each triangle was allowed to define its own rotation around the new joint (rotate ϕ_1 and ϕ_2 instead of a common rotation, in the example), then this joint would indeed be a reasonable common joint for the triangles. However, we do not want to introduce more unique rotations, as such would imply more billboard clouds. The solution is to *average* the rotations obtained by mapping the trajectories of all triangles animated by one bone onto their centroid rotation spheres around the new joint. This average is unfortunately not guaranteed to be a good one.

We believe that a better end joint might be obtained by considering both the error, E_T, introduced by a joint, and how precise an average can be created for each bone rotating around this joint. A such approach will not be discussed further.

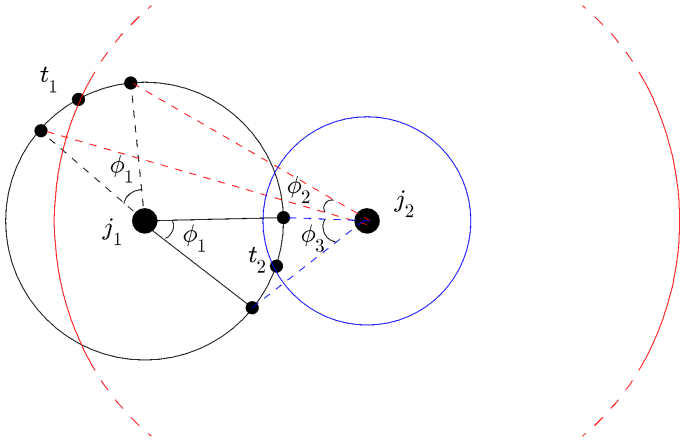


Figure 7.8: Triangle t_1 and t_2 are attached to the same bone, and both perform the rotation ϕ_1 around joint j_1. Around the new joint j_2, t_1 defines the red centroid rotation sphere, and t_2 defines the blue centroid rotation sphere. The two triangles yield different rotations by mapping their trajectories to the respective centroid rotation spheres, namely ϕ_2 and ϕ_3.

7.4 Loss of Bone and Trunk Mesh Correspondence

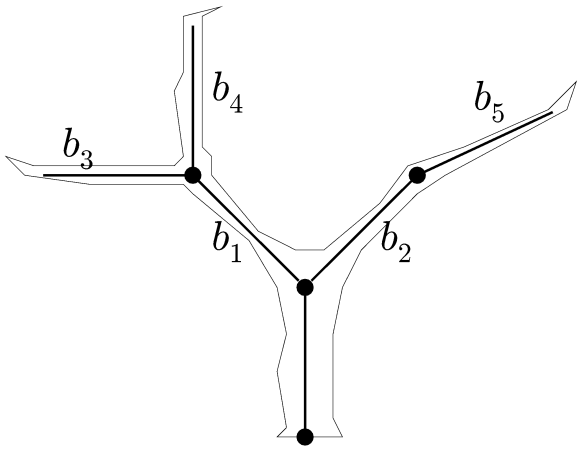
A very important observation is that when performing a number of joint reducing skeleton simplifications based on sibling collapses, the assumed correspondence between the skeleton bones and the trunk mesh is lost. When a collapse replaces two end joints with a new common end joint, a new rotation for each bone at the original joints is defined as a rotation around the new joint. In effect, the triangles attached to these bone will perform a new rotation around a new joint.

This, however, has an unfortunate effect on trunk triangles attached to the bone, as the connected trunk mesh does not retain its connectivity. This is illustrated in Figure 7.9 on the next page.

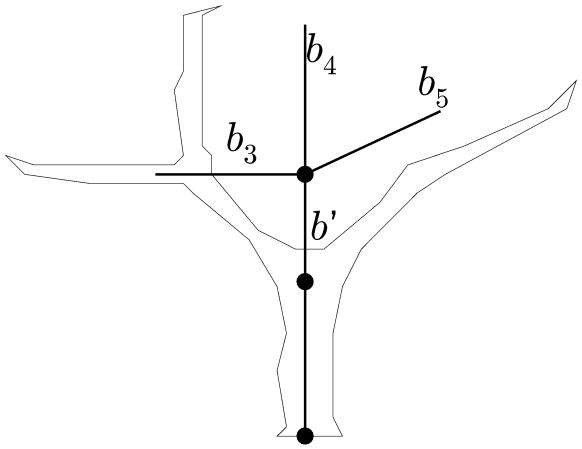
As foliage is modeled by unconnected triangles, rotating such around a new joint does not introduce any connectivity artifacts. When non-terminals are collapsed to reduce the number of joints, our solution is to animate the trunk with the original skeleton to maintain trunk connectivity, and hence only animate the foliage with the new skeleton. Consequently, the foliage and the trunk will animate independently, which can obviously be a noticeable visual artifact. At far away LODs we expect it to be acceptable, though.

7.5 Skeleton Billboard Cloud Simplification Algorithm

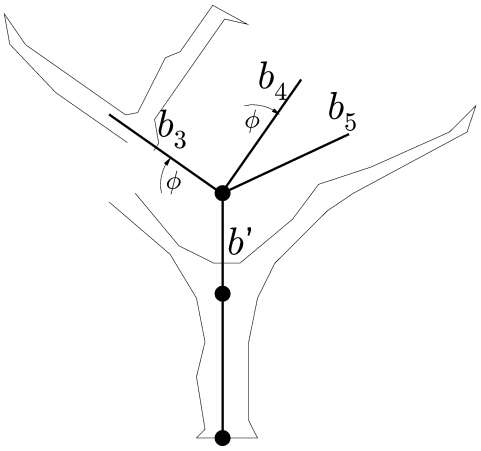
The algorithm of the Skeleton Billboard Cloud Simplification solution, $SBCS$, is one based on both common joint rotation simplifications and non-terminal sibling collapses. Due to the discussed connectivity issues, we propose to perform as much common joint rotation simplification as possible, and only apply collapses of non-terminals when few enough billboards cannot be obtained



(a) A trunk mesh rigged with a simple skeleton.



(b) The two non-terminal bones b_1 and b_2 are collapsed into b', and their children bones are now siblings at the end joint of this bone.



(c) The children bones b_3 and b_4 both rotate ϕ around the new joint, resulting in their associated branches becoming disconnected from the trunk mesh.

Figure 7.9: Losing connectivity.

using common joint rotation simplification alone. We choose to keep the algorithm simple, and briefly discuss a more advanced approach.

The input is a CASTM, denoted $TreeModel$, the desired number of billboard clouds, BC, as well as the parameters taken by the Stochastic Billboard Cloud Algorithm, which have been omitted in the pseudo-code.

$SBCS(TreeModel, BC)$

1. While the number of joints in the skeleton is larger than BC:
   (a) Minimize the error measure E_T (see Equation 7.4 on page 124) for each pair of non-terminal siblings, where T refers to all foliage triangles rotating around either of the sibling end joints. This yields a reasonable common end joint for the two bones.
   (b) Collapse the sibling pair with smallest error E_T.
2. Let $TreeModel'$ denote the CASTM obtaining after performing a number of non-terminal sibling collapses in the previous step. Run $CJR(TreeModel', BC)$ to yield sets of common sibling rotation, $C_1, C_2, ..., C_{BC}$.
3. For each common rotation set, C_i, apply the animation $average(C_i)$ to all the bones in the set.
4. Run the Stochastic Billboard Cloud Algorithm on the foliage geometry attached to bones in each rotation set, C_i.
5. If no sibling collapses were done in Step 1, then output the single CJR simplified skeleton, with both trunk and foliage attached. On the other hand, if sibling collapses were necessary, output a trunk animated with the original skeleton, and a foliage animated with the simplified skeleton.

7.6 Analysis of the Simplification

We conclude this chapter with a discussion of the simplification performed by the Skeleton Billboard Cloud Simplification solution, followed by a comparison with the Spectral Clustering solution.

7.6.1 Quality of the Simplification

The Skeleton Billboard Cloud Simplification solution simplifies the input foliage by creating a set of separate animated billboard clouds. Each billboard cloud is created for a set of triangles attached to the same bone or, in case common joint rotation simplifications or non-terminal sibling collapses are performed, triangles that are attached to siblings. In any case, the billboard clouds are created for triangles attached to bones that are closely related in the skeleton structure, and the billboards in the different billboard clouds will in general not intersect, due to them being ”isolated” from one another. We expect that at least

two or three billboards are necessary per billboard cloud, in order to reduce the risk of the foliage represented by a billboard cloud being ill-represented from certain view angles.

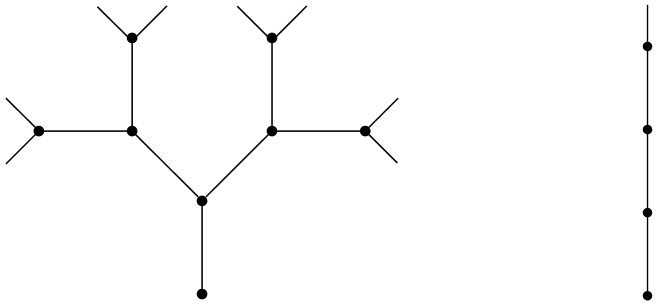
The visual fidelity of the simplifications produced by the solution is greatly dependent on how reasonable a bone animation description type can be defined. The amount of visual error introduced by a common joint simplification depends on the ability to select the pair of sibling bones with rotation animations visually the most alike, and define a reasonable common animation for these bones. We have only defined a very simply bone animation description and argued why we believe a more advanced approach based on spectral analysis of rotation would be better. We expect the quality of the bone animation description type to have a very direct impact on the visual error metrics.

The solution obtains a large reduction of the number of billboard clouds required for simplification by performing non-terminal sibling collapses. The main issue when performing these collapses is to find a common end joint for the two siblings, around which both their children should rotate. We find the position of this end joint by minimizing the error introduced to each individual triangle rotation, when this rotation is mapped onto a sphere centered at the new joint. A shortcoming of this approach is that triangles attached to the same bone map to different rotation animations around the new end joint, and these should be averaged in order to define a single rotation animation for the bone. It is hard to predict the quality of this simplification.

7.6.2 Preservation of Skeleton Levels

An example of an animation skeleton is illustrated in Figure 7.10(a) on the next page. By performing non-terminal sibling collapses, and common joint rotation simplifications, the Skeleton Billboard Cloud Simplification solution cannot simplify the skeleton further than the simplified skeleton in Figure 7.10(b) on the following page. In order to reduce the number of bones further, parent/child collapses should be performed, which are not part of the solution. In the illustrated example the solution can hence reduce the number of billboard clouds needed for simplification from fifteen to four, but not further.

Not being able to simplify further is a limitation of the solution. However, as a consequence of the solution not considering parent/child collapses, the number of levels in the animation skeleton are preserved. Assuming that each foliage triangle is attached to the outer level branches, the animation of each foliage triangle would be defined by n rotations (four, in the example above). These rotations have different inherent frequencies, due to the difference in branch rigidity at different branch levels in a tree (see observations 1.5 on page 29). In a simplification produced by the Skeleton Billboard Cloud Simplification solution, the animation of each foliage triangle is still defined by n rotation animations around different joints, and the inherent frequencies at the different levels are retained, assuming a reasonable bone animation description type.



(a) An example of an animation skeleton before performing simplification.

(b) The simplified animation skeleton after performing as many sibling collapses and common joint rotation simplifications as possible.

Figure 7.10: Preservation of skeleton levels illustrated.

7.7 Comparison with Spectral Clustering

It is interesting to compare the Skeleton Billboard Cloud Simplification solution with the Spectral Clustering solution. Which one is the better is hard to generalize, as it depends on the input model, the applied description types, and whether an animated billboard cloud LOD model is to be observed at close or far range. However, some essential similarities and differences can be discussed.

7.7.1 Similar Input

The two solutions both perform a budget-based clustering of triangles with the number of billboard clouds as budget, which attempts to minimize animation error within each cluster. After this clustering, all triangles in each cluster are assumed suitable for agreeing a single animation. The error-based Stochastic Billboard Cloud Algorithm is then applied in both solutions to construct the actual billboards. Consequently, the two solutions take similar parameters, which makes it easier to compare their results.

7.7.2 Skeleton Rotations

The Skeleton Billboard Cloud Simplification solution constructs a simplification where the foliage billboards are animated using bones. This is judged a great advantage, even when end joints are collapsed, compared to the artificial appearance expected of the simplifications produced by the Spectral Clustering solution, due to not analyzing and averaging animation in terms of rotation.

Both solutions are, however, able to retain the different inherent frequencies in the input model; the Skeleton Billboard Cloud Simplification due to its preservation of skeleton levels, and the Spectral Clustering solution due to its representation of animation using frequency spectra.

7.7.3 LOD Distances

In general, we expect the Skeleton Billboard Cloud Simplification solution to be better at producing precise simplifications to be observed from short range. For example, if only about a halving of the triangle count is needed, this solution can settle with a billboard cloud per bone. Consequently, next to no animation error is introduced, as each billboard is animated precisely as the triangles it simplifies. Also, as long as no non-terminal sibling collapses are needed, the foliage and the trunk mesh can use the same animation, hence avoiding the artifact of foliage not following branches. This artefact is unavoidable using the Spectral Clustering solution, which in itself makes it less attractive for producing LODs to be observed up close. On the other hand, the Spectral Clustering is judged better for producing LODs to be observed from far distances for several reasons.

Assume an animated billboard cloud LOD model where the foliage should only be represented by approximately ten billboards. If the skeleton of the model to be simplified contains more than ten levels, then this alone makes the Skeleton Billboard Cloud Simplification solution unable to produce this LOD. Suppose that the number of levels is lower (it usually is). As an example, say it is five. In this case the solution still has problems. Say that each billboard cloud should contain at least three billboards, in order to avoid bad observation angles. Consequently, this yields a minimum of 15 billboards, which again is more than the desired ten. As a final example, assume only three skeleton levels. The solution is now able to reduce the number of billboard clouds required for simplification to three, after which it can produce a billboard cloud for each bone level containing at least three billboards.

We hypothesise the Spectral Clustering solution to be better than the Skeleton Billboard Cloud Simplification solution for producing the LODs containing very few triangles. The latter produces its animated billboard cloud simplification by constructing a set of separate billboard clouds that simplify sets of localized triangles attached to bones that are close in the skeleton structure. This implies in general that billboards from different clouds do not intersect. The Spectral Clustering solution also creates a set of separate billboard clouds, but an important difference is that the triangles simplified by these are potentially from very different positions and bones in the tree. This difference entails that billboards from different clouds will intersect each other, which helps hiding the two-dimensional nature of the simplification. It also increases the efficiency of the increased foliage improvement. For this reason, the Spectral Clustering solution is judged better at producing simplifications using few billboards, and hence for LODs to be observed from large distances.

Using the Skeleton Billboard Cloud Simplification solution all billboards in each separately created billboard cloud are animated by the same bone and perform thus the same animation. On the contrary, all the billboards contained in a Spectral Clustering simplification have a unique animation, each defined by computing an average of the animations of the triangles the billboard in question simplifies. We judge these independent animations an advantage for LODs containing few billboards to be observed at a far distance, because it gives

an illusion of structural complexity.

Chapter 8

Implementation and Results

Having presented and discussed our solutions in Chapter 6 and Chapter 7 the topic of this chapter is implementation and experiments. We will discuss our implementation of the Stochastic Billboard Cloud Algorithm, as well as a prototype implementation of the Skeleton Billboard Cloud Simplification solution.

Details on the implementations are given in Section 8.1, and a visual evaluation of the simplifications produced is given in Section 8.2. Several experiments have been conducted on the two implementations to analyze how different aspects of the simplification are affected by different parameters. The setup for these experiments is presented in Section 8.3, and the results are presented and analyzed in Section 8.4.

8.1 Implementation

We have implemented the Stochastic Billboard Cloud Algorithm and a prototype of the Skeleton Billboard Cloud Simplification solution. The implementations will be described in the following sections.

8.1.1 Stochastic Billboard Cloud Algorithm Implementation

The Stochastic Billboard Cloud Algorithm has been implemented exactly as prescribed in Section 3.5.1 on page 51. Furthermore, both the view penalty improvement and the increased foliage density improvement have been implemented (see Section 3.6 on page 53). The program is invoked with the following parameters:

- ϵ specifies the maximum allowed permutation distance.
- N specifies how many random planes are considered before choosing the one with maximum density.
- $increasedFoliageDensity$ is a boolean parameter that specifies whether the increased foliage density improvement is enabled or disabled.
- $viewPenalty$ is a boolean parameter that specifies whether the view penalty improvement is enabled or disabled.

- ϕ_v specifies the view penality angle, if the view penality improvement is enabled.

To be compatible with the prototype implementation of the Skeleton Billboard Cloud Simplification solution, the Stochastic Billboard Cloud Algorithm can be invoked for a subset of all foliage triangles. The triangles of this subset are the only triangles that are considered for the simplification.

Implementation Issues

Even though the Stochastic Billboard Cloud Algorithm is simple, an implementation involves several technical details. We have implemented the algorithm using the open source rendering engine Ogre [27] to handle meshes and the rendering of billboard textures. Given a mesh input file, our implementation operates as follows. We first load the mesh vertex and index buffers from the mesh file (the vertex buffer contains the vertices of the mesh and the index buffer keeps track of how the vertices are connected to each other). The Stochastic Billboard Cloud Algorithm then runs on the triangle set defined by the vertex and index buffers. This outputs a set of triangle clusters, each of which is associated with a plane that should simplify the triangles in the cluster.

For each cluster, the triangle vertices are orthogonally projected onto the associated plane and the least bounding rectangle is found. We have implemented a least bounding rectangle algorithm using rotating calipers [22]. The result is four vertices defining the billboard quad for a given triangle cluster.

The triangles in a cluster are then rendered onto the associated billboard quad, again using an orthogonal projection, and the result is stored in a texture, which is associated with the billboard quad. We do not include realistic lighting in our simplifications (cf. our project limitations in Section 2.4 on page 35), so we simply render the billboard textures using ambient shading.

Rendering a billboard texture involves uploading a vertex and an index buffer defining the triangles to be simplified, to the GPU. When the triangles are rendered, the view transformation matrix is set to align the camera with the billboard quad edges, and the projection matrix is set to an orthogonal projection. The view window has a fixed size in world space units, so when performing an orthogonal projection, most triangles are likely to be projected outside the view window, and hence be view frustum culled. The solution is to let the projection matrix scale the x- and y-components with the billboard quad dimensions, resulting in a view window with the same size and dimensions as the quad, which consequently stretches the geometry to be rendered. The geometry is rendered to a texture, and when mapping this texture to the billboard quad, the geometry represented on the billboard will appear correct (i.e. not stretched). The resolution and dimensions of the texture are chosen from the quad dimensions, in order to get a reasonable texture resolution.

A billboard cloud mesh with correct uv-coordinates is constructed and stored. As mentioned in Section 1.3.3 on page 21, the textures of the different billboards should be packed into a single texture in order to reduce render state switches [17], and hence improve fps performance. This has not been

implemented, and the fps performance of our billboard clouds is therefore not judged representative.

8.1.2 Prototype Skeleton Billboard Cloud Simplification

A series of experiments are performed using a prototype implementation of the Skeleton Billboard Cloud Simplification solution. This prototype will be described in this section.

The Skeleton Billboard Cloud Simplification solution consists of two types of simplification; common joint rotation simplification (represented by the CJR algorithm), and non-terminal sibling collapses. The solution performs only non-terminal sibling collapses, if few enough billboard clouds cannot be obtained using common joint rotation simplification alone. The prototype implementation includes common joint rotation simplification only, and hence reduces the number of billboard clouds needed for simplification by defining common rotation animations for siblings. It furthermore applies the Stochastic Billboard Cloud Algorithm implementation for billboard cloud creation per common joint rotation set.

The parameters taken by the prototype are:

- The number of desired billboard clouds, BC. If the input CASTM contains J joints, the prototype can only obtain the desired number of billboard clouds, if $J \leq BC$.
- All parameters needed by the Stochastic Billboard Cloud Algorithm implementation.

The prototype can be used to evaluate common joint rotation simplification, for which a prototype bone animation description type has been implemented. This prototype bone animation description type is simplified compared to the one defined in Section 7.2 on page 115, and the one discussed in Section 7.2.6 on page 119. Since the prototype of the Skeleton Billboard Cloud Simplification algorithm does not include non-terminal sibling collapses, it cannot be used to evaluate our strategy for defining a common end joint for two sibling bones (see Section 7.3 on page 120). Also, since non-terminal sibling collapses are required to reduce the number of billboard clouds to a very low number, the prototype cannot be used to evaluate how well the Skeleton Billboard Cloud Simplification solution can produce simplifications containing very few billboards, and hence its ability to produce LODs to be observed from a large distance.

Prototype Bone Animation Description Type

As mentioned, a prototype bone animation description type has been implemented. This prototype is used by the CJR algorithm in order to construct common rotation sets. The intuition behind the prototype is to let the bones with most triangles attached define the animation of the triangles attached to

their siblings. By doing this, a correct animation is preserved for the largest number of triangles possible.

The functions constituting the type are defined as follows:

$description(b)$**:** A description, $d = \langle b, T \rangle$, for a bone, b, contains a reference to the bone, as well as the number of triangles attached to it.

$average(B)$**:** An average description $\langle b_a, T_a \rangle$ for the descriptions of a set of sibling bones, B, is built as follows. The bone reference b_a in the average description equals the bone reference of the descriptions in B, that has the largest number of triangles attached to it. The amount of triangles T_a equals the amount of triangles attached to b_a.

$deviation(d_1, d_2)$**:** The deviation between two descriptions $\langle b_1, T_1 \rangle$ and $\langle b_2, T_2 \rangle$ equals $min(T_1, T_2)$. If the bones referred to by d_1 and d_2 are not siblings, their deviation equals ∞.

$apply(d, b)$**:** A description, d, is applied to a bone, b, by reassigning the triangles of b to the bone referred to by d (unless it is the same bone, in which case no reassignments are done).

The CJR algorithm initially defines a common rotation set for each bone in the skeleton. It then iteratively builds up larger common rotation sets, by uniting the rotation sets with least deviation between their average descriptions, until the amount of rotation sets equal BC.

A single animation is defined for a common rotation set by reassigning the triangles of the different bones to the bone referred to by the description yielded by $average$, i.e. the bone with most triangles attached. In order for CJR to unite the sibling rotation set pair that implies the least amount of triangles getting their animation changed, the $deviation$ function is evaluated between the average descriptions of each pair of rotation sets.

It should be noted that we are well aware that this animation description type is a simplification compared to the type discussed in Section 7.2 on page 115. The sole reason for opting for this type is lack of time.

It should further be noted that the no bones get their animation changed by this simplification procedure. When the triangles attached to two sibling bones are to share animation in order to be able to share billboards, the triangles of one of the bones is simply reassigned to the other. The effect of this corresponds to the average animation bones improvement presented in Section 7.1.2 on page 113, and the average bone using the prototype bone animation description type is simply one of the original bones. The advantage of this improvement is that animation error introduced to the foliage triangles of a bone does not influence the foliage triangles attached to its children. The disadvantage is that foliage triangles being reassigned no longer follow the animation of their branches. Also, the animation of foliage triangles attached to the children of a bone getting its triangles reassigned will appear less related to the foliage of their parent. We should be able to observe and reflect on these artefacts with our prototype implementation.

8.2 Visual Evaluation

A brief visual evaluation of the simplifications produced by our implementations are given in this section.

8.2.1 Stochastic Billboard Cloud Algorithm

Examine the example simplifications of our input data tree model in Figure 8.1. These examples demonstrate how the Stochastic Billboard Cloud Algorithm effectively reduces the number of triangles, and is successful in simplifying the example model to a variable degree of rendering complexity. The first simplification in Figure 8.1(b) is created with a small ϵ value, and it reduces the number of triangles from 1384 to 126. The result is visually very close to the original tree model, and it retains high visual fidelity from all angles. The simplification in Figure 8.1(c) is created with a relatively large ϵ. It reduces the number of triangles from 1384 to 30. The two-dimensional nature of the foliage has become noticeable, and the visual fidelity no longer seems consistent from different view angles. However, as the simplification only contains about 2% of the original number of triangles, it could be used effectively for distant LODs, at which we believe it to resemble the original accurately. A final example simplification is shown in Figure 8.1(d). This simplification has been created with a large ϵ value, with the improved foliage density improvement enabled, and contains 6 billboards. Generally, we observe that using large ϵ values, the foliage tend to appear too sparse (as discussed in Section 3.6.2 on page 53), but, as seen in the figure, the improved foliage density improvement successfully remedies this. The result has very low polygonal complexity, and from unfortunate view angles it loses some of its visual fidelity.

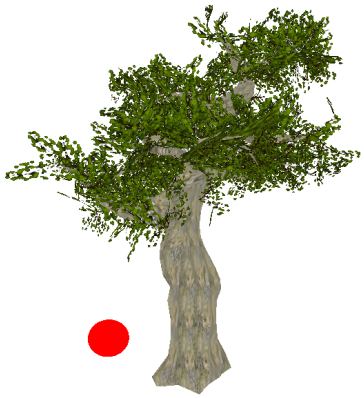
Overall, the results are positive. The algorithm is successful in creating simplifications for different observation distances, i.e. different LODs. The results furthermore indicates that the Stochastic Billboard Cloud Algorithm is suitable for simplifying foliage models. It would be interesting to run the algorithm on other input tree models, especially models with a much larger number of foliage triangles.

Visual Artefacts

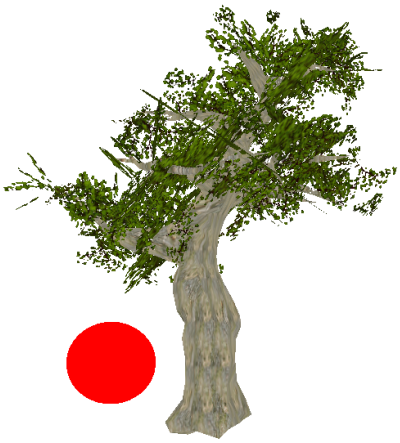
We observe several visual artefacts in our billboard cloud simplifications. First and foremost, the billboards do become visually noticeable when large ϵ values are used, but this is to be expected. However, our implementation suffers from some technical issues regarding the rendering of billboard textures, also.

In general, the billboard textures have a too dark colour intensity, compared to the textures of the input model. We do not currently know the exact reason for this reduction in colour intensity, but we guess it is related to texture aliasing. Furthermore, the foliage silhouette seems slightly blurred in the billboard textures.

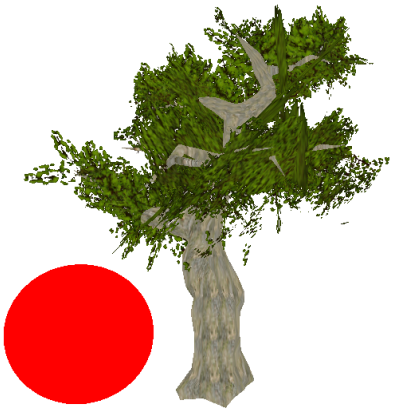
(a) The original tree model.



(b) A billboard cloud simplification with $\epsilon = 3$. The billboard cloud contains 63 billboards (= 126 triangles).



(c) A billboard cloud simplification created with $\epsilon = 6$. The billboard cloud contains 15 billboards.

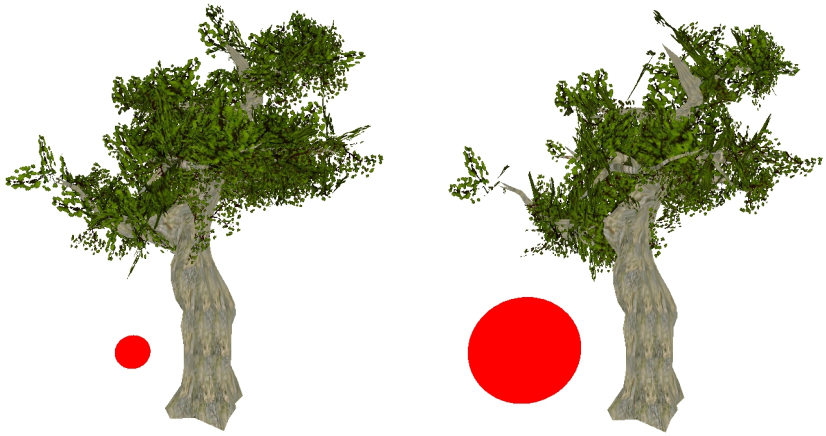


(d) A billboard cloud simplification created with $\epsilon = 11$. The increased foliage density improvement is applied in order to avoid too much reduction in foliage density, due to the large ϵ value. The billboard cloud contains 6 billboards.

Figure 8.1: Our input CASTM and two billboard cloud simplifications created with our implementation of the Stochastic Billboard Cloud Algorithm. All billboard clouds utilize view penalty to increase visual fidelity. Each billboard cloud is illustrated with its associated validity sphere of radius ϵ.

8.2.2 Prototype Skeleton Billboard Cloud Simplification

The Skeleton Billboard Cloud Simplification can be used to produce simplifications of high visual quality, by creating a billboard cloud per bone in the skeleton. Two example simplifications are illustrated in Figure 8.2.



(a) A simplification produced by creating a billboard cloud per bone, using $\epsilon = 2.5$. The number of foliage billboards has been reduced from 692 to 307, and the simplification appears with correct animation.

(b) A simplification produced by creating a billboard cloud per bone, with $\epsilon = 8$. The number of foliage billboards has been reduced from 692 to 86, and the simplification appears with correct animation.

Figure 8.2: Two simplifications produced by the Skeleton Billboard Cloud Simplification prototype. Each billboard cloud is illustrated with its associated validity sphere of radius ϵ.

We expect that at least a few billboards is needed per isolated billboard cloud, in order to avoid unfortunate viewing angles. Figure 8.2(b) demonstrates, however, a surprisingly good result, where most bones only have a single billboard attached. We believe the large number of bones, and only few foliage triangles per bone, in the input model to explain this result.

Common joint rotation simplification can be applied in order to obtain fewer billboards in the skeleton, without reducing ϵ further. As a result, some foliage triangles get their animation altered, which is hard to notice in the result though.

Visual Artefacts

We observe the expected artefact, that some foliage triangles appear unconnected to their branches, when common joint rotation simplification is applied. It is hard to notice though, as the branches in our input model do not perform large rotations, and also due the branches being partially covered by foliage.

As the prototype uses the Stochastic Billboard Cloud Algorithm implementation for creating billboard clouds, we naturally observe the artefacts related to billboard textures mentioned for this implementation.

8.3 Experiments Setup

To gain a more profound understanding of the Stochastic Billboard Cloud Algorithm and the prototype of the Skeleton Billboard Cloud Simplification solution and their effectiveness, we investigate how the different parameters in the implementations affect the quality of the simplifications.

8.3.1 Quality Experiments

Recall that we measure quality in terms of error and cost; error expresses the visual fidelity of the simplified model to the original model, and cost expresses the rendering performance of the simplified model. We assume that our simplifications are to be used in a first-person perspective application where the observer is positioned on the same ground level as the trees, and evaluate them in this context. In our experiments, only foliage is simplified, and when computing the visual error and cost, only the foliage is considered.

Measuring Visual Fidelity

In Section 5.1.5 on page 76 it was argued that if a simplification yields low values with the colour metric as well as an animation error metric, it suggests visual fidelity in every frame and coherent animation structures. To evaluate the visual fidelity of the simplifications produced by our implementations, we thus employ the colour metric defined in Section 5.1 on page 70. This metric is defined with animation in mind, meaning that it compares the colour values in images obtained by sampling the animation cycle. To measure the colour error of a static billboard cloud simplification the same metric can be used, with the number of frames considered set to one. We refer to this specialized version as the *static colour metric*. Unfortunately, we have not had the time to implement any one of the animation metrics, so our experiments hence measure visual fidelity in terms of colour error only.

The colour metric evaluates the general visual fidelity by analyzing the difference in rendered foliage pixels in the input CASTM and the billboard cloud, respectively. Intuitively, the colour metric expresses how much overlap there is between the silhouettes of the renderings, and thus how similar they appear to an observer. Since we evaluate our simplifications to be used in an application where the trees are observed from a ground level, the colour error only compares images of the foliage rendered from this perspective, and not from a top-view, for example. To avoid texture filtering and similar aspects of the rendering process influencing the result, we simply evaluate whether a given pixel in the output has the background colour or not. If it doesn't have the background colour, foliage must have been rendered in that pixel.

The colour metric can be computed using any number of view angles, animation frames, and in any resolution. In all of our experiments, we have chosen a resolution of 800 x 600. Furthermore, we have chosen to always render five different view angles, and when animated models are compared, five sample animation frames. Intuitively, the number of view angles and animation frames

should be as high as possible, but as our implementation of the colour metric is fairly slow, we keep both low to save computing time. The five view angles are distributed evenly on a full rotation of the tree model, and the animation frames are uniformly distributed during the animation cycle. We have deliberately avoided using a number of view angles that is even, to avoid getting very similar renderings from opposite directions of the tree model.

Measuring Rendering Performance

To evaluate whether a billboard cloud simplification of a model is effective, some objective metric of rendering cost should be defined. To be able to evaluate performance without depending on a specific hardware configuration, and without considering the added complexity of a graphics system like Ogre, we must identify key properties of a billboard cloud that determine the relative rendering cost of a it.

The performance of rendering a billboard cloud depends on the number of triangles to be projected onto the screen, as well as the area of the triangles, as these determine the complexity of rasterization. We define a *hardware-independent cost metric* for a billboard cloud as:

$$< billboard\ count, total\ billboard\ area >,$$

where $total\ billboard\ area$ is measured in world-space units.

The fps performance bottleneck of rendering a billboard cloud can either be in the geometry stage or the rasterization stage, as discussed in Section 1.3.3 on page 22. For this reason we include both billboard count and total billboard area in our performance metric. The bottleneck is often rasterization, as a billboard cloud replaces many triangles with a single larger billboard. However, on far-away distances, a billboard cloud may cover only very few pixels, making the geometry stage the bottleneck of rendering it. Each billboard in a billboard cloud has a unique texture, hence total billboard area is also a resolution independent texture memory measure. Note that our cost measure does not include the cost associated with animating the billboards.

In some application running on a given hardware platform, the exact fps performance can be computed for our simplifications by computing the *average frame rate* measured in fps. Fps should be measured as a function of the distance between the billboard cloud and the observer, in order to be able to identify the expected performance gain when increasing the distance between the billboard cloud and the observer.

Statistics

The Stochastic Billboard Cloud Algorithm searches for high density planes using pseudo-randomness. Thus, the algorithm will yield different results for consecutive runs with the same input and parameters, and likewise will the the prototype Skeleton Billboard Cloud Simplification solution. When performing an experiment of either, the implementation can be invoked repeatedly with

the same parameters, and subsequently apply simple statistic functions to the obtained metric results.

We compute the average of the resulting metric values. To investigate the range of values in the results, we also find the boundary values, minimum and maximum, as well as compute a measure of variance. The variance measure employed is the *average absolute deviation*, which, for $i = 1..n$ metric values x_i, is given by:

$$\frac{1}{n}\sum_{i=1}^{n} \mid x_i - \overline{x} \mid,$$

where $\overline{x}$ is the average value of the n metric values. The average absolute deviation simply computes the average difference of the results from the average result.

8.3.2 Input Data

The input data used in our experiments is a tree model modelled and animated manually by an artist, named David Longacre. Certain choices by the artist may influence the results of the experiments. Firstly, each foliage triangle represents 50 leaves connected to twigs. Thus, the foliage of the input model may be considered a set of small billboards. However, this may often be the case with manually created tree models, as adding an amount of leaves that corresponds to a real tree is an unsurmountable task, and the model may be considered a reasonably typical example of a manually created tree model for real-time usage.

Here we list a few facts about the input model:

- The foliage part of the input model contains 1384 foliage triangles defining 692 billboards.
- The skeleton contains 101 bones, and the depth of the skeleton is 16 levels.
- 70 of the bones have foliage triangles attached, and each of these bones have approximately 10 foliage triangle.

The input model satisfies the requirements to CASTMs as listed in Section 2.1 on page 32, except for the requirement that vertices of a foliage triangle may not be associated with more than a single bone. However, this is remedied by adding a pre-processing step to the algorithms that associates all vertices of a triangle to the same bone, decided as either the bone to which the most vertices are assigned, or the bone being at the lowest level in the tree.

8.4 Experiments Results

In this section we present the results of different experiments conducted on the implementations of the Stochastic Billboard Cloud Algorithm and the prototype of the Skeleton Billboard Cloud Simplification.

8.4.1 Stochastic Billboard Cloud Algorithm Experiments

Since the Stochastic Billboard Cloud Algorithm relies on pseudo-randomness in its search for high density planes, it is interesting to examine the stability of the results produced by it, in terms of cost and error. Such are presented in the following. The view penalty and increased foliage density improvements have been implemented, and it is interesting to test whether these influence cost and error as expected. Finally, we have conducted experiments in order to determine how cost and error are affected by the maximum allowed permutation parameter distance, ϵ, the result of which is demonstrated.

Stability Experiments

Due to the stochastic search for high density planes performed by the algorithm, the algorithm will yield different results for consecutive runs with the same input and the same parameters. The following experiments evaluate the stability of the Stochastic Billboard Cloud Algorithm. Here, stability of an algorithm is considered the property that the algorithm yields similar results on the same input and given the same parameters. These experiments analyze the variance of the output metrics results. Both the stability of the visual results of the algorithm as well as the stability of the performance metric results are investigated.

► **Experiment: Stability of performance**

This experiment tests whether the Stochastic Billboard Cloud Algorithm outputs a similar hardware-independent performance metric for a sequence of runs with the same input and parameters. If the metric varies a great deal between runs of the algorithm, the algorithm may be considered impractical.

The stability of the algorithm depends on the number of sample planes evaluated in each iteration, N. The larger N is, the more stable is the algorithm expected to be, as it is more likely to consistently find a high density plane. We also expect the algorithm to be most stable with low ϵ values. A random plane is created by selecting a random seed triangle, and translate its vertices with a maximum distance of ϵ in the normal direction. The lower ϵ, the less different billboards can be created, hence making the algorithm more stable.

The Stochastic Billboard Cloud Algorithm is run on the input model with $\epsilon = 10$ and the number of billboard selection iterations $N = 100$. The experiment is repeated with $\epsilon = 4$. The average hardware-independent performance metric is computed as well as the average absolute deviation of the results. The view penalty improvement is enabled, as it will be in most of the experiments. The increased foliage density improvement is irrelevant for this experiment.
Results of 40 tests with $\epsilon = 10$:

- average number of billboards: 5.73
- average total billboard area: 10500
- average absolute deviation of the number of billboards: 0.630

- average absolute deviation of the total billboard area: 1000

Results of 40 tests with $\epsilon = 4$:

- average number of billboards: 36.2
- average total billboard area: 45600
- average absolute deviation of the number of billboards: 1.04
- average absolute deviation of the total billboard area: 1570

To relate the variance of the two sub-experiments, the variance can be expressed relatively to the average number of billboards. Similarly, the variance of the total billboard area can be expressed relatively to the average total billboard area.

The sub-experiment with $\epsilon = 10$ results in six billboards on average, and the average absolute deviation is 0.630. Intuitively, this means we can expect the number of billboards to be between five and six. The relative variance of the number of billboards is $\frac{0.630}{5.730} \approx 0.110$. The relative variance of the total billboard area is $\frac{1000}{10500} \approx 0.0952$.

The sub-experiment with $\epsilon = 4$ results in 36 billboards on average, and the average absolute deviation is approximately 1.04, which means that we can expect the number of billboards to be between 35 and 37. The relative variance of the number of billboards is $\frac{1.04}{36.2} \approx 0.0287$. The relative variance of the total billboard area is $\frac{1570}{45600} \approx 0.0344$.

The variance in the number of billboards of both experiments is very reasonable, a variance of a single billboard should be acceptable for most practical purposes. The total billboard area has a higher relative variance in the first test, which is expected, as the relative variance in the number of billboards is higher in the first test, and adding or subtracting a billboard from the cloud will add or subtract a considerable amount of unused area to the total.

In general, our results show that the algorithm is stable for the tested input model using only $N = 100$, and even better stability can be obtained by increasing it. The relative variance is notably lower using the smallest ϵ value, which was expected.

► **Experiment: Stability of colour metric**

This experiment tests whether the Stochastic Billboard Cloud Algorithm produces simplifications with similar visual error according to the static colour metric for a sequence of runs with the same input model and parameters.

The test setup is identical to the setup in the previous experiment, and when ϵ is small, the average static colour metric is expected to be most stable.
Results of 20 tests with $\epsilon = 10$:

- average static colour metric: 0.440
- average absolute deviation of static colour metric: 0.0250

Results of 20 tests with $\epsilon = 4$:

- average static colour metric: 0.310
- average absolute deviation of static colour metric: 0.0140

The relative deviation with $\epsilon = 10$ is $\frac{0.0250}{0.440} \approx 0.056$, and the relative deviation with $\epsilon = 4$ is $\frac{0.0140}{0.310} \approx 0.045$. The large ϵ value hence yields the most variable colour error results, as expected. In any case, we judge the algorithm fairly stable, and with $\epsilon = 10$ and $N = 100$, the results vary only about 6% from their average. The variance is even less with $\epsilon = 4$.

Quality of Improvements Experiments

These tests investigate the effects of the view penalty and increased foliage density improvements, as they have been implemented in the Stochastic Billboard Cloud Algorithm.

► Experiment: Static colour metric with and without the increased foliage density improvement

This experiment tests the efficiency of the increased foliage density improvement to the Stochastic Billboard Cloud Algorithm. An example of the visual result of this improvement in our implementation was given in Section 3.6.2 on page 53.

The improvement is expected to yield somewhat better results in terms of colour error, especially when there are few billboards in the billboard cloud, i.e. when ϵ is large. When this is the case, the foliage area will have been reduced the most, if the improvement has not been applied.

The effect of the increased foliage density improvement is tested by rendering a series of 10 billboard clouds using the Stochastic Billboard Cloud algorithm with the increased foliage density improvement enabled, and a serires of 10 with the improvement disabled. The experiment is performed with three different ϵ values, namely 4, 10, and 20. The number of sample planes considered in each iteration is $N = 100$, and the view penalty improvement is disabled. Note that the cost metric is irrelevant for the increased foliage density improvement, as it does not influence the process of choosing planes for billboard cloud creation.

Results of 10 tests with $\epsilon = 4$:

- improvement disabled: average static colour metric: 0.306 (absolute deviation: 0.00871)
- improvement enabled: average static colour metric: 0.304 (absolute deviation: 0.000827)

Results of 10 tests with $\epsilon = 10$:

- improvement disabled: average static colour metric: 0.463 (absolute deviation: 0.0192)

- improvement enabled: average static colour metric: 0.464 (absolute deviation: 0.0127)

Results of 10 tests with $\epsilon = 20$:

- improvement disabled:
  average static colour metric: 0.590 (absolute deviation: 0.0205)
- improvement enabled:
  average static colour metric: 0.490 (absolute deviation: 0.0392)

The benefit of the increased foliage density improvement is negligible with lower ϵ, and it may actually yield worse performance than without it being enabled, when ϵ is close to zero. When ϵ is 20 or higher, the improvement becomes very useful. In this case the algorithm only outputs three billboards on average, and the improvement yields thus best colour improvement when only using few billboards for simplification.

In our visual evaluation, we judge the improvement effective also for $\epsilon = 10$. However, the improvement has a tendency to make the foliage appear too dense, which might be the reason for the poor effect on the static colour metric.

► Experiment: Static colour metric and hardware-independent performance with and without the view penality improvement

This experiment tests the efficiency of the view penalty improvement to the Stochastic Billboard Cloud Algorithm.

The view penalty improvement is expected to yield better visual results, and better colour metric results, when the tree model is observed from the specified view angle. However, the view penalty improvement is expected to have an adverse effect on the hardware-independent performance metric, as high density planes are potentially discarded, due to their normal directions being disfavored by the view penalty.

The efficiency of the view penalty improvement is tested by rendering a series of 10 billboard clouds with the Stochastic Billboard Cloud Algorithm with the view penalty improvement enabled, and a series of 10 with the improvement disabled. When the view penalty is enabled, ϕ_v is set to favor vertical billboards, as the assumed observation of the tree models are from ground perspective. The experiment is performed with $\epsilon = 4$ and $\epsilon = 10$ for $N = 100$. The increased foliage density improvement is disabled.

Results of 20 tests with $\epsilon = 4$:

- improvement disabled:
  - average static colour metric: 0.343 (absolute deviation: 0.0321)
  - average number of billboards: 36.2 (absolute deviation: 0.710)
  - average total billboard area: 45800 (absolute deviation: 1400)
- improvement enabled:

  - average static colour metric: 0.306 (absolute deviation: 0.0116, boundaries $0, 0$)
  - average number of billboards: 36.3 (absolute deviation: 0.830)
  - average total billboard area: 45100 (absolute deviation: 1200)

Results of 20 tests with $\epsilon = 10$:

- improvement disabled:
  - average static colour metric: 0.536 (absolute deviation: 0.0378)
  - average number of billboards: 5.7 (absolute deviation: 0.56)
  - average total billboard area: 10100 (absolute deviation: 650)
- improvement enabled:
  - average static colour metric: 0.441 (absolute deviation: 0.0238)
  - average number of billboards: 5.85 (absolute deviation: 0.425)
  - average total billboard area: 10400 (absolute deviation: 683)

The algorithm clearly benefits from the view angle improvement with regards to the static colour metric. The absolute deviation is significantly decreased, so the visual results of the algorithm with this improvement enabled is certainly also more reliable than with it disabled. The improvement seems to have an increase in rendering cost as expected, but it is a minor increase compared to the decrease in colour error.

Contrary to expectations the average number of billboards does not increase when the view penalty improvement is enabled.

Quality of Simplification Experiments

ϵ is the parameter that defines the maximum allowed vertex displacement in the Stochastic Billboard Cloud Algorithm, and hence the size of the triangle validity domains. Obviously, ϵ affects the static colour metric and the hardware-independent performance metric. In the following experiments the affect of ϵ is investigated in more detail.

▶ **Experiment: Static colour metric as a function of epsilon**

This experiment evaluates the effect of ϵ on visual fidelity, measured in the static colour metric.

If $\epsilon = 0$, billboards representing foliage triangles are not allowed to be displaced from the original positions at all. Thus, the static colour metric should yield close to zero. To yield a perfect zero, texture aliasing should not have any effect on the static colour metric, which in general is not true. As described in Section 8.2.1 on page 137, we observe several artefacts related to the billboard textures rendered in our implementation, and hence expect far from zero colour error for $\epsilon = 0$.

As ϵ increases, the static colour metric is expected to increase, because triangles are allowed displacement further from their original positions.

The experiment computes the static colour metric for a Stochastic Billboard Cloud Algorithm simplification with ϵ in the range from zero to 30. Both view penalty (favoring vertical billboards) and the increased foliage density improvements are enabled, as these were shown to yield better results in the preceding experiments. The algorithm runs with the parameter $N = 100$.

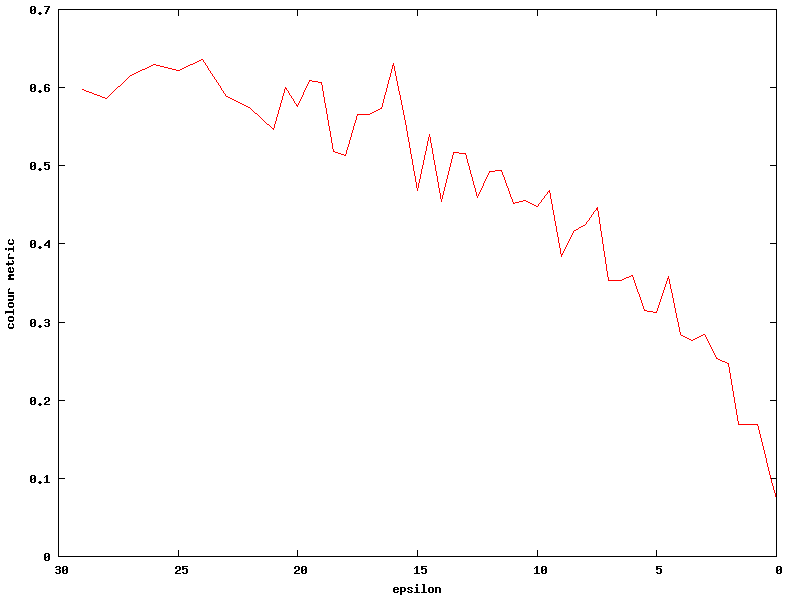


Figure 8.3: The static colour metric as a function of ϵ.

It can be observed that even with $\epsilon = 0$, the static colour metric is 0.074, i.e. 7% of all pixels have incorrectly been rendered to. This demonstrates the colour error introduced by the visual artefacts of our billboard textures, even when disregarding colour intensity error. The error value is higher than expected, since the simplification look identical to the original model (aside from the darkened colour artefact).

As expected, the static colour metric increases steadily as ϵ increases. It can be noted that the colour error seizes to increase much when $\epsilon > 15$.

► **Experiment: Performance metric as a function of epsilon**

The hardware-independent performance metric may be evaluated for different values of ϵ, thus determining the benefit of the billboard cloud simplification.

When $\epsilon = 0$, we expect the number of billboards to be equal to the number of foliage triangles that are non-coplanar in the input model. Each foliage triangle in our specific input model is part of a textured quad consisting of two coplanar triangles (ignoring any round-off errors). As the input model contains 1384 foliage triangles, we expect the number of billboards to be $\frac{1384}{2} = 692$. The billboards rendered for $\epsilon = 0$ are expected to have the same area as the original foliage quads, since the billboards are created using a minimum bounding box

algorithm on the vertices of the quads. As ϵ increases, we expect the number of billboards to be reduced.

In Section 3.2.2 on page 43 we mentioned a tendency of billboard cloud algorithms to consider cost simply as the number of billboards. This seems acceptable, assuming that increasing the number of billboards also increases the total area. This experiment can help us see to what degree this is true.

The experiment computes the static colour metric for simplifications created by the Stochastic Billboard Cloud Algorithm with ϵ sampled in the range $[0; 30]$. The implementation is invoked with the parameters $N = 100$, and with both the increased foliage density and view penalty (favoring vertical billboards) improvements enabled. The number of billboards and the total billboard area are illustrated for each value of ϵ.

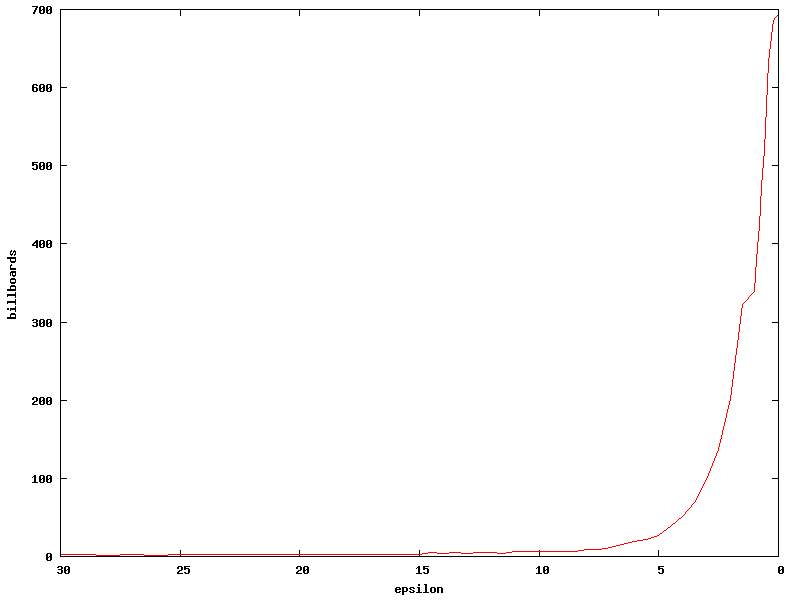


Figure 8.4: The number of billboard clouds as a function of ϵ.

First and foremost, it can be observed that the number of billboards in the billboard cloud decreases fast, when ϵ is increased. For $\epsilon > 10$, the number of billboards is very low, and becomes more stable.

When ϵ is large, the total billboard area is small, in fact much smaller than the total area in the original foliage model. When ϵ decreases, the number of billboards in the simplification increases, and so does the total billboard area. That is, until approximately $\epsilon = 1.5$, where the number of billboards is 322, and the total area begins to decrease when ϵ is lowered further. In general, the assumption that the number of billboards also dictates the total billboard area seems to hold, unless a very small ϵ is used, and hence the number of billboards is very large.

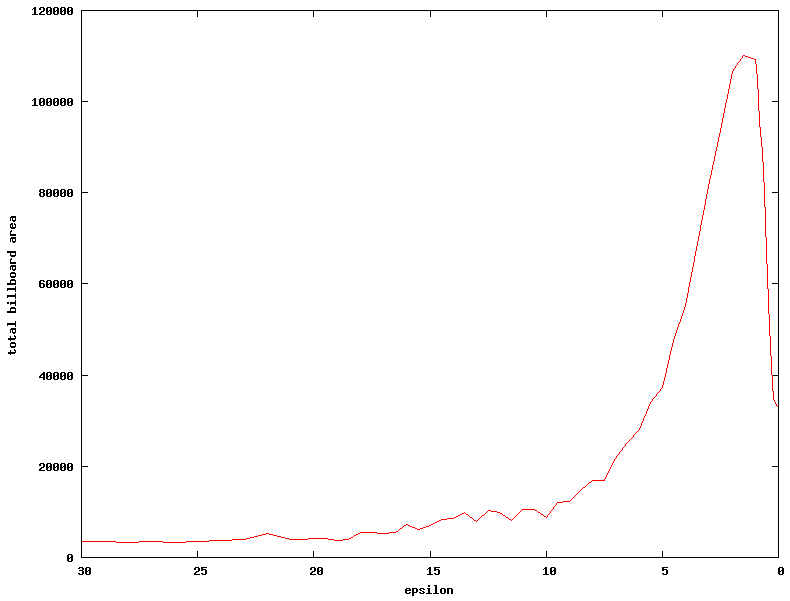


Figure 8.5: The total area of billboard clouds as a function of ϵ.

Results in Perspective

In overall, the results of the experiments are as expected, but it would be interesting to run the experiments on other input models. Our input only contains 1384 foliage triangles, which are arranged as 692 small billboards, and the input model can hence be said to already be a billboard cloud. We would prefer a foliage model with large polygonal complexity, as billboard cloud simplification is more substantial for such models. In fact, our input model is likely to be rasterization bound in a practical application already, so reducing its number of triangles has limited implication. Nonetheless, we believe that our experiments do demonstrate the effectiveness of the simplification technique.

It should be noted that the Stochastic Billboard Cloud Algorithm is rather fast. With the ϵ and N values used in the presented experiments, a billboard cloud is constructed in a matter of seconds. The rendering of textures handled by Ogre is, on the other hand, relatively slow, and become thus the dominating time factor in our implementation.

8.4.2 Prototype Skeleton Billboard Cloud Simplification Experiments

In the following, the experiments conducted on the Skeleton Billboard Cloud Simplification prototype are presented. The Stochastic Billboard Cloud Algorithm implementation is used, with $N = 100$ and the both mentioned improvements enabled.

► **Experiment: Colour metric as a function of individual billboard clouds**
The colour metric is evaluated for $\epsilon = 0$ and for a range of the number of

common joint rotation simplifications. The purpose is to evaluate how much visual error is introduced by altering sibling bone animations alone, without any triangle displacement.

Recall that the input model contains 70 bones with foliage attached. If no common joint rotation simplification is done, a billboard cloud is created for each of these bones. The bones with foliage are arranged in the skeleton such that 24 joints contain two siblings with foliage, and a single joint contain three siblings with foliage. Consequently, our prototype solution is able to reduce the number of individual billboard clouds from 70 to 44, at most. This corresponds to 26 of the 70 bones reassigning their foliage triangles to a sibling, and assuming a uniform distribution of the 692 foliage billboard amongst these 70 bones, this corresponds to approximately 37% of the foliage getting their animation altered.

Assuming that sibling bones in the skeleton define different rotation animations, we expect the reassignment of 37% of the foliage triangles to have a noticeable impact on the colour metric error.

The number of individual billboard clouds to be constructed is defined by BC, which hence defines the amount of common joint rotation simplification performed, and the parameter values used are in the range $[44; 70]$. The experiment results are shown in Figure 8.6, and has been created using 3 view samples.

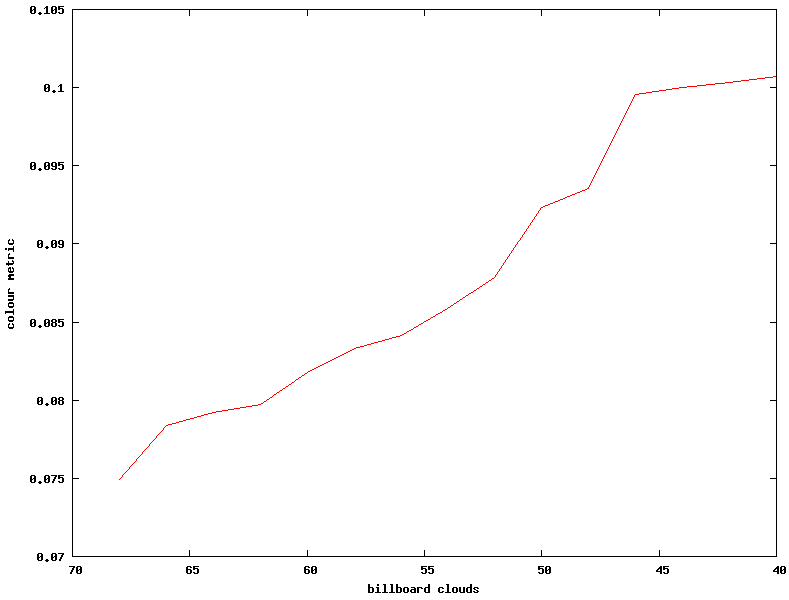


Figure 8.6: The colour metric as a function of the number of billboard clouds.

When no common joint rotation simplification is performed, a billboard cloud is created per bone, and as ϵ is 0, the colour metric is expected to demonstrate the error introduced by texture artefacts only. The colour error in results is approximately 0.075, which is almost exactly the same error observed for the stochastic billboard in Section 8.4.1 with $\epsilon = 0$. This is to be expected, as $\epsilon = 0$ implies no triangle displacement.

As the number of individual billboard clouds is reduced, i.e. BC is lowered, the colour error increases steadily. When most common joint rotation simplification is performed ($BC = 44$), the colour error is largest and equal 0.100. This error represents the effect of reassigning the triangles of 26 bones to their siblings, and thus altering the animation. Relatively to the error with $BC = 70$ this is a relative increase of $\frac{0.100-0.070}{0.070} \approx 0.429$, i.e. 42.9%.

The effect common joint rotation simplification has on colour the metric is noticeable, but is minor compared to the effect of changing epsilon in the Stochastic Billboard Cloud Algorithm documented in Section 8.4.1 on page 145, with out example input tree model.

▶ **Experiment: Performance metric as a function of sibling collapses**

We have conducted experiments to examine the behaviour of the performance metric, as a function of the number of individual billboard cloud, BC, i.e. the amount of common joint rotation simplification. We have used $\epsilon = 4$.

The number of billboards is expected to decrease, when BC is decreased, as the triangles of more sibling bones will be allowed to share billboards. The result is illustrated in Figure 8.7, and show how the number of billboards are reduced from 187 to 148, by decreasing BC from 70 to 44. The reduction when having $\epsilon = 4$ is not impressive, but do demonstrate the fact that triangles attached to sibling bones can share billboards, if the two bones are given a common rotation animation, and if the triangles are co-planar with respect to ϵ.

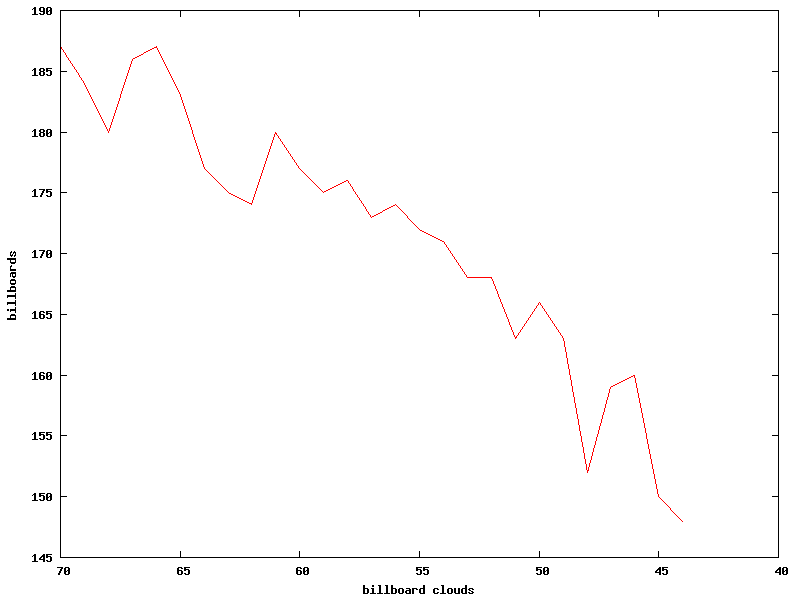


Figure 8.7: The number of billboards as a function of the number of billboard clouds.

Figure 8.8 shows the behaviour of the total billboard area when decreasing BC. As can be seen, this value fluctuates much. It total area has increased from approximately 31.000 to 33.000, when BC is decreased from 70 to 44. An increase was expected, since the new billboards introduced for triangles

attached to siblings given a common rotation, will in generally expand a larger area.

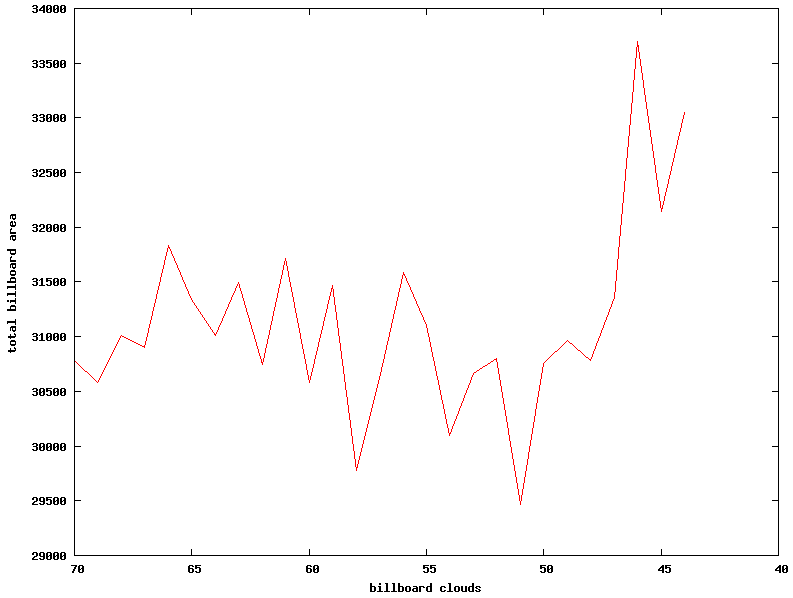


Figure 8.8: The total area of the billboards as a function of the number of billboard clouds.

Results in Perspective

For several reasons, the results are not as informative as one could hope for. Our input tree model does not contain many foliage triangles per bones, and it is therefore difficult to obtain large performance difference between our simplifications, and the original input tree model.

We demonstrated how the colour metric error increased, when 26 bones got their triangles reassigned to a sibling triangle. The increase was not substantial compared to the colour error introduced by changing the ϵ value in the Billboard Cloud Algorithm. The reason is believed to be, that the bones with foliage attached do not perform large rotations in the animation, and furthermore that the siblings tend to rotate with similar animations.

The implemented prototype stands as a proof of concept, but does not do the Skeleton Billboard Cloud Simplification solution justice. It would, however, be interesting to run some experiments on an input model more suitable for common joint rotation simplification, i.e. one with more siblings amongst the foliage triangles.

Chapter 9

Conclusion

We have developed two solutions for simplification of the foliage part of CASTMs, that combine the billboard cloud technique with our own solutions for animation simplification. The solutions implement two different simplification strategies and can be used as a part of a discrete LOD scheme, either separately or in combination. Each solution may prove best suited for different rendering distances, and, as such, they may complement each other when used in a practical application. To be able to measure the quality of the animated billboard cloud simplifications, we have developed a set of error metrics, that together describe various aspects of the subjective notion of visual fidelity.

We have implemented a functional prototype of our Skeleton Billboard Cloud Simplification solution, and performed experiments on the produced animated billboard cloud LODs of a CASTM. We have shown a potential performance gain by reducing the number of triangles, and the visual fidelity of the solution is reasonable, both when measured using a colour error metric and when observed.

We hypothesize that the Spectral Clustering solution is mostly suitable for producing LODs with few triangles to be observed from afar, while the Skeleton Billboard Cloud Simplification solution is suitable for producing detailed LODs for short observation distances. In order to be able to validate this hypothesis, at least a prototype implementation of each solution should be created. The two prototypes and their corresponding strategies can be objectively compared using the visual error and performance metrics. It would furthermore be interesting to obtain more example tree model data, for example concrete data from some actual real-time application.

Future research could involve how to apply our simplifications efficiently in the context of a practical rendering application. This includes specific performance issues arising when rendering a number of our simplifications corresponding to the scale of a forest, such as how to group the data in order to reduce the number of render calls per frame. Also, solutions for smooth transitions between our LODs should be considered, and transition solutions specifically designed for our solutions might be devised.

Appendix A

Tools

A.1 Normalized Error Function

The *normalized error function* NE calculates the normalized error value of a given error value of some type for an object to the range $[0; 1]$:

$$NE : \mathbb{R}^+ \cup \{0\} \rightarrow [0; 1]$$

For example, an object could be a foliage triangle of a CASTM simplified by a billboard, and the error value could be of type distance between the original position of a foliage triangle in a CASTM and its simplified position on a billboard over the animation cycle.

Let a denote an error value of some type A for some object. NE is defined as

$$NE(a) = 1 - \frac{1}{1 + a} \tag{A.1}$$

For $a = 0$, $NE(a) = 0$, and for $\lim_{a\rightarrow\infty} NE(a) = 1$. Relating it to the triangle distance example, $a = 0$ implies that there is no distance between the original position of the triangle and its simplified position on a billboard, while $a \rightarrow \infty$ implies that the distance becomes infinitely large. NE can be used to compare normalized error values for objects, e.g. the normalized distance error values of two foliage triangles.

A.2 Normalized Error Product Function

The *normalized error product function* NEP calculates the total normalized error of two error values of different types for a pair of objects:

$$NEP : \mathbb{R}^+ \cup \{0\} \times \mathbb{R}^+ \cup \{0\} \rightarrow [0; 1]$$

Again, the objects could be foliage triangles of a CASTM simplified by the same billboard. For a triangle, a is an error value of type distance as before, and b is an error value of type speed between the original triangle and the billboard

simplifying it. For each triangle object, NEP then calculates the total normalized error of the distance and speed errors. The total normalized errors of the two triangle objects can then be compared in order to determine how well they are simplified by the same billboard.

For some object, let a and b denote two error values of types A and B. NEP is defined as

$$NEP(a,b) = 1 - \frac{1}{1+a} \cdot \frac{1}{1+b} \tag{A.2}$$

For $a = 0$ and $b = 0$, $NEP(a,b) = 0$. When either $a \to \infty$ or $b \to \infty$, then $NEP(a,b) = 1$.

NEP states the value of the *total* normalized error of two error values. Consequently, if either a or b is large, NEP will yield a large value.

A problem may arise if either error type in general takes larger values than the other. In this case it will dominate the entire NEP term, which can be unfortunate. To address this issue, the weighted normalized error product function can be applied.

A.3 Weighted Normalized Error Product Function

As with NEP the *weighted normalized error product function* ($WNEP$) is used to calculate the total normalized error of two error values. However, to address the issue of one a these two values dominating the total error value, and to be able to specify the importance of an error value in the total expression, a weight is specified for each of the two values:

$$WNEP : \mathbb{R}^+ \cup \{0\} \times \mathbb{R}^+ \cup \{0\} \times \mathbb{R}^+ \cup \{0\} \times \mathbb{R}^+ \cup \{0\} \to [0;1]$$

Let a and b denote two error values of types A and B, and w_a and w_b two weight values. $WNEP$ is defined as:

$$WNEP(a,b,w_a,w_b) = 1 - \frac{1}{1+a \cdot w_a} \cdot \frac{1}{1+b \cdot w_b} \tag{A.3}$$

The values of the weights are to be determined from application specific requirements.

A.4 Average in a Cyclic Range

In this section we specify how to find the difference of values specified in a cyclic range, and the average of such values.

Assume two angles specified in the cyclic range $[0; 2\pi]$. One attempt to find the difference between two such values is simply to consider the subtraction of the values as numbers. This is, however, not suitable for a cyclic value range, as the distance between 0.1 and $2\pi - 0.1$ would be concluded $2\pi - 0.2$. Taking the cyclic behaviour of the range into account, the difference between the two angles is only 0.2. The solution is to consider the phase value as angles, convert

these angles to two-dimenstional vectors, and consider the angle between these vectors.

An average of a set of angle is obtained by converting the angles to vectors, sum the vectors, and finally convert the result back to an angle. This is illustrated by an example in Figure A.4.

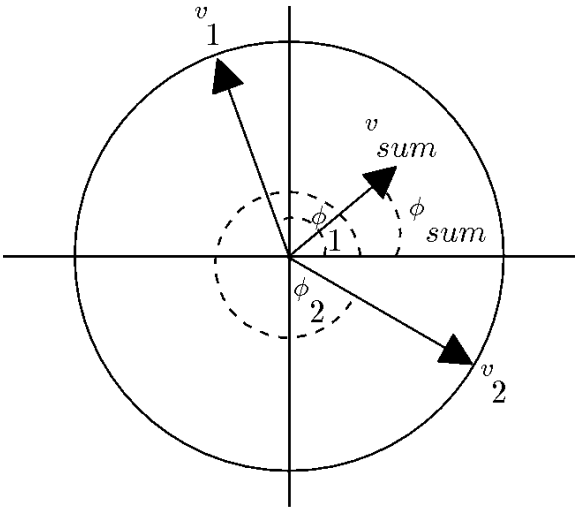


Figure A.1: The two angles $\phi_1 = 110°$ and $\phi_2 = 330°$ are converted to the unit vectors v_1 and v_2. These vectors are summed to yield the vector v_{sum}, which is converted back to an angle. The resulting angle $\phi_{sum} = 40°$ is the correct average of ϕ_1 and ϕ_2, taking the cyclic behavior of the angle range into account.

If the values to be averaged lie in a general cyclic range $[a; b]$, this range is simply mapped to the range $[0; 2\pi]$, after which the values can be considered as vectors. The result obtained is mapped back the range $[a; b]$.

Appendix B

Skinning

Skinning is the process of updating vertex positions in a mesh that is rigged with and animated by a skeleton. In this section we will explain how vertex positions are updated, when the skeleton changes pose. Our descriptions are partly based on [11].

As mentioned in Section 1.4.2 on page 25, a vertex of a mesh can be associated with more than one bone in the skeleton, which, among other improvements, has the effect of preserving elasticity around joints, as it prevents cracks around joints and reduces creasings [3]. In effect, a branch in the trunk part of a polygonal tree model that sways will not appear unconnected to its parent branch and its animation will be more ”smooth”.

Associating a vertex to more than one bone makes sense for closed meshes, such as the trunk in a polygnal tree model. However, for a mesh such as the foliage part of a polygonal tree model, which consists of unconnected triangles, connectivity does not have to be preserved. Therefore, for a triangle, it is reasonable to associate its vertices with one and the same bone for these meshes.

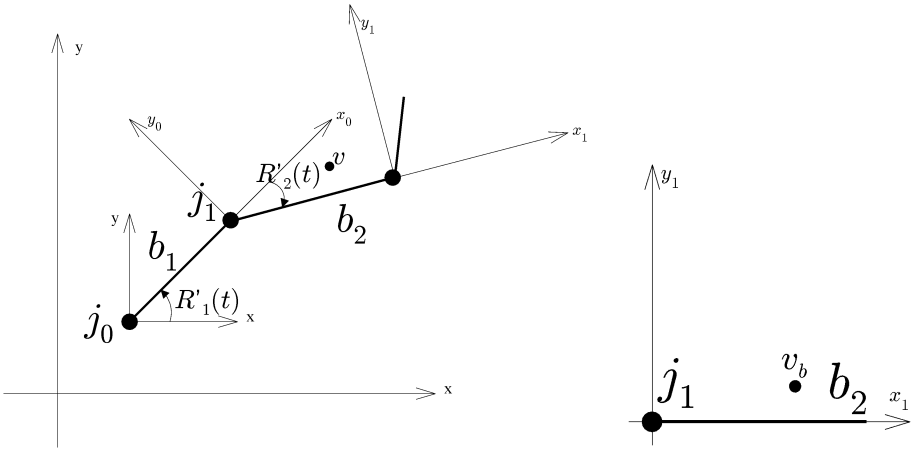
When updating the position of a vertex, the vertex is transformed from world space to bone space of the model. When the skeleton has been updated to a new pose, the vertex is transformed back again, from bone space to world space, to its new position.

Bone space is the local coordinate system at the start joint of a bone, rotated by the bind pose rotated followed by the rotation animation of the bone. In bone space the coordinates of the vertex are invariant, so when the skeleton changes pose, the coordinates of the vertex remain the same in bone space. Bone space is illustrated in Figures B.1(a) and B.1(b) on the next page.

Figure B.2(a) on the facing page shows the bind pose of an animation skeleton, and vertex $\mathbf{v}$ rigged to bone b_2 in world space coordinates. Figure B.2(b) on the next page illustrates $\mathbf{v}$'s new position, $\mathbf{v}'$, that we want to find, when the skeleton has changed pose.

Let $\mathbf{X}_0$ be the translation vector yielding the position of the root joint. For $i > 0$, if $\mathbf{X}_i$ is the translation vector of bone b_i with $i-1$ parents and $\mathbf{R}_{bind_i}$ is the bind pose rotation of b_i, the bind pose rotation matrix $\mathbf{B}_i$ of b_i is given by

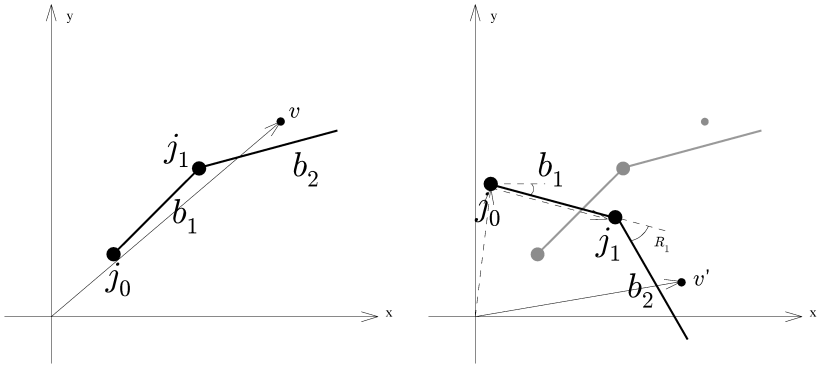
$$\mathbf{B}_i = \mathbf{R}_{bind_i}\mathbf{X}_{i-1}\mathbf{R}_{bind_{i-1}}\mathbf{X}_{i-2}\cdots\mathbf{R}_{bind_1}\mathbf{X}_0$$



(a) The local coordinate system in j_1, which is the world coordinate system rotated by $\mathbf{R}'_1(t)$, is rotated by $\mathbf{R}'_2(t)$ to yield the bone space of b_2.

(b) Vertex $\mathbf{v}$ in bone space of b_2, denoted $\mathbf{v_b}$ (notice that the coordinate system is the local coordinate system in j_1 rotated by $\mathbf{R}'_2(t)$).

Figure B.1: Illustration of bone space.



(a) A skeleton in its bind pose. $\mathbf{v}$ is attached to bone b_2.

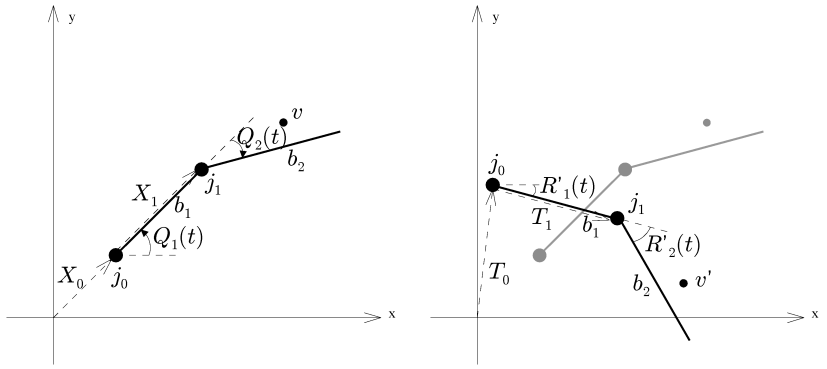
(b) The skeleton has been transformed from its bind pose to a new pose. We want to determine the new position of $\mathbf{v}$, $\mathbf{v}'$.

Figure B.2: The transformation of a skeleton from its bind pose to a new pose. The new position of $\mathbf{v}$, $\mathbf{v}'$, is to be found.

Remember that $\mathbf{X}_i$ can be found from Equation 1.1 on page 28 using $\mathbf{R}_{bind_i}$ instead of $\mathbf{R'}_i(t)$. This is shown in Figure B.3(a).

In Figure B.3(b) the skeleton is updated to a new pose. For $i > 0$, $\mathbf{T}_i$ now denotes the translation vector of bone b_i with $i - 1$ parents (found by Equation 1.1 on page 28) and $\mathbf{R'}_i(t)$ is the rotation animation of b_i. The current pose rotation matrix $\mathbf{P}_i$ of b_i is given by

$$\mathbf{P}_i = \mathbf{R'}_i(t)\mathbf{T}_{i-1}\mathbf{R'}_{i-1}(t)\mathbf{T}_{i-2}\cdots\mathbf{R'}_1(t)\mathbf{T}_0$$



(a) The bind pose rotation illustrated. (b) The current pose rotation illustrated.

Figure B.3: Illustrations of the bind pose rotation and the current pose rotation.

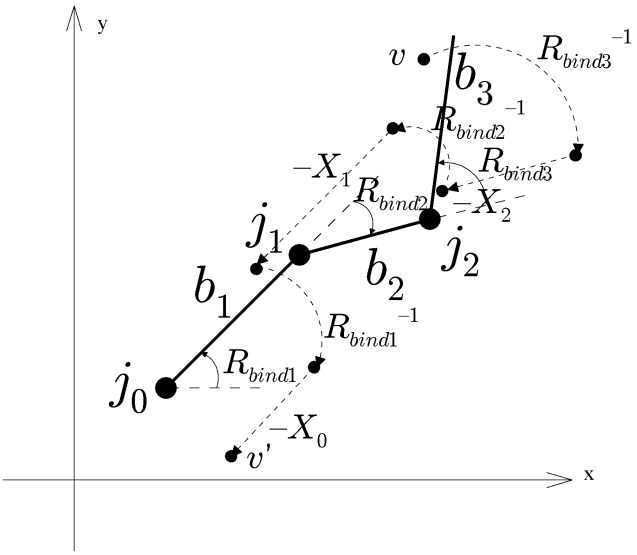
When updating $\mathbf{v}$ to $\mathbf{v}'$ we multiply $\mathbf{v}$ by $\mathbf{B}^{-1}$ to transform the vertex from world space to model space, and we then multiply that by $\mathbf{P}$ to transform the vertex back into world space. Generally, $\mathbf{v}'$ attached to bone b_i is thus given by

$$\mathbf{v}' = \mathbf{P}_i\mathbf{B}_i^{-1}\mathbf{v}.$$

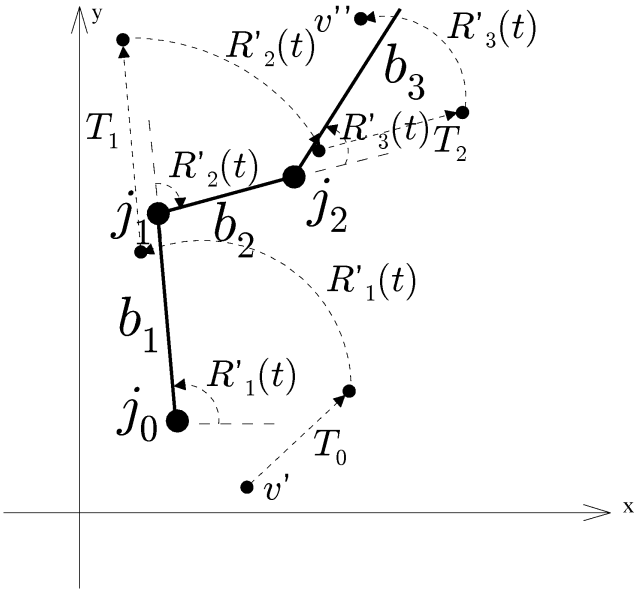
Figures B.4(a) and B.4(b) on the facing page illustrate the process of updating the position of a leaf vertex, $\mathbf{v}$, of an animated polygonal tree model, which is attached to the bone b_3. The skeleton rotates around joint j_0, j_1, and j_2 to be transformed from the bind pose, shown in Figure B.4(a), to the pose shown in Figure B.4(b).

To update the position of $\mathbf{v}$ we first transform it from world space to bone space of b_3. This is done by first applying the inverse rotation $\mathbf{R}_{bind_3}^{-1}$ around joint j_2 followed by the inverse translation $-\mathbf{X}_2$, and so forth. In this way, $\mathbf{v}$ is transformed to the bone space of b_3, where it is marked $\mathbf{v}'$.

The skeleton is then rotated around j_0, j_1, and j_2 to be in the pose shown in Figure B.4(b) on the next page at time t. $\mathbf{v}'$ is transformed from bone space of b_3 to its position in world space, $\mathbf{v}''$, by first applying the translation $\mathbf{T}_0$, which is the translation from the origin of the world coordinate system, then the rotation $\mathbf{R'}_1(t)$ around j_0, then the translation $\mathbf{T}_1$, and so forth.



(a) Leaf vertex $\mathbf{v}$ attached to bone b_3 (in the skeletal model bind pose) and transformed to the bone space of b_3, where it is denoted $\mathbf{v}'$.



(b) The leaf vertex in the bone space of b_3 (denoted $\mathbf{v}'$) and transformed to its new position, $\mathbf{v}''$, in world space, when bone b_1 is rotated around joint j_0 with $\mathbf{R}'_1(t)$, bone b_2 is rotated around joint j_1 with $\mathbf{R}'_2(t)$, and bone b_3 is rotated around joint j_2 with $\mathbf{R}'_3(t)$.

Figure B.4: An example of skeletal animation. The position of a leaf vertex, $\mathbf{v}$, which is attached to the bone b_3, is updated. The skeleton rotates around joint j_0, j_1, and j_2 to be transformed from the pose in Figure B.4(a) to the pose in Figure B.4(b) at time t, hence resulting in $\mathbf{v}$ being transformed to $\mathbf{v}''$.

B.1 Software and Hardware Skinning

Skinning can be performed on both the CPU and the GPU, which is referred to as *software skinning* and *hardware skinning*, respectively.

When doing software skinning the CPU updates the bone matrices, multiplies them with the vertex position vectors, and uploads the transformed vertices to the GPU. This happens in each frame.

When doing hardware skinning the GPU needs the bone positions per frame from the CPU [9]. Hence, the CPU updates the transformation matrices and uploads them to the GPU. One way of handling the vertex transformations, then, is to use a vertex shader.

There are different reasons for doing skinning on either the CPU or the GPU. If the application is the bottleneck, doing hardware skinning could reduce the load on the CPU. On the other hand, if, for example, collision detection is to be performed on skeletal animated meshes, or if shadows should be cast from these meshes, it is problematic to perform skinning on the GPU, since the CPU needs access to the transformed vertices [2]. Waveren [29] and Davies [6] detail an efficient way of using the CPU to do the skinning. Osborne explains how hardware skinning works in [21].

Appendix C

Spectral Clustering Improvements

C.1 Spectral Cluster Initialization

In this section we discuss how to initialize the k clusters in the clustering of spectral descriptions.

The K-means Clustering Algorithm is not guaranteed to find the best set of clusters, i.e. the set with global minimum squared error, where error is animation deviation in our application. As demonstrated in Section 3.1.3 on page 41, the clustering result depends largely on how the k clusters are initialized. No perfect solution exists to this problem, and a common attempt to get around it is to run the clustering algorithm several times with random initializations, and select the best clustering obtained.

In addition to random cluster initialization, we choose to also run the algorithm on clusters that already initially contain spectral descriptions with small deviation. This initialization is realized using the animation skeleton of the input CASTM. The squared error of the clustering obtained using this initialization is compared to the squared errors obtained using random initialization, and the best result is chosen.

Two triangles attached to the same bone are likely to have similar spectral descriptions, as they are subject to the same set of bone rotations. They are, however, not guaranteed to have identical spectral descriptions, and will in fact most likely not have, as two triangles positioned differently on the same bone do not have equal animation. Remember that a bone dictates a rotation around a joint, i.e. the origin of the bone, and given some rotation, a triangle far from the joint will move a larger distance than a triangle close to the joint. This implies a difference in animation properties, and thus yield different spectral descriptions. However, two triangles attached to the same bone are more likely to have similar spectral descriptions than two triangles attached to different bones, as they are subject to the same rotations.

We choose to initialize the k clusters by clustering the spectral descriptions of triangles according to which bones they are attached to. Let b be the number of bones in the skeleton:

- Create a cluster for each bone. If a bone has no foliage triangles associated (usually the trunk has none, for example), then the cluster for that bone will be empty. Delete all such empty clusters.

  - If $k = b$: A cluster is created for each bone, and the spectral descriptions of all triangles associated with this bone are assigned to the cluster.
  - $k > b$: Until there is no longer too few clusters, do the following:

    For each cluster, compute the average description of the associated spectral descriptions, and compute the average deviation the triangle descriptions has to this average. This average deviation is squared to yield variance.

    Variance indicates the amount of variation in a cluster. The cluster with the most variance is divided into two clusters. This is done by calculating the average distance the triangles have to the start joint of the bone associated with the cluster, and then divide the triangles into those that have further distance to the origin of the bone than the average distance, and those that have equal to or shorter distance to the start joint than the average distance.

    The intuition is that triangles with equal distance to the start joint are rotated an equal amount of distance when the bone rotates, and hence are more likely to have similar spectral descriptions than triangles with different distances to the start joint of the bone.
  - If $k < b$: Until there is no longer too many clusters, do the following.

    For each cluster, compute the average of the associated descriptions. Then for each pair of clusters, compute the deviation between their two average descriptions. The cluster pair with the smallest deviation between their average descriptions is joined to form a new cluster.

This process of merging and splitting clusters resemble the Hierarchical Clustering algorithms described in Section 3.1.2 on page 39, although no dendrogram is created.

C.2 Optimization Using Animation Zones

In Section C.1 on the preceding page it was argued that two triangles attached to the same bone are more likely to have similar spectral descriptions than two triangles attached to different bones. As explained, however, the two triangles will most likely not have identical spectral descriptions, as they are differently positioned relative to the bone and thus differently positioned relative to the rotation point around which the bone rotates.

This optimization is based on further elaboration on this observation. Two triangles positioned identically and attached to the same bone have equal animation and will yield identical spectral descriptions. Furthermore, two very

closely positioned triangles attached to the same bone have spectral descriptions with a small deviation. This leads to the introduction of *animation zones*.

An animation zone is a volume associated with a bone, within which all triangles attached to this bone are considered to have equal animation. A single common spectral description can therefore be created for all the triangles in an animation zone, which can improve greatly upon the performance of the Spectral Clustering Algorithm in terms of computation time. Employing animation zones in the algorithm requires the definition of the actual zones, and a definition on how to use spectral descriptions common for several triangles.

We propose to include a distance parameter, ϵ_z, in the Spectral Clustering Algorithm, which indicates the maximum Euclidean distance between two triangles in order for them to share a common spectral description. In other words, ϵ_z is used to perform an error-based clustering on the positions of all triangles attached to a bone, and each resulting cluster is an animation zone. We propose to use an error-based clustering algorithm. We propose to do an error-based clustering instead of a budget-based clustering, as we want to guarantee that all triangles in an animation zone are positioned within a given error bound, and furthermore it is not obvious how many clusters should be created for a bone if a budget-based approach were to be used.

For each bone, a clustering of the triangles is performed, resulting in a number of animation zones. A single spectral description is created for each zone by analyzing the animation of a centroid point for the zone. The centroid for an animation zone is the average world space position of all the triangles in the zone. Since the triangles are all attached to the same bone, the distances between individual triangles remain constant over the animation cycle, so the centroid can simply be computed from the triangle positions in the first animation frame. This average point is then attached to the bone, and its animation during the animation cycle is analyzed to yield the spectral description.

The Spectral Clustering Algorithm is modified to operate on spectral descriptions for animation zones instead of triangles. Basically, the algorithm just clusters animation zones instead of triangles. The average function of the spectral description type is modified to take the amount of triangles in a given animation zone into account when creating an average description for a cluster of animation zone spectral descriptions. Each spectral description is simply weighted by the number of triangles that are contained in the associated animation zone.

The result when using animation zones will differ from a result created without using animation zones, as triangles with slightly different animations would be considered to have a single shared animation.

Animation zones are introduced as a performance improvement to the Spectral Clustering algorithm, and determining values for ϵ_z that provides satisfactory results is subject to experimentation.

Appendix D

Summary

Billboard clouds, originally presented by Décoret et al. [7], is a well-established image-based rendering technique used for extreme model simplification. It simplifies polygonal models of arbitrary complexity to a greatly reduced set of textured polygons that replace the original geometry. However, a recognized weakness of existing billboard cloud algorithms is their lacking ability to simplify animated models.

In this thesis we focus on the simplification of animated polygonal tree models. Our goal is to improve the billboard cloud technique with our own solutions for simplification of the animation of polygonal tree models; an extension we refer to as animated billboard clouds. Specifically, we combine the billboard cloud simplification technique with our own developed animation simplification techniques in order to create discrete animated billboard cloud LOD models of polygonal tree models rigged with animation skeletons. These LODs consist of an animated billboard cloud for the foliage part and a simplified cyclic animation of the closed mesh constituting the trunk and large branches.

One of the benefits of the billboard cloud technique is that it can be used to create simplified tree models of high visual fidelity to the original model, both for models that should be viewed at close range and for models that should be viewed from afar. Similarly, our solutions are able to simplify animated polygonal tree models whilst remaining visually faithful to the original models, such that the simplified animated polygonal tree models are suitable for being displayed close up as well as and from far-away distances.

We present two solutions for simplification of animated polygonal tree models. Both solutions operate on polygonal tree models rigged with animation skeletons, although one of the solutions does not require a skeleton to simplify the animation. The solutions use our own implementation of a billboard cloud algorithm developed by Lacewell et al. [17], specifically aimed at the simplification of the foliage of static polygonal models.

In order to evaluate the visual fidelity of an animated billboard cloud model we define a set of error metrics that enable us to objectively measure the quality of the simplification. The metrics can be combined in different ways to measure the simplification quality expressed from different parameters.

A prototype implementation of one of the solutions has been created as

part of the project. This implementation, along with our implementation of the chosen billboard cloud algorithm, will be tested and evaluated.

Bibliography

[1] Activision and Treyarch. Call of duty 3, 2006. http://www.callofduty.com/.

[2] Tomas Akenine-Möller and Eric Haines. *Real-Time Rendering*. A. K. Peters, Ltd., Natick, MA, USA, 2002.

[3] Eike F. Anderson. Real-time character animation for computer games. http://ncca.bournemouth.ac.uk/gallery/files/innovations/2001/Anderson_Eike_220/CharacterAnimation.pdf.

[4] Christian Bauer, Chris Dams, Vladimir V. Kisil, Richard Kreckel, Alexei Sheplyakov, and Jens Vollinga. Ginac is not a cas. http://www.ginac.de/.

[5] J. Beaudoin and J. Keyser. Simulation levels of detail for plant motion. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 297–304, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association.

[6] Leigh Davies. Optimized cpu-based skinning for 3d games. http://www.intel.com/cd/ids/developer/asmo-na/eng/172123.htm.

[7] Xavier Décoret, François Sillion, Frédo Durand, and Julie Dorsey. Billboard clouds for extreme model simplification. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 689–696, New York, NY, USA, 2003. ACM Press.

[8] Christopher DeCoro and Szymon Rusinkiewicz. Pose-independent simplification of articulated meshes. In *I3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 17–24, New York, NY, USA, 2005. ACM Press.

[9] Bryan Dudash. Sdk white paper. matrix palette skinning, an example. http://download.nvidia.com/developer/SDK/Individual_Samples/DEMOS/Direct3D9/src/HLSL_PaletteSkin/docs/HLSL_PaletteSkin.pdf.

[10] Anton Fuhrmann, Eike Umlauf, and Stephan Mantler. Extreme model simplification for forest rendering. *Natural Phenomena 2005: Eurographics Workshop on Natural Phenomena*, pages 57–66, 2005. http://www.vrvis.at/vr/billboardclouds/TR_VRVis_2004_039_Full.pdf.

[11] Jason Gregory. Animation in video games. http://www-scf.usc.edu/~gamedev/animation.ppt.

[12] I.-T. Huang, K. L. Novins, and B. C. Wünsche. Improved billboard clouds for extreme model simplification, 2004. http://www.cs.auckland.ac.nz/~novins/Publications/Huang04.pdf.

[13] Inc Interactive Data Visualization. Speedtree, 2006. http://www.speedtree.com.

[14] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Comput. Surv.*, 31(3):264–323, 1999.

[15] G. Jungman and B. Gough. Gsl - gnu scientific library. http://www.gnu.org/software/gsl/.

[16] V. C. Klema and A. J. Laub. The singular value decomposition: its computation and some applications. *ieeeac*, AC-25:164–176, 1980.

[17] 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. http://www.cs.utah.edu/~lacewell/billboardclouds/billboardclouds.pdf.

[18] Jon Leech. Tessellate a sphere with triangles (source) ii, 1989. http://www.gamedev.net/reference/articles/article427.asp.

[19] Matteo Matteucci. A tutorial on clustering algorithms. http://www.elet.polimi.it/upload/matteucc/Clustering/tutorial_html/kmeans.html.

[20] Microsoft and Epic Games. Gears of war, 2006. http://gearsofwar.com/Emergenceday/.

[21] Richard Osborne. Matrix palette skinning using vertex shader 1.1. http://www.cyberkreations.com/kreationsedge/index.php?url=tut/Games/GPUSkin.

[22] Hormoz Pirzadeh. Rotating calipers homepage, 1999. http://cgm.cs.mcgill.ca/~orm/rotcal.html.

[23] P. Prusinkiewicz and A. Lindenmayer. *The algorithmic beauty of plants*. Springer-Verlag, New York, 1990. http://algorithmicbotany.org/papers/#abop.

[24] I. Remolar, M. Chover, O. Belmonte, J. Ribelles, and C. Rebollo. Geometric simplification of foliage, 2002.

[25] Julius O. Smith. *Mathematics of the Discrete Fourier Transform (DFT)*. W3K Publishing, http://www.w3k.org/books/, 2003.

[26] Bethesda Softworks and 2K Games. The elder scrolls iv: Oblivion, 2006. http://www.elderscrolls.com/games/oblivion_overview.htm.

[27] The OGRE Team. Ogre 3d : Open source graphics engine, 2006. http://www.ogre3d.org/.

[28] Eike Jens Umlauf. Image-based rendering of forests. Master's thesis, Technischen Universität Wien, August 2004. http://www.vrvis.at/TR/2004/TR_VRVis_2004_032_Full.pdf.

[29] J.M.P. van Waveren. Fast skinning. http://www.intel.com/cd/ids/developer/asmo-na/eng/dc/games/293451.htm.

[30] James Van Verth and Lars Bishop. *Essential Mathematics for Games and Interactive Applications: A Programmer's Guide*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[31] Jason Weber and Joseph Penn. Creation and rendering of realistic trees. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 119–128, New York, NY, USA, 1995. ACM Press.

[32] David Whatley. Toward photorealism in virtual botany. *GPU Gems 2*, 2005. Chapter 1.