Frame-Coherent 3D Stippling for Non-Photorealistic Computer Graphics

Dissertation

zur Erlangung des akademischen Grades

Doktoringenieur
(Dr.-Ing.)

angenommen durch der Fakultät für Informatik
der Otto-von-Guericke-Universität Magdeburg

von M. Sc. Oscar Ernesto Meruvia Pastor
geboren am 14. November 1972 in Mexico, D.F.

Gutachter: Prof. Dr. Thomas Strothotte
Prof. Dr. Eduard Gröller
Prof. Dr. Wolfgang Strasser

Magdeburg, den 17. Oktober 2003

Zusammenfassung

Die Dissertation befasst sich mit der Frage, wie nicht-photorealistische gestippelte Animationen anhand dreidimensionaler Modelle produziert werden können. Nicht-photorealistische Animationsverfahren sollten drei wichtige Konzepte berücksichtigen, die durch die Anwendung von zweidimensionalen Zeichenstilen auf dreidimensionale Modelle entstehen: Framekohärenz, Skalierbarkeit und Formänderungen. Als Lösung werden Punkthierarchien vorgeschlagen, mit denen die oben genannten Aspekte in eine einzige Lösung integriert werden können. Es wird beschrieben, wie solche Punkthierarchien aufzubauen sind und welche Eigenschaften die Punkte der Hierarchie haben müssen, um framekohärente, sichtabhängige, skalierbare Animationen statischer und dynamischer Modelle zu ermöglichen. Zwei unterschiedliche Verfahren zum Aufbau von Punkthierarchien werden präsentiert und Anwendungen im Bereich der Animationen für das Echtzeit-Stippling statischer Modelle, der Herstellung von Zeichentrickfilmen, der Archäologie und der medizinischen Illustration werden erläutert.

Abstract

This thesis deals with the question of how to produce non-photorealistic stippling animations of 3D models. Non-photorealistic animation techniques ought to take into account three important issues that arise when applying 2D drawing styles to 3D model rendering: frame-coherence, scalability and view-dependance, and rendition of animated models. To deal with these issues, the author proposes point hierarchies as an integral and modular technique to generate non-photorealistic animations, and describes how point hierarchies are used to produce stippling animations of static and animated models which are frame-coherent, view-dependant and scalable. Two different approaches to create such point hierarchies are provided and applications of animated stippling are shown in the areas of cartoon animation, medical illustration, real-time rendering, and archaeology.

Foreword

When I started my Ph.D four years ago, I was given the task of further developing existing work in NPR done here at the Institute of Simulation and Graphics. This thesis is the result of this request. As I started to write it, I wanted to make it not only comprehensible for the specialized reader, but also I wanted any interested reader to understand the place of this work within the area of Non-Photorealistic Rendering / Computer Graphics, for which the first two chapters were mostly devoted.

For the specialist in Computer Graphics, I also wanted to write a detailed description of both the challenges that I faced at the beginning of the work, my solutions to these challenges, and the reason other solutions were not considered or discarded at the time. The final chapters of the thesis are the results of the work, they deal with the areas of application and the possibilities which are left open for future research.

There is a number of people who should receive credit for their friendly support and collaboration during this work. I would like to thank my supervisor Thomas Strothotte, who provided me with a number of challenges, visions and strategic support during the development of this work, in conjunction with a great deal of research independence.

Thanks a lot to Felix Ritter and Nick Halper for their "technical support" in the form of advices, tips and hints during the development (programming) of the stippling system. Whenever there was a problem with OpenGL, or with the way the first approximations to the final solution looked like, I always received help and feedback from them.

Of course, I would like to thank my senior editor and associate, Lourdes Peña Castillo, who has always been on my side, giving not only "non-technical support", when things looked not so good, but also technical support in the areas of editing, proof-reading, and readability (this paragraph not included, of course) for the dissertation and my publications.

Many thanks to my mother and brothers for their unconditional support delivered from a foreign country far, far away, and to all the friends I met in the Institute and at the University, which made life in Magdeburg a lot nicer, with a special mention to Petra Specht who was there from the first day of my stay on German soil.

Contents

1 Introduction

Producing photorealistic images (or reproducing reality) has been an ever present aspiration in the computer graphics community, which has constantly tried to create computer-generated imagery which resembles reality in such a way that a computer-generated image cannot be differentiated from a photograph. In the area of Non-Photorealistic Animation and Rendering (NPAR), graphics researchers pursue several goals related to the rendition of models in a variety of styles which avert from the photorealistic.

The fact that the area of Non-Photorealistic Rendering (NPR for short) is defined as a negation of the photorealistic has given raise to the concern that its field of action is not well defined. Nevertheless, it is fortunate for some researchers in this field that the emulation of artistic rendering techniques has fallen directly into the zone of competence of NPR right from the beginning of this discipline. This encompasses all the graphic arts that have emerged along the development of humanity, such as African drawings, Chinese and Japanese painting, impressionism, pointillism, cartoon illustration, Mexican painting and so on. Scientific illustration styles based on pen-and-ink illustration styles like cross-hatching, hatching and stippling are also included in NPR, In sum, any illustration style from any time, and any civilization is a valid subject for NPR.

Researchers in NPR have developed algorithms which let us emulate pencil and charcoal illustrations, watercolor rendering, and most of the graphic arts previously mentioned. A systematic overview of such algorithms and techniques is provided by Strothotte and Schlechtweg [Stro02], and Gooch & Gooch have provided an overview of literature on NPR [Gooc01].

The work described in this thesis falls within this context. Our research addresses a fundamental question in the field of NPR: how to produce animations using NPR styles in a frame-coherent style. Until recently, little attention has been given to producing animation for NPR which is frame-coherent. In this thesis, we want to em-

phasize the relevance of including frame-coherence as a fundamental attribute of non-photorealistic animation. In particular, we explore the creation of point hierarchies for Non-photorealistic Animation and Rendering which are suitable for producing illustrated animations of 3D models with particular emphasis on the reproduction of the rendering technique known as stippling. [1]

1.1 Combination of 2D Rendering Styles with 3D Graphics

In this work, we want to use the stippling style for rendering 3D models. There are a number of aspects that must be considered when transferring a rendering style which is originated from 2D into the realm of 3D graphics. This comes from the fact that a single image is static, while 3D graphics are dynamic by nature. There are a number of issues related to this unique relationship between 2D artistic rendering styles and 3D computer graphics. In the following lines we discuss those issues that we have found to be of most relevance within the context of our work.

1.1.1 Frame-Coherence

Frame-to-frame coherence, also referred to as frame-coherence, is the term that describes coherence of drawing primitives and objects between subsequent frames of an animation. Frame-coherence responds to the expectation that a viewer has that during an animation, object transitions and depth information varies in a way that corresponds to the real world. Viewers expect frame-coherence in an animation, because it is in the nature of the real world that objects are not constantly jumping around or jumping back and forth, appearing and disappearing as a scene evolves.

When dealing with non-photorealistic rendering systems we need to differentiate between frame-coherence at the object level and frame-coherence at the particle level. Frame coherence at the object level refers to objects being smoothly displaced across the frames of an animation, this also includes the object attributes such as shading (in which case we refer to shading changing smoothly across frames).

Frame-coherence at the particle level implies a higher level of coherence across frames where not only the object, but also the primitives which are used to give it shape, color or tone, are frame-coherent. Frame-coherence at the object level does not imply in any way frame coherence at the particle level. To more clearly illustrate this

[1] The difference between the terms NPR and NPAR is that NPAR includes animation or animated models, where NPR alone describes mostly single rendition and static models. For simplicity, many researchers use the term NPR to refer to NPAR.

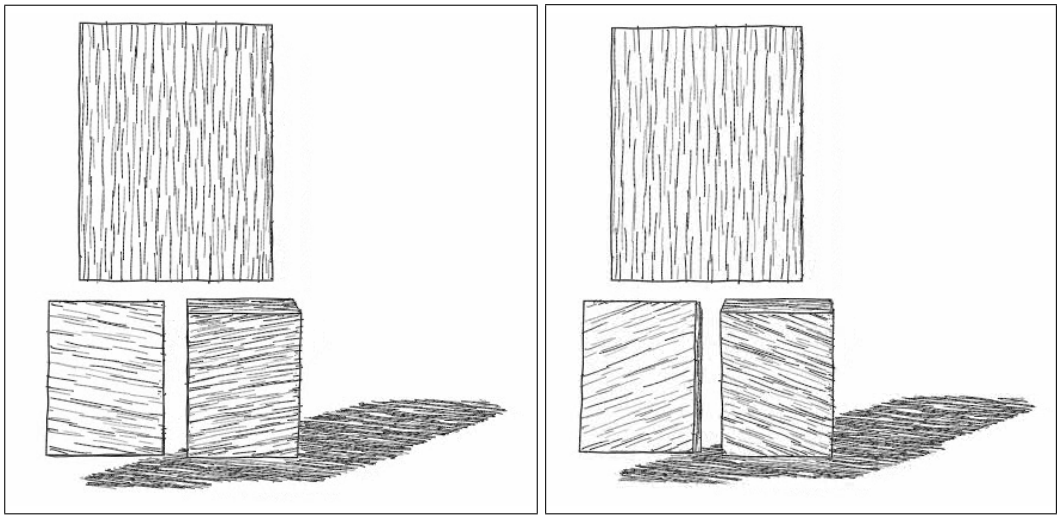

Figure 1.1: Lack of frame-coherence at the stroke level during an animation: while the box objects are slightly different, the strokes that are used to shade them have noticeably changed [Haga01].

difference between the two types of frame coherence we have Figure 1.1 and Video 1-*WalkingBoxes*[2].

In Video 1 we observe that the animation is coherent with respect to the shading level and the model displacement (spatial-temporal coherence), but if we look at the individual strokes on the surface of the objects, we notice how they change even when an object stops moving. Figure 1.1 shows two consecutive frames of Video 1. We can observe that strokes of the object on the left are not in the same orientation than the strokes of the object on the right. The video shows that animations which do not enforce frame-coherence at the particle level are noisy, and this noise has a negative effect on the perception of the animation by the user, in a similar way as the lack of frame-coherence at the object level affects the user experience in virtual environments [Mack93, Ware94, Elli99]. The issue of having two different types of frame-coherence arises only in non-photorealistic rendering, because photorealistic rendering is meant to be fully frame-coherent.

1.1.2 Scaling and Projection

Another natural attribute of computer graphics is scalability. In 3D graphics, we model objects from the real world and take for granted that is it possible to view the models within a wide range of scales. It is normal for a modeler for example to scale up, or

[2]The animations mentioned in this thesis are included in the electronic version delivered with this work and are also publically available under http://isgwww.cs.uni-magdeburg.de/~oscar/.

zoom in, a certain part of a model work on it, and then scale the model so that it fits in the entire screen and judge the aspect of his modifications. It is normal for engineers or scientists to take a closer look at modeled parts of factories or molecules to figure out their role as part of a whole.

In NPR scalability is a seemingly difficult issue, because 2D illustrations are typically not scalable in the way 3D models are, with the exception of the work of Salibury et al., who considered scalable rendering of pen-and-ink illustrations [Sali96]. That not being enough, a 2D artistic image does not provide many clues as to what an artistic image of a 3D model should look like, because the rendering styles have evolved under precisely opposite premises: images and paintings try to capture a world which exists in 3D and figure out ways to adequately represent it in 2D, even at the expense of distorting some parts of the models or objects so that they can be adequately perceived in the mind of a viewer and even when they cannot really exist as such in 3D (take cubism as an example) [Coop95].

Another issue present while scaling NPR imagery is to decide how the individual strokes should be scaled. It is a matter of artistic nature to decide how individual strokes should scale. A simple solution would be to represent strokes as objects in 3D space which would scale together with the object. Unfortunately, this approach does not work successfully in NPR, because a user does not expect a stipple dot to grow to the size of a large circle in the middle of the screen, or a line stroke to grow into a large polygon which has no artistic expression anymore. This is the reason why we propose the regulation of the presence of the drawing primitives using a point hierarchy to indicate the size of the stipples at different levels of detail.

1.1.3 Morphing and Animation

Another possibility available in 3D graphics is that of having objects change in shape. This happens for example, in character animation, in the visualization of simulations of objects being deformed, model morphing or during interaction with deformable objects. This is also a novel aspect to think of in NPR graphics, because the traditional techniques do not provide hints as to how a rendering style should be modified to account for morphing. It is not clear whether strokes should appear or vanish during the deformation, or whether the strokes should maybe adapt to the deformation to some extent. As with the previous issues, what should occur to the individual strokes or stipples during deformation, is a question of artistic preference.

1.2 Stippling

Stippling is the art of painting with dots. Stippling belongs to the group of pen-and-ink illustration techniques, together with hatching and cross-hatching. This technique is used in scientific illustration, archaeology, and for artistic illustrations. A definition

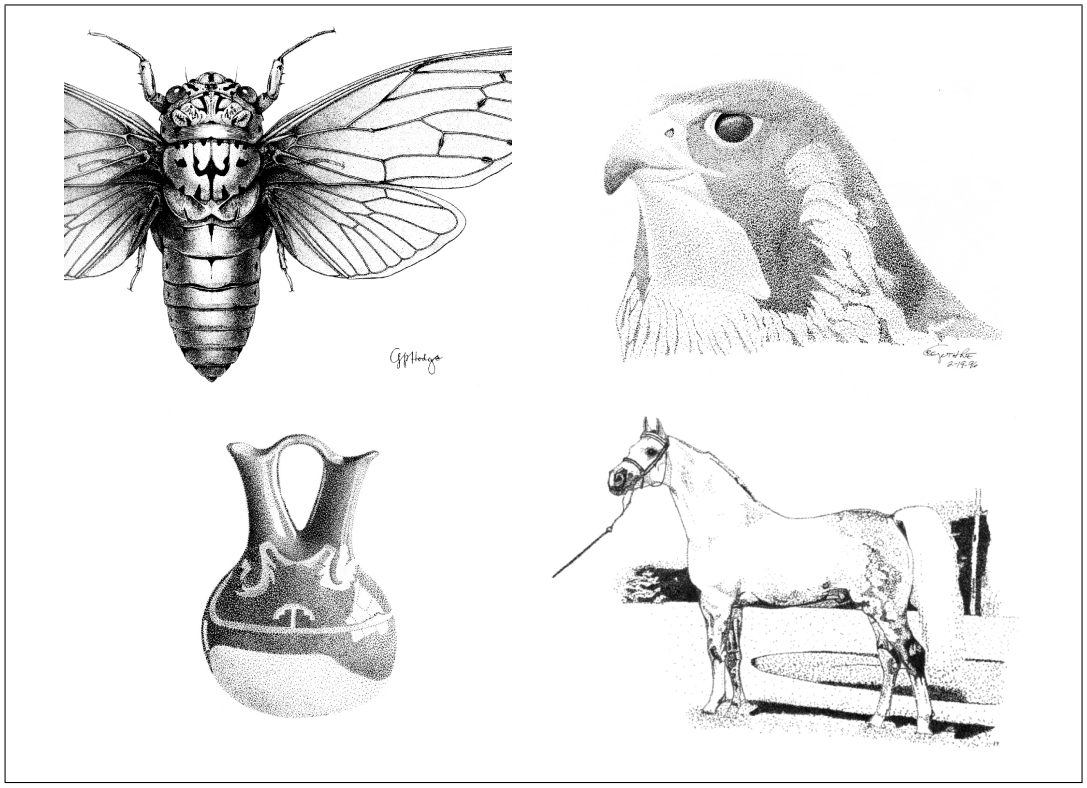

Figure 1.2: Stippled drawings made by scientific illustrators and artists. In the figure at top-left and bottom right, artists use line drawings to emphasize relevant features of the models. On the top-left: image by Hodges, next two images are by Ron C. Guthrie, Image on the bottom right by JandJ Designs.

of stippling [Rigg99] says: " stippling is a drawing technique in which dots rather than lines form an image. Groups of small dots placed close together will read as a patch of gray tone from a distance. By altering the size and spacing of the dots, it is possible to create a full tonal (or value) range. ... Stippling is useful as a texture-producing technique, and colored stippling is used for the pointillist technique of color mixing". The tools required to produce stippled drawings are simple, one only needs a point pen, ink, and appropriate paper.

The set of images in Figure 1.2 show stippled illustrations from various artists. These images show that almost any topic of nature and the arts can be a subject for stippling.

Figure 1.3: Stippled drawing and detail by George Robert Lewis. Notice the regular spacing between dots and the silhouette on the image on the right which lifts up the object from the background [Hodg88].

The common denominator in stippled images is that dark tones are obtained by placing together a large amount of dots and that light tones are obtained by placing a few dots which are also regularly spaced, as shown in the detail on the right side of Figure 1.3.

In *scientific illustration* [Hodg88], Hodges describes a process of stippling which involves carefully distributing a group of dots and then evaluating the areas which require more dots and filling in as required. Another characteristic of this style is that stippled images do not scale well. In fact, Hodges mentions that a stippled image can only be successfully reproduced within a small range. Stipples cannot be too small, because they would vanish if the image is scaled down. When a stippled image is excessively scaled down the collection of dots tends to combine with each other and a uniform grey tone replaces the individual dots.

Stippling is also used in conjunction with other rendering styles. For instance, many stippled drawings are outlined to enhance the profile of the rendered objects, which

happens commonly in scientific illustration (as in Figure 1.3). Line strokes can also be used in conjunction with stippling as shown in the insect drawing in Figure 1.2.

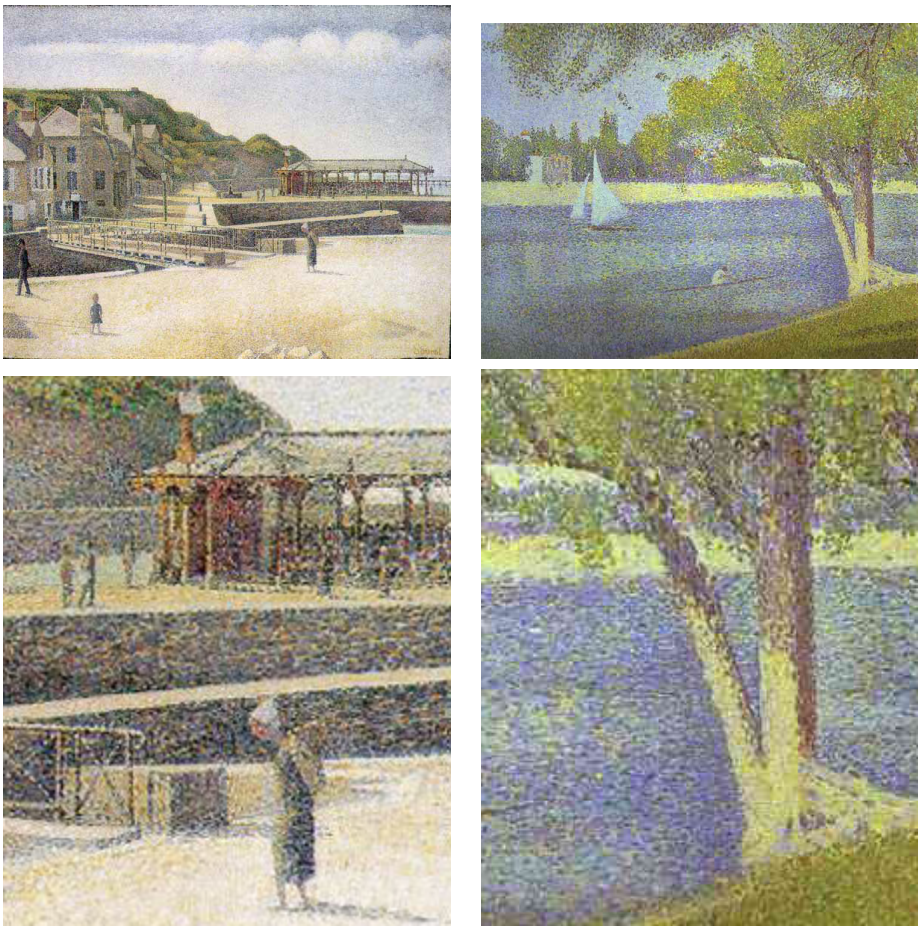

Figure 1.4: Pointillist paintings by Georges Seurat: On the left, "Port-en-Bessin" (1888). On the right, "The Siene at La Grande Jatte, Spring" (1888) (Original Sizes: 83x66cm, detail sizes: 18x33cm).

Traditional stippling is done in black-and-white, and some stippling artists describe their work as pointillism. The technique called pointillism was introduced in the end of the 19th century, and is represented in the works of neo-impressionist painters Georges Seurat and Paul Signac. Pointillism is defined as "a technique whereby solid forms are constructed by applying small close-packed dots of unmixed color to a white background". Figure 1.4 (top row), shows examples of paintings in the pointillist style. We have included details of these images in the bottom row to illustrate the individual dots which make up the images (we indicate the original measures of the paintings as $width * height$). Stippling and pointillism are closely related, but a fundamental difference is that in stippling shading is given by the density of dots in a region, while

in pointillism, dot density is mostly uniform and shading is given by the darkness of the selected color.

1.3 Computer-Generated Stippling

Since the early 1990's, graphics researchers have worked on the emulation of the stippling technique. In a paper on Computer-Generated Pen-and-ink illustration [Wink94], Winkenbach and Salesin propose the use of *stroke textures* to represent both texture and tone in a texture array (see Figure 1.5).

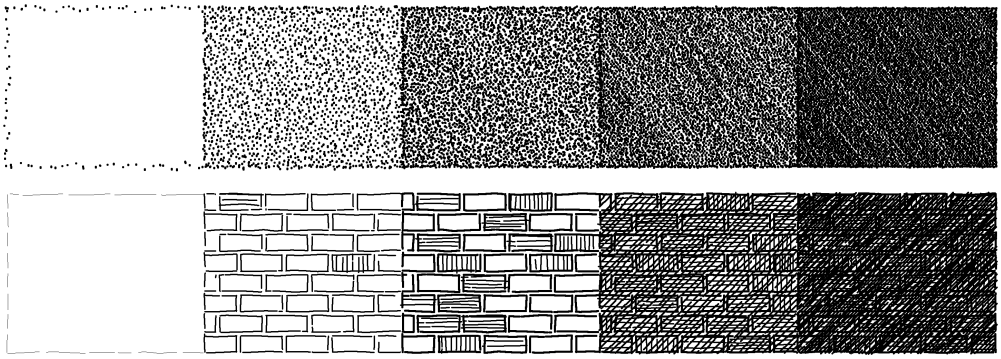

Figure 1.5: Stroke textures can provide both tone and texture [Wink94].

In their approach, the stippling style is implemented using textures filled randomly with an ever increasing amount of stipples that represent different levels of darkness. The issue of how stroke textures should be scaled is also discussed, but the emphasis of the work is on solving level of detail issues for complex stroke textures that represent objects like brick walls or the windows in a house in the style of pen-and-ink illustrations.

The same authors also proposed a way to produce stippled and hatched images using parametric surfaces [Wink96]. The amount of stippling in a certain region of the image depends directly on the desired shading tone. The authors emphasized the production of highly detailed single illustrations in the pen-and-ink style, Thus, issues on frame-coherence are not discussed. In Figure 1.6 we see an image produced for parametric surfaces where the stipples are randomly distributed in the shaded areas. The feeling of a rough surface is conveyed by the random distribution of the dots in the shaded areas. In addition, the overall distribution of the stipples, where no sharp boundaries are shown, indicates a surface with low reflectance (non specular).

Deussen et al. [Deuso0] found that one important feature of stippled images is that points in stippled images are evenly distributed and match the underlying shading

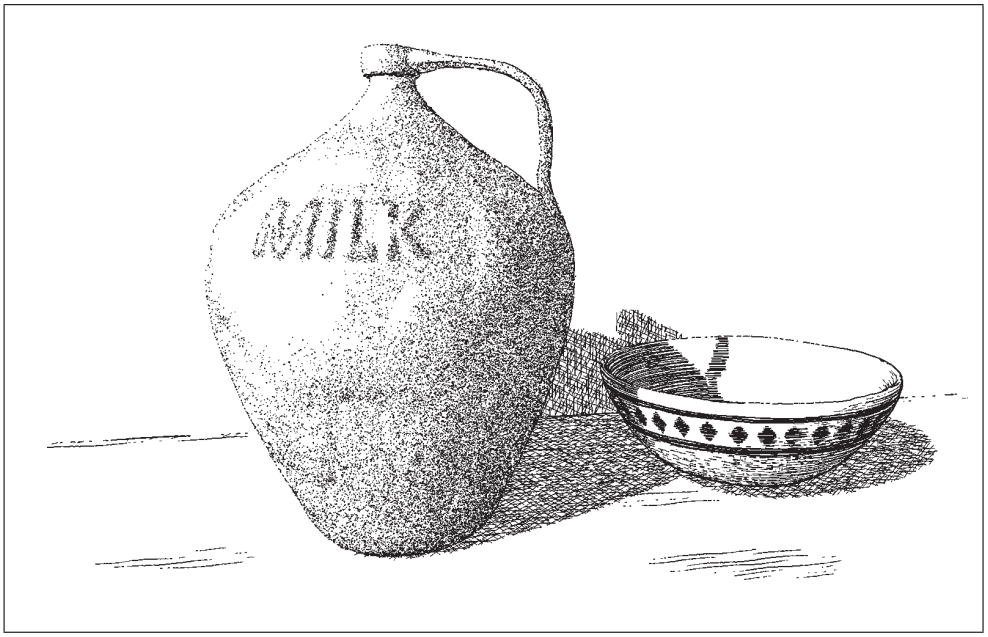

Figure 1.6: Computer-generated illustration of a Ceramic Jug and Bowl by Winkenbach and Salesin [Wink96].

tone. Contrary to a point distribution produced randomly, a regular point distribution in a stippled image conveys a cleaner look to the image. They proposed a system based on point relaxation using Voronoi diagrams. In their approach a grey scale image is rendered in the stippling style by first distributing points at random on a canvas in such a way that the stipples matched the target shading level. After that, the Voronoi diagram of the distributed particles is computed. A Voronoi diagram is a space partition scheme on the image plane where each particle (or stipple in this case) is contained in a 2D cell. The cell includes all those points which have the particle of the cell as its closest neighbor. The cell is limited by those points which are at equal distance from one or more particles. In the approach of Deussen et al., the center of the cells in the Voronoi diagram are iteratively relaxed, which means that the stipples in the Voronoi cells are subject to the influence of their neighbors and are lightly displaced as a result of repelling forces exerted by its neighbors. After several iterations, a set of evenly space stipple points is obtained (see Detail in Figure 1.7).

The system of Deussen et al. includes a tool to help artists control the boundaries of the stippled regions and retouch the image to produce a better looking image. In a similar approach, Secord presented a method which was also based on the relaxation of Voronoi cells [Seco02a]. In this system, the user only needs to setup a couple of parameters about the stipple quantity and size so that images at different scales can also be automatically produced. An advantage of this approach is that this stippling systems can use any grey scale image as input, not only 3D model renditions. On the

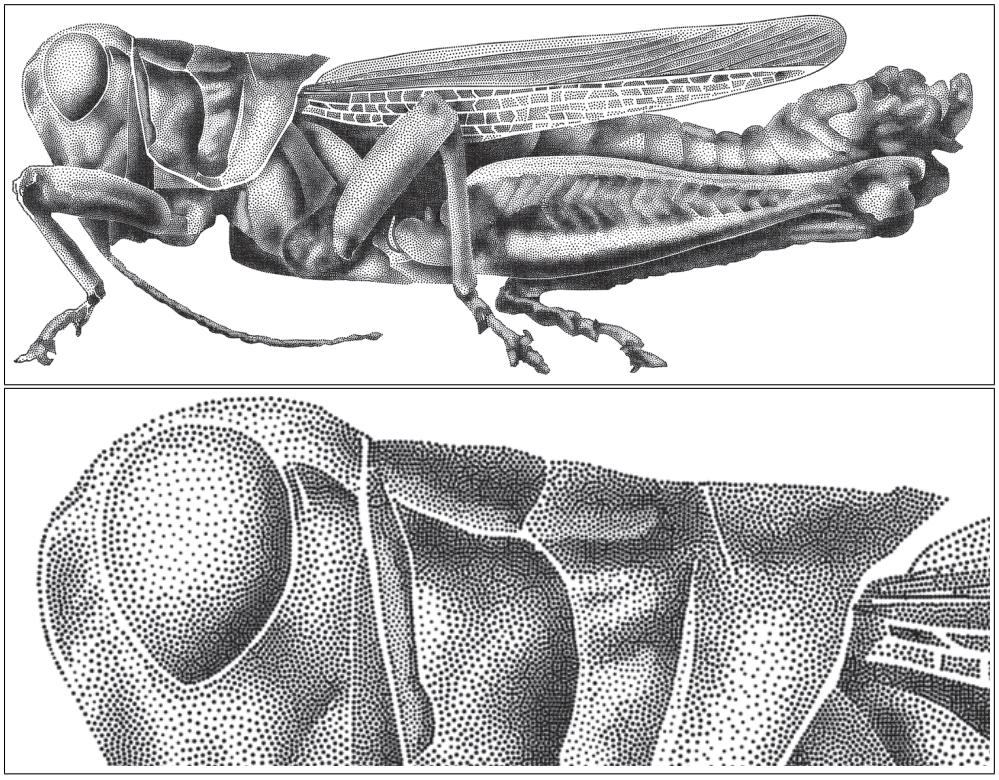

Figure 1.7: On the top, a computer-generated stippled image produced by Voronoi relaxation, on the bottom a detail of the grass-hopper's head (images by Deussen et al. [Deus00]).

other side, these systems are designed to produce single images, so they can hardly be used in the production of animation sequences, because they lack frame-coherence.

1.4 Thesis Contributions

This work presents point hierarchies as a solution for producing non-photorealistic animations. The point hierarchies presented in this work are meant as an integral solution to the problem of distributing particles on the surface of 3D models. The use of point hierarchies in the framework here presented, allows the production of view-dependent, frame-coherent animations of static and animated models. In addition, there is an emphasis in producing appropriate point distributions on the surface of the model such that the point distribution successfully emulates drawing in the stippling style.

In the following lines we provide an overview of the work developed in the context of this thesis:

Development of a large number of functions to inspect the geometry of the input models. We have developed a library of functions and modules which allow us to produce several applications which assist the task of generating point hierarchies and animations in the stippling style.

File conversion tools which allow the manipulation of models and the import and export of polygonal models into our systems' file format, Open Inventor: We have developed modules to convert quad-based models into triangular-based models, to convert between VRML formats to Open Inventor, to revert the face-ordering, and to convert output produced by 3D Studio Max animations to the Open Inventor format. With these conversion tools it is possible to convert animations created with the external software 3D Studio Max into the native format of our application (Open Inventor). Our internal graph model represents 3D models as a mesh of triangular faces, thus, we need to convert models in other formats, such as quads or other polygonal representations into triangular meshes.

Modules for Mesh inspection We have produced modules to perform basic mesh inspection tasks, for modifying input meshes, for computing normals in both flat-shading and gouraud-shading styles [Fole93], and for consulting information about mesh connectivity such as vertex and edges connectivity and similar which are required for performing other tasks described below. We have used as our base connectivity data structure the directed graphs implemented in the Leda Algorithms Library [Mehl00].

Applications and Modules for mesh simplification and subdivision We have produced two main applications which we use to generate point hierarchies. The first application produces point hierarchies based on mesh simplification and mesh subdivision. We have produced the modules required to perform these tasks separately and we have developed several applications which perform either simplification and subdivision and one application which combines both approaches to produce seamless point hierarchies by mesh simplification and subdivision. The mesh simplification modules is based in the idea of Progressive Meshes [Hopp96, Hopp97], while the mesh subdivision module was independently developed.

Modules for Surface Subdivision We have developed applications to perform surface subdivision of polygonal meshes. Our surface subdivision algorithms create closed regions which we call polygonal patches, and are our original work [Meru02a]. These polygonal patches are used for visibility preprocessing and for creating patch hierarchies and performing patch-based token relaxation, as described below.

Applications and Modules for Patch-based Token Relaxation An alternative approach for creating point hierarchies is by performing point relaxation and

patch-based token relaxation based on the work of Turk [Turk91, Turk92]. In this approach, we first create patch hierarchies by means of patch fusion procedures and then we use these patch hierarchies to distribute tokens over the surface of a model based on existing schemes for point relaxation, which is our original idea.

Applications and Modules for producing Stippling Animations We have developed tools to capture key-frame information and to generate off-line animations of stippled models. Our modules are distributed among several applications. The first is the key-frame capturing tool, which allows the user to produce files which encode an animation sequence based on changes in illumination and camera positioning. During off-line rendering, our applications make use of the key-frame files to produce animation sequences by interpolating the information contained in the key-frame files. For the case of animated models, the key frame files also contain pointers to the shape of the model at the given key frame.

Applications for performing Real-Time Stippling In the context of this work, an application which makes use of the point hierarchies created by either simplification and subdivision or token relaxation was implemented. The application runs on computers equipped with hardware graphics acceleration cards and shows several models which have been imported to that application, including models from archaeology.

Stippled Animations and Illustrations The group of modules and applications we have developed in the context of this work has enabled us to find a systematic way to produce illustrations and animations of static and animated models in the stippling style. We have made experiments with several models and have proven the generality of our techniques, which means that we can use any polygonal model and apply our rendering techniques.

1.4.1 Thesis Structure

We have structured the presentation of our work in eight additional Chapters. In Chapter 2 we describe systems which integrate non-photorealistic rendering styles with 3D graphics, and describe how they address the issues of frame-coherence, scalability and morphing and to which extent. In Chapter 3 we provide a solution to deal with these issues. We present a general framework to produce NPR animations using point hierarchies and describe how point hierarchies deal with the problems mentioned before in an integral way. We also explain how it is possible to decouple the process of point hierarchy creation from the actual rendering and how stippling can be used in conjunction with other rendering styles, traditional and non-photorealistic.

We have developed two solutions to create point hierarchies suitable for use in NPR. In Chapter 4 we describe the first solution method, where point hierarchies are created by means of mesh simplification, subdivision and randomization. Work described in this Chapter has been published in the IEEE Computer Graphics and Applications Special Issue on Non-Photorealistic Rendering, which came out of the press in the July/August 2003 issue of the magazine [Meru03].

In Chapter 5 we describe our second solution method to create point hierarchies, which is based on point relaxation. The technique was developed with the purpose of improving the point distribution, and is based on point relaxation. Point relaxation has been used in other two dimensional stippling systems, as well as for texture synthesis and for performing mesh simplification. The technique is described as Token-Based Patch Relaxation, but for simplicity we refer to it as Patch Relaxation or Token Relaxation.

In Chapter 6 we describe how point hierarchies can be used to produced stippled renditions of animated models which maintain scalability and frame-coherence. Here we describe the issues of point distribution under deformations, and how to adapt the point hierarchy to these deformations, presenting both a theoretical and a practical solution. Since we address the problem of animations in NPAR, we have included a set of animations which are part of the documentation of our work. Our animations are available in the electronic version of this work, and also on the web site of the Department of Simulation and Graphics of the University of Magdeburg.

In Chapter 7 we present an informal analysis on the applications of stippling by using point hierarchies and deal with such issues as model suitability, shading styles, and animation effects. Stippling is a technique used frequently in Archaeology to document findings from the excavation sites. In Chapter 8 we describe our collaboration with the Office of Archaeology of the State of Sachsen-Anhalt, where we took objects found in their excavation sites, scanned them in 3D and produced stippled drawings for their assessment. In this collaboration we received valuable feedback as to the feasibility of applying our rendering technique to archaeological models.

In the last Chapter (Chapter 9) we summarize our results and present interesting directions of future work, with an emphasis on those aspects which have more relevance to the area of NPAR.

2 Non-Photorealistic Rendering Systems for 3D Models

In the previous Chapter, we mentioned three main issues that need to be considered when using 2D rendering styles for 3D models: frame-coherence, scaling and morphing. Several researchers have proposed different solutions to deal with these aspects. We can group these solutions according to the structures used to represent and place primitives in NPR as follows: particle systems, textures, silhouettes, volume and point-based renderers, and image-based particle systems. In this Chapter we describe these methods and the benefits and disadvantages that they have, and put our technique in the context of these solutions.

2.1 Particle Systems

A particle system is a collection of points located on the surface of a 3D model. Each point indicates the position where a drawing or rendering primitive should be drawn. The rendering primitive can be a paint stroke or a line stroke, a stipple dot, or a small graphical element.

The seminal piece of work on particle systems for NPR is that of Barbara Meier [Meie96], *Painterly Rendering*. In this work, the concept of a particle system where particles are randomly distributed on the surface of a 3D model is presented. The reason that Meier used particles in her system was that particles satisfy the requirement of frame-coherence. Since the particles are attached to the surface of a model, the particles move along with the model as it moves, or as the scene changes. Meier used the particles to distribute small paint strokes on the surface of the model, the paint strokes simulated different effects, such as watercolors and oil painting. Each paint stroke is represented using a textured surface. The overall system combines three different inputs: one was the particles system, which determined the position of the

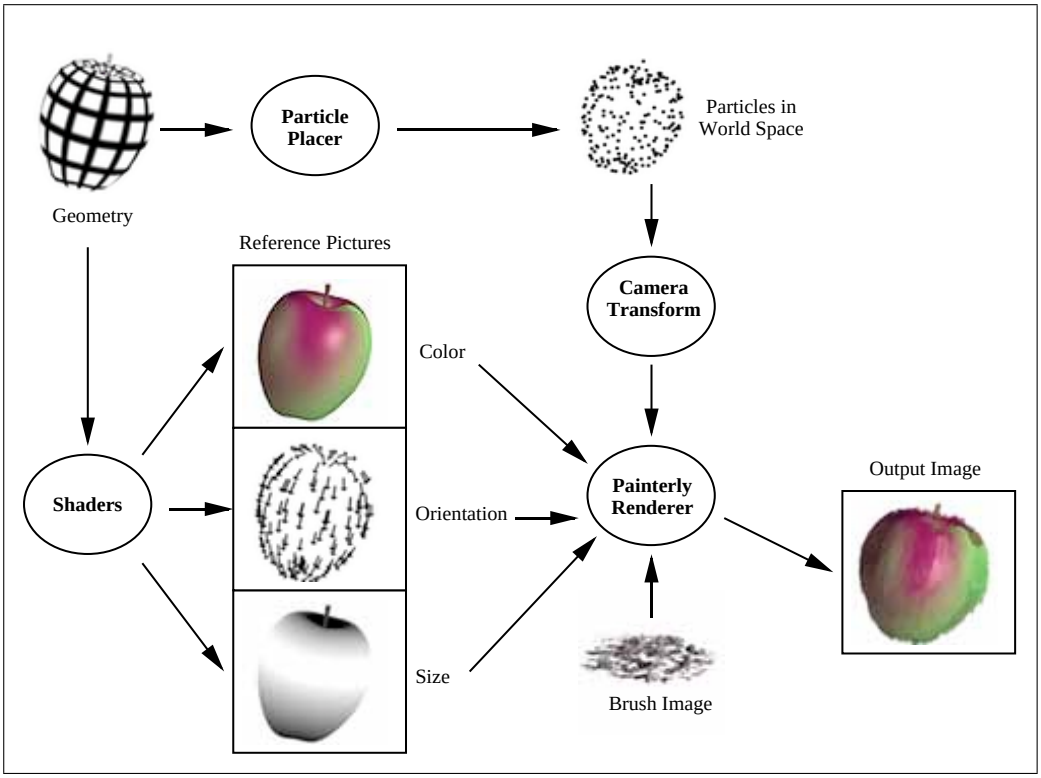

Figure 2.1: Structure of the particle system for painterly rendering by Meier [Meie96].

individual strokes; the other was the painting strokes and the last element corresponds to lighting (see Figure 2.1). The major contribution of Meier was the realization that one way to obtain frame-coherence in NPR is to attach particles to the surface of the model. The initial work presented by Meier, however, was not complete in the sense that while the particle system presented by her did achieve frame-coherence, it was not scalable, which means that the models could only be seen at a given distance, but not at arbitrary viewing distances.

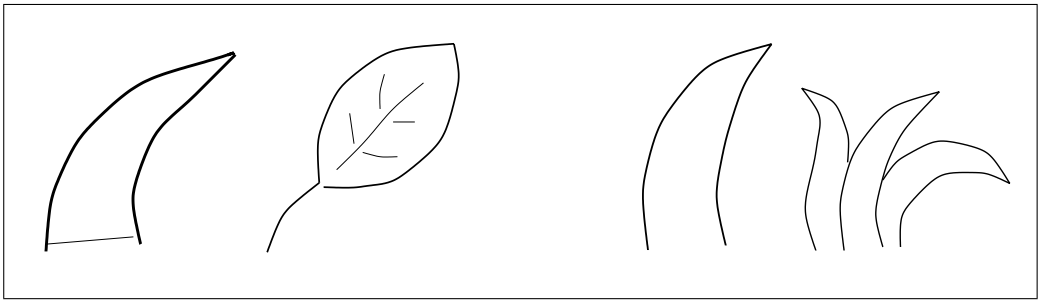

Figure 2.2: Left: sample graphtals composed of several primitives. Right: A 'tuft' rendered at two levels of detail [Mark00].

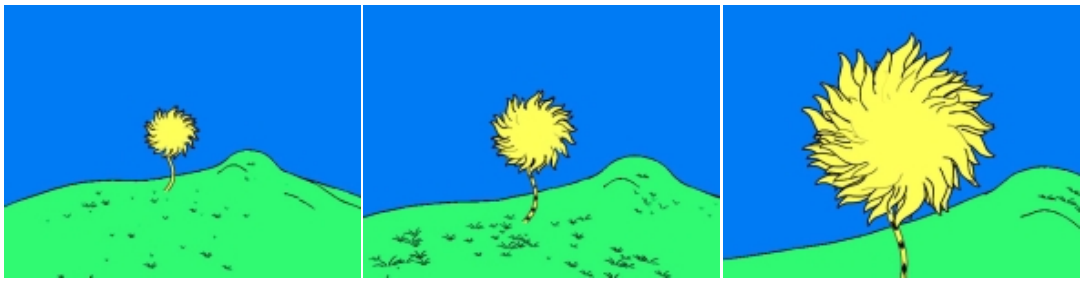

Figure 2.3: Groups of graphtals at different levels of detail [Mark00].

This problem was addressed in the work of Markosian et al. [Mark00], where particles are used to define the position of small graphical objects, denominated *graphtals* or *geographtals*. Graphtals are used to represent cartoon-like illustrations of a landscape. Let us assume one of the elements of this landscape is a grass patch. A grass patch can be clearly appreciated in a scene at a certain distance, but if the user moves farther from the point where the grass is placed, the representation for the grass needs to change, since it occupies a decreasingly small region of the entire scene. This is done by creating several representations, or levels of detail (LODs), for the grass patch and other objects in the scene. Figure 2.2 shows two different graphtals on the left, while a graphtal at two levels of detail is shown on the right. Using this approach, a complete scene can be described at several levels of detail, as is shown in Figure 2.3. Markosian et al. included a scaling attribute for each graphtal which is used in combination with the change in viewing distance to determine the level-of-detail to be chosen.

In most cases, rendering primitives in NPR can not be scaled up (or zoomed in) beyond a certain screen-space threshold. This is because drawing primitives are not meant to be seen from too close, otherwise its lack of detail becomes apparent, and this has a negative impact on the esthetic perception of the object (imagine for example, the Graphtals in Figure 2.2 seen at a close distance). For this reason, Kaplan et al. [Kaplo0] presented a particle system where the rendering primitives scale up to a certain maximum size in the image space. To fill the area not covered by existing primitives, new rendering primitives appear on the surface of the model as the viewer zooms into the model. In their implementation, particles are randomly distributed on the surface of the model, and are given specific radius values which are used in conjunction with the scaling and projection values. The number of particles is controlled interactively by the user during an interactive session, and newly added particles are given lower relevance values than existing ones, so that they are rendered only after particles on higher levels are rendered. In addition, they suggest changing the shape of the rendered primitives to achieve different effects (see Figure 2.4). The system of Kaplan has the advantage that it addresses the problem of scaling by automattically controlling the amount of visible primitives. However, generating additional primitives is not automatically done, and the distribution of primitives on the surface of

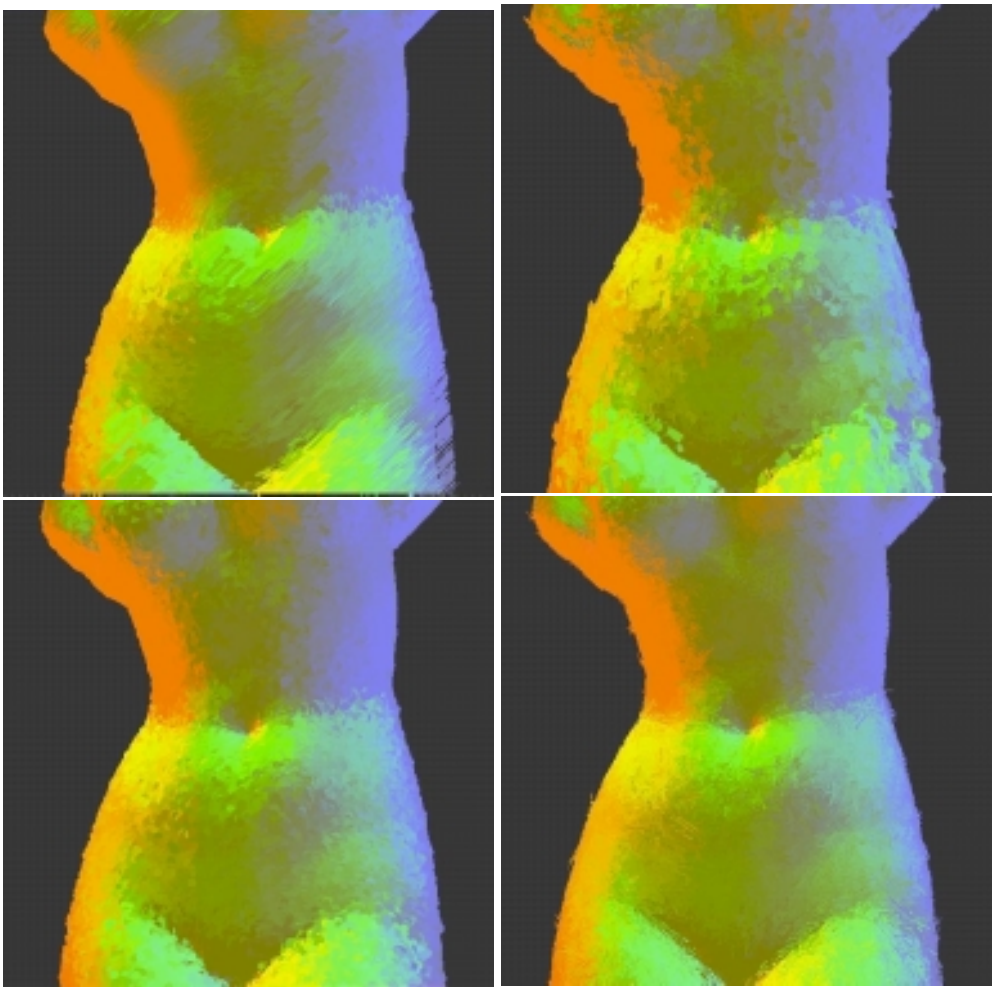

Figure 2.4: Effects obtained by changing primitive shape, by Kaplan et al. [Kapl00].

the model is random, while some drawing styles, like stippling, require having regular spacing between stipple dots.

2.2 Textures

Texture mapping has been widely used in 3D graphics to provide more detail to the surface of models, often to increase the look of realism of a model. An additional function of a texture is to replace model geometry with images. The result is a model that has a detailed look, but at the same time has a low polygon count and can thus be rendered rapidly. A texture is an image which is mapped to the surface of a model. Textures were originally meant as a replacement of complicated geometry that make a model look more realistic, but in NPR, textures contain strokes or drawing primitives which

are rendered on the surface of a model. Using textures to emulate non-photorealistic styles is part of the seminal work in this area [Wink94]. An advantage of using textures is that since textures are fixed to the model surface, they satisfy the frame-coherence requirement for NPR styles. In addition, textures can be rendered rapidly, since there is special hardware for rendering textures in current graphics cards.

There is a large set of techniques which rely on texture-mapping hardware to produce NPR renditions in styles like charcoal rendering and pen-and-ink illustration. Among the first publications that describe the use of textures for NPR is the one by Winkenbach and Salesin [Wink94], where the use of an array of prioritized stroke textures was presented. The idea here is that a texture can be represented at several levels of detail, and it is possible to create a hierarchy of textures to represent rendering styles related to hatching and cross-hatching. The original work of Winkenbach and Salesin proposed the use of several textures at different levels to adapt to changes in the viewing angle. The different levels of the textures are stored in *mip-maps* which contain the texture at different levels of detail. Nowadays, texture mapping hardware allows the combination of several textures in real-time. This has found use in the work of Praun et al. [Prau01] who propose the use of *tonal-art-maps* to re-create NPR hatching styles. *Tonal-art-maps* solve the question of scaling and shading by using a two dimensional array of textures instead of a one-dimensional array as the one proposed by Winkenbach and Salesin (see Figure 2.5).

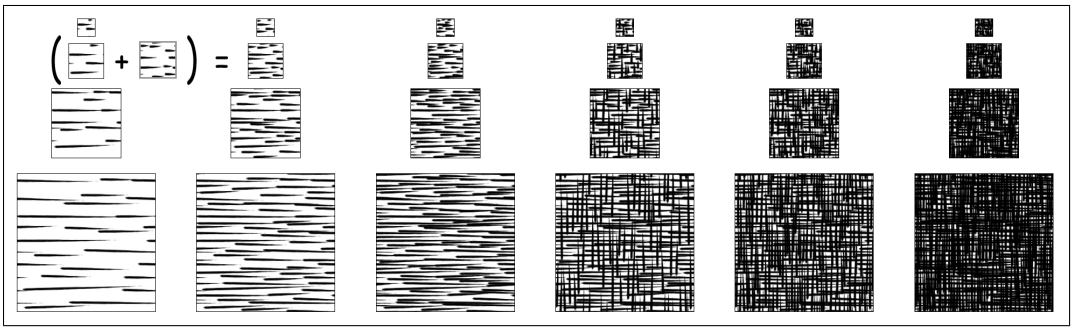

Figure 2.5: The *Tonal Art Maps* by Praun et al. [Prau01] ensure frame-coherence at the stroke level and take into account changes in shading and scaling.

This separation of scaling and shading gives freedom for the creator of the texture to define the aspect that a model should have when changes in scale, shading, or a combination of them, occur. The strokes in the tonal-art-maps of Figure 2.5 are placed such that a stroke is present in all the textures on the right of it and in all the textures below it. This enforces frame-coherence at the particle level, because the strokes will appear and are removed in a sequential way, according to shading and scale. Figure 2.6 illustrates several models rendered using Tonal-art-maps.

Having said that, it might look like textures could also be used for stippling, since they are also constituted in a hierarchical structure, and are frame-coherent. In fact,

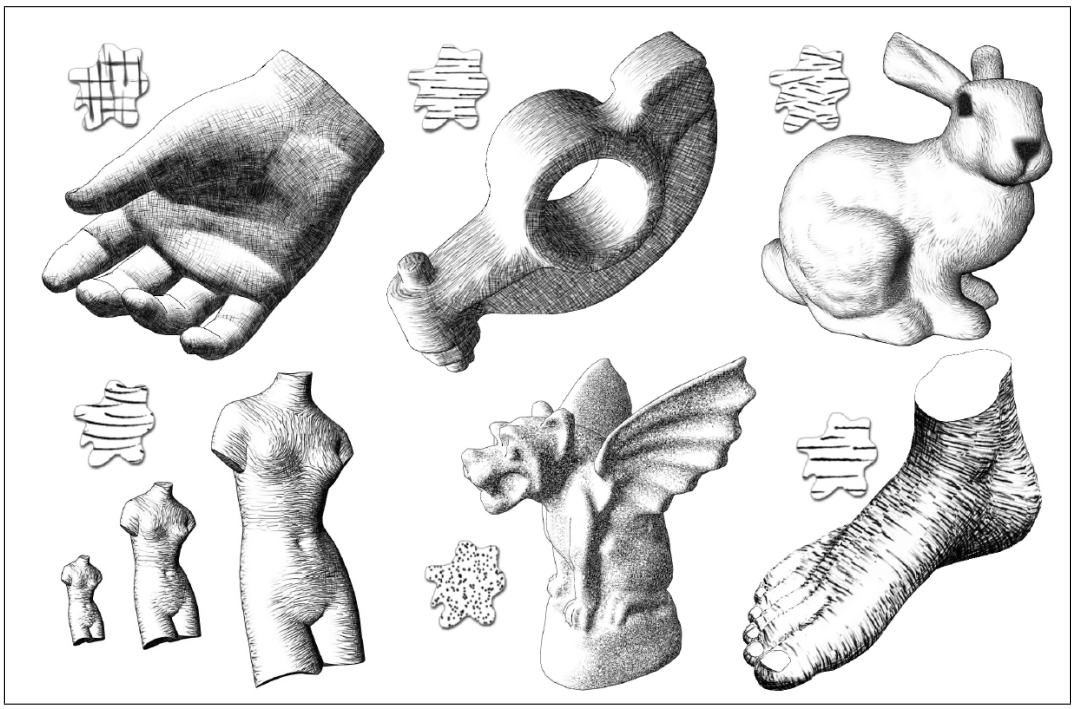

Figure 2.6: Examples of models drawn in several NPR styles using *Tonal Art Maps* [Prau01].

some authors have presented proposals to emulate the stippling style using textures [Prau01, Wink94], however the resulting images have not yet achieved anywhere near the level of quality that images in the traditional stippling style have. An example of a texture that aims at emulating the stippling style is shown at the lower bottom (center) of Figure 2.6. By inspecting this image we can notice that the overall aspect of the model is more that of a granite block, and the spacing among dots is rather random. A big problem with texturing is that is is hard to create and mix high resolution textures at different levels of detail. This problem likely arises from the way textures are mapped to the surface of 3D models.

Textures are normally subject to several hardware-based filtering processes during which they lose resolution. If the texture's resolution is high, the effects of these transformations are not noticeable, but if the model is zoomed-in, the texture will at some point lose resolution. The column on the left of Figure 2.7 illustrates how textures lose detail when seen at a close distance. In the case of stipples, however, we cannot afford to have large dots when the object is seen at a close distance, because the dots would become circles.

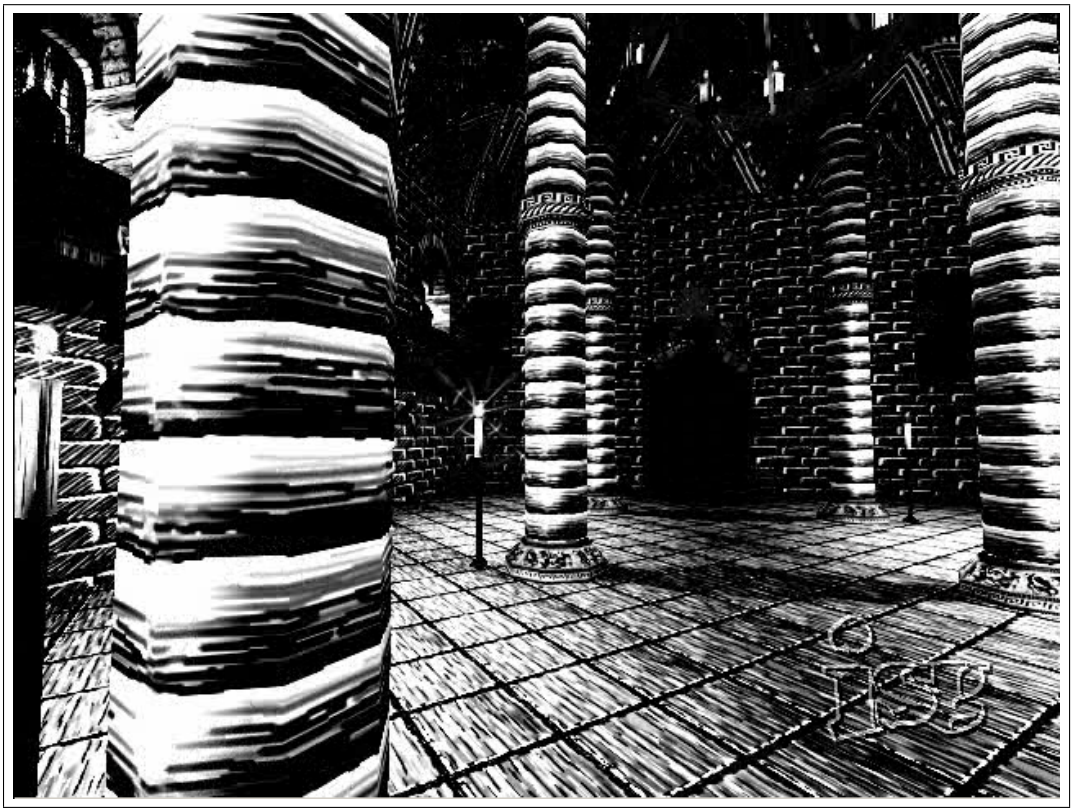

Figure 2.7: Non-photorealistic scene based on stroke textures by Freudenberg et al. [Freu02].

2.3 Silhouettes

Silhouettes are an essential part of NPR renditions. The reason for this is that silhouettes enhance the shape of the model, separating it from the background. There are a number of techniques which deal with the generation of silhouettes using 3D geometry. In the work of Isenberg et al. [Iseno3], a guide for NPR developers is presented which describes a number of techniques available to produce silhouettes.

Most silhouettes are generated as a set of lines of constant width which delimit the boundaries of a 3D model. There are also stylized silhouettes, which are rendered as paint strokes and can be used to enhance the artistic look of non-photorealistic renditions. The work of Kalnins et al. [Kalno2] integrates several real-time rendering techniques in a complete NPR painting system which enables artists to produce NPR styles interactively. In Figure 2.8 we show a set of cups obtained by having artists anotate on the same 3D cup model. Notice that all cups have stylized silhouettes, and the variety of stylies which can be obtained by using different colors and styles.

Figure 2.8: Cups obtained by the interactive painting system Wysiwyg NPR [Kaln02].

2.4 Volume and Point-Based Rendering

Point-based rendering has emerged recently as a feasible rendering technique which can efficiently deal with large complex models. In principle, point based rendering works under the premise that many existing models contain large amounts of points and that it is possible to render the model as a cloud of points rather than as a mesh of polygons. For this reason, most sample models for point based rendering are obtained from 3D scanned objects, since it is common that large point clouds are generated when using this technique.

The work of Pfister et al. [Pfis00] and Zwicker et al. [Zwic01] concentrated in producing photorealistic representations of objects using points and surfels as rendering primitives. In addition, point-based rendering is scalable, can be performed in real-time and can be combined with polygonal rendering. Dachsbacher et al. [Dach03] presented Sequential Point trees, an approach which makes use of a point tree hierarchy which makes it possible to render several objects with hundreds of thousands of points at interactive rates, by means of the reordering of a point hierarchy in a sequential array. Rusinkiewicz and Levoy presented QSplats, a technique which permits rendering of scanned model containing millions of points at interactive rates [Rusi00].

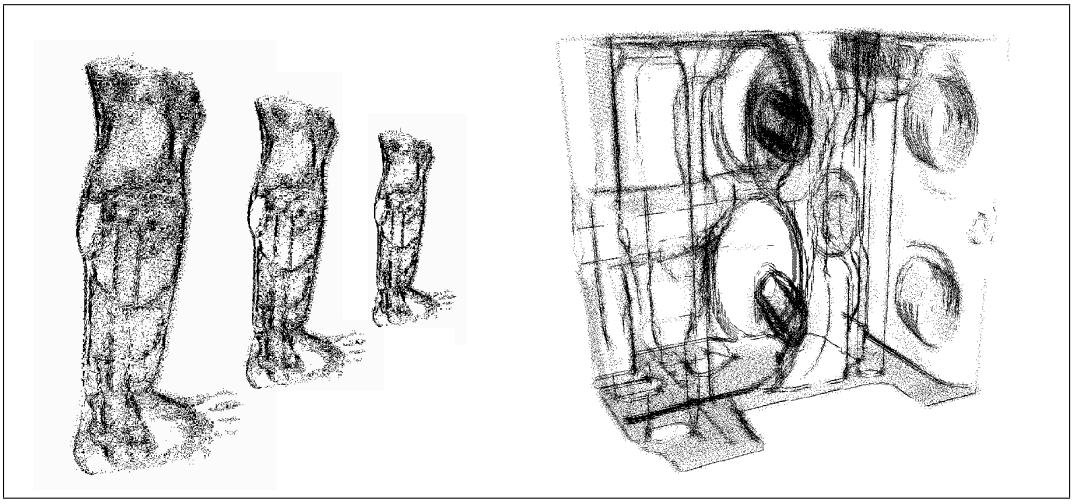

Figure 2.9: Examples of volume rendering using Stippling Techniques [Lu02].

Little research has been done to combine point-based rendering and non-photorealistic rendering. However, this might change soon. Lu et al. [Lu02] have recently presented a point-based rendering system to obtain stippled renditions of volumetric data (the information about an object is represented by a three-dimensional array of data). In this approach, a point hierarchy is produced using a spatial partition scheme (an octree). The approach consists in controlling the size of the stipples according to their level at the octree and information computed from the volumetric data. Stipples are rendered in 3D space, and the system can control the stipple density according to the viewing parameters, when the user zooms in, more stipples are generated, and when the user zooms out fewer stipples appear, in this way, their renditions are scalable and frame-coherent (see Figure 2.9, left). In addition, their system takes advantage of hardware acceleration to render models at interactive rates. However, the fact that there is no underlying polygonal mesh makes it difficult to discern between different layers of the input model, because these layers overlap in the screen plane (see Figure 2.9, right). Another system for the visualization of volumetric data is presented by Lum and Kwan-Liu [Lum02], who illustrate other non-photorealistic styles (see Figure 2.10), like Gooch and Gooch warm and cold-color schemes [Gooc98], and hybrid scenes with photorealistic and non-photorealistic styles which yield impressive results.

There is a strong relationship between point-based rendering and non-photorealistic rendering, which consists in that point-based rendering systems can be modified to produce point-based stippled renditions, as discussed in the last Chapter of this thesis. However, stippling is not the only rendering style which could be applied to point-based systems, since other primitive shapes could be used, as shown by the work of Kaplan et al. (Figure 2.4).

Figure 2.10: Non-photorealistic volume rendering by Lum and Kwan-Liu. On the left, the skin surface is made visible using gradient based feature enhancement. On the right, skin and flesh are rendered in a more photorealistic style, while the bones are rendered using non-photorealistic techniques [Lum02].

2.5 Particle Distributions in the Image Plane

An alternative approach to texture mapping and to particle systems is the possibility of enforcing frame coherence on the image plane. Secord et al. [Seco02b] have explored the possibility of distributing particles in a frame-coherent way on the image plane. Under this approach, a 3D model is rendered as a grey-scale image, and after each frame is rendered, a particle distribution tries to enforce frame coherence between subsequent frames. A similar approach is presented by Haga et al. [Haga01], where a technique that also performs image-based coherence is presented. In this approach, a stroke center which determines the location of a stroke is defined. This center is updated from frame to frame according to the movement of the object, giving the frame coherence. In this system, strokes fade in and out of the image smoothly, However a problem with frame-coherence appears because the stroke orientation is a function of the model surface normal with respect to the screen plane, which leads to strong variation of stroke orientation as the model moves. An additional problem that reduces frame coherence is that the life of the particles is short, and new particles do not emerge at the location where previous particles were located, which introduces noise in the animations.

The main advantage of using image-based approaches to obtain frame-coherence is that tone is guaranteed to be kept, which is not the case for particle-based systems, as

it will be later discussed. Another advantage is that it is not necessary to preprocess the 3D models for the purpose of the particle distribution. A third advantage is that the distribution of particles occurs in the 2D plane and is fast since the number of strokes/particles is defined by the size of the image and not by the complexity of the underlying model.

A disadvantage of carrying out the particle distribution in the image plane is that even when frame-coherence in the image plane is enforced, the particles do not necessarily move correspondingly to the surface of the model. This can be understood as having particles 'float' on the surface of the model. This effect is interesting per se, but is somewhat counterintuitive, because it does not quite correspond to what is expected to occur in the real world. Video 2-*Fast Primitive Distribution* shows these effects, and we can observe that the rendered strokes displace as if they were floating on the surface of the model. In addition, the orientation of the strokes is not constant with respect to the surface of the model, which also undermines the coherence between the stroke orientation and the model surface. As a counterexample, texture-based NPR systems enforce frame coherence also with respect to the surface of the model, because the orientation of the strokes is fixed with respect to the model's surface.

The question that image-based particle distribution systems makes us arise is whether it is possible to obtain regular point distributions in 2D by distributing points regularly on the surface of a 3D model. This is a question that we are confident will be answered throughout this work.

2.6 Contribution of Point Hierarchies to NPR

In this thesis we depart from the concepts present in the literature previously mentioned to create a point hierarchy which is appropriate for stippling and other NPR styles.

Taking into account the three main aspects that need to be taken into account when choosing a rendering technique for NPAR, namley frame-coherence at the particle level, scalability and the possibility of applying deformations to 3D models, we have decided to develop a hierarchical particle system where each particle is a point in 3D space. A particle system inherently provides full frame-to-frame coherence, and in our case, is used both for controlling the stipple density (according to the desired shading tone) and for controlling the number of particles used for rendering (as the model is scaled, or the viewing distance or screen projection of the model changes). Furthermore, by slightly changing the coordinate system of the points in the hierarchy, we can use it for animated and deformable models. In addition, to enforce a regular point

distribution, as it is done in image-based stippling, the point hierarchy creation approaches presented throughout this work are constructed in a way that enforce appropriate spacing among the point particles. This requirement of the spatial distribution of particles is unique to the purposes of stippling. However, once created, the particles could be of use for other non-photorealistic rendering styles, because the particle system is one component of the rendering system, and it could be used in conjunction with other rendering primitives as well, in the way of Kaplan et al [Kaplo0].

Our choice of using the point hierarchies instead of textures let us dynamically control the number of particles we want to render. We have full control of the stipple dot size at every frame, and can regulate the maximum size of a dot, even as the model is seen from a short viewing distance.

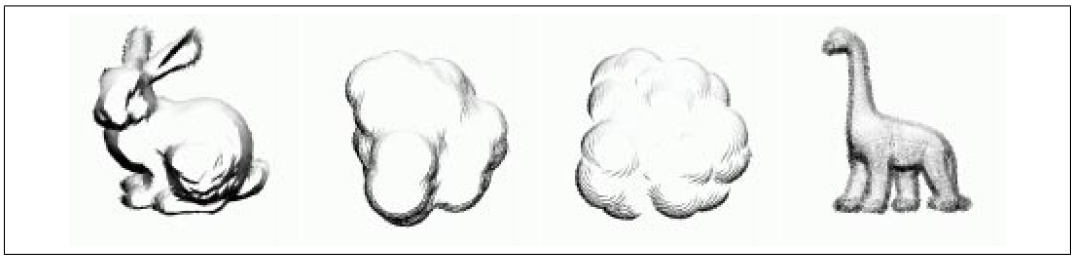

Figure 2.11: Examples of the rendering styles obtained with the particle system of Cornish et al. [Corn01].

In this thesis, we present two new and different approaches to generate particle systems for NPR, where the spatial distribution of the particles is taken into account during the creation of the point hierarchy. The first approach is inspired in the work done by Cornish et. al. [Corn01], who presented a system for interactive rendering of complex models based on the mesh simplification system of Luebke and Erikson[LueEri:97]. In Cornish's system, strokes to be rendered were derived through dynamic mesh simplification. When a model is seen from far away, a screen-space-based threshold determines if an edge is to be visible or not. The edges are used as the locations of stroke lines which result in the images presented in Figure 2.11. Our work differs from that of Cornish in two main aspects. First, we can process both coarse and highly tessellated models. With our system, point hierarchies of any size are created through mesh simplification and subdivision, regardless of the degree of tessellation of the input model, where in the case of Cornish, the size of the point hierarchy is determined by the number of polygons of the input model. This let us produce stippled renditions in situations where the original amount of vertices in the model is not sufficient to fill the canvas with the appropriate number of stipples (see Chapter 4). The second difference is that we flatten the point hierarchy in a sequential point array, which permits fast hardware processing for performing real-time stippling with a large number of dots using graphics accelerators, as described in Chapter 7.

The second approach that we use to create well-spaced point hierarchies is also inspired from the literature on level of detail, but in this case is related to the creation of levels of detail by mesh re-tiling and point relaxation. In 1991-1992, Turk presented a system to create regularly distributed point sets on the surface of a model to generate synthetic textures [Turk91] and to perform mesh simplification [Turk92]. Inspired by this approach we present a way to distribute points on the surface of a 3D graph by token distribution. Figure 2.12 shows a stippled rendition produced using our rendering system. Note that the image at the bottom is not a detail taken from the image at the top, but the rendition our system produces as the viewer zooms at the model, filling in the shaded areas with additional stipples. The token relaxation approach emphasizes appropriate spacing among stipple dots at each level of the point hierarchy. The main visual difference between the approach of point hierarchies created by mesh simplification and subdivision and those created by token relaxation appears at the lowest level of the point hierarchy (this is descibred in Section 5.7.3). Our token relaxation approach also illustrates how point hierarchies can be created with different approaches, mostly related to mesh simplification and mesh compression [Allio1], and show that additional approaches for creating point hierarchies are still open for research.

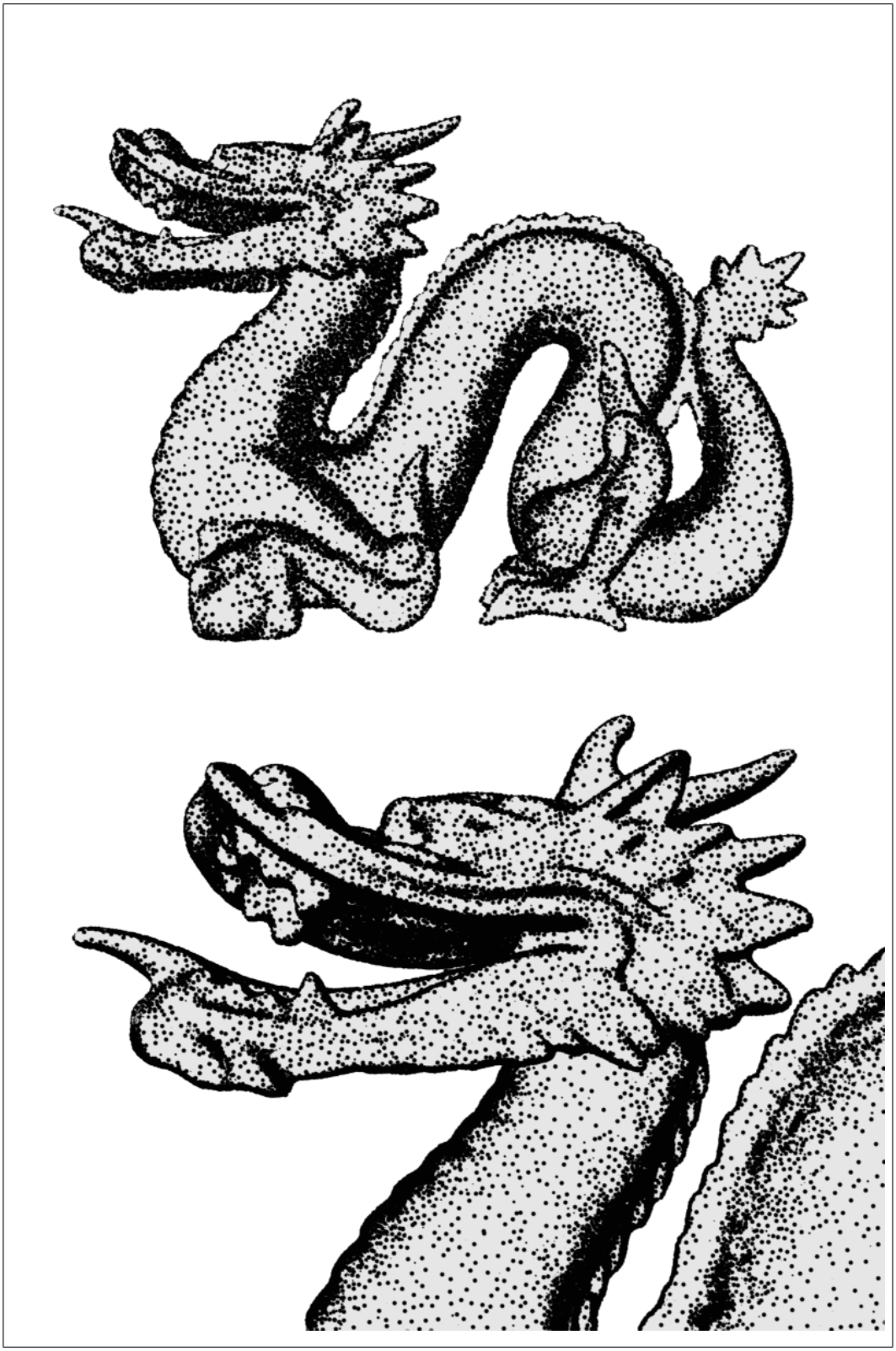

Figure 2.12: With our system, it is possible to produce adaptive stippled renditions for 3D models. As the user zooms at the dragon (top), more dots appear to maintain the tone and stippling style (bottom) (original sizes: 800x600).

3 A General Framework for a Particle-Based Non-Photorealistic Rendering System

In the first part of this Chapter we describe how to distribute points on the surface of a model in a way that the particle distribution adapts to changes in shading, scale, and viewing angle. In the second part of this Chapter, we describe how to construct a non-photorealistic renderer to produce a stippled rendition by taking as input a 3D model and a point hierarchy obtained from the 3D model.

3.1 Going from 2D to 3D with Adaptive Point Hierarchies

In this section we explore the relationship between stippling in the plane (2D stippling) and stippling in 3D as the model is moved and the viewing conditions are changed, including scaling, slope and lighting. This discussion will help us determine which functions can be used to approximate stippling using point distributions in 3D space.

When we have a stipple distribution on a plane, and we want to keep a constant tone on the surface of a model, the stipple density must somehow be regulated. In the following section, we consider how to regulate stipple density in relationship with scaling and shading. Later, we consider controlling stipple density for changes in the viewing direction.

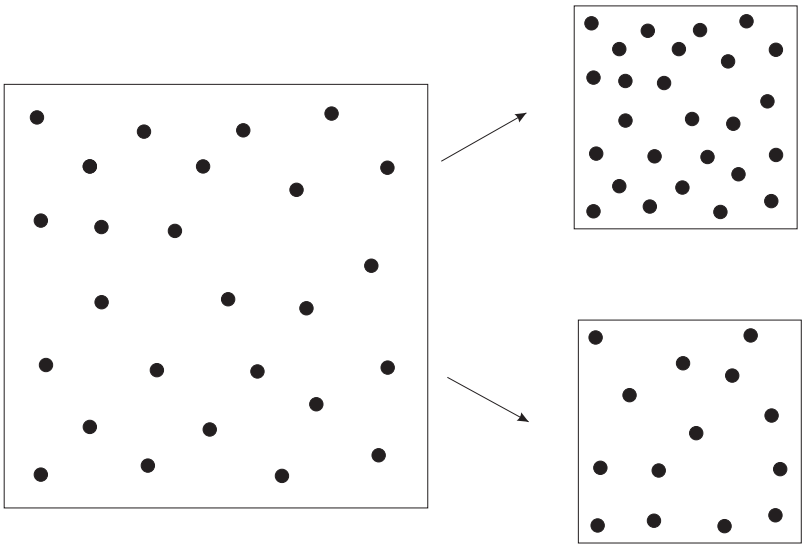

Figure 3.1: Scaling with points of fixed size. The images on the right are scaled-down versions of the image on the left. In the bottom, points have been removed from the image.

3.1.1 Adapting to Scale and Shading

If we scatter a number of stipple dots of fixed size on a square face in 3D and scale it, the overall darkness of the image will vary as an inverse of the scaling, when the model is small, points will be placed close together producing a dark tone and when the model is large, points are more spaced and the overall tone becomes lighter. This is shown in the image on the top right side of Figure 3.1. On the other hand, the image on the bottom right of the Figure shows how the initial tone available on the left side can be maintained by removing some stipples from the original image. To preserve tone independent of scaling, we want to control automatically the amount of stipples rendered on a certain region, while maintaining tone constant as the model is scaled up and down.

When using particles on the surface of a 3D model, it is important to keep a constant target tone by controlling the amount of stipples rendered. For this, we must define a function that let us compute which points should be discarded as a model changes in scale.

To define this function we make use of the following definitions:

1. **targetTone** is a variable which goes from zero to one, if targetTone is zero, then a totally white tone is indicated or desired, if targetTone is one, then a totally black tone is desired.

2. **PtSize** indicates the point's diameter in pixels. PtSize must be kept relatively small, since a large point size will produce a circle instead of a stipple. In general, we limit its size (MaximumPtSize) to a user-defined value which is typically around 3.0, but this value greatly depends on the size of the rendition relative to the resolution of the output device.

3. **CanvasArea** indicates the size of the canvas in pixels.

The actual tone in a stippled image is the average of points over a certain region. The following formula gives us the shading tone obtained by distributing points of a given size on a canvas:

$$\text{actualTone} = \text{PtSize} * \frac{\text{numberOfPoints}}{\text{CanvasArea}}$$

Let us assume for a moment that we have scattered a limited set of regularly distributed points on the canvas, and we need to decide (on a point-by-point basis) which points in this set should be drawn, as a function of the targetTone at each point's location.

First, we need to determine the region that a point covers with respect to the CanvasArea and the other points:

$$\text{RegionOfInfluence} = \frac{\text{CanvasArea}}{\text{numberOfPoints}}$$

Then, we can determine what is the tone given by a full point in comparison with the area covered by its region of influence:

$$\text{GivenTone} = \frac{\text{PtSize}}{\text{RegionOfInfluence}}$$

We can estimate for an individual point whether a point can be lit or not, using the following algorithm by comparing the target tone value with the tone value that a full pixel would give:

```
if     (targetTone >  GivenTone )
then   outputPtSize = maximum point size
else   outputPtSize = 0
```

Algorithm 3.1: Discrete point selection for rendering.

The function previously described is a discrete function to determine the point size. The result of using such a function during interactive rendering is that points pop in and out of the image, which is an artifact that introduces noise.

To produce smooth transitions of point size we can modify the previous function and set it as a function of the proportion of the difference between the target tone with respect to the tone given by a full point:

```
if (targetTone/GivenTone > smoothSlope)
then   outputPtSize = 1.0
else if (targetTone/GivenTone < 1.0)
then   outputPtSize = 0.0
else   outputPtSize =
              (targetTone-GivenTone)/GivenTone /(smoothSlope-1)
simplifying:
              ((targetTone/GivenTone)-1)/(smoothSlope-1)
outputPtSize = outputPtSize * MaximumPtSize
```

Algorithm 3.2: Smooth point rendering doing point size interpolation.

In this algorithm, smoothSlope (a user parameter restricted to values higher than one) indicates the proportion in which targetTone must exceed the value of givenTone such that the point reaches its maximum size. Ideally, the transition should be fast, so smoothSlope should be small, but if the transition is too fast, the points will suddenly pop in or out of the image.

While the target tone is not guaranteed to be correct for values other than zero and the maximum point size, we obtain in exchange a smooth transition for enabling and disabling points, which is more important than obtaining an exact point transition.

This algorithm takes into account scaling, as long as the value in CanvasArea is the projection of the square face on the output image or output window. If CanvasArea is considered a dynamic variable under this assumption, the algorithm works appropriately during scaling.

Up to this point, we have worked on the assumption that we have a constant set of input points. This constant set of input points are appropriate to render only a small spectrum of targetTones. If the canvas area is small, all the points in the set will exceed the target tone, and none will be rendered. On the other side, if the canvas area is large, all the points in the set will be below the target tone and will be set to their maximum size.

It is clear now that we need additional sets of points to cover different shading tones. For example, a set of two points of size 1 on a canvas of area 10 is adequate to provide

shading of 0.20 black/white, but if the canvas is of area 100, we will need a set of approximately 20 points of the same size to represent the same shading tone as before.

For this reason, we need to distribute several groups of points on the surface of the model, in such a way that a small group of points provides appropriate shading for a small canvas, and a large group of points provides appropriate shading for a large canvas. Every group of points should be evenly distributed on the surface of the model. In addition, to ensure frame-to-frame-coherence, we must make sure that the groups grow incrementally. This means that the points in the smallest groups should be included in the following group. Otherwise, we will have an excess of points i.e., darker tones as expected.

In this work, we call this arrangement of points a point hierarchy. The point hierarchy contains points which correspond to different scales or shading tones. It contains different groups of points distributed over the same surface to successfully represent different levels of targetTone. In our case, we can differentiate the group of points by grouping them on the region of influence they cover. Points which cover large regions of the canvas are at the highest levels of the hierarchy, and additional points, which cover smaller regions of the canvas are further down the hierarchy.

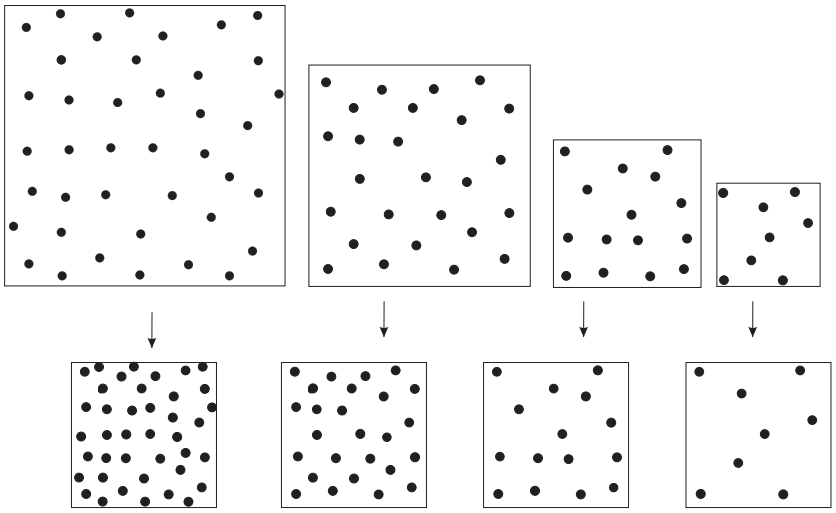

Figure 3.2: The same point hierarchy is used to accommodate changes in scale while preserving the target tone (upper row), and changes in shading by reducing the point density (lower row).

Using the algorithm previously described, we have a way to distribute points on a canvas where the targetTone and the scale can be arbitrarily modified using the same point hierarchy. Figure 3.2 illustrates this, in the upper row, we scale down the square face, and remove some points to keep the initial tone. In the lower row, we have a canvas of constant size, and remove the same points to achieve lighter tones.

3.1.2 Adapting to Slope

An additional consideration now arises with respect to the slope, or angle of inclination of the points.

When a square polygon covered with stipple dots is rotated around an axis perpendicular to the face normal, linear patterns arise in the distribution of the dots, even if the point density on the face is regular (see Figure 3.3).

If we want to avoid the presence of these patterns, the point density should adapt to the axis and the angle of rotation. For example, if the axis of rotation is on the X-axis, the points should probably vanish in such a way that they avoid the formation of the vertical linear patterns on the projected images shown in Figure 3.3.

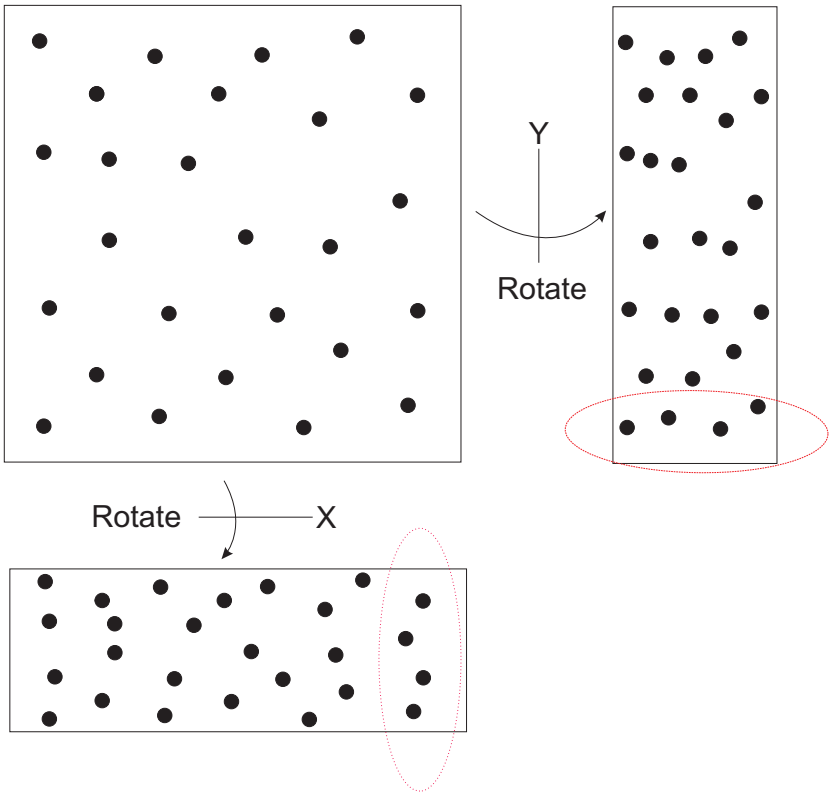

Figure 3.3: Effect of rotations on the point distribution in the image plane. The image on the top left illustrates a face covered with a random point distribution with regular spacing. Images on the top right and bottom left show the linear patterns which result from rotating the face around the X- and Y-axis.

In general, we need to make point distributions which are tailored to the rotations depending on the axis of rotation, and apply them only in the case of rotation on this axis. But the question then becomes, what attributes must we give to the points so

that they behave appropriately under each combination of scaling and rotation. The situation becomes more complicated when we consider that rotations around X and Y are only two standard rotation types, but in fact, there is an infinite set of rotations which can be used to rotate the input polygon. A possible solution would be to have a dynamic evaluation procedure where the stipples are removed using a mathematical expression or a function which determines for a combination of shading tone, scale and viewing angle, the size of the stipple. In the frame of this work, we explored several functions that could be used to do this. However, a general solution was not found.

Another option could be to assign attributes to each point in the hierarchy to consider each possible rotation. This, however, is not an efficient solution because the set of attributes per point would be large.

We propose the use of a linear solution which is applied on all points which considers only the deviation of the face with respect to the standard orientation (which is front-facing):

$$\text{NewRegionOfInfluence} = \text{RegionOfInfluence} * \text{FaceNormal.dot.ViewingVector}$$

In this case, every point will adapt to some extent to the changes in rotation, eliminating the points as the face is rotated. However, this will not prevent the appearance of some linear patterns when faces are quite sloped with respect to the normal position, leading to the effect mentioned in Figure 3.3.

3.2 A Particle-Based NPR Renderer

There are two parts to a particle-based NPR renderer, the first is the creation of a point hierarchy, and the second is the rendering stage, where the point hierarchy is used to produce stippled renditions. These two stages are divided for the sake of flexibility and modularity: since particle generation is a time-consuming process and point rendering can be done in real-time, both processes are performed separately; in addition, the point hierarchy can be created using different approaches, and can be described in a standard file format which is used later by the rendering system.

The point hierarchy consists of points in 3D space scattered over the surface of the model, each containing its hierarchy attributes. During rendering, the input model is rendered using a plain color and the z-buffer. The model is rendered slightly behind the particles (using constant offset values), so that the particles which are on the visible surface of the model are not occluded by the surfaces of the model where they lie.

It is a task of the renderer to evaluate the point hierarchy and to determine which particles are visible and what their size should be, according to the lighting conditions,

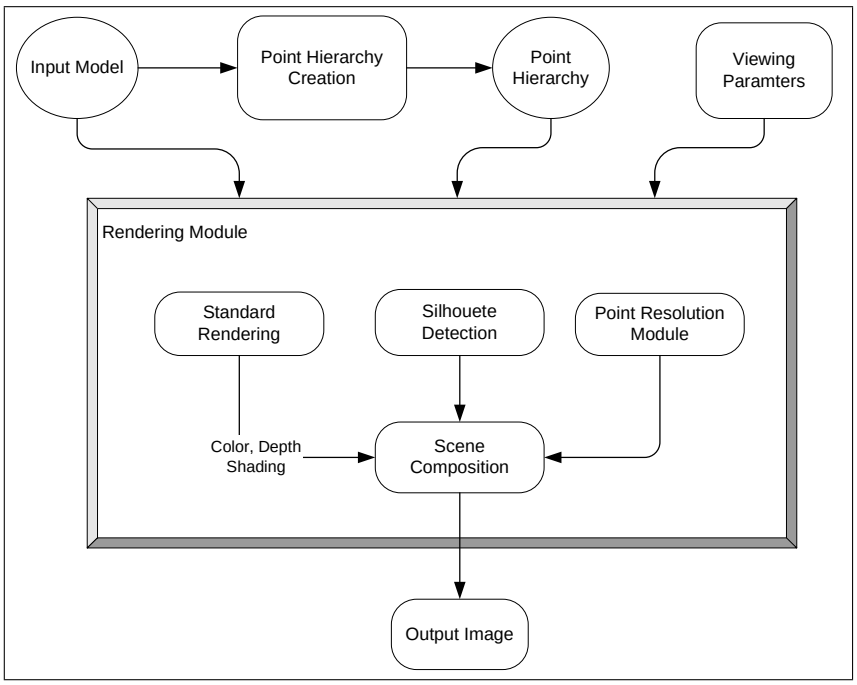

Figure 3.4: Rendering pipeline for producing renditions with a point hierarchy.

viewpoint, and screen-space projection. For this, each point in the hierarchy must be assigned a "relevance" value (also called radius) which is used by the renderer. This value is normally a floating-point value (for real-time rendering), but it can also be computed from a collection of points in the neighborhood of the particle set (which is useful for animated stippling).

For animated models and morphing, the particles should be defined relative to the surface of the input polygons where they lie on. In this case, the rendering system is able to transform the coordinates of the particles according to the change in the coordinates of the input polygon.

To enhance the profile of the model, a silhouette rendering algorithm is also included which renders edges that connect front and back facing polygons of the input model. This is the basic algorithm, however, research in NPR also has focused in producing stylized silhouettes effects, and there is a number of algorithms for silhouette rendering. Isenberg et al. have produced a compilation of such algorithms which can be of interest for the reader [Isen03].

In Figure 3.4 the general rendering pipeline for a particle hierarchy is shown. Mixed rendering styles are produced by using the particle system to stipple certain parts of the model, while other parts are rendered using other rendering techniques.

The particle renderer takes as input the point hierarchy and the rendering parameters. The rendering parameters (see Figure 3.5) include information about the model itself,

the location of the particles on the model, the rendering parameters such as light direction, viewing distance, and viewport size, to determine the size of each individual particle. As indicated, a relevance value is an important parameter used to decide the size of a point with respect to other points. Figure 3.5 illustrates the point resolution module, which is part of the whole rendering pipeline.

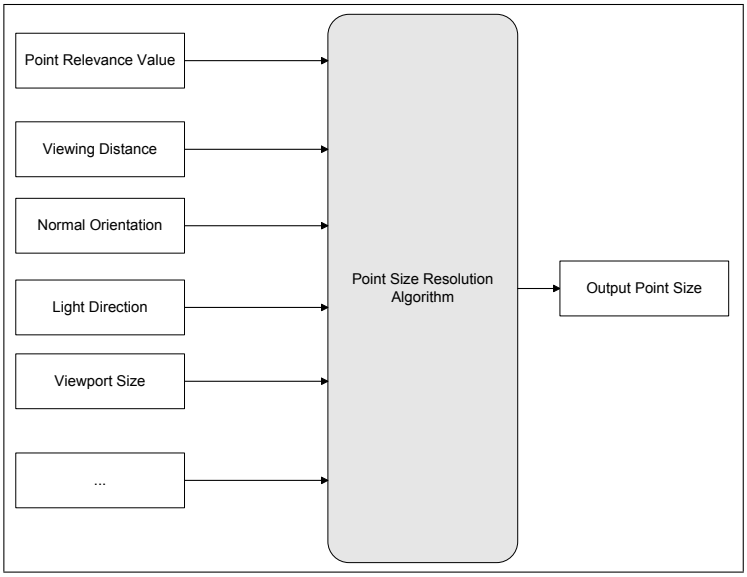

Figure 3.5: Particle rendering module and its inputs.

3.3 Generation of Point Hierarchies

In this section we describe how polygonal mesh simplification algorithms, which deal with level-of-detail generation and mesh compression, are suitable for the production of point hierarchies. Mesh simplification techniques are meant to save rendering time by sending the minimum amount of polygons to the graphics engine, while preserving the visual aspect of a model. A polygonal model's Level-of-Detail describes the amount of geometric complexity that a model has, which is commonly measured by the number of polygons the model has at a given level.

Level-of-Detail representations are produced in most cases by analyzing the geometry of the input model and by simplifying it using a wide range of simplification operators. In fact, there is a large number of simplification techniques, which vary considerably among each other For example, one technique may decide to collapse the longest edges of a polygonal mesh as its basic simplification operator, while other technique may decide to cluster a number of vertices using regularly spaced voxels as a simplification step.

In several cases, the result of this simplification is the production of a mesh hierarchy, which contains all the different levels of detail of a model, and a mapping, used at the rendering stage, that defines how to produce a transition between different LODs.

It is important to have smooth transitions between levels of detail, because this reduces the presence of *popping artifacts*, which refers to noticeable differences between levels of detail that appear when switching between levels of detail during interaction.

There are several aspects where the mesh simplification theory coincides with the production of point hierarchies:

1. There is a parallel between a LOD sequence and a point hierarchy: while a low LOD is a representation of a model with few polygons, a higher level in the point hierarchy requires just a few number of particles to be spread over the surface of a model. High levels of detail correspond to the lower levels of a point hierarchy and are obtained by visiting the lower levels of the mesh simplification tree.
2. Another aspect is that both areas are bound by the requirement of coherence between different levels of the hierarchy. While the levels of detail of a polygonal mesh have to coincide between each other to enforce smooth transitions, there must be a degree of coherence between the levels of the particle hierarchy, so that frame-coherence at the particle level is kept while scaling the point hierarchy, or while varying the shading tone of a surface.
3. A third aspect is the preservation of features versus the distribution of points on the surface of a model. Most LOD techniques emphasize the production of levels of detail where the lower detail representations are meant to keep features which are present at higher levels of the hierarchy to the highest degree possible. Nevertheless, other LOD techniques are meant to reduce uniformly the polygon count and keep a regularly tessellated mesh at different levels-of-detail. Regularly tessellated meshes are preferred in graphics because they reduce the risk of having floating-point inaccuracies when computing other geometric attributes of the mesh (like the normals of the faces, for example). The parallel between regularly tessellated meshes and point hierarchies is present when a mapping from the polygon mesh to the points of a point hierarchy is defined. This connection highly depends on the technique used to distribute points on the surface of the mesh.

In sum, mesh simplification techniques can be used as a ground to produce point hierarchies suitable for Non-Photorealistic Animation and Rendering. The large amount of available techniques also provides a number of possibilities to produce point hierarchies that cover the requisites of frame-coherence, scalability, and regular point distribution. In the following chapters we discuss in detail two techniques inspired by literature on level-of-detail and mesh simplification which are implemented within this context to produce point hierarchies.

4 On the Generation of Point Hierarchies Using Mesh Simplification and Subdivision

In *scientific illustration* Hodges [Hodg88] points out that artists create stippled drawings by first placing some groups of dots in a region of interest and then filling in until the desired tone is achieved. We follow the strategy proposed by Hodges by creating a hierarchy of vertices in 3D space which represent point locations on the surface of the model. The point hierarchy here presented is defined in such a way that spacing of points is taken into account when adding and removing points: new points (which are inserted lower in the particle hierarchy) are placed at locations roughly in the middle of existing points. Alternatively, when points are removed from the surface of a model, points at the bottom of the hierarchy vanish first. In Figure 4.1 we illustrate how the concept of regular spacing of points is taken from the one dimensional case to the three dimensional case. On the left side of the image, we can observe how relevance values can be assigned to the points according to their hierarchy level. This distance is a key value used to determine whether a point should be rendered or not. On the right side of the image, it is possible to observe that if we consider the vertices of a polygonal mesh as potential point locations, a mesh simplification and subdivision approach can be used to produce a point hierarchy in 3D space. Points in the higher levels of the hierarchy are sparsely distributed, and new points are added between existing ones.

In some cases, the number of vertices in the input model is so high that we have to discard many of them to produce a light shading tone. To discard vertices from a highly tessellated model we perform mesh simplification. Mesh simplification approaches, such as *progressive meshes* by Hugues Hoppe [Hopp96] and *view-dependent polygonal simplification* by David Luebke [Lueb97], generate simplification hierarchies which are used to produced continuous levels of detail transitions by traversing a simplification tree created by applying a series of edge-collapse operations on the edges of an input

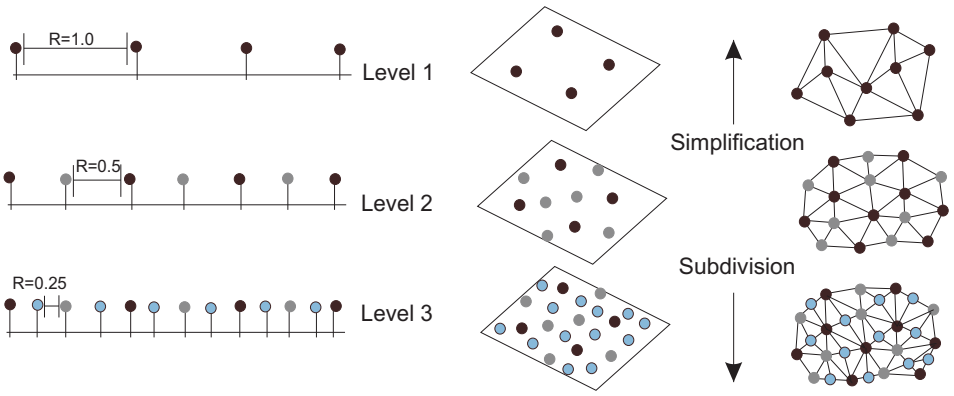

Figure 4.1: Point hierarchies for the one- (left), two- (middle) and three dimensional cases (right). Points at the lower levels of the hierarchy have smaller radius values, which determines their relevance in the hierarchy. For 3D models, a continuous level of detail is created using mesh simplification and subdivision.

mesh. The model is iteratively simplified from its most complex level-of-detail by using simplification operations until a few polygons in the model are left. A vertex hierarchy is the result of this sequence of operations, the lower levels of the tree represent the edges that were initially simplified, and the higher level represent subsequent edge simplifications that were done on top of previous simplification operations (see Figure 4.2).

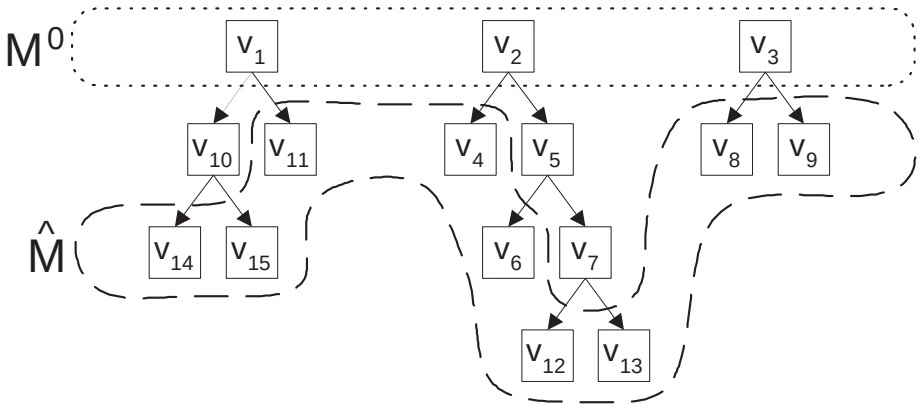

Figure 4.2: Vertex hierarchy created through simplification for progressive meshes. M^0 represents vertices at the lowest level of detail, $\hat{M}$ represents vertices of the original mesh. [Hopp96].

The idea of using a mesh simplification approach for NPR was presented by Cornish et al. [Corno1]. Cornish et al. takes the simplification system of Luebke, and applies it to an input model to create a hierarchy of edges. This hierarchy of edges produced by the

simplification stage is used for rendering in such a way that edges which are lower in the hierarchy, appear last and vanish first than edges which are higher in the hierarchy. In our approach, we use the edge-collapse operation defined by Hoppe in Progressive Meshes and create a vertex hierarchy, based on the sequence of refinement operations applied to the input model.

Depending on the viewing distance and the number of polygons in the input model, the number of vertices of the input model might not be enough to cover dark areas. To fill these areas, additional vertices are generated on the surface of the model by mesh subdivision. After each simplification and each refinement step, the resulting vertex is assigned a relevance value (alternatively, a list of neighbors) that we use to decide which points should be included in a particular rendition.

To ensure that the appropriate level of detail is obtained at most viewing ranges, mesh simplification and mesh subdivision are combined to provide seamless levels of detail regardless of the resolution of the input model (see Figure 4.1, right). Vertices at the top of the hierarchy are the initial group of dots spatially distant from each other; the vertices down the hierarchy fill-in the space between existing vertices, so that new points always come up to fill-in uncovered regions of the canvas until the desired tone is achieved. This initial approach for frame-coherent stippling was presented in *frame-coherent rendering* [Meru02b].

An additional stage introduced to improve point distribution is mesh randomization, which has the goal of reducing the presence of visible linear patterns embedded in the geometry of the input model. Linear patterns can also result during the process of mesh subdivision. To avoid these patterns, randomization is applied on vertices produced by mesh subdivision.

We have implemented a system that integrates these steps and generates a point hierarchy as follows:

1. Compute a connectivity graph to operate on the input polygonal mesh.
2. Apply a randomize phase on the vertices of the input mesh to reduce the presence of regular patterns in the point distribution (see Section 4.4).
3. Perform mesh simplification on the input mesh, creating a hierarchy for the vertices in the input mesh.
4. Perform mesh subdivision on the input mesh, up to a desired level of detail, or a desired point count, expanding the existing point hierarchy with the new vertices.

We describe each stage in detail in the following sections; the complete rendering system is described in [Meru03].

4.1 Setting up the Connectivity Graph

To generate the point hierarchy, a connectivity graph based on the input polygonal mesh is created. The graph data structure we use is a Leda [Mehloo] directed graph where every face is made up of three edges. The edges store the connectivity information of vertices in the input mesh. Each edge contains a source and a target vertex, according to the orientation of the faces (either clockwise or counterclockwise). Adjacent faces are related to each other by assigning to each edge connecting the same vertices a pointer to the edge owned by the adjacent face (the edge *reversal*). Figure 4.3 illustrates the graph structure corresponding to a sample patch of polygons.

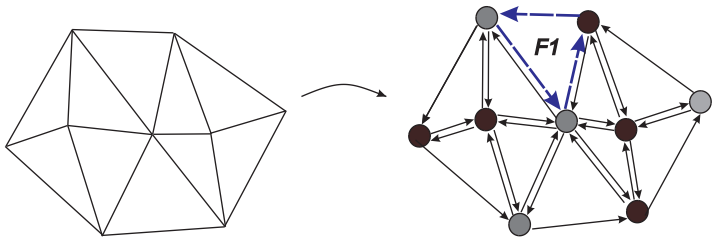

Figure 4.3: Conversion of a polygonal patch in a directed graph. Notice that each face has its own set of edges. As an example, the edges in face F1 are dashed. The direction of the edges depends on the ordering of vertices in the input models (counterclockwise in this case). Adjacent edges are *reversals* of each other.

The graph stores information about the vertices in the data structures **graph Node** and **VertexInfo**. Information about the edges is stored in **graph Edge** and **EdgeInfo**. Data specific to the faces is stored in **FaceInfo**. This information is used for mesh simplification, subdivision, randomization, and vertex projection. Table 4.1 contains a list of these structures with its most important components.

4.2 Mesh Simplification

In mesh simplification we create a vertex hierarchy by applying a series of edge collapse operations in the input mesh until the model is simplified to a few vertices. The operator that we use for mesh refinement is a variant of *edge collapse* (`ecol`) introduced by Hoppe [Hopp96] where one of the vertices is removed by displacing its connected edges to the other vertex (see Figure 4.4).

The following algorithm is used to simplify the model:

Graph Node
1. A pointer to VertexInfo
2. A list of incoming edges (edges directed towards the node)
3. A list of outgoing edges

Graph Edge
1. A pointer to EdgeInfo
2. A pointer to the edge's source node
3. A pointer to the edge's target node
4. A pointer to the reversal of the edge

VertexInfo
1. VertexID
2. VertexNormal
3. VertexCoordinates in space
4. FaceInfo pointer to a face in the input model
5. A list of its relevant neighbors
6. Relevance value (Point Radius)

EdgeInfo
1. A pointer to FaceInfo where the vertex lies
2. A pointer to the parent VertexInfo node in the point hierarchy
3. The edge's length in object space

FaceInfo
1. The face Normal
2. A list of the edges of the graph that make up the face

Table 4.1: Contents of the data structures that make up the connectivity graph used for mesh simplification, subdivision and randomization.

while not all edges have been simplified:
1. **find** the longest edge $e\mathrm{Longest}$ in the mesh
2. **select** one of the vertices of $e\mathrm{Longest}$ for removal
3. **save** a list of the nodes connected to the vertex that will be removed (the relevant neighbors)
4. **collapse** $e\mathrm{Longest}$ by removing the selected vertex using *ecol*

Algorithm 4.1: Simplification of the input model.

By performing mesh simplification we can take models of complex geometry and render few points on top of them by using the vertices at the top of the hierarchy (see Figure 4.5).

In the frame of this work, we also implemented a version of View-Dependant simplification of Polygonal Meshes [Hopp97], where the vertex hierarchy is used during

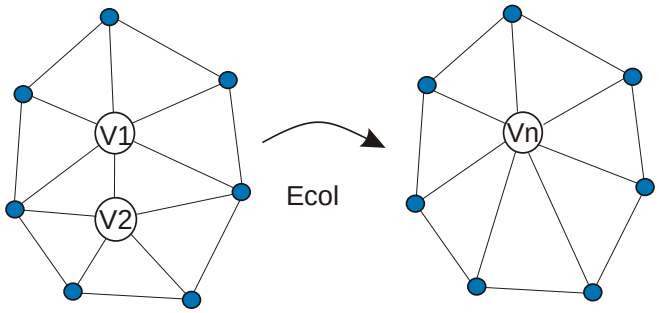

Figure 4.4: The edge collapse operation used in mesh simplification [Hopp96].

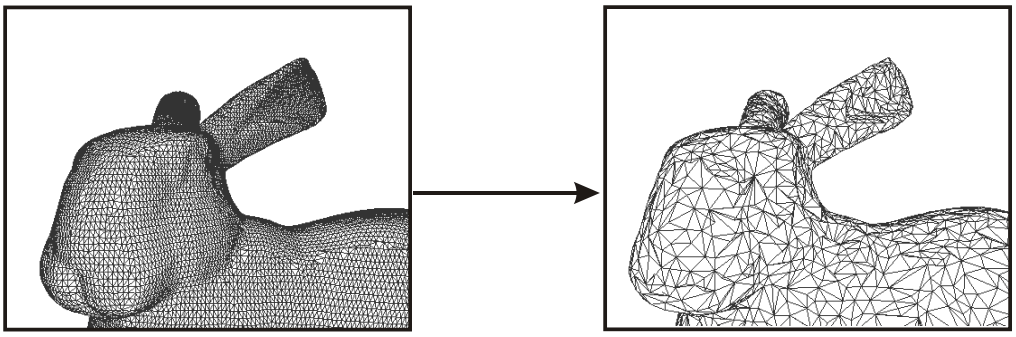

Figure 4.5: Wireframe view of the original bunny model (left) and the model after a series of simplification steps (right).

rendering to guide the process of refinement present in the output mesh. During rendering, the degree of refinement allowed is made dependent on three parameters:

1. **viewing frustum**, which permits simplification of edges which fall outside the viewing region.
2. **surface orientation**, for simplifying edges which belong to back-facing faces.
3. **screen-space geometric error**, for simplifying edges which are small enough on screen-space to be removed without affecting the resulting rendition.

The implementation of Progressive Meshes(PM) was useful to find appropriate criteria for point size resolution during rendering. However, Progressive Mesh simplification involves the overhead of continuously updating the geometry during rendering. In addition, mesh simplification occurs within a *selective refinement framework*, which establishes a set of legality conditions for the edges that can be collapsed or expanded during PM rendition (such as the presence of all the original faces previous to an edge collapse operation) [Hopp97]. This set of legality conditions is necessary for reconstructing adequately the geometry of the input mesh at different levels-of-detail, but has a negative influence on the distribution of points for NPR, because it creates a

point distribution that does not necessarily fit the desired shading when using points as stipple locations. Due to this severe drawback, we do not use the legality conditions presented by Hoppe.

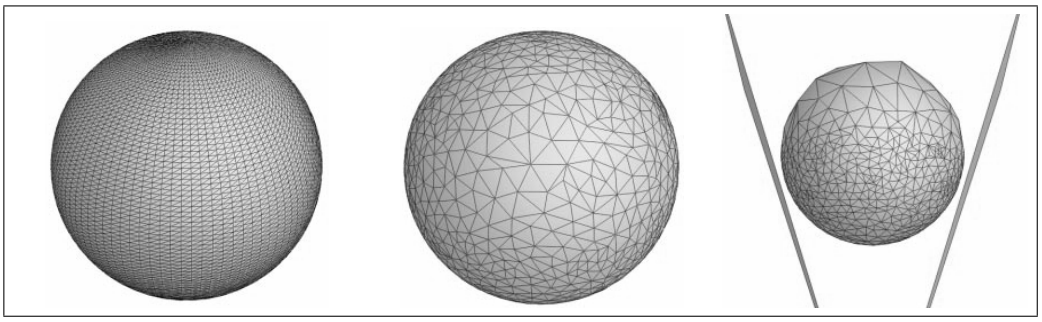

Figure 4.6: A sphere refined using view-dependent refinement of progressive meshes to preserve contour and detail on the visible side of the sphere [Hopp97]

In Figure 4.6 we observe that parts where the mesh does not require any detail, for example the back part of the sphere, still contain a significant number of vertices, these vertices are not necessary for a particle renderer, but are necessary for incremental addition or removal of detail during interaction.

The overhead of updating the geometry present in mesh-simplification techniques is not present when using particle rendering systems, because the geometry of the model does not need to be reconstructed for the purposes of particle distribution when using static models. In our implementation, the rendering system only needs to control the presence and size of the particles on the surface of the model, for this reason, the only parameter that is required to produce a point hierarchy after a refinement operation is a list of neighboring nodes, or alternatively, the average of the distance to these nodes, obtained after each refinement step.

4.3 Hierarchical Subdivision

We generate the point hierarchy by subdividing (refining) an input 3D model iteratively until a user-defined number of vertices in the model has been reached, or when the longest edge in the refined model falls under a certain threshold in object space. The operator that we use for mesh subdivision is an edge-split that creates a point around the middle of two vertices of the edge to be split (see Figure 4.7).

The base algorithm used to generate new points during mesh subdivision is the following:

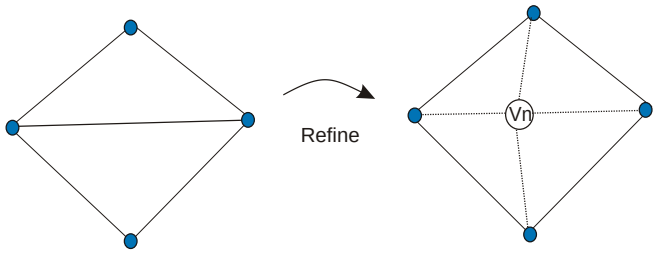

Figure 4.7: The refinement operation used in mesh subdivision.

while the target vertex count has not been reached:

1. **find** the longest edge eLongest in the mesh
2. **apply** the subdivision operator on eLongest
3. **save** a list of the nodes connected to the newly created vertex
4. **compute** the normal and other information for the new vertex

Algorithm 4.2: Generation of new vertices using mesh subdivision.

The longest edge is taken to avoid creating extremely thin triangles, which would appear if only a specific region of a model is refined.

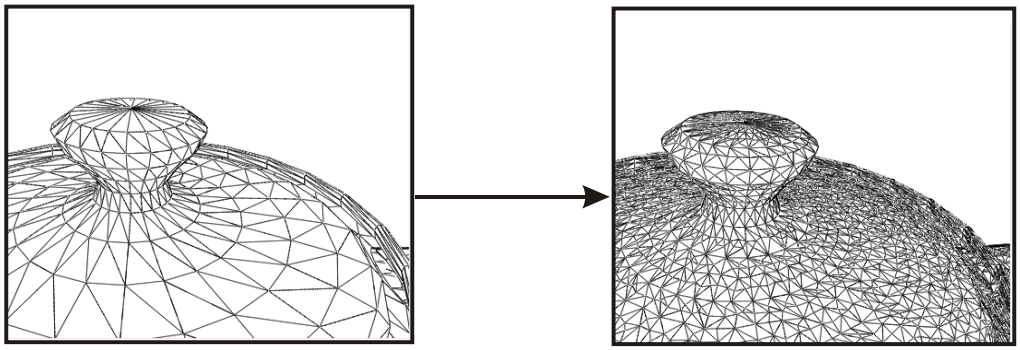

Figure 4.8: Wireframe view of the original teapot model (left) and the model after a series of refinement steps (right).

Figure 4.8 shows a wireframe view of the teapot before and after mesh refinement. It is important to notice that this approach increases the geometric complexity of a model in a dramatic way. As an example, if we take an input model with N faces and subdivide all its edges, we obtain $4 * N$ faces, and the average edge length is reduced to the half of its previous value. If we wanted to halve again the average edge length, $4 * 4 * N$ faces would be produced. This indicates that the growth of faces is exponential with respect to the inverse of the average edge length. To control the geometric growth of

our model during refinement, we prefer to set a refinement limit by defining a target number of vertices rather than by setting a threshold on the longest edge of the refined model.

4.3.1 Interactive and Local Mesh Refinement

An initial proposal for our particle rendering system deals with the idea of refining the model during user interaction. If the rendering system detects that the level-of-detail, or the amount of particles in the system is not enough to provide an adequate shading tone, more particles are generated through additional mesh refinement. In this approach the model is not refined more than it is necessary. In addition, the system contemplates local refinement, which means that only specific regions of the model where refinement is needed are refined.

Local refinement consists of refining only those edges in the model which are visible and which pass a series of tests, the most important of which is the screen-space projection of the edge. If the length of a visible edge in screen-space is found above a computed threshold, then it is refined. However, if edges are arbitrarily refined, the chance that long triangles appear is high, because neighboring triangles might not be tessellated at the level of detail of the new length. The order of refinement determines the quality of the triangulation. This is illustrated in Figure 4.9. The problem with having thin triangles is that they favor the presence of errors in the projection stage, because it is harder to project a ray on a thin triangle than to project a ray in a regular triangle, due to the loss of floating-point precision.

To perform appropriate local refinement we introduce a legality condition to determine when a polygon can be refined. Our legality condition leads to the recursive function refineLocal():

refineLocal(inputEdge)

1. **find** the longestEdge at the faces bound to inputEdge or its reversal.
2. **if** longestEdge is inputEdge apply the subdivision operator on inputE and exit.
3. **else** refineLocal(longestEdge)
4. **repeat** from step 1.

Algorithm 4.3: Constrained mesh subdivision for local mesh refinement.

The algorithm avoids the creation of thin triangles during refinement, because it enforces subdivision of the longest edges in the triangle before subdivision of the shortest edges occur. On the right side of Figure 4.9 a mesh refined with this subdivision strategy is shown.

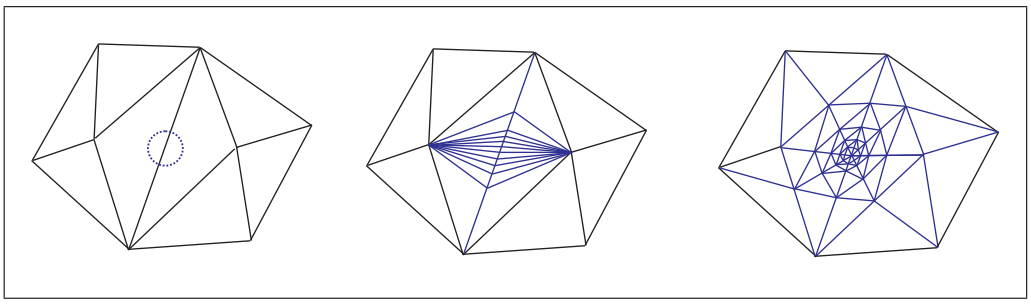

Figure 4.9: Local refinement strategies. On the left, the original mesh, with the region to be refined enclosed in a circle. In the middle, refinement starting by the shortest edge in the region. On the right, refinement using legality conditions in refinelocal().

The interactive mesh refinement system has inherent drawbacks. The first is that the geometry of the refined model must be kept in memory during rendering. If we consider that the geometry of the model grows exponentially during refinement, we find that memory resources become rapidly scarce. The second drawback is that the rendering system needs to continually monitor the existing edges of the mesh and test which edges require refinement. This monitoring process requires extra processing time which affects interaction and is directly dependent on the complexity of the refined model. The third and most important drawback is that mesh refinement is a time-consuming task. While refinement occurs, the interaction with the user stops. In sum, all the drawbacks that are present in the interactive refinement system reveal that it is not viable in terms of computational complexity and existing computational resources. We separate point hierarchy generation from rendering, as described in the rendering framework in Chapter 3, which makes it possible to have interactive rendering of particles systems, as will be shown later in this thesis.

4.4 Improving the Point Distribution

The distribution of vertices in the original model is most of the times quite regular. This becomes noticeable when we render each vertex as a point and is a problem because linear patterns are introduced in the final rendition which are not desirable from an esthetic point of view. In addition, the subdivision operator also tends to generate linear point distributions, since it creates vertices along existing edges of the model.

To reduce the presence of these patterns, "randomize and project" operations are applied to the vertices of the input model before proceeding to mesh simplification and to the vertices generated by mesh subdivision.

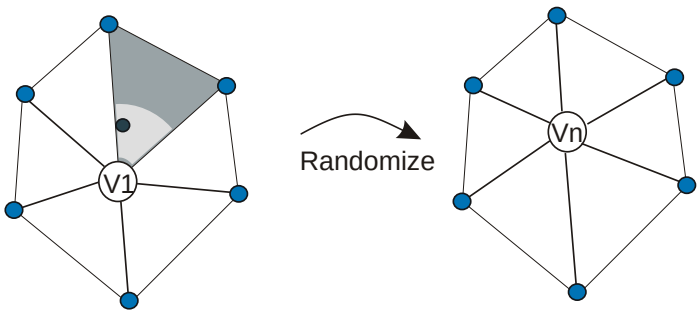

Figure 4.10: The random operator displaces the input vertex to a new location within the neighboring faces.

4.4.1 Randomization

The randomize operator receives as input the vertex to be moved and the set of faces sharing the vertex.

The vertex is displaced within the region enclosed by these faces, we first select a neighboring face at random and then displace the vertex to a point in this face which falls within a range that covers a small region (user-defined) between the input vertex and the other vertices of the face (see Figure 4.10). The region cannot be the whole face, because we could displace the vertex to a position too close to the other vertices and we would generate thin polygons, which we want to avoid in general.

Figure 4.11 illustrates the effect of the randomize operator on the overall point distribution. The image on the top left shows the original vertex distribution of the bunny model. The following images show the effects of the randomize operator at increasing degrees of randomization: the range of the randomization takes place within 15%, 35%, and 75% of the distance from the input vertex to the other vertices of the selected face.

4.4.2 The Projection Operator

A side-effect of the randomization operator is that the randomized mesh does not coincide with the input mesh. Figure 4.12 illustrates this situation. The image on the left shows a sample input mesh, whose edges represent faces of the model which are seen from above and are oriented outwards. In the center image, the vertices of the input mesh have been displaced on the faces of the input model using the randomize operator. The edges connecting the randomized vertices are not aligned with the surface of the input mesh, because the randomized vertices do not coincide with the input vertices.

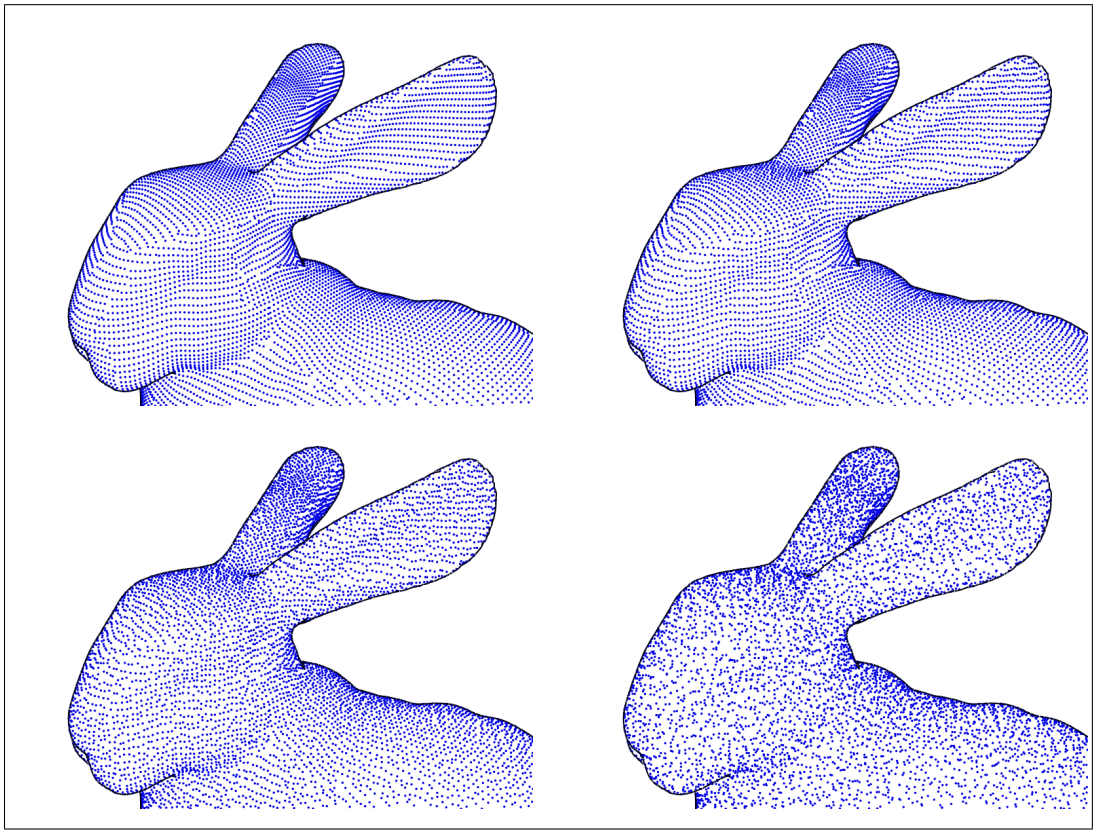

Figure 4.11: Effects of randomization on the point distribution. On the top left, the original vertex distribution in a close-up of the bunny model. The following images show the effect of the randomize operator set at increasing degrees of randomization.

Because each randomization operation can potentially displace an existing vertex to the inner region of a convex model (or to the outer region of a concave model), and each subdivision operation can generate vertices which do not lie on the surface of a model, a projection operation is applied after each randomization operation in a processing stage previous to mesh simplification and after each subdivision operation during mesh refinement. In the central image of Figure 4.12 a new vertex Vn is shown, inserted during mesh refinement. Vn lies on the inner part of the input model and would be occluded by the faces of the model, because the model itself and the randomized vertices, which represent the particle system, are rendered using the z-buffer to determine visibility.

The only case where a randomized or a newly created vertex is guaranteed to lie on the surface of the input model is when all the vertices of the polygon fan around it are coplanar and lie on the surface of the model. Otherwise, the vertex has to be projected back on the surface using a projection operator.

The projection operator implemented in our system defines a projection ray depart-

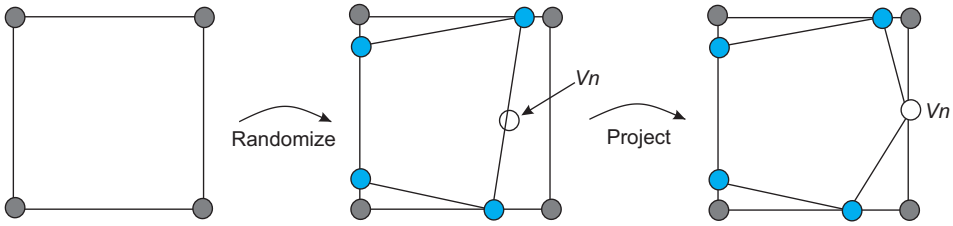

Figure 4.12: Effect of randomization on the input mesh and its correction through the projection operator.

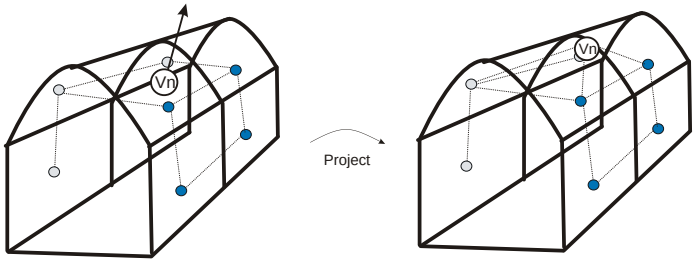

Figure 4.13: The projection operator takes a randomized vertex and displaces it to the surface of the input model. In this illustration, black lines represent the input model and thin lines represent the particle mesh.

ing from the input vertex towards the direction of the sum of normals of the vertices connected to the input vertex (see Figure 4.13). The faces of the input model which are connected to the neighbor vertices are tested for intersection with the projection ray. If more than one intersection is found, we select the one that lies closest to the input point and which is not invalid. An invalid projection is one that creates a fold in the particle mesh, which can occur if the vertex is projected past an edge that connects two neighbors of the input vertex. In the case of an invalid projection, we have chosen to discard the operation. Invalid projections are typically the result of floating point inaccuracies, which mostly occur when the model is highly tessellated. We have found that invalid projections rarely occur and can be undone without affecting the overall result.

Prior to mesh simplification, we randomize the vertices of the input model using the following algorithm which makes use of two meshes. The first is the polygonal mesh of the input model M_i, which contains the original geometry of the input model, and the second is the mesh which contains the randomized vertices M_r:

duplicate input Mesh Mi and produce Mr
for all vertices Vi in Mr:

1. **randomize** the current vertex Vi
2. **project** Vi on the input mesh Mi

Algorithm 4.4: Initial randomize and projection algorithm.

The modified algorithm to perform mesh refinement with randomization and projection follows:

while the target vertex count has not been reached:

1. **find** the longest edge eLongest in the mesh of randomized nodes
2. **apply** the subdivision operator on eLongest
3. **randomize** the newly created vertex Vn
4. **project** Vn on the input mesh
5. **if** the projection is not valid, discard projection and continue
6. **save** a list of the nodes connected to the newly created vertex
7. **compute** the normal and other information for the new vertex

Algorithm 4.5: Mesh refinement and randomization.

The computational cost of randomization and projection is high, because each newly created vertex has to be projected on the mesh and because the operation has to be validated. However, randomization and projection are essential to obtain point distributions that are esthetically admissible, as the ones shown in the bottom of Figure 4.12.

4.5 Defining the Point Set Hierarchy

A point covers an area of influence which extends beyond its projection in the image plane, and is determined by its size with respect to the canvas size, as described in Chapter 3. The amount of points in a certain region determines the region's amount of darkness, as shown in the work of Deussen et al. [Deus00] and Secord [Seco02a]. In our approach, a region of influence of a point is defined in object space by the set of its nearest neighbors. During rendering, a relevance function can be defined where a particle is drawn depending on the desired darkness at the vertex and the screen-space distances between the vertex and a group of relevant neighbors.

The process of mesh simplification creates a point hierarchy where the vertices higher in the hierarchy have more priority than the vertices lower in the hierarchy. We implement this relevance concept by storing on each point, a list of the neighboring nodes present before the vertex is removed by simplification. A relevance value, also called

'radius' can be computed from this list of neighbors by averaging the distances in object space from the vertex that is to be removed to the relevant vertices. Because the spacing between remaining vertices constantly increases during simplification, the vertices which are higher in the hierarchy have larger relevance values than the vertices located lower in the hierarchy. Figure 4.14 shows the relevant edges and the definition of the relevance radius for a given node.

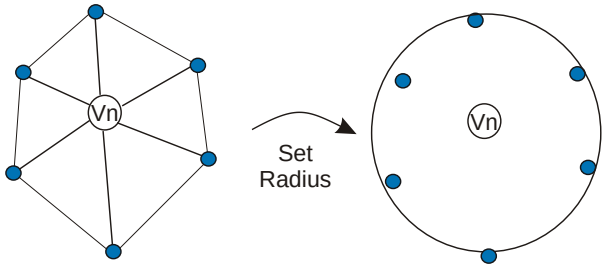

Figure 4.14: The vertices connected to a point affected by an edge collapse or an edge split are saved in the list of relevant neighbors of the resulting vertex, and determine the radius associated with the point.

In an analogous way to what is done in simplification, mesh subdivision creates a point hierarchy where the newly inserted vertices are located lower than existing vertices. The list of relevant neighbors for a vertex generated by mesh refinement is saved after applying either the subdivision, randomization and projection operations for a new vertex. Since new vertices are created between existing ones, the average distance to their neighbors is smaller than the distance of existing vertices. A continuum of relevance values from newly created vertices to simplified vertices created during simplification is produced in this way. The rendering system can decide whether it takes into account the vertex list of relevant neighbors, as is the case in off-line rendering system, or whether it takes a unique relevance value, as in the case of real-time rendering.

4.5.1 Additional Vertex Attributes

Since all vertices in the point hierarchy lie on the surface of the input model, we can define a mapping between each vertex in the hierarchy and the polygon in the input model where the vertex lies. We do this by defining the barycentric coordinates of the vertex with respect to the corresponding face in the input model (which we call the host face), and save this information in the vertex. This is used for animated stippling. Last but not least, each vertex is assigned a normal which can be obtained either by interpolating the normals of the neighboring vertices (for gouraud shading) or by consulting the normal of the host face (for flat shading). After the mesh simplification and

subdivision stages have taken place, each point contains the information described in Table 4.1.

To make use of the point hierarchy, the renderer decides to draw points with shorter edges (or shorter radius) only after points with larger edges (or larger radius) have been drawn, assuming these points lie on an evenly shaded surface (one which has constant tone). If this is not the case, the rendering algorithm determines for each point the required distance that will allow it to show up in the image, depending on the desired darkness at the point's position. Since both mesh simplification and subdivision are driven by spatial criteria (closest vertices in object space are removed first, and new points are placed roughly in the middle of existing points), points down the hierarchy appear or vanish between existing points during rendering.

4.6 Results

In this section, we present we present stippled renditions and frames from stippling animations obtained using the point hierarchies generated by mesh simplification and subdivision. In addition, we present statistical results which describe the performance of our rendering system.

The animations were produced using offline renderers embedded in a key-frame based animation software we have produced exclusively for the stippling system. The key frame files contain information about light and camera position, timing, and in the case of animated models, vertex arrays which determine the shape of the object at each key frame. Our software produces a sequence of images by linear interpolation of the information from the key frames. Further discussion on how point hierarchies are used for stippling animated models is presented in Chapter 6. The animations referred in this thesis and other additional animations are publicly available under http://isgwww.cs.uni-magdeburg.de/~oscar/.

4.6.1 Visual Results

The point hierarchy has been successfully used for producing stippled renditions at different scales. In Figure 4.15 we show the horse model at several resolutions.

In addition, in Figures 4.16 through 4.18, we show examples of our stippled renditions of some of the models described in Table 4.2. In Figure 4.16, we observe two images of the Brain model where the images have been scaled down to show a small point size. figure 4.17 illustrates the bunny model with a larger point size. The difference between the bunny on the top and the bunny on the bottom is the degree of saturation or darkness of the regions which are meant to be black. The number of stipples for the bunny

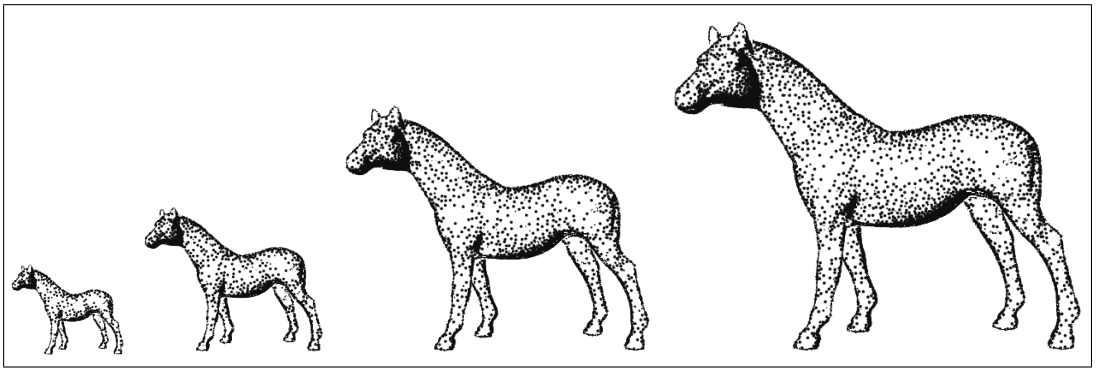

Figure 4.15: This sequence shows how stipple density increases to fill-in shaded areas as the horse model increases in size (Original sizes: 205x173, 237x205, 295x257 and 463x392).

on the top is 2,578, while the bunny on the bottom was rendered using 11,500 points (see Chapter 8 for further discussion on this issue). We control the degree of saturation by specifing a lowest value for the distance between stipples, so that no stipples can be too close together for the darkest tone. A side effect is that the degree contrast between regions of the rendition decreases. In Figure 4.18, we show two images of the horse model, rendered using a small point size. Again, the image of the horse on the bottom has a strong contrast, which was achieved by setting the light source as coming from its left side, not by changing the degree of saturation.

Videos 3-*teapot Gouraud* and 4-*brainHiRes* show animations of the teapot and the brain models obtained using our technique. All animations produced are frame coherent and have points smoothly fading in and out of the renditions as shading and scale change. When point size is small (1.0 or less), some vibration appears in the animations. This is due to the effect of aliasing: since small points have to move between existing pixels, the points are not consistently represented by the output device due to the relatively coarse resolution of current displays, this effect is reduced by applying several rendering passes with the camera being moved slightly (OpenGL antialiasing).

4.6.2 Point Hierarchy Generation Time

In Table 4.2 we show performance statistics related to the generation of point hierarchies using mesh simplification and subdivision using an SGI Onyx 2 with two processors.

From Table 4.2, we can observe that the time required for creating point hierarchies depends on the complexity of the input models, in terms of polygon count. In the case of simple models (less than 10,000 polygons), the system can easily generate tens of thousands of points in less than two minutes using mainly mesh subdivision. In

Figure 4.16: Frames from our stippled renditions of the brain model (Original sizes: 602x501 & 730x554).

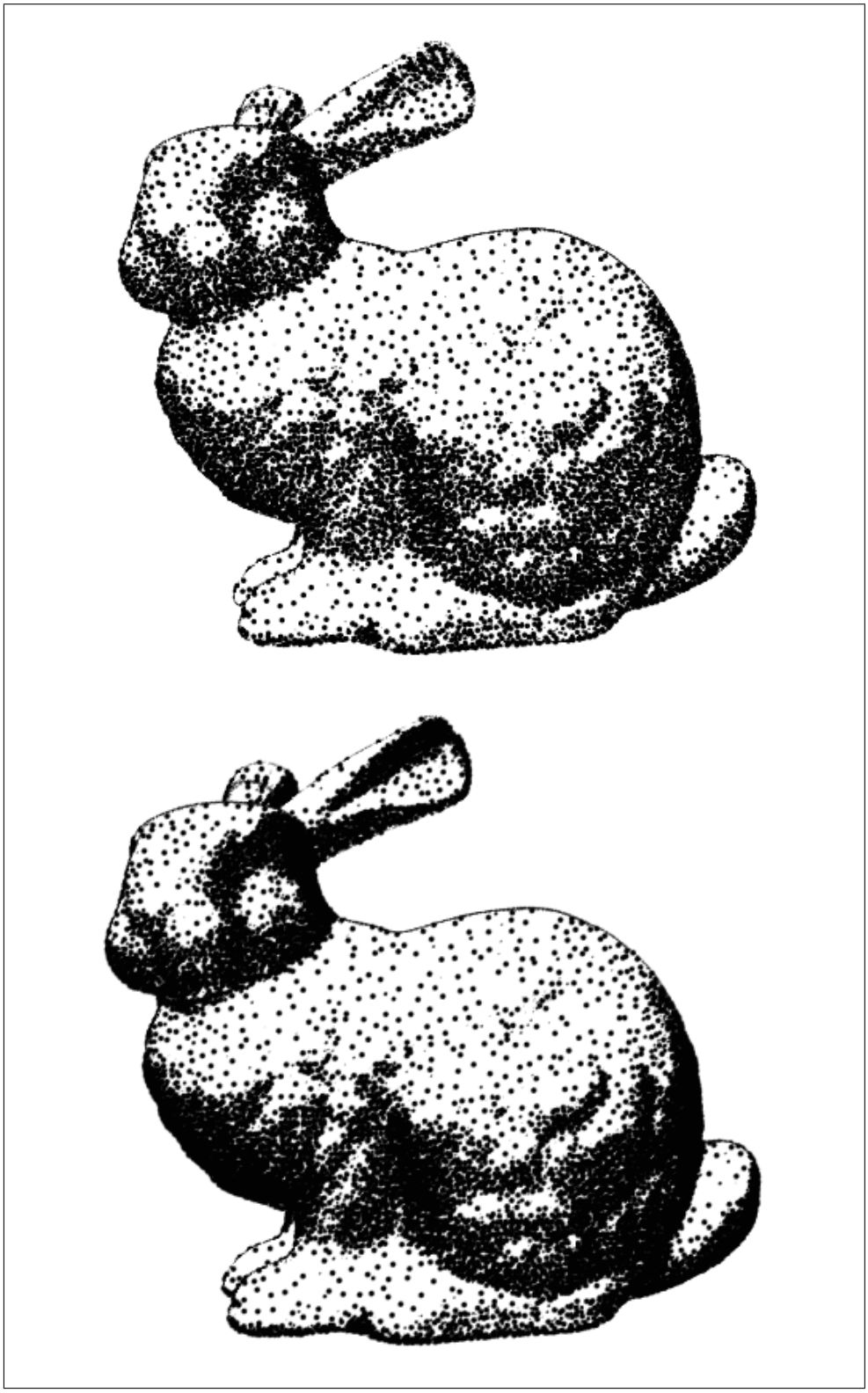

Figure 4.17: On the top, we show a rendition of the Stanford bunny where the darkest tone is not totally black, while the image on the bottom shows the darkest tone possible and a higher contrast (Original sizes: 455x425).

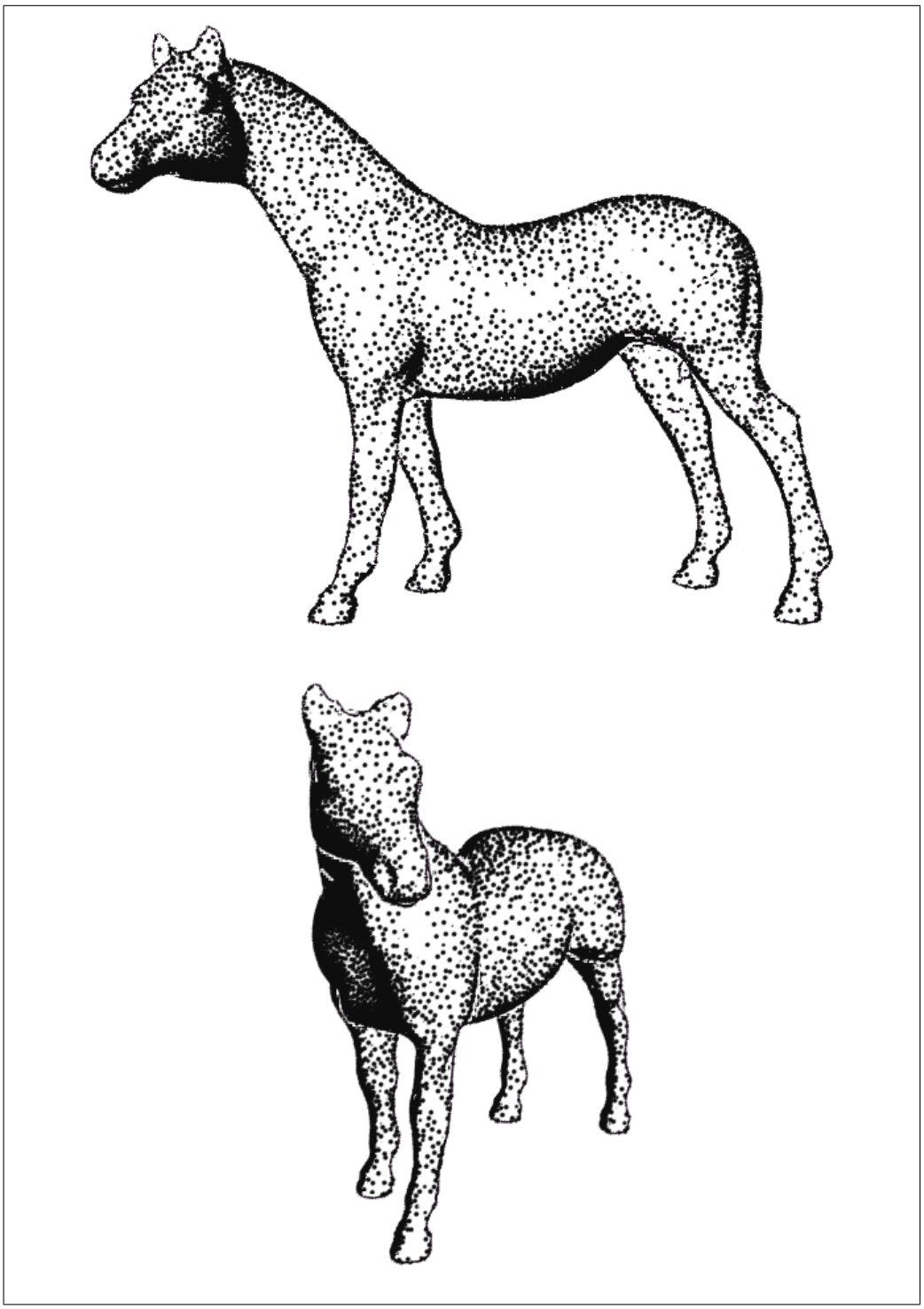

Figure 4.18: Stippled renditions of the horse model illustrating changes in position and illumination. (Original sizes: 578x477 & 335x479)

Model	Polygons	Points generated	Generation Method	On-line Rendering	Off-line Rendering	Memory Required	Setup Time	Simplif. Time	Subdiv. Time	Total Time
Teapot	2,256	70,000	Subdivision	0.74s	1.8 s	74Mb	1s	1s	39s	41s
Crocodile	1,684	40,000	Subdivision	0.5s	0.6s	48Mb	1s	1s	44s	46s
Hand	2,878	80,000	Subdivision	0.9s	2.0s	86Mb	1s	1s	50s	52s
Heart Vents	3,852	34,000	Subdivision	0.8s	2 s	88Mb	41s	1s	19s	1m 1s
Bunny	69,459	34,834	Simplification	1.5s	2.3 s	72Mb	17s	1m 20s	–	1m 33s
Bunny	69,459	70,000	Simp. & Subd.	1.9s	2.7 s	104Mb	17s	1m 20s	26s	1m 59s
Horse	96,966	48,485	Simplification	1.8s	2.3s	94Mb	25s	1m 46s	–	2m 11s
Bunny	69,459	120,000	Simp. & Subd.	2.4s	3.1 s	148Mb	17s	1m 20s	58s	2m 31s
Horse	96,966	100,000	Simp. & Subd.	3.7s	3.84 s	142Mb	25s	1m 46s	34s	2m 45s
PiggyBnkSml	86,040	120,000	Simp. & Subd.	2.82s	4.0 s	156Mb	17s	1m 28s	1m 04s	2m 49s
PiggyBnkLrg	172,078	86,041	Simplification	2.4s	6 s	192Mb	38s	3m 29s	–	3m 59s
Brain	288,334	144,171	Simplification	6.3s	15s	262Mb	1m 9s	5m 41s	–	6m 50s
Mosaic	400,000	200,002	Simplification	8s	— s	363Mb	1m 30s	8m 30s	–	10m
Heart Stip	173,274	70,000	Simp. & Subd.	16s	21 s	76Mb	–	–	—	–s

Table 4.2: Statistics for creating Point Hierarchies by Mesh Simplification and Subdivision

these models, almost 100% of the total time used for generating the point hierarchy is spent in mesh subdivision. For the largest models (200,000 polygons and above), only simplification is performed, since the number of existing points is more than enough to obtain stippled renditions which are scalable in a number of cases, and no subdivision is required. For models of middle scale (between 10,000 and 200,000 polygons), a mix between simplification and subdivision is performed to obtain point hierarchies with some 70- to 100-thousand points. The time spent between mesh simplification and mesh subdivision in these cases tend to be dominated by mesh simplification. For example let's take the case of the small piggy-bank model (PiggyBnkSml), which has roughly 43,000 vertices, but is subdivided to obtain 120,000 vertices. That is, during subdivision, 80,000 points are generated. The timing results reveal that the system spent 57% of the time during mesh simplification and 43% during subdivision, although two thirds of the points were generated using mesh subdivision. In the case of the Horse model with 100,000 points, 50% of the points were generated using mesh simplification and the rest using mesh subdivision. However, the time required for mesh simplification was 75% of the total, which reveals that the simplification task tends to dominate over the subdivision task.

4.6.3 Rendering Time

We measured rendering times for on-line (on-screen) and off-line rendering (rendering an image to be stored in a file). On-line rendering times are acceptable for some tens of thousands of points, which means that the user can still manipulate the model and wait to see the results. For models containing 100,000 points and above, on-line manipulation becomes impossible in the SGI platforms. It is important to notice that several user studies demonstrate that interactive applications require a refresh rate of 30 frames per second, while 60 frames per second is standard in existing real-time interactive environments. Since we compute the point size by software, it is practically impossible in our system to achieve such high frame rates for point sets having above 10,000 points. On the other haned, in Chapter 7 we describe an implementation of point hierarchies using vertex programs, which makes possible the rendition of stippled models at interactive rates.

Our recorded animations make use of offline rendering, which takes more or less 60% additional time for the creation and storage of each individual image for an image which has the same size as the viewport on the display (typically 440x358 pixels). However, off-line rendering takes much longer time when the output resolution of the images is large, and this effect is enhanced when the input model itself is large. With respect to rendering times Table 4.2 also shows that the very largest models constitute a problem in memory resources and time required for point hierarchy creation. For example, in the case of the Mosaic model, which contains 400,000 polygons, the on-line rendering times peaks to 8 seconds.

5 On the Generation of Point Hierarchies Using Patch-Based Point Relaxation

In this Chapter, a description of an alternative approach to create point hierarchies by point relaxation is described. The initial proposal to generate regular point distributions by relaxation was done by Deussen, who pointed out that the work done by Turk for Re-tiling of Polygonal Meshes [Turk92] could be useful in the context of stippling. In fact, the point distributions obtained for polygonal re-tiling and texture synthesis using reaction-diffusion [Turk91] are evenly spaced and are thus interesting for producing high-quality stippling renditions, which is our main motivation.

In the technique proposed by Turk, a set of points is randomly distributed on the surface of a polygonal model. This set of points is distributed by relaxation on the surface of the model by having points repel one another. The neighborhood information is obtained by regular spatial subdivision, and the points are projected in a plane before computing the forces that the neighboring points exerted on a given point. Points are displaced by moving them on the surface of the polygonal mesh, which involves translating a point from polygon to polygon, as a result of the repulsion forces.

The approach we present is based on the work of Turk previously mentioned, but we have modified the algorithm as follows:

1. We have a graph-based approach to distribute points at the higher levels of the point hierarchy, instead of using the original model surface: We simplify the approach of Turk by performing relaxation of points only between the nodes of a graph, which eliminates the need of performing geometrical operations required by Turk's approach. In addition, the graph where the points are defined often has fewer nodes as the polygonal mesh of the input model, which also results in a fast relaxation process when generating the higher levels of the point hierarchy.
2. We avoid the use of a regular point space subdivision scheme to save memory resources: In our approach, the set of neighboring points for the purposes of

relaxation is determined by inspection of the region around a polygon or a patch using connectivity information. In the approach of Turk a space-subdivision data structure is used to determine the set of neighboring points.

3. We avoid polygonal re-tiling, since it is not necessary to generate geometric levels of detail to create the point hierarchy.

We have implemented a system that generates a point hierarchy by point relaxation as follows:

1. Compute a connectivity graph to operate on the input polygonal mesh.
2. Randomly distribute a set of points on the surface of the input model.
3. Apply a relaxation phase on the initial point distribution (see section 5.2).
4. Create a patch hierarchy by iterative patch fusion.
5. For each level of the patch hierarchy:
   a) Perform particle relaxation by token (point) displacement.
   b) Compute the point radius for the particles at the current level.

We describe each stage in detail in the following sections.

5.1 Setup and Initial Point Distribution

The first step for point relaxation is to compute a connectivity graph in the same way as done for mesh simplification and subdivision presented in Chapter 4. This connectivity graph is used to create the lowest level of the patch hierarchy. At the lowest level of the patch hierarchy, the term polygon and patch can be used interchangeably, because each polygon is considered a patch, such that the initial patch distribution is the polygon mesh itself. After the initial polygon mesh is computed, each polygon is assigned a random point on its surface, which we refer to as the polygon 'center', and indicates the location of a potential particle.

5.2 Primary Relaxation of the Initial Point Distribution

An artifact which emerges when assigning one stipple to each polygon in the input model is the presence of linear patterns that emerge as a reflection of the tessellation

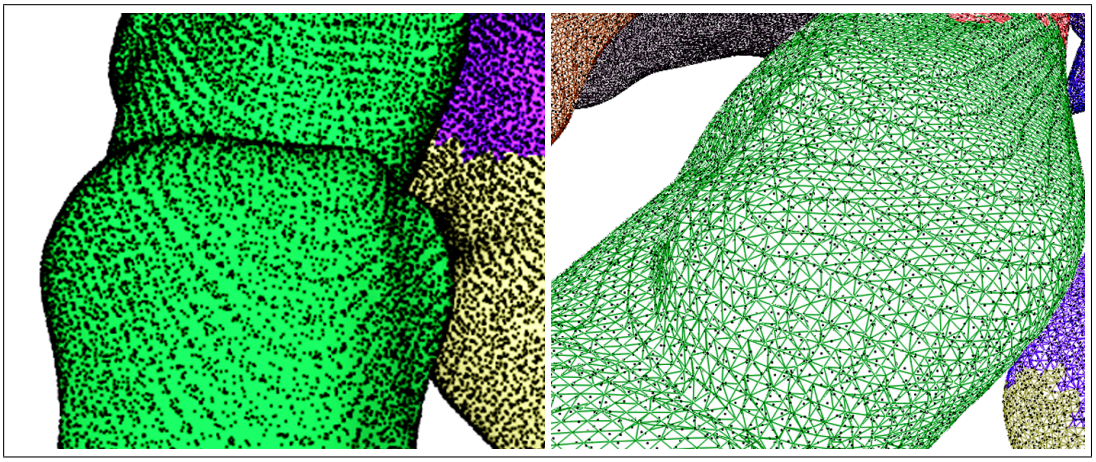

Figure 5.1: Point distribution in irregular meshes. The image on the left shows linear patterns formed along the zones of higher tessellation of a sample model. On the right, we observe the wireframe view which explains these patterns. In both images, a point has been randomly placed at each face (original sizes: 600x489 & 700x571).

of the input model. Areas which contain a higher density of points are visible in the zones of higher tessellation of irregularly tessellated meshes(see Figure 5.1).

One way to solve the presence of this artifacts is to apply mesh subdivision until the mesh is regularly tessellated with small polygons. However, if the input mesh already has a large vertex count (we refer in this case to models that have 400,000 triangles or more), it is not practical to further increase the number of polygons of the model.

A better solution is to perform point relaxation on the initial point distribution to improve the spacing between these points. Our approach consists of applying a number of relaxation iterations on the initial point distribution, which also reduces the noise artifacts typically present in random point distributions. Our goal is to improve the aspect of the initial point distribution, such that the stipples are regularly distributed at the lowest level of the point hierarchy. The relaxation algorithm we use is the following:

loop until a target number of iterations is performed
    for each polygon in the input mesh
        retrieve point Pt associated with the polygon
        determine neighbor points of Pt
        compute and store the repulsive forces that the neighbor points exert on Pt
    for each point Pt in the input mesh
        displace Pt on the surface of the model according to the repulsive forces.

Algorithm 5.1: Relaxation of the initial point distribution.

This algorithm is basically the one presented by Turk. However, contrary to Turk, we do not use regular spatial subdivision to determine the location of neighboring points.

We use the connectivity information of the input model to determine the neighboring points using the following algorithm:

for each vertex Vt in the input polygon Poly
    retrieve the list of polygons connected to Vt
    insert in the list of relevant neighbors of Poly all the polygons sharing vertex Vt, removing duplicates and Poly itself

Algorithm 5.2: Retrieving the neighbouring vertices of a given polygon.

During relaxation, the points will start repelling each other. If the forces applied by the neighbors of a given point push it outside of the boundaries of its host polygon we let it 'jump' to the neighboring triangle. To determine the position of the vertex in the new polygon, we produce a displacement vector in the direction of the repulsing force, using a projection such that the vertex is displaced on the surface of the new face. Finally, we set the new face where the vertex lies as the new host face, and include the vertex on the list of contained points for the host face and remove it from the previous face where the point came from. The algorithm to retrieve the neighboring points is modified such that the relevant neighbors is the set of all the points in the neighboring polygons and all the other points which share the same host face.

We have found that the appropriately spaced point distributions are obtained after some 5 to 15 iterations are done, after that, the effect of the iterations is not visually significant, i.e. the points are already relaxed. We want to minimize this number of iterations, since the time required for the initial relaxation is directly dependent on the number of initial iterations. In Figure 5.2 we show the initial point distribution and the point distribution after some iterations.

5.3 Graph-Based Point Relaxation

To create a point hierarchy using relaxation, we present a graph-based approach where point relaxation occurs iteratively on each level of a patch hierarchy.

The patch hierarchy is created by patch fusion (described in section 5.4), starting from the lowest patch level, the one obtained during setup. The patches created at each level of the hierarchy are used as the 'playing field' where tokens are going to be distributed and relaxed. The 'playing field' is a graph whose nodes are points in 3D space located on the surface of each patch (we refer to each of these points as the 'center' of the

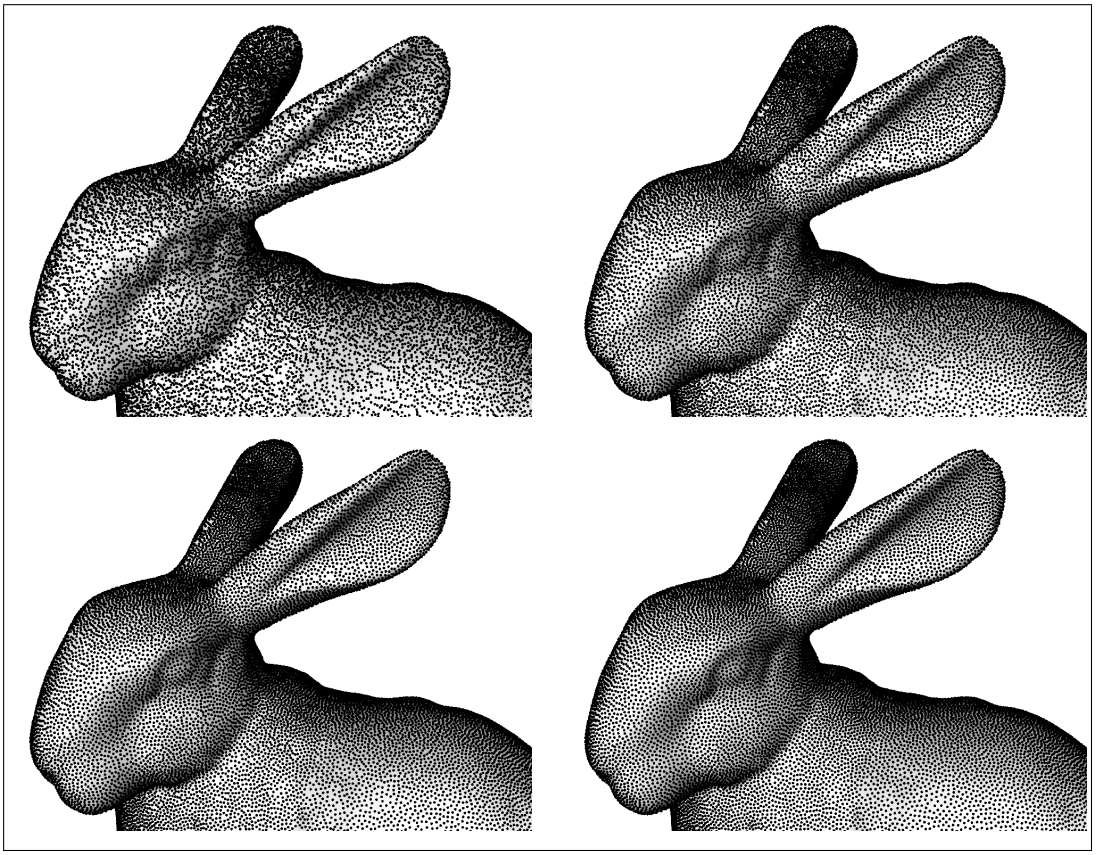

Figure 5.2: Effects of the primary relaxation on the point distribution. On the top left, the original vertex distribution in a close-up of the bunny model. The following images show the effect after 5, 10, and 15 relaxation iterations.

patch), and whose edges are formed between each pair of patch centers that belong to neighboring patches (see Figure 5.5).

After obtaining the graph, we proceed to distribute a number of 'tokens' within the nodes of the graph. These tokens represent point particles that are going to be the subject of the relaxation process, which is explained in section 5.5.

The relaxation process is done first on the highest level of the point hierarchy. Once the relaxation of the tokens has been finished at a certain patch level, we fix the tokens on the surface of the model, go down one level in the patch hierarchy, generate additional tokens and repeat the relaxation process on the new tokens. This is done until all levels of the patch hierarchy have been traversed.

After each relaxation step, we assign to each of the relaxed points a radius value which indicates the average of the distances to its neighbors. The particles that are distributed at the higher levels of the hierarchy have longer radius values than the particles relaxed

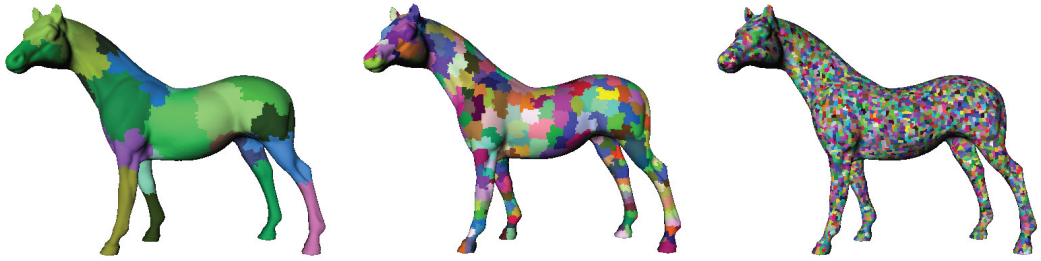

Figure 5.3: Patch hierarchies on the horse model. The first image (from left to right) shows the model with 25 patches, the second and third models have their surface subdivided in 350 and 6144 patches, respectively.

at lower levels of the hierarchy. This information is saved in a file and used in the real-time, animated stippling stage.

5.4 Patch Hierarchy Creation

The patch creation approach we use was originally developed for visibility preprocessing. [Meru02a] We use a variation of this approach to produce patch hierarchies intended specifically for point relaxation. The initial patch list is constructed from the graph connectivity information of the input model: each polygon becomes a patch, and the polygons which share an edge with a given polygon are inserted in the list of neighboring patches of this polygon.

The patch hierarchy creation works by iterative fusion of the patches available at a given level, starting from the level where each polygon in the input model is considered a patch.

Initially, all the patches of a given level are stored in a binary search tree (BST) ordered by the amount of surface they cover. In each fusion step, the smallest patch found in the BST is combined with a number of neighbors (four in our case), creating a patch at the next level. To select which neighbors will be combined with the smallest patch, we choose those neighbors that minimize the span of the bounding box of the original patch (my measuring the lenght of the line which joins the minimum and maximum points of the box), with the restriction that only patches at the same level can be fused together. After a patch is combined with another, the patch connectivity information needs to be updated. This is done by adding the neighbors of the old patch in the list of the neighbors of the new patch and by removing duplicates as well as by updating the information of the old neighbors of the removed patch with the identification of the newly created patch.

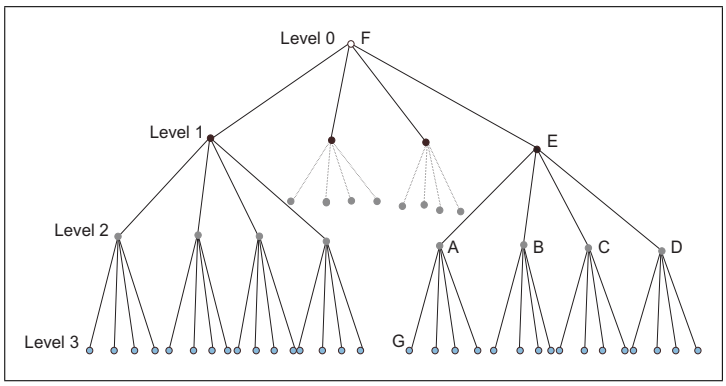

Figure 5.4: Illustration of a point hierarchy resulting from the patch hierarchy. At the lowest level of the hierarchy, there is a one-to-one correspondence between a point and a patch.

This connectivity information is used for creating the next level in the patch hierarchy, and to guide the process of particle relaxation. The patches that were combined are removed from the BST and the next smallest patch available is used at the next fusion step until all patches of a given level have been removed from the BST.

The patch hierarchy creation process finishes once a certain number of target patches at the highest level are reached or after a certain number of levels have been produced. In our case we stop when we have less than 10 patches at a certain level, which means that we have no more than 10 stipples at the highest level of the particle hierarchy. In Figure 5.3 we show a series of patch hierarchies for the horse model.

A patch hierarchy is initialized with patch 'center' values by randomly assigning one point to each of the patches at the lowest level (which are the base polygons of the model). Patches of the higher levels are associated with points selected from their descendants. For example, in Figure 5.4, the patch E that joins patches A, B, C and D, has as 'center' either A, B, C, or D's 'center' values.

5.5 Particle Relaxation by Token Displacement

Now we are ready to carry out a token relaxation process to distribute particles on the surface of the model. The goal of the relaxation process is that the distribution of the particles does not conform with any linear or regular pattern, and that the spacing among particles is regular at each level of the point hierarchy.

We perform the process of particle relaxation as a token distribution approach. A token represents the place where a rendering particle is located. We borrow the term token from the computer networks literature, where a token is passed along the nodes

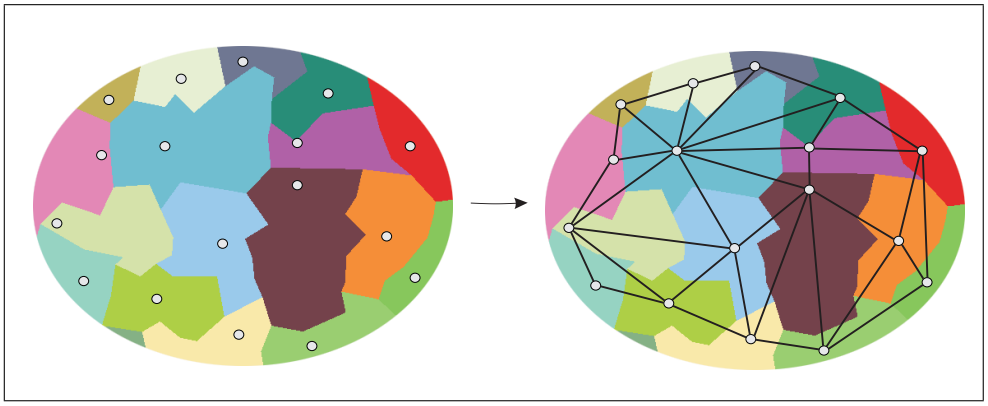

Figure 5.5: Left: a sample patch array showing the points on the patches. Right: the graph obtained from the patch array based on the neighbors information.

of a graph and does not belong to any specific node until relaxation finishes. The graph where the tokens are distributed is defined by a list of all the patches and their connectivity information at a given level: the nodes are the points in 3D associated with each patch and the edges are formed by the list of neighboring patches of each patch in the list (see Figure 5.5).

The relaxation process starts by pseudo randomly distributing tokens among the nodes of the graph. After that, tokens are brought apart from each other by iteratively applying a relaxation step where each token is pushed away by the tokens in its neighborhood. At each iteration, we count how many tokens were actually displaced (changed their position in the graph). Relaxation stops when the displacement count is zero or when we detect this count has stopped decreasing. In summary, our algorithm for the relaxation process is as follows:

distribute tokens on the graph
loop until the displacement count stops decreasing
    for each token T on the graph
        determine neighbor tokens of T
        compute the repulsive forces that the neighbor tokens exert on T
        determine whether the token needs to be displaced, and in this case, which of the neighbor nodes should receive the token based on the repulsive forces
    for each token T on the graph
        if the token needs to be displaced, displace the token T to the new host node, and increase the displacement count

Algorithm 5.3: Graph-based token distribution.

In the next sections, we describe each step of this algorithm in more detail.

5.5.1 Token Distribution

Initially, we randomly select, for each patch at a given level in the hierarchy, one descendant found n-levels below in the patch hierarchy, and assign this descendant a token (as an example, node G in Figure 5.4 is a descendant located three levels below node F). The goal of using patches at lower levels in the hierarchy is to relax the tokens in

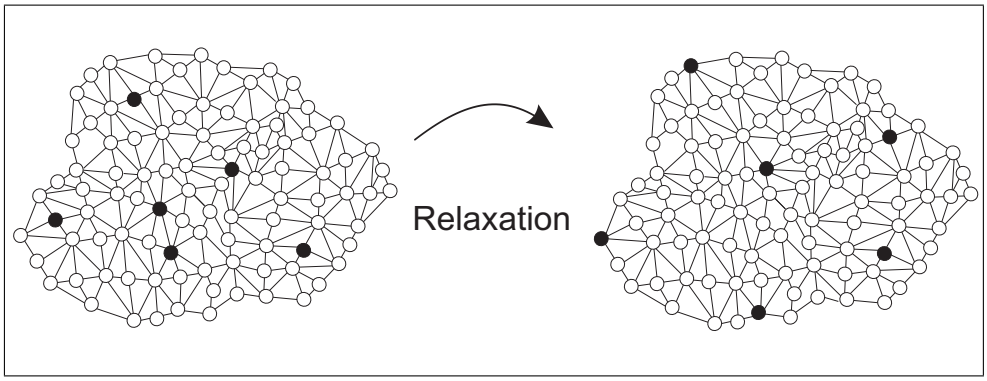

Figure 5.6: Left: a sample graph where some tokens have been distributed among the nodes of the graph (nodes with a token are shown in black). Right: the token distribution after relaxation.

a graph with such a number of extra nodes, that there is enough room for the tokens to be freely displaced. In our case, we have chosen n to be 3, so that each selected descendant is displaced within a graph containing approximately 4^3 nodes without a token.

Figure 5.6 illustrates a possible token distribution on a graph. If we had many more tokens in the graph, it would be much harder to evenly distribute the tokens, since there would be just too few positions where the tokens could be moved to.

5.5.2 Determining the Set of Neighboring Tokens

The first step to perform relaxation on a given token is to determine which tokens are located in its neighborhood and store them in a list. First, we include in the list the tokens of the patches which are neighbors of the high-level patch which currently holds the token. After that, each token explores its neighborhood using the patches at the lowest patch level where the tokens are located. In this case, we explore the immediate neighboring patches and we also visit all the descendants of its ascendants, up to two levels higher in the hierarchy. Each node in this set is queried as to whether the node is host of a token. All the nodes which are hosting a token are inserted in the list of neighbor tokens, which is used to compute the repulsive forces exerted on

the original token. An alternative approach to find the list of neighbors is to visit all nodes which are within a certain distance from the node. We have tried this using a Breadth-first search algorithm which operates only within a certain radius of action and traverses the graph from the token outwards, but this was more time-consuming than visiting the set of descendants of the ascendants.

5.5.3 Computing the Repulsive Forces

To compute the repulsive forces, we use those tokens in the list of neighbor tokens which are within a region of influence determined by a *repulsion radius*. The repulsion radius is obtained as:

$$\mathrm{repulsionRadius} = k\sqrt{a/n}$$

where k is a constant which varies from 1 to 2 (a value of 1 is sufficient for closed models, a value of 2 enlarges the area of influence and is more appropriate for open-ended models), a is the area of the surface of the input model and n is the number of tokens being distributed.

We compute the repelling forces $\mathrm{ForceVec}$ by computing the vectors $\mathrm{NT\_Vec}$ which go from the position of the neighboring tokens N to the input token T, and then scaling each of these according to their distance, and in proportion of the repulsion radius, using the formula

$$\mathrm{ForceVec} = (\mathrm{repulsionRadius} - |\mathrm{NT\_Vec}|) * \mathrm{NT\_Vec}/|\mathrm{NT\_Vec}|$$

After that, we average all the neighboring forces and produce a vector in 3D which indicates the direction of the overall force that the neighbors within the repulsion radius exert on the input node T.

5.5.4 Determining the Displacement

If the overall force is different from zero, we need to determine which of the empty neighbors around T is the best candidate for where the token should be moved. This is done by computing the dot product between the computed force and the vectors leaving from the host node to its neighbors. The neighbor which maximizes this dot product is the one chosen as the candidate to where the token should be passed. This candidate node is stored as the node that will hold the token, and another token is processed.

Once the displacement of each token is computed, the actual displacement of the tokens is done. Figure 5.6 illustrates this relaxation process. Since the tokens are restricted to move on the graph, a token can constantly jump back and forth between

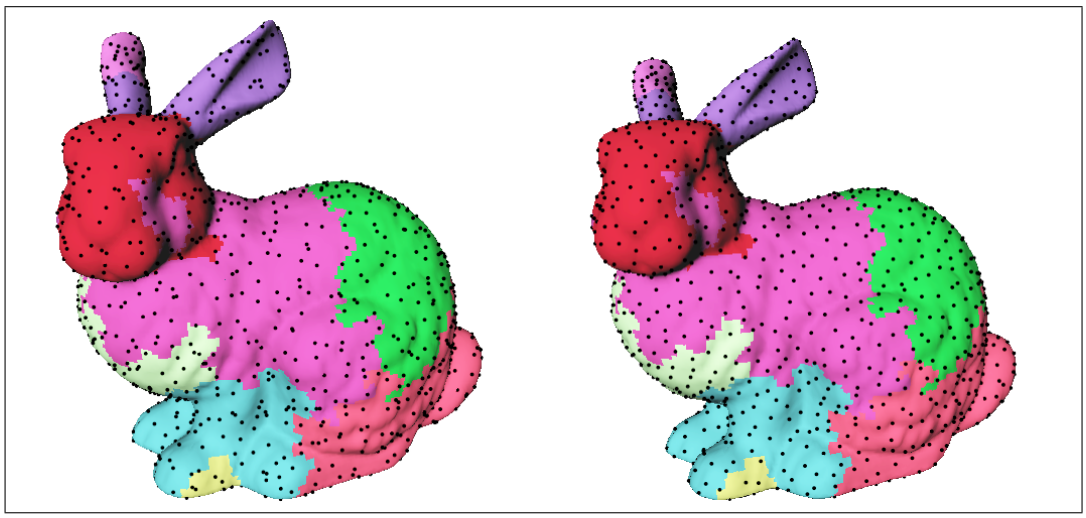

Figure 5.7: On the left, the bunny model with 1500 points. On the right, the bunny with the same points after relaxation.

two nodes in search of the optimal position. Hence, the relaxation process stops once the total energy (the sum of all the forces exerted on all the tokens) has stopped decreasing, when we consider the system has reached stability. Most of the time our system reaches a point of stability (the displacement count stops decreasing) within less than 30 iterations on any given level. In Figure 5.7, we see the initial point distribution and the result of the relaxation process after the system has reached stability.

5.5.5 Creation of the Point Hierarchy

We have so far described how we can distribute particles on the surface of a model for a specific patch level. To create a point hierarchy, we follow the idea proposed by Turk [Turk92] for mesh re-tiling. We start the relaxation process with the patches at the highest hierarchy level, then we fix the results of the relaxation and proceed to distribute and relax tokens at the next level, until we have traversed the complete hierarchy. The algorithm of this process is the following:

while not all patch levels have been expanded:

1. **select** a number of tokens for each patch on the highest level of the hierarchy.
2. **perform** graph-based relaxation.
3. **fix** the relaxed tokens on the graph and on the patch hierarchy.
4. **expand** the patches' graph by introducing all the descendants from the patches at current hierarchy level.
5. **if** not all patch levels have been expanded, iterate from step two.

Algorithm 5.4: Creation of the point hierarchy using patch-based token relaxation.

Figure 5.8 shows a sequence of levels of the point hierarchy for the hand bones model after relaxation. In addition, the accompanying videos illustrate how this graph-based relaxation process works. In Video 5-*HorseRelaxation* we show an interactive session where the user generates a series of points and indicates that they should be relaxed, after some iterations the user fixes the points, generates new ones and starts relaxation again (the user interaction has an illustrative purpose only). In Video 6-*HorsePtHierarchy* we show the session where the complete point hierarchy is automatically generated for the horse model. Notice that the session has been shortened to show only the patch hierarchies at different levels, in reality, the first levels of the point hierarchy are computed very fast, because they contain few elements. As the point hierarchy is created, the relaxation consumes more time.

5.6 Results

Similar as in Chapter 4, here we present stippled renditions and frames from stippling animations obtained using the point hierarchies generated by the method presented in this Chapter. We also present statistics on the performance of our system for creating point hierarchies by means of token-based relaxation.

5.6.1 Visual Results

In Figures 5.10 through 5.13, we present frames from our animations produced using token-based relaxation. In addition, in Figure 5.9 we show the bunny model at several resolutions. The point distribution obtained by means of token relaxation is similar to that obtained by mesh simplification and subdivision. In general, we can observe in the Figure showing the bunny at several resolutions that the point distribution is kept regular in most cases. In addition, it is not possible to identify linear patterns at levels of the hierarchy different from the last one. The last level of the hierarchy will be compared in more detail in the Section 5.7. Video *7-point distribution* shows the point distribution obtained by relaxing points at a certain level for the horse model. Points are evenly distributed on the surface of the model.

5.6.2 Point Hierarchy Generation Time

In Table 5.1 we present our statistiacl results. We have split the time required for the primary relaxation (column 5) and the time required for the actual patch-based token relaxation (column 6) and observe that the time spent during primary relaxation is normally less than the time spent for the token relaxation.

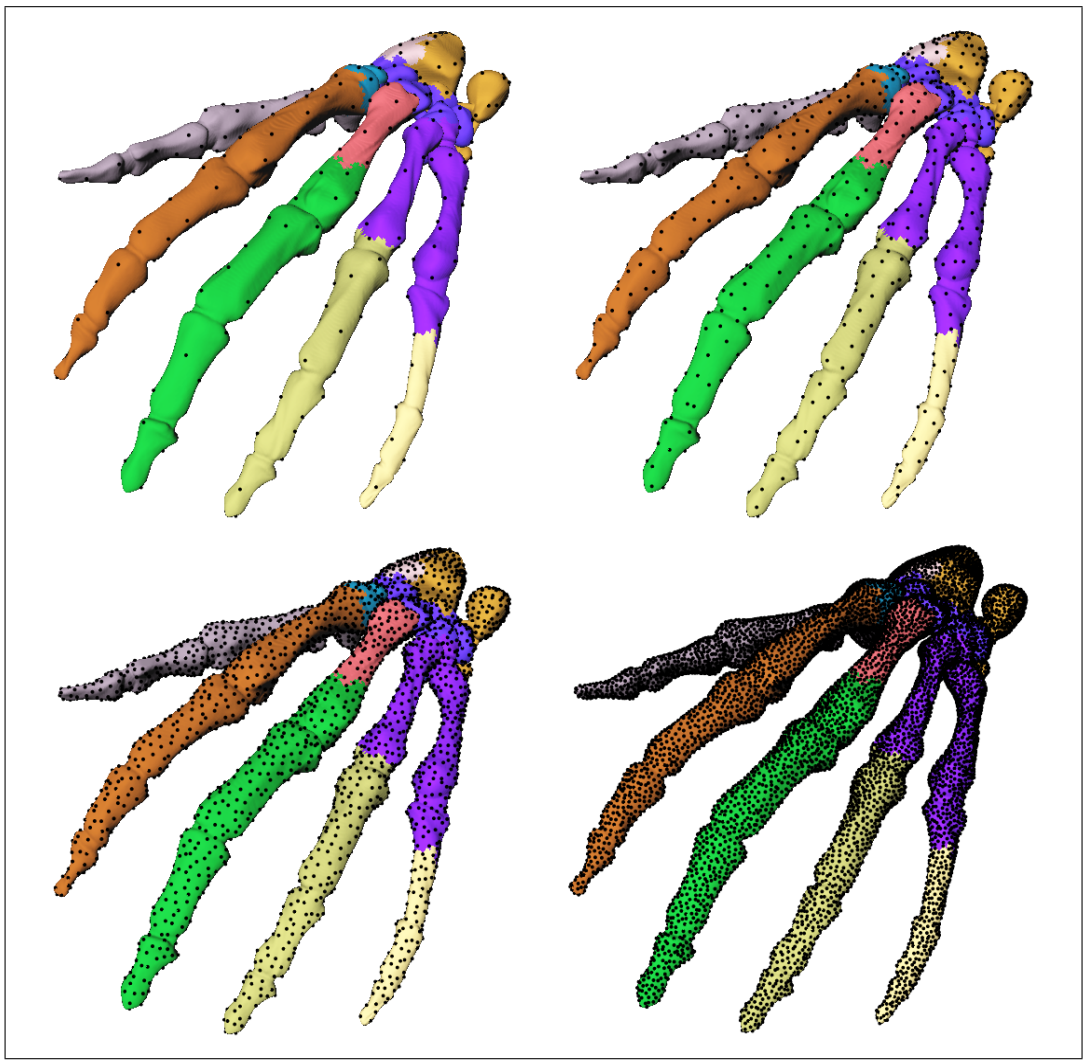

Figure 5.8: Four levels of the point hierarchy after patch-based relaxation, the hand-bones model is shown with 332, 1,192, 4,277 and 15,184 points respectively (original sizes: 527x535).

It is important to note that in our results, we applied a constant number of six primary relaxations on all input models. We did this, because we observed that this number of iterations offered a good compromise between point distribution and time required to perform the relaxations. Initially, we had designed the primary relaxation technique as an additional way to improve point distributions. For the token-based relaxations, we set a maximum of 50 iterations per level, or a maximum of 15 iterations without decrease in the total tension between the relaxed particles. This limits were set so, after observing that for most cases, the system reached stability, or no improvement after some 15 iterations are performed for a given level. However, the maximum limit of 50 iterations could be reduced to reduce point hierarchy generation time.

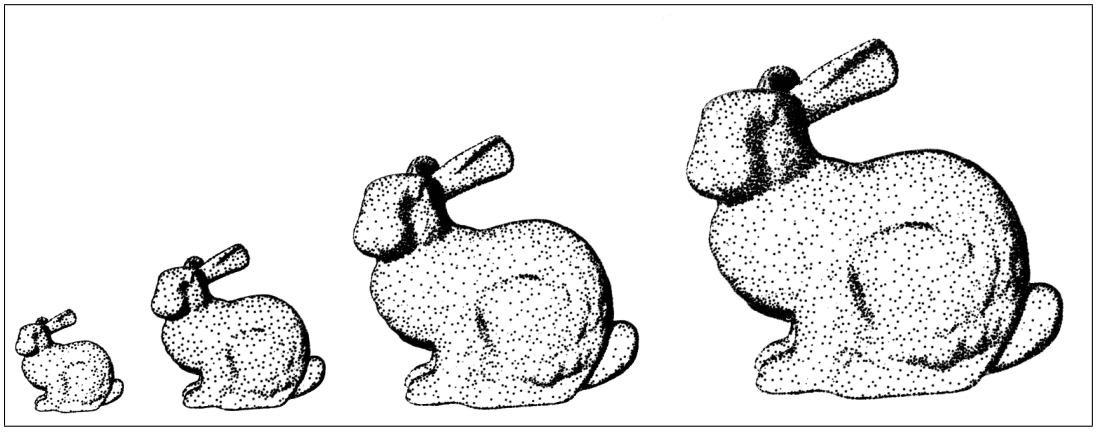

Figure 5.9: This sequence shows how stipple density increases to fill-in shaded areas as the bunny model increases in size (original sizes: 201x195, 279x261, 476x452 and 608x579).

Model	Points Generated	Memory Required	Setup Time	Primary. Relax.	Patch Relax.	Total Time
Bunny	69,451	48 MB	22s	52 s	1m 03s	2m 15s
PiggyBnkSml	86,040	59 MB	18s	1m 6s	2m 16s	3m 40s
Horse	96,966	66 MB	28s	52 s	2m 23s	3m 43s
PiggyBnkLrg	172,078	110 MB	41s	1m 44s	4m 32s	6m 57s
Kachel	400,000	247 MB	2m 8s	8m 45s	11m 40s	22m 13s
Dragon	870,877	51 MB	5m 50s	35m 10s	34m	1h 15m

Table 5.1: Statistics for creating point hierarchies by patch-based point relaxation.

5.6.3 Memory Requirements

An advantage of the technique presented in this Chapter with respect to the technique presented in Chapter 4, is that it requires less memory resources, which makes it more easy to handle large models. Table 5.2 illustrates the resolutions of the models and the number of points that were actually rendered for the total model at each resolution for the dragons in Figure 5.14. This number refers to the total of points used to render the model, not the total number of visible stipples. The time required to render each of these images was approximately one minute when rendering online and 3 minutes when rendering offline.

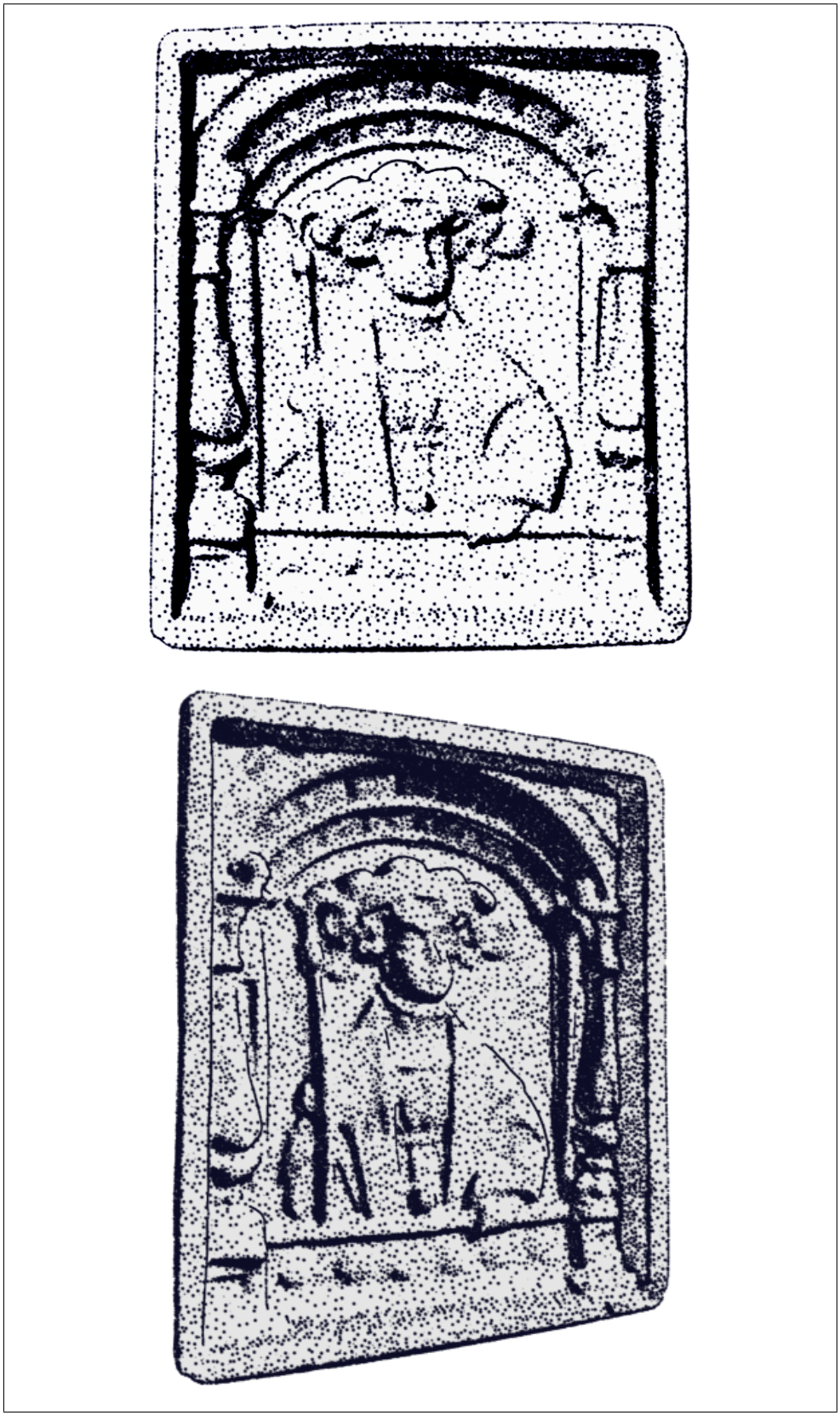

Figure 5.10: Stippled renditions of the mosaic model (original sizes: 441x502 & 409x541).

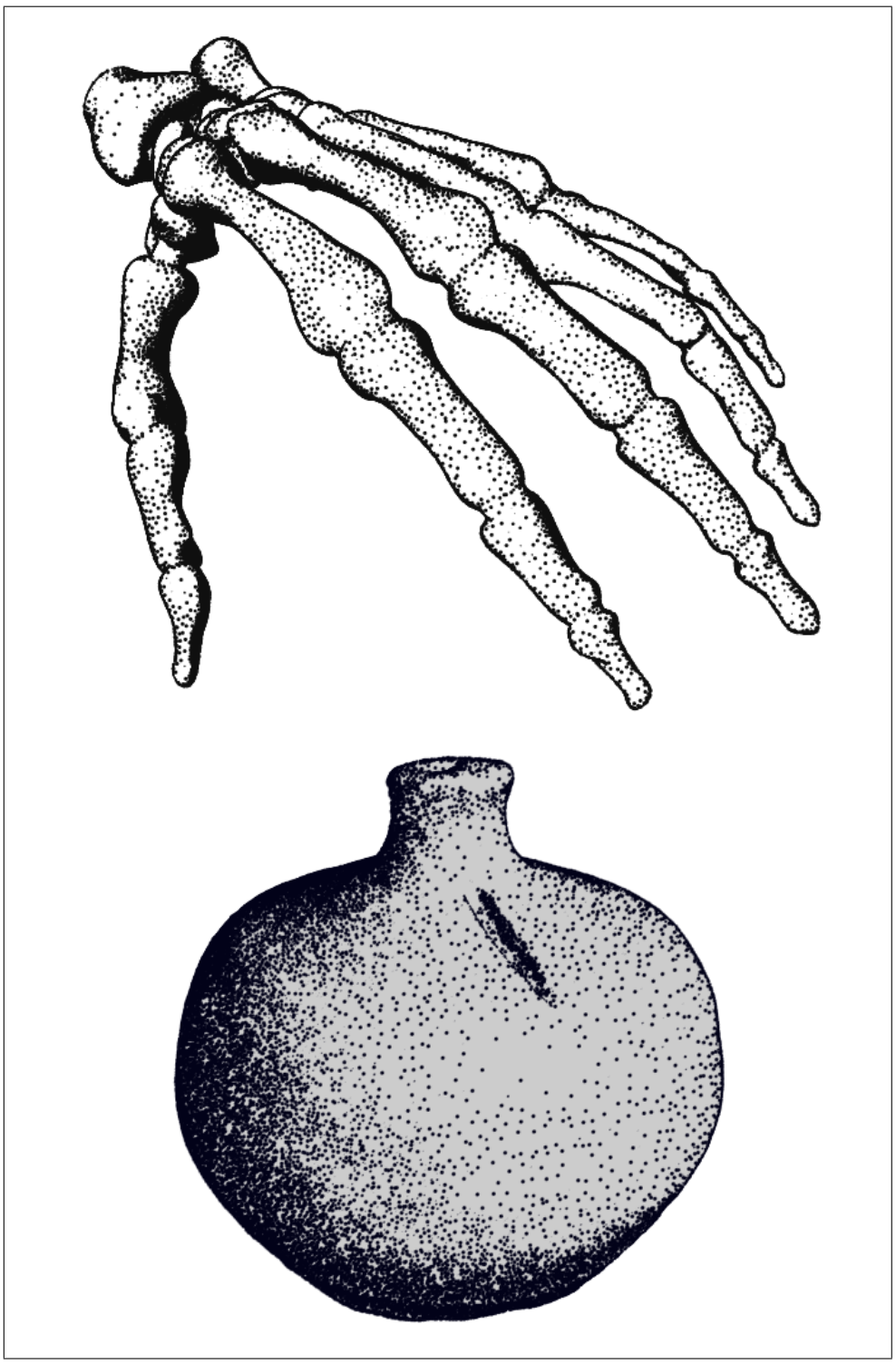

Figure 5.11: Stippled renditions of the hand-bones model trying to grab the medieval piggy-bank (original sizes: 622x559 & 409x541).

Resolution	Rendered Points
320x240	12,753
640x480	61,353
1000x750	123,737
1280x960	165,963

Table 5.2: Image resolution and total number of rendered points for the Dragon model shown in Figure 5.14.

5.7 Simplification and Subdivision Versus Token-Based Relaxation

In this section we present a comparison between our two approaches for creating point hierarchies, we compare threes aspects: hierarchy generation times, memory requirements and visual aspects.

5.7.1 Generation Times

A comparison of the system performance between our two point generation techniques can be done by taking a look at Table 4.2 and Table 5.1. All the models reported in Table 5.1 have also been tested in Table 4.2. When we compare the total time required by mesh subdivision and relaxation with the time required for patch relaxation it becomes clear that for small models, there is little difference in point hierarchy generation time. However, this difference becomes more accentuated as the size of the model increases. If we consider the total point generation time, we observe that the time required for patch relaxation is 35 to 100 per cent more than the time required for mesh simplification and subdivision. However, the number of points generated differs among both approaches: in the case of mesh simplification and subdivision, the generated point count is pretty much related to the vertex count, and is thus the vertex count (when no subdivision is performed). On the other hand, the number of points generated using patch relaxation depends on the number of polygons in the input model. Thus, the latter point hierarchies contain roughly two times more points than point hierarchies obtained by mesh simplification, which compensates the extra amount of time required by patch relaxation to generate point hierarchies. Based on the complexity of the model, the user can choose either technique, for very large models, the point-relaxation algorithm offers a memory-efficient solution, while for simple to large models, mesh simplification and subdivision is a better choice.

5.7.2 Memory Requirements

An advantage of the patch-based relaxation technique with respect to mesh subdivision and simplification is the reduction in the amount of memory required to create the hierarchies. This reduction occurs because we require less data structures to perform the process of relaxation than we do to perform mesh simplification and subdivision. This reduction in memory resources is important, since it allows us to create point hierarchies for larger models, like the Dragon (see Figure 5.14), which has roughly 800,000 polygons, for which we required approximately 515Mb of storage in main memory, while for mesh simplification and subdivision we require a bit less than 1 Gigabyte of memory, but then memory paging becomes a problem, since current SGI platforms have approximately 1 Gigabyte of Memory, and the system starts becoming unable to handle the memory requests.

5.7.3 Visual Comparison

The most visible difference in the point distributions obtained with both approaches is noticed when models are shown at a close distance (see Figure 5.15). In these situations, the points at the lowest levels of the hierarchy are visible, and the point distributions and patterns at this level become visible. In Figure 5.15 we have close-ups of the bunny model rendered using both techniques. For purposes of illustration, we have deactivated the stippling effect, such that all available points are visible. The model on the left has 34,834 stipple points (no subdivision was perfomed) and the model on the right has 69,459 points, which corresponds to the number of faces of the bunny model. The best point distribution in this case is obtained by point relaxation, since random point distribution is noisy. It is interesting to notice that using point relaxation other patterns appear which are similar to point relaxation techniques which operate in 2D (see Chapter 1) in the work of Deussen et al. [Deus00], and in the work of Secord [Seco02a].

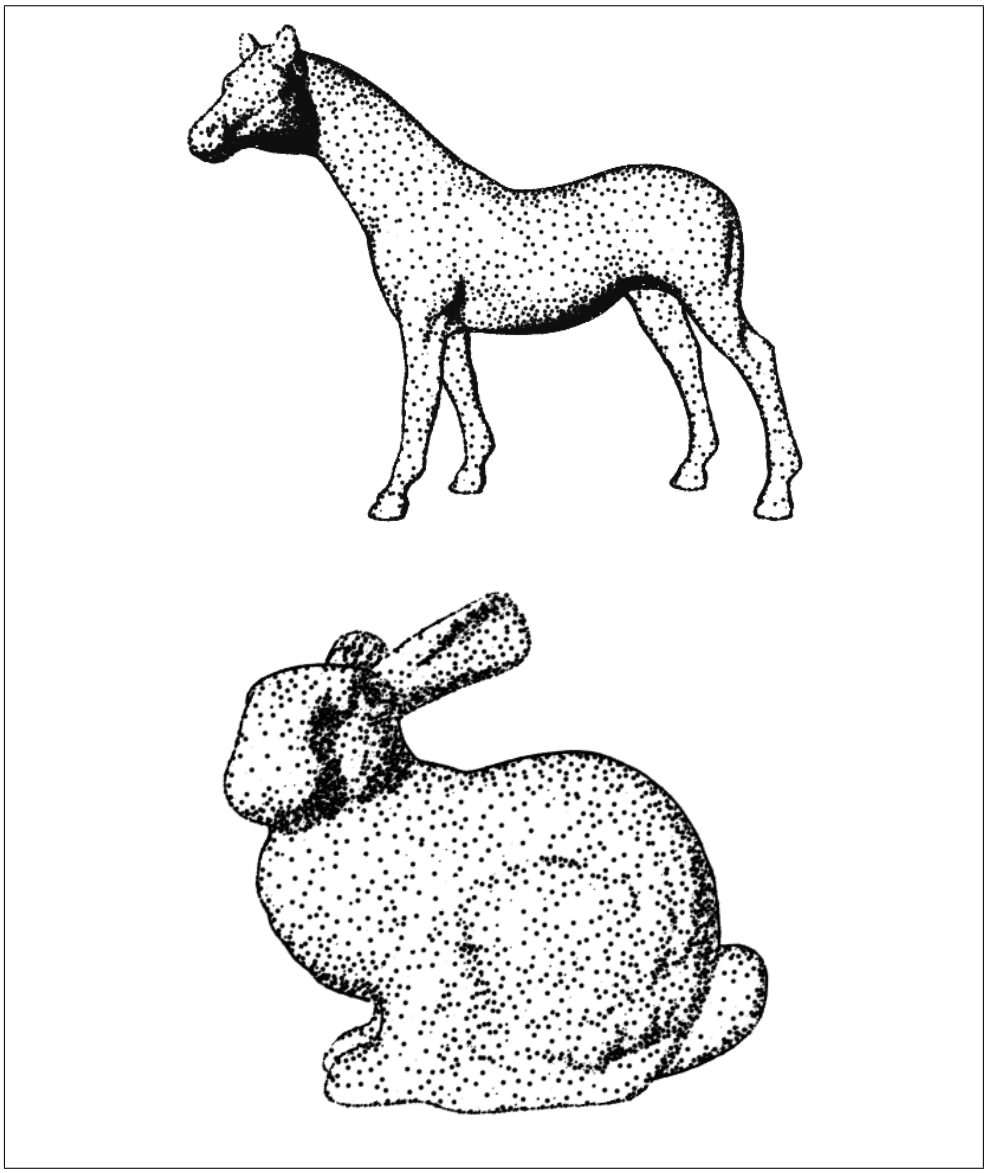

Figure 5.12: Stippled renditions of the horse and bunny models (original sizes: 553x450 & 451x428).

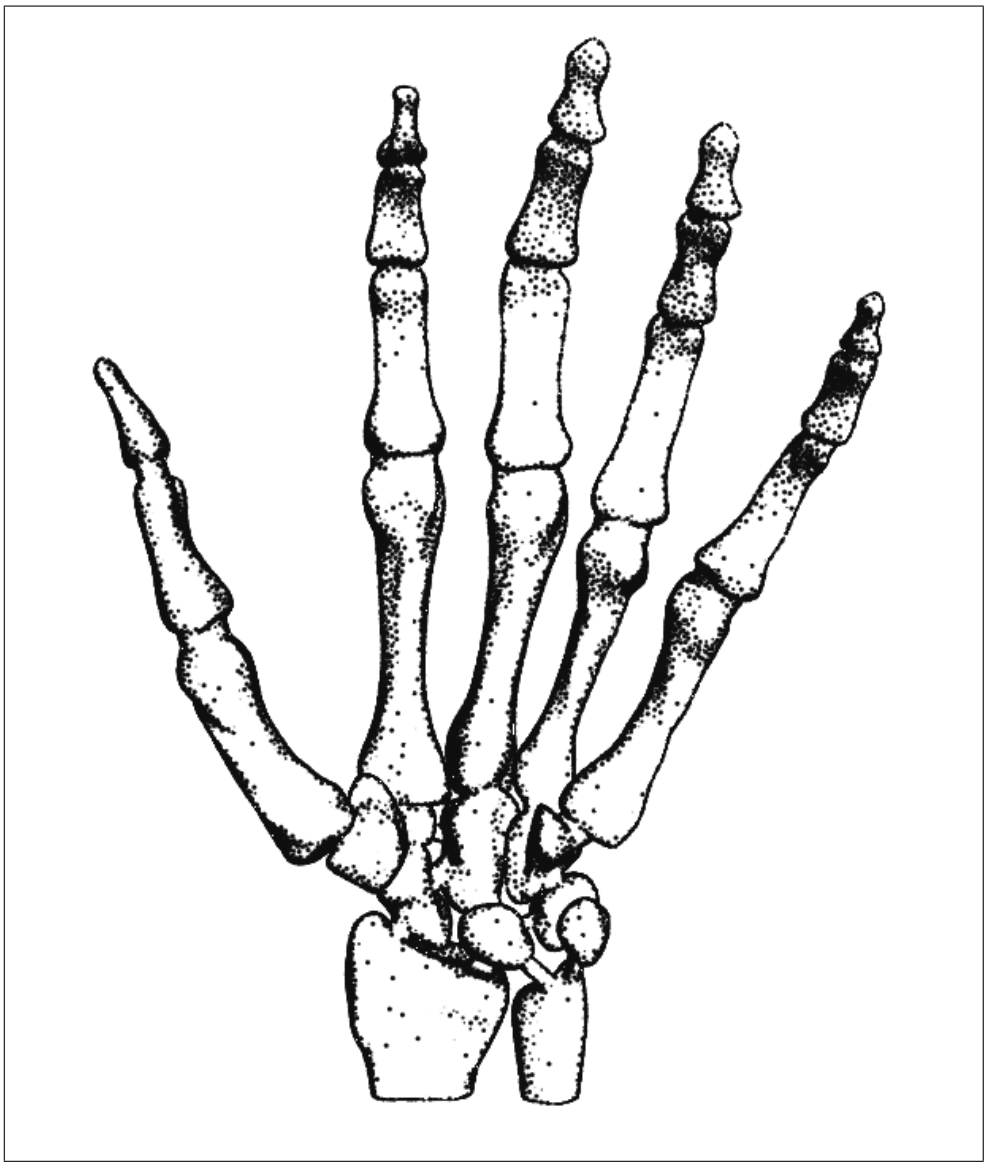

Figure 5.13: Stippled rendition of the hand bones model (original size: 551x712).

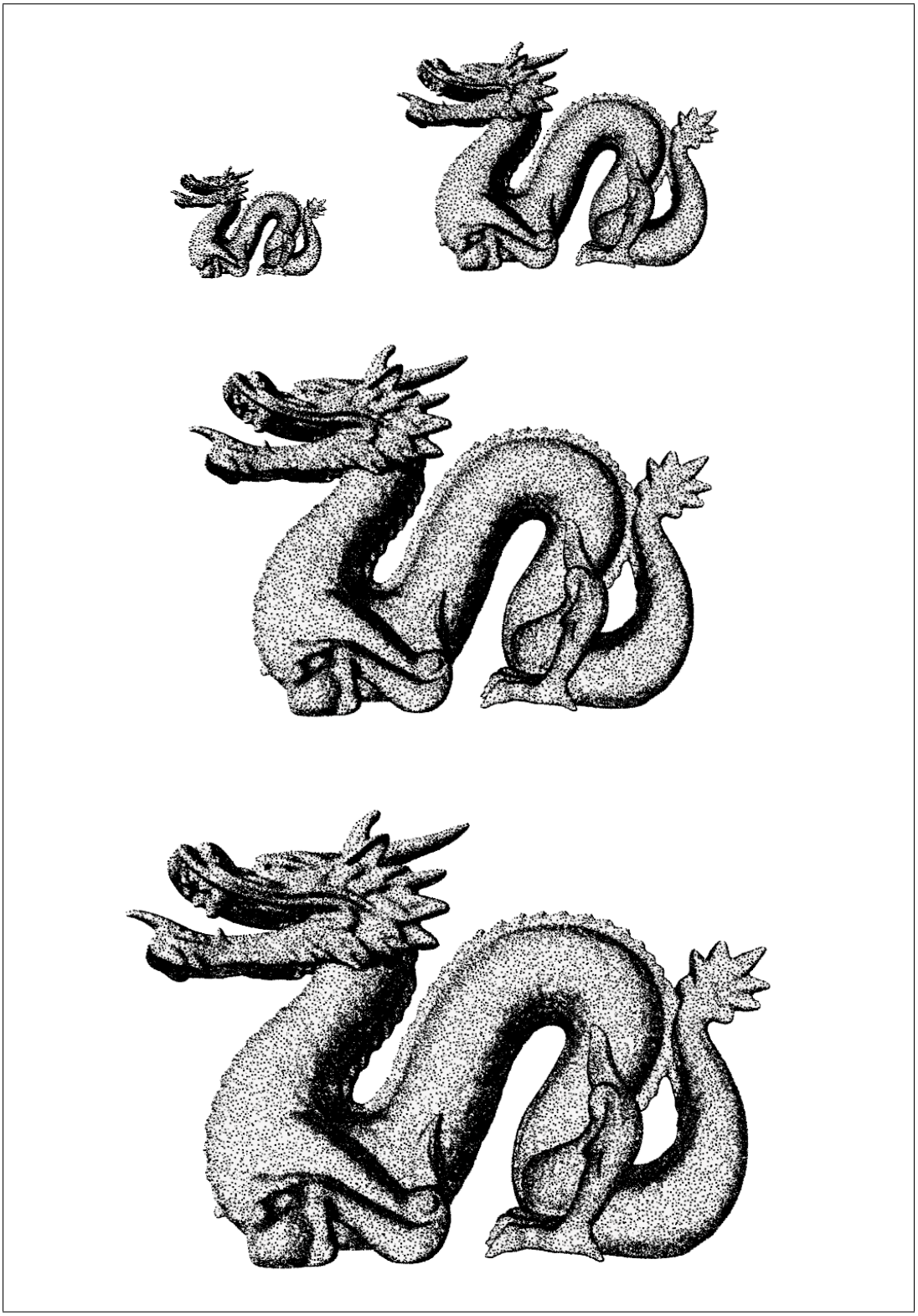

Figure 5.14: Dragon model rendered at several scales (for original sizes see Table 5.2).

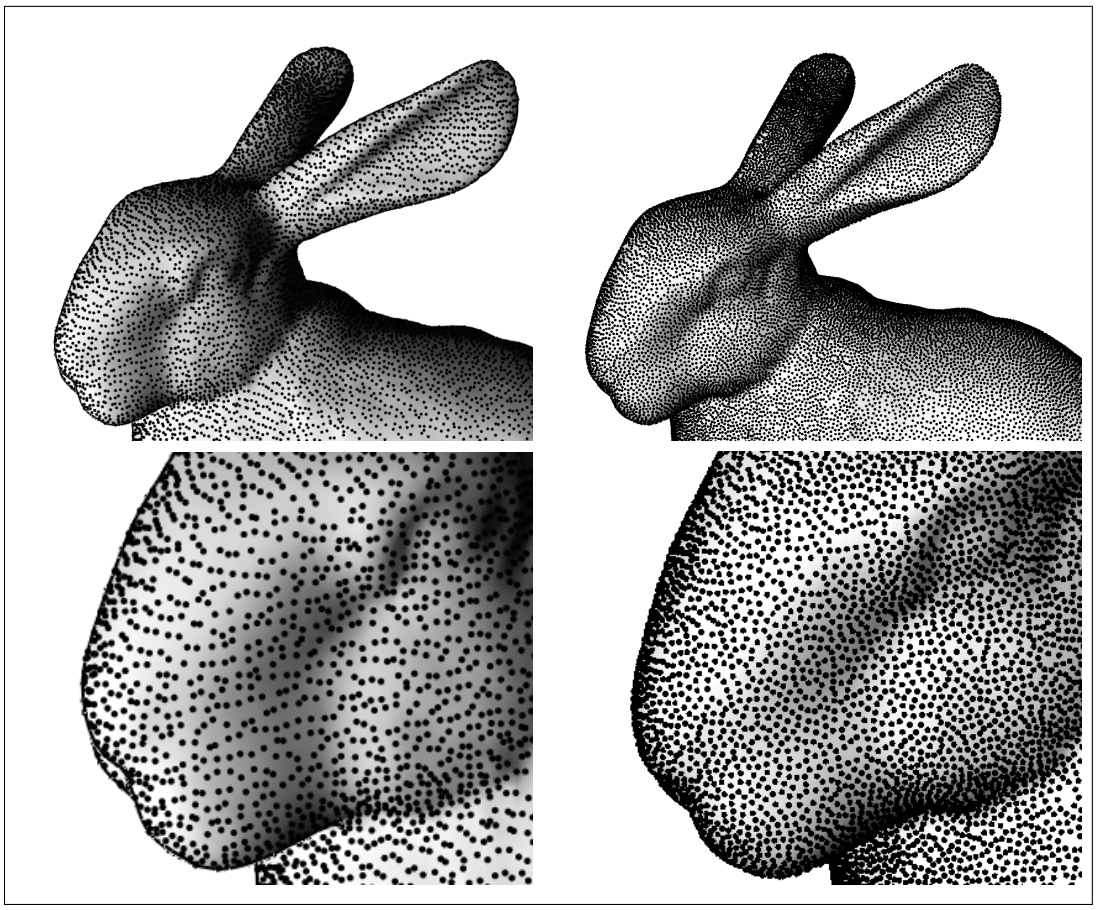

Figure 5.15: Comparison between point hierarchies by mesh simplification and subdivision and token relaxation. On the left, we show the points at the lowest level of detail for the first technique. On the right, we show the points at the same level for the second technique, with six initial relaxation steps. The bottom row shows a detail of the point distribution in the region of the bunny's eye.

6 Particle Distribution of Deformable Models

Most of the non-photorealistic techniques which have been applied to models in 3D space are applied to static models, but relatively little work has been done to apply non-photorealistic effects to animated models [Kaln02]. The challenge lies again in scaling and in defining how non-photorealistic particles or strokes should adapt to changes in the shape of a polygonal mesh.

In this Chapter we describe solutions we have developed to distribute particles on the surface of deformable models. First, we describe a solution for the problem of controlling the density of particles for deformable models, and we describe why this solution is computationally prohibitive. In the second section, we describe an alternative solution to the problem of particle distribution on the surface of the models, which offers a good compromise between stippling behavior under deformations and computational cost.

In this discussion, we assume as input a set of polygonal meshes which have the same topology, but different geometry. This convention includes several animation formats. The most simple one is to have one polygonal mesh and two sets of vertices, the source vertex positions and the target vertex positions. In this case we only need to interpolate between both positions to obtain an animation. Another convention is to have an input mesh and a vector field which indicates the directions of the vertices at each time step. Most animation software, however, make use of a skeleton which is attached to the input model and which is distorted as a simplified version of the complete model. Depending on the control variables, the skin, i.e., the input model, will be deformed as a function of the skeleton.

Animations can also be done using mesh morphing. In mesh morphing, the topology of the mesh is not necessarily preserved [Alex01]. This introduces a challenge, because the correspondence between faces of the input and the target model might

be lost. Hence, if a morphing system is used to produce the animation, an important requirement is that a face correspondence is defined for the morphing. Once a face correspondence exists, we can map particles on the surface of the input model using barycentric coordinates.

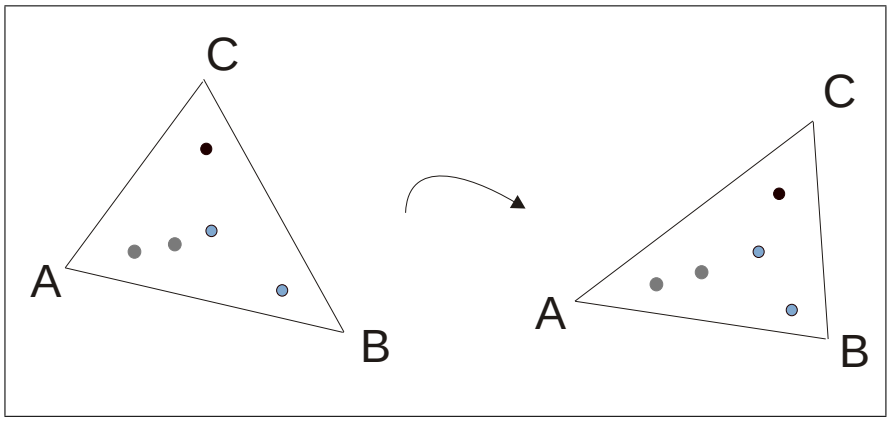

Figure 6.1: The points on the surface of the triangle are defined using barycentric coordinates, that is, they are defined relative to the vertices of the triangle. As a result, the points on the surface move along with the triangle as the position of its vertices changes.

To attach the particles to the surface of the mesh during the animation, we redefine the positions of the points in the hierarchy using barycentric coordinates (see Figure 6.1). In the barycentric coordinate system the position of the points contained in the plane of a triangle are defined with respect to the vertices of the triangle. Using this coordinate system we can displace the vertices of the triangular mesh in 3D space, and then recompute the positions of the particles as a function of these vertices.

Since normally there are several particles per face, each point in the particle set contains an index to the face of the model where the point lies (the host face for that particle) and the barycentric coordinates of the point within that face. At each step in the animation, the point coordinates are recomputed using the barycentric coordinates and the vertex positions of the host face. The effect achieved is that particles behave like an elastic texture attached to the model's surface. If we did not make use of barycentric coordinates, the model geometry would be changing, but the particles would stay in they same place as before an would slowly be detached from the surface of the model.

6.1 A Theoretical Model for Frame-Coherent Point Distribution of Deformable Models

In this section we present an approach to provide view-dependent frame-coherent stippling on the surface of a model under extreme deformations. The approach presented here has been implemented in a pilot program to test the feasibility of our approach. This approach is computationally expensive, since it assumes the use of Voronoi diagrams on the surface of a 3D model which contains a large number of points.

Recall from Chapter 1 that a Voronoi diagram is a space subdivision scheme where a surface which contains a number of points or centers is subdivided in cells called Voronoi regions or Voronoi cells. Each cell contains a central point, and the cell contains the set of all points in the surface which are closest to that central point. Thus, the boundaries of the cell are determined by those points which are at an equal distance from two or more central points. Computing the Voronoi regions on a 2D surface is a computationally expensive task: there are a number of algorithms to compute the Voronoi diagram [New 01, Aure91, Aure00], but the plane sweeping algorithm is, according to DeBerg et al. [Berg00], an optimal algorithm for the 2D case, with complexity $O(N\log(N))$ where N is the number of points. Algorithms for computing 3D Voronoi diagrams are discussed in more detail later in this Chapter.

Initially, we distribute a set of points over the surface of a 3D model. Each point represents a potential stipple and is the center of a Voronoi region. As the model is stretched and compressed, Voronoi regions are subdivided and blended together in a view-dependent way: Blending of cells occurs when the cells are squeezed together, and subdivision occurs when the cells are stretched along a given direction. Our goal is to keep the particle density constant as the model is being distorted under these conditions. To simplify our discussion, we assume that we are handling a front-facing square, which is subject to distortion. To maintain a constant particle distribution, we need to insert or remove particles from the surface of the model, depending on the nature of the distortion: when cells are squeezed together, points are removed and the Voronoi diagram is updated. Conversely, when cells are stretched, new points (cell centers) are generated and the Voronoi diagram is updated accordingly.

6.1.1 Stretching Surfaces

We first illustrate the case where a cell is subdivided due to stretching, because it is the most intuitive case. Figure 6.2 left, shows a model where a surface is split in a regular grid of Voronoi regions. In the middle, we observe the initial surface stretched along the X-axis. On the right, we observe two diagrams. The diagram on the top shows an scheme where the cells in the middle are divided in half, each giving origin to a new

cell. The center of each cell is displaced to a location close to the geometrical center of the newly generated cell. The diagram on the bottom shows another subdivision scheme. In this scheme, new cells are generated between the boundaries of existing cells, exactly at the positions which are farthest from existing points. As a new cell is generated, a new Voronoi diagram is computed based on the new point distribution.

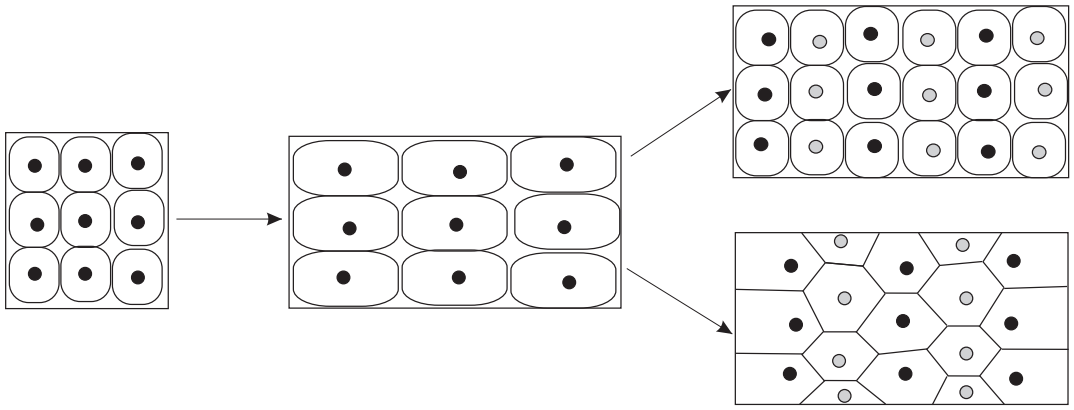

Figure 6.2: Point distribution for stretching surfaces. In the middle, the input surface shown on the left has been stretched without adapting point density. On the top right, the original cells are divided to give origin to new cells. On the bottom right, new cells are generated in the areas between existing cells. Newly generated cells have their point centers in a gray tone.

By comparing both approaches, we prefer the approach in the bottom, because it enforces frame-coherence: In the first approach, cells are subdivided and their centers have to be relocated. In the second approach, existing point centers remain in their previous locations and new centers appear between existing ones, which enable frame coherence. In addition the approach illustrated in the bottom eliminates the formation of linear patterns.

Figure 6.2 illustrates stretching only on the X-axis, but stretching can occur in any direction, with the corresponding distortion of the Voronoi diagram. Two rules can be used to split a cell based on its shape when projected on the screen. If the cell elongates significantly in any direction as a result of the deformation, the first rule is to split the cell by its half with a cut perpendicular to the direction of elongation (Figure 6.2, top right). The second rule indicates that a new cell is generated along the axis of elongation between existing cells, as far as possible from existing cell centers (Figure 6.2, bottom right). Both rules ensure that the spacing of the stipples on the screen plane is kept regular, regardless of the direction of deformation.

6.1.2 Contracting Surfaces

As can be seen from our previous discussion, generating new points for filling in existing cells can be carried out in an intuitive way. However, the opposite task of removing particles from a contracting surface is less intuitive than the task of expanding. The question here is, which of the existing points can be removed from the structure, or in other words, what is a good order in which to remove points. In Figure 6.3 we can notice that without a specific hierarchy among existing particles, little can be done to remove cells in a consistent way. To deal with this issue, we start by hypothesizing that some sort of alternation is required to select points from the existing set. We also believe that this selection should ideally produce the inverse result of the process of point generation illustrated in Figure 6.2.

We have two scenarios: In the first, we have a point distribution which is the result of a point generation process. In this case, we can assign the new particles an attribute or an identification field which signals that the particle is among the first ones to be removed in case of a compression in the X-axis. This attribute can be a number or a floating point threshold value. In the second scenario, the original model is the input and needs to be compressed on the X-axis without any previous information and without the presence of a hierarchy.

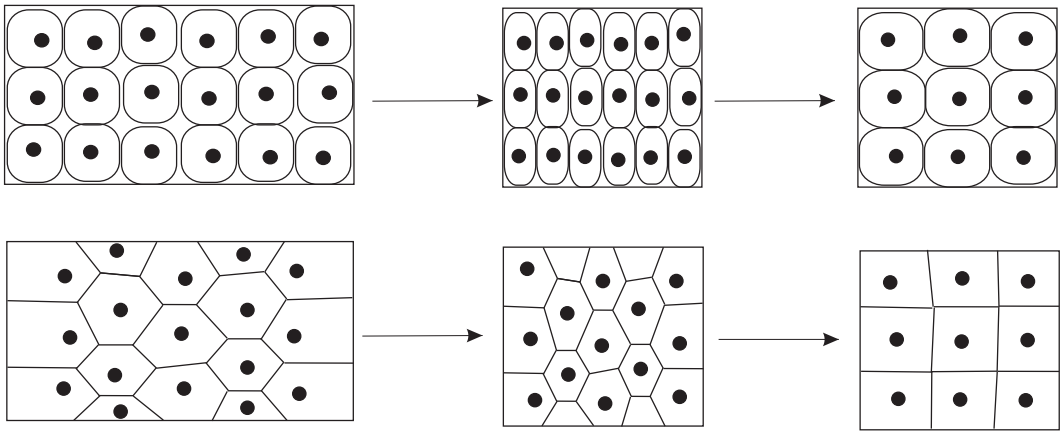

Figure 6.3: The compressing process should find a way to remove points to achieve an even point distribution, the inverse to what is done under stretching.

In this case, we need to make use of a strategy that puts cells together in an ordered way. To deal with these cases, we have designed the following algorithm:

```
Input: Graph(Cells, cellNeigbhorEdges) and startCell ∈ Cells
insert(queue,startCell)
while size(queue)>0 {
   cell = queue.pop()
   test RemovalFromGraph(cell)
   if RemovedFromGraph(cell) then update(Graph)
   explored(cell) = TRUE;
   for(neigbhorCell,cell) ∈ cellNeigbhorEdges {
      if explored(cell) = FALSE then
         insert(queue,neighborCell)
      }
   }
```

Algorithm 6.1: Algorithm for point removal during contraction.

This algorithm is a variant of the Breadth-First-Search Algorithm (BFS) and it has complexity $O(cellNeigbhorEdges+removedCells)$ in the graph, since each neighbor cell is visited only once, and each removal Operation requires updating the graph if the cell is removed.

The test for removal is designed to remove cells according to the contraction. If the cell is compressed along a given direction, the cell takes an elongated shape perpendicular to that direction (as in Figure 6.3) and needs to be fused with a neighbor on the elongated side. For this, the test needs to identify the extents of the cell after the compression, such that if the cell becomes too narrow, it is removed from the graph. Updating the graph is a task which is not too complicated, since the graph only needs to be updated in the neighborhood of the removed cell, which means updating the connvectivity information for a limited number of cells (the cell neighbors).

The presented algorithm has the advantage that is it fast (it runs in linear time), but has a potential drawback: lack of frame-coherence. Lack of frame-coherence occurs if the cells in the graph are not consistently removed at each animation iteration, which gives place to a complete different point distribution at each rendered frame. The requirement that the first cell is chosen consistently could be satisfied by assigning a unique identification number to each of the cells, and then by starting the analysis with the cell with the highest number, creating an artifical priority ordering. If we need to choose among different cells, we can test the cell with the highest priority. In this case, however, we need to replace our original queue with an ordered heap, which leads us to an algorithm similar to Dijkstra's algorithm for finding the shortest paths:

```
Input: Graph(Cells, cellNeigbhorEdges) and startCell ∈ Cells
for cell in Cells {
   idNumber(cell) = count;
   }
insert(heap,startCell)
while size(heap)>0 {
   cell = heap.popMaxCell()
   test RemovalFromGraph(cell)
   if RemovedFromGraph(cell) then update(Graph)
   explored(cell) = TRUE;
   for(neigbhorCell,cell) ∈ cellNeigbhorEdges {
      if explored(cell) = FALSE then
         insert(heap,neighborCell)
      }
   }
```

Algorithm 6.2: Algorithm for ordered point removal during contraction using a heap.

According to [Mitz03], the complexity of Dijkstra's algorithm is bound by the implementation of the heap. If we implement the heap as a linked list, the complexity is $O(graphCells^2)$, if it is implemented as a binary heap, then it is $O(neighborCellEdges * \log(graphCells))$. However, since we are considering hundreds of thousands of cells which need to be updated during the animation, this would be prohibitive.

6.1.3 Retriangulation During Interactive Stretching

According to the computer geometry literature, the Delaunay triangulation is the dual of the Voronoi diagram, and it is obtained by drawing a line segment between two Voronoi vertices if their Voronoi regions have a common edge. As mentioned by Van Laerhoven [Laer03], there is a natural bijection between the two which reverses the face inclusions. This is illustrated in Figure 6.4.

A Delaunay triangulation exists when the vertices of a mesh are joined by triangles such that the circles which pass through the vertices of each triangle do not contain any of the other vertices available in the mesh. In addition, these vertices connect the cell centers of a Voronoi diagram . According to Attali and Boissonnat [Atta02], the task of computing such a triangulation in 3D can be of complexity $O(n^2)$, and the number of tethraedra (polygon cells) can be quadratic. Hence, having to compute the triangulation for a set of hundreds of thousands of points at each frame of an animation is prohibitive.

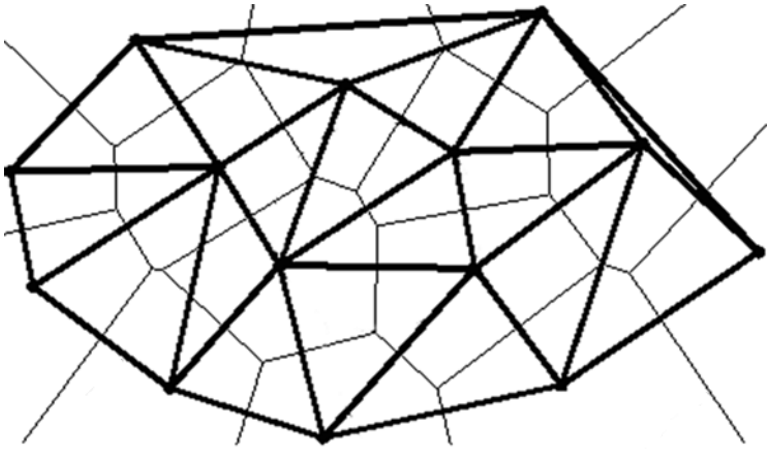

Figure 6.4: Sample Delaunay triangulation (drawn with thick lines) and its dual, the Voronoi diagram (drawn with thin lines) [Fili96].

We produced a pilot program to test the possibility of dynamically computing an approximation of the Delaunay triangulation of the model depending on model deformation. In this program, we take the input mesh and its connectivity information. The user is given the possibility to stretch or compress the input model on the X- or Y-axis in model space. As the model is compressed or stretched, the connectivity information is updated. Every edge which is bound to two adjacent faces is tested against the edge joining the vertices of the faces it binds which are not connected to the edge. If this edge is shorter than the edge being tested, the edges are swapped, i.e. the edge which joins the vertices not connected to the input edge becomes the edge which joins the two faces, and the edge being tested is removed. This is done for every edge in the input mesh.

Figure 6.5 shows two snapshots of the interactive remeshing program. On the left, we show the original model. On the right, we show the effect of two distortions: the distortion on the top illustrates an expansion along the X-axis and a compression along the Y-axis. The distortion on the bottom illustrates an expansion along the Y-axis. In both cases it is possible to reorganize the polygonal mesh to approximate a Delaunay triangulation.

However, the retriangulation could only be successfully accomplished on flat parts of the model's surface. Whenever we tried to adjust triangulation for an edge connecting faces that were non-planar, an artifact was introduced in the model. Figure 6.6 illustrates the artifacts that are introduced when a 3D mesh is retriangulated around faces which are not coplanar. Due to these disadvantages, this algorithm was not inserted in our system.

The reason we had artifacts which produced counterintutitive cracks is that we did not consider the interaction between convex and concave shapes. During interactive

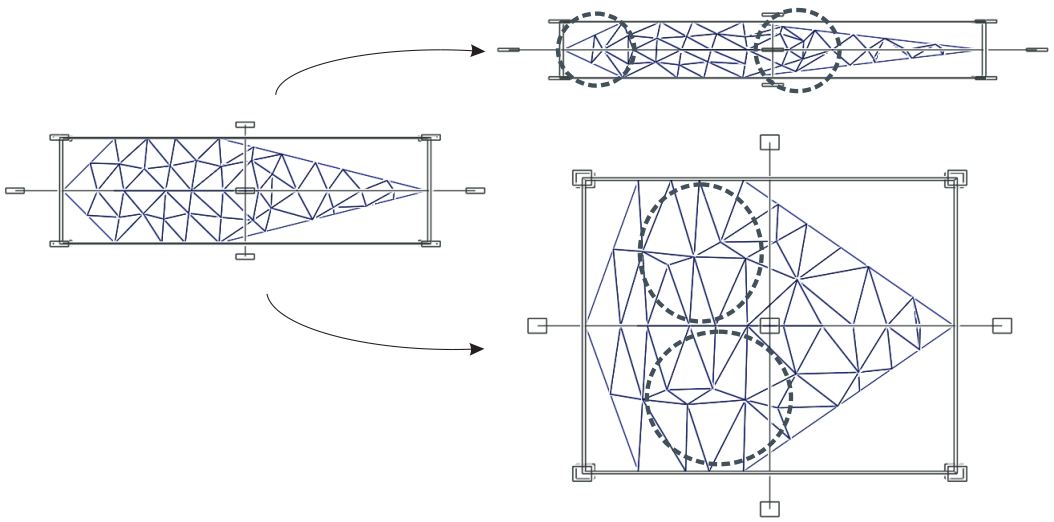

Figure 6.5: Retriangulation of a polygonal mesh (left) under interactive deformation. On the top right, we observe the input mesh expanded along the *X*-axis. On the bottom right, we observe the input mesh expanded along the *Y*-axis. The circled areas show changes in the triangulation as a result of the deformation.

deformation a surface which is mostly flat can be stretched along an axis which will make it look like a peak, and the distances which should be considered for such surfaces are not the euclidean distances, but the geodesic distances, since these are the real distances between particles on the surface of the model. Due to the complexity of this task, we decided to stop the development of this program, since a view-dependent modification of the mesh using a dynamic Delaunay retriangulation based on geodesic paths seemed too much of a complex task. Instead we looked for an alternative solution, which offered us a compromise between computational cost and our goals as to the particle distribution.

6.2 A Practical Model for Stippling Deformable Models

As we have observed before, the process of performing view-dependent distribution for animated models can be seen as a process of view-dependent retriangulation of a mesh which contains as many vertices as stipples are rendered. We have discarded this solution due to the prohibitive amount of time it requires for processing large point datasets. In the following discussion, we describe an alternative approach, animated stippling, a technique that allows us to use the point hierarchies described in Chap-

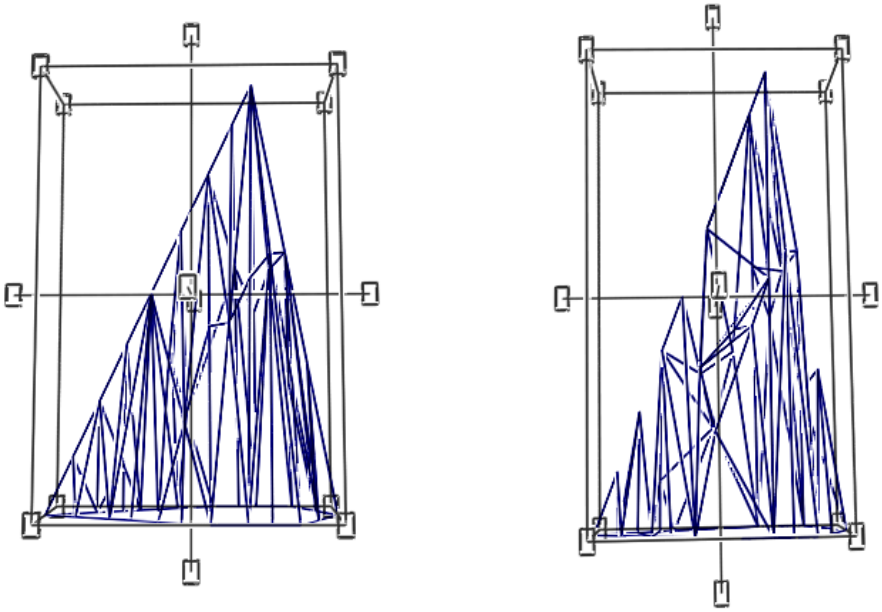

Figure 6.6: Artifacts of retriangulation on 3D models. The image on the left shows an extrusion of a 3D model without retriangulation. The image on the right shows the extruded model with some vertices retriangulated. The small peaks on both sides of the mountain are the result of the retriangulation.

ter 4 and 5 to stipple animated models using a linear amount of time and memory resources.

6.2.1 Animated Stippling Algorithms

We have found that stippling as a rendering style is well suited for producing computer animations, because the point hierarchy described in Chapter 3 can be used as an elastic texture which is attached to the surface of the model. To achieve this effect under animated stippling, we take as input the original model and the point hierarchy and use the barycentric coordinates of each point to recompute the position of the particles under deformation. Algorithm 6.3 describes the basic approach for animated stippling.

In animated stippling, we use the same rendering function as the one used for conventional and real-time rendering, where each point is assigned either a set of relevant neighbors or a relevance value which will enable the renderer to determine whether a point should be visible or not. Animations produced using Algorithm 6.3 already give the effect of stipples being attached to the surface of the model as an elastic texture.

```
input: modelGraph(Vertices,modelNormals, Edges),
   AnimationFrames, pointHierarchy(points,baryCoords)
for frame ∈ AnimationFrames:
   updateVertices(frame, modelGraph,Vertices)
   updateNormals(modelGraph,modelNormals)
   for point ∈ pointHierarchy {
      pointCoords = getCoords(baryCoords,Vertices)
      pointNormal = getNormal(baryCoords,modelNormals)
   }
render(pointHierarchy)
```

Algorithm 6.3: Use of barycentric coordinates to produce animated stippling.

However, the point density in these models is left untouched during deformations. If we were to perform extreme deformations on the model, the particle density will not adapt to the deformations, since it directly depends on the nature of the distortion: if the model is compressed, the point density increases, and the renditions become darker; conversely, if the model is stretched, less stipples appear, and the renditions become relatively lighter shaded.

The reason for this is that as the model is stretched or compressed, the relevance values for each point in the hierarchy are not scaled with the distortion. If this scaling does not take place, the relative radiae of the stipples with respect to the distortion change, such that for a given surface and without any change in the rendering variables (specially the target shading tone) more stipples appear on the same surface boundaries when the model is contracted and less stipples appear when the model is contracted. To counter this effect, we need to modify or adapt the point hierarchy such that the point density corresponds to the new shape of the model. In particular, we need to modify the relevance values of each point according to the deformations.

The solution we have developed to compensate for the changes in point density due to deformation is to use the set of relevant neighbors of each point to dynamically re-scale its relevance value. Recall from Chapter 4, Section 5 that the relevance value is obtained as the average of the distances to the relevant neighbors of a particular point. Under our approach, we can go back to this set of relevant neighbors and recompute the relevance value out of this set of relevance neighbors. The only consideration we need to make is to recompute the relevance values once we have updated the barycentric coordinates of the points in the mesh. This is shown in Algorithm 6.4.

By defining the set of relevant neighbors in barycentric coordinates, the position and the radius of the particles can adapt to mesh deformation to some extent. The point density varies proportionally to the distance from the particle to the original neighboring points, as illustrated in Figure 6.7. While this introduces an additional computa-

```
input: modelGraph(Vertices,modelNormals, Edges),
   AnimationFrames, pointHierarchy
for frame ∈ AnimationFrames:
   updateVertices(frame, modelGraph,Vertices)
   updateNormals(modelGraph,modelNormals)
   for point ∈ pointHierarchy {
      pointCoords = getCoords(baryCoords,Vertices)
      pointNormal = getNormal(baryCoords,modelNormals)
      pointRelevanceValue = getRV(Vertices,pointHierarchy)
   }
render(pointHierarchy)
```

Algorithm 6.4: Accounting for mesh deformation during animated stippling.

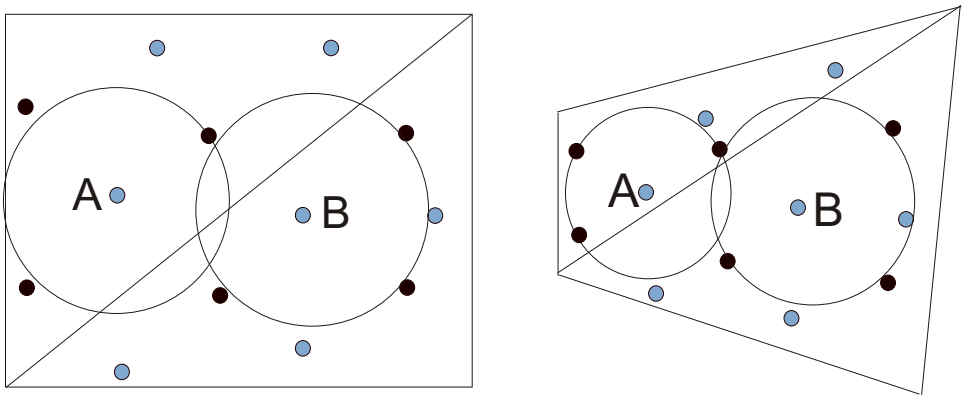

Figure 6.7: On the left, two sample points A and B are given a radius value which is the average distance to its relevant neighbors (the dark points). On the right, we observe the two points with their new average radius after the distortion has taken place. Since the distance to their neighbors has changed after the transformation, the values for the radius of the particle points also change, but in different proportion for each point.

tional effort during rendering, this step ensures that the particle hierarchy originally computed is kept consistent as deformations take place.

6.2.2 Results

We have produced several stippling animations using this technique. Videos of these animations can be observed at http://isgwww.cs.uni-magdeburg.de/~oscar/. In Figures 6.8 through 6.10 we show frames of the animation sequences produced with this technique.

In Video 8-*cocodrilo* we show an animated cartoon crocodile character who suddendly gets angry as the lighting changes and leaves him on the dark, the light source has been purposely placed behind the model, to enhance the stippling effect as the light position changes (see bottom image of Figure 6.8).

Video 9-*closing hand* shows a human hand as it opens and closes, and video 10-*thumbs-up* shows the same hand making a thumbs-up sign. For the hand animations, the deformation is greater than in the case of the crocodile. However, in both cases, the stippling still adapts nicely to the deformations and also responds adequately to changes in shading.

Video 11-*beating heart* shows our animation of the beat of a heart, which can be looped to have a continous heart-beat motion (see Figure 6.10). The animation of the heart is not physically based, it is modelled by an artist after the real heart-beat motion. Two animations were produced for the heart, one with the original model rendered opaque, and another with the lower heart shell being indicated only by the presence of stipples on its surface, which allows the viewer to take a look into the heart chambers (see Figure 7.7 in Chapter 7).

There is a still a drawback in our solutions which emerges from the simplicity of our model. In the case of extreme distortions along a specific axis, the point distribution will not be optimal. This occurs because the original point distribution is uniform, and the resulting point distribution is not. The effect is that some stripes of points will appear as the model is stretched. This occurs because the point density along the axis of distortion is different than the point density along the other axis of distortion. Fortunately, this effect is only visible under extreme cases, and was not noticeable within the set of animations we have explored.

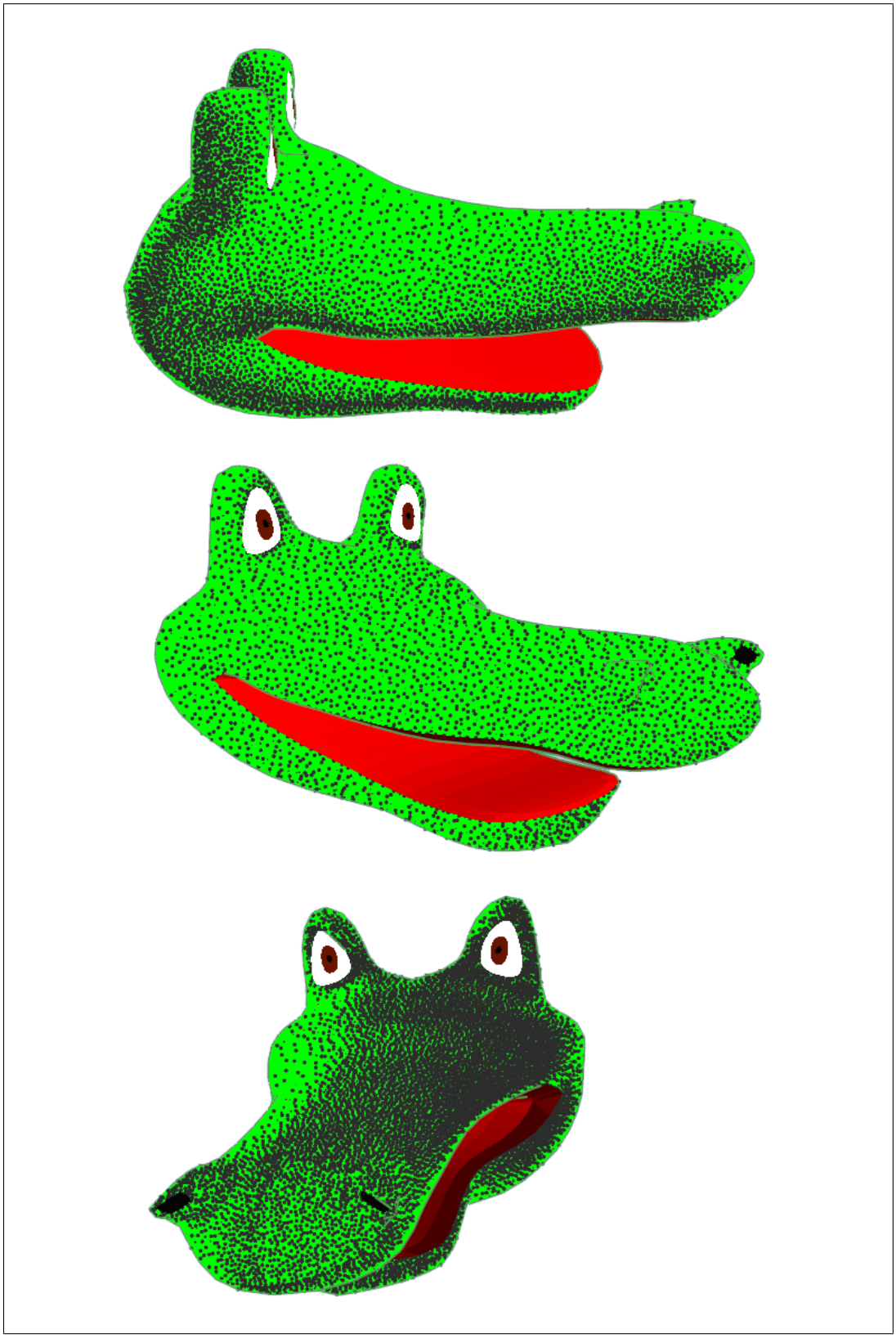

Figure 6.8: Frames from our animation "Upsetting the crocodile" (original sizes: 700x380).

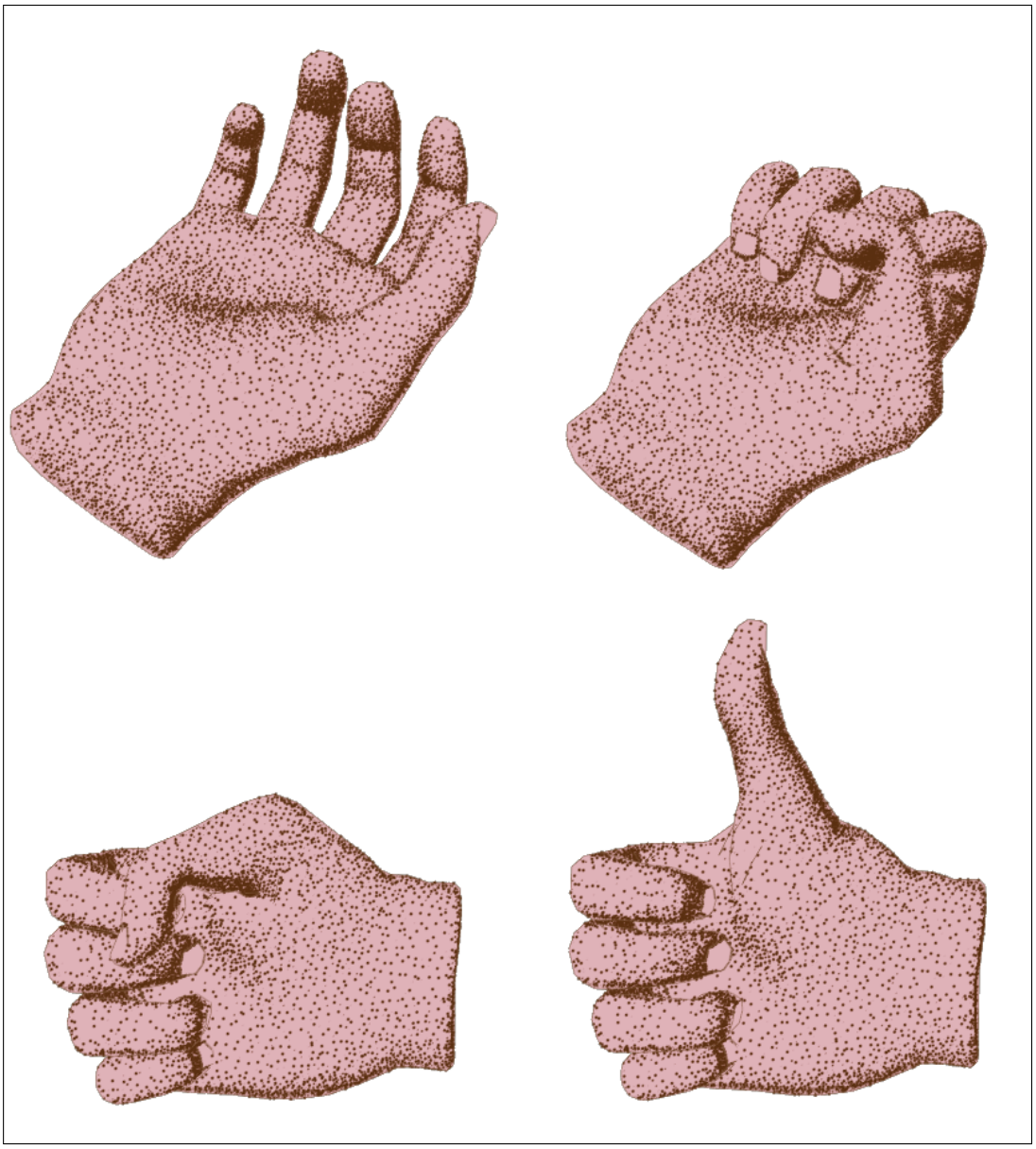

Figure 6.9: Frames from our animations "Closing hand" and "Thumbs up" (original sizes: 400x450).

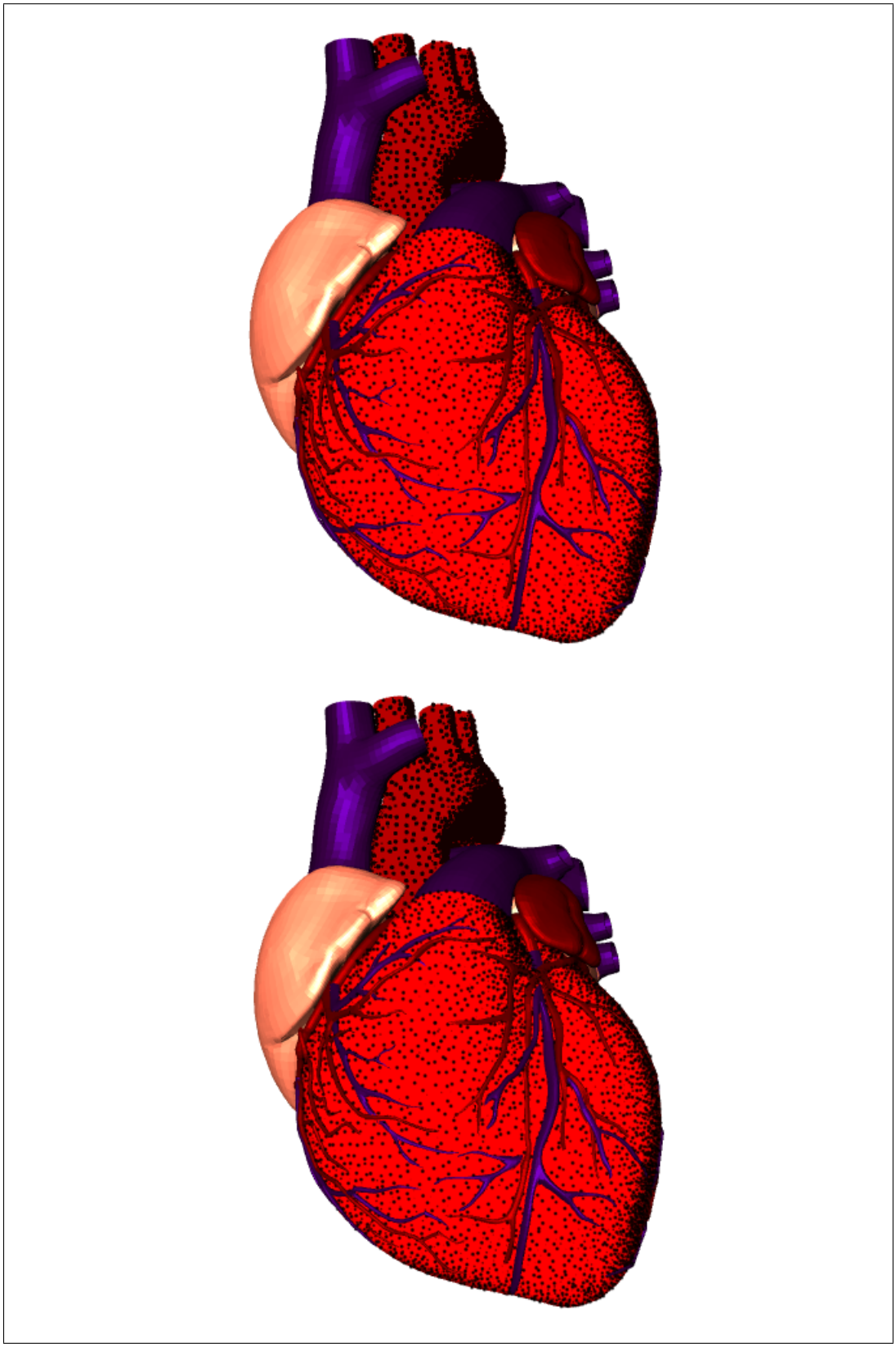

Figure 6.10: Frames from our animations of a beating heart (original sizes: 600x500).

7 Informal Assessment on the Applications of Stippling

In this Chapter we make an in-depth discussion on the limitations and applications of frame-coherent stippling. We present several results of stippling applied to different models and under different contexts, and discuss the limitations of our implementation and its advantages when applied on several models. We do not include the specific application of 3D stippling for archaeology, because we will discuss this as a special case in Chapter 8.

The discussion presented in the following stems from informally received feedback, and from our personal appreciations. It is a discussion on the possible uses and limitations of stippling from the point of view of the author, and is meant to provide insight to what can and what cannot be done with 3D stippling.

We discuss stippling under different viewpoints, and we divide our observations in the following themes: shading styles, model suitability, real-time rendering and use of transparency.

7.1 Shading Styles

The first models we tried for stippling were the Utah Teapot and the Stanford Bunny. Stippling worked quite well for the Utah Teapot. With this model, we explored the possibility of applying different shading styles with stippling. We first tried flat shading, which works quite well for models with sharp edges. Then we tried phong shading [Fole93], which works well for round-shaped models, since it reduces the effect of tessellation (see Figure 7.1).

When using particles for stippling, we assign to each particle a unique normal. To render the model using flat shading we assign the normal of each face to each stipple

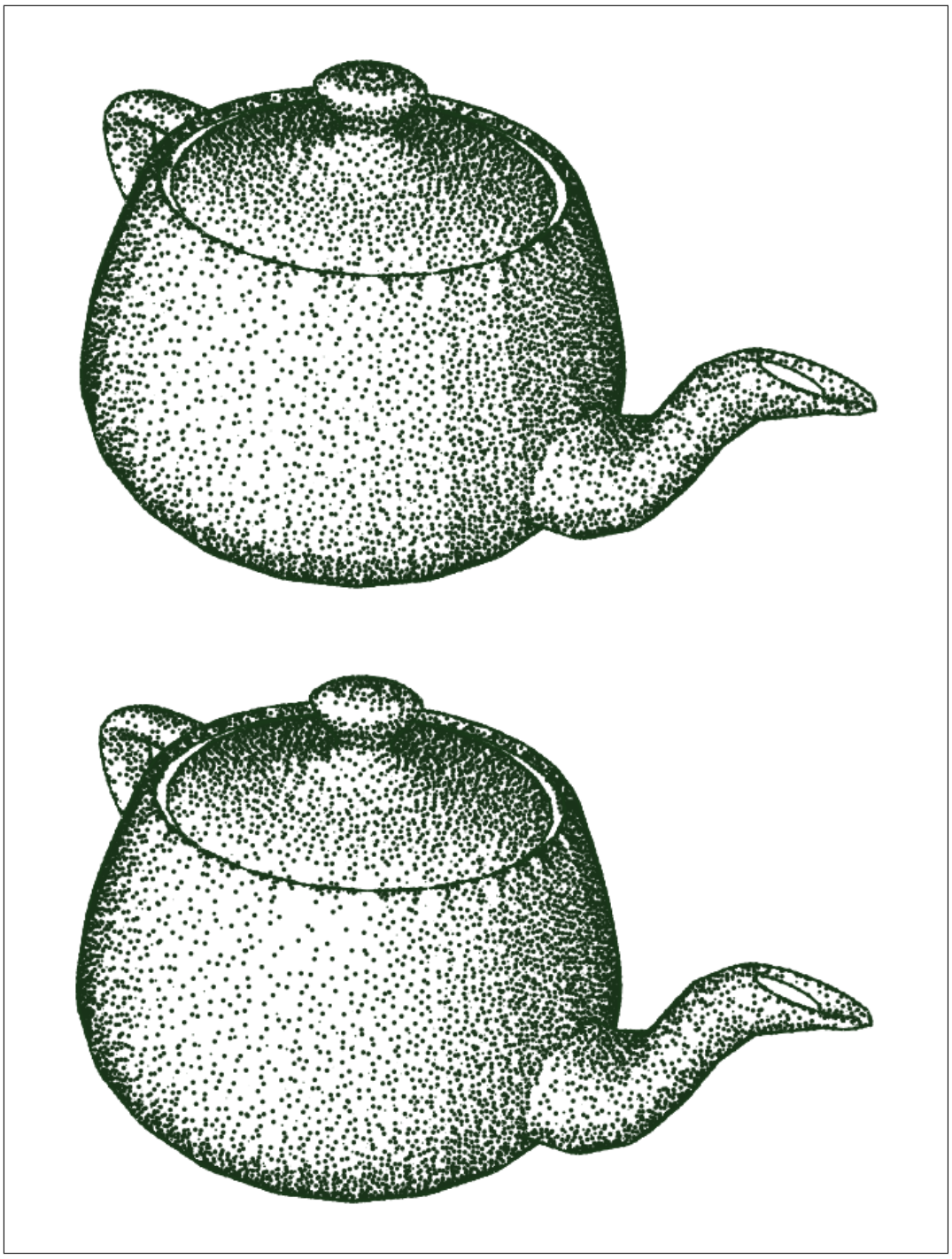

Figure 7.1: Changing shading styles. On the top, the Teapot model rendered using flat shading. On the bottom, the Teapot model rendered with phong shading (original size: 630x450).

lying on that face. For phong shading, the normal is obtained by interpolation among the normals of the faces connected to each vertex of the face where the stipple lies (which is the basis for gouraud shading [Fole93]).

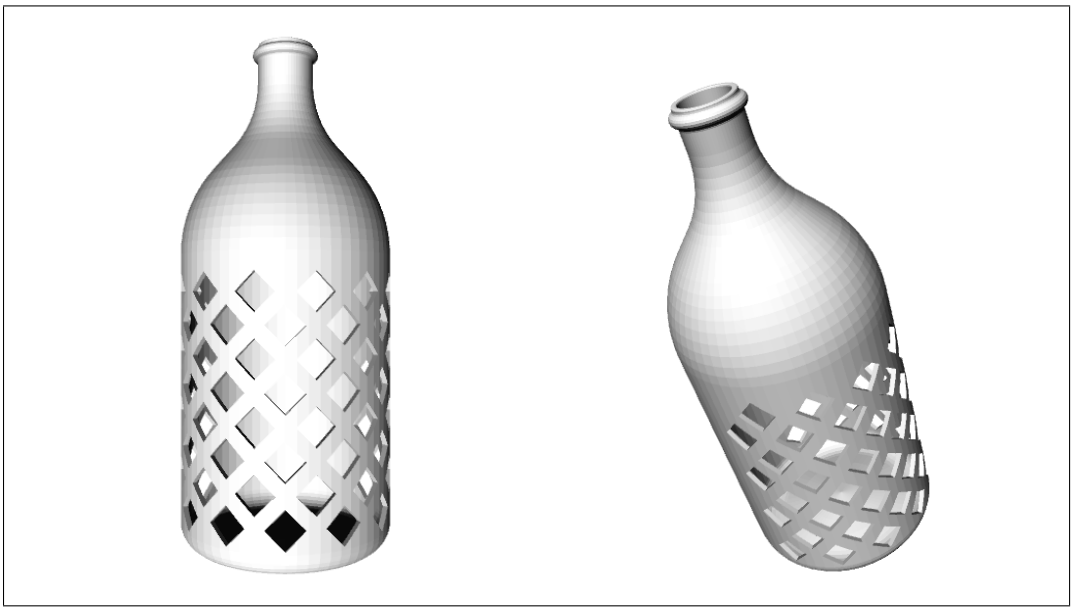

Figure 7.2: The upper part of the bottle model would be optimally rendered using gouraud or phong shading, while the bottle's grid would be better represented using flat shading.

For some models, a combination of flat and gouraud shading is ideal, since many models contain both rounded and sharp surfaces. We have such a model in Figure 7.2. The upper part of the bottle is a rounded surface and would be optimally shaded using gouraud or phong shading, while the grid on the side of the bottle needs to be shaded using flat shading, since it contains sharp edges where the illumination needs to be separated. To combine both shading styles, we need to use single normals per vertex per face. That is, each vertex at each face must have its own normals. If the vertex belongs to a sharp edge, then each vertex at each face should be oriented according to the normal of the face. If the vertex belongs to an edge which is not sharp, then the normal at the vertex should be the same for all the faces in the vertex.

7.2 Model Suitability

Not all 3D models are well suited for stippling. This is an esthetic consideration rather than a scientific one, which has come out consistently as the stippling technique has been presented to people in the field of computer graphics who have observed our work.

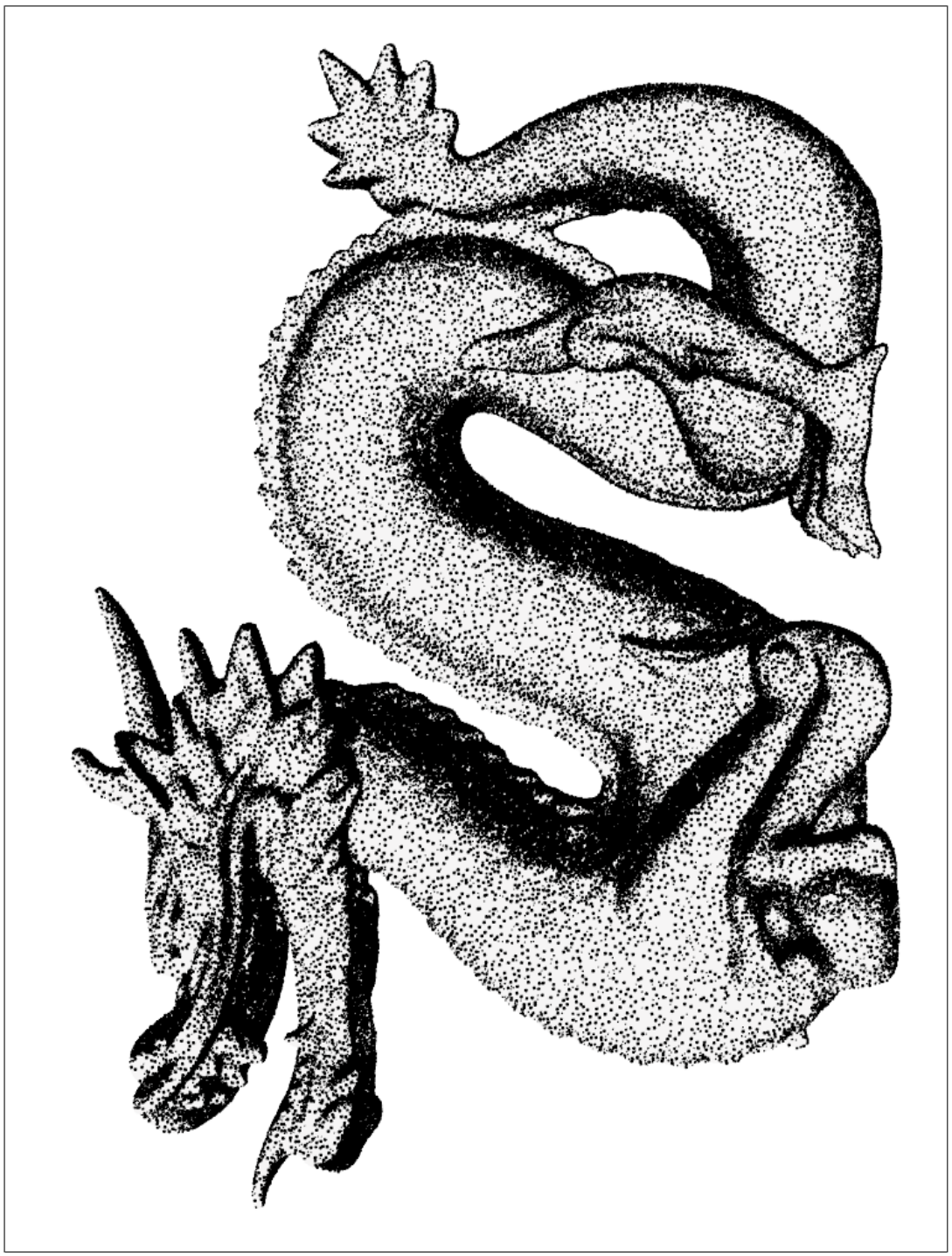

Figure 7.3: Dragon model (original size: 1000x750).

Objects which are definitely good for stippling are the ones associated with pottery or ones which have a stone-made look. For example, the teapot and objects from archaeology. A model which came out particularly well after applying our stippling technique is the model of a Chinese Dragon (Figure 7.3), which is a pretty large model publicly available from the Georgia Tech repository of large models. The horse model shown in Chapters 4 and 5 also turned out very pleasing when stippled.

Stippling enhances the aspect of this model by conveying it the aspect of being carved out of stone, which is credible also because of the nature of the model. Here we find a potential application for the entertainment industry: it is possible to have an object which initially looks like a static model made out of stone, which suddenly comes to life during the animation.

The original stippling technique is meant to produce black and white illustrations. In this thesis, however, we have taken the freedom to try colored stippling, obtaining interesting results in some cases. For example, we produced a toy-crocodile animation (Figure 7.4), where the crocodile has bright green skin, cartoon-like eyes and a red mouth. This was covered with dark green stipples, and the animation obtained was found entertaining. We observed this during a number of presentations and demos given, where people always smiled when looking at the animation. This leads us into believing that stippling has also an application as a rendering style for shading cartoon-like characters.

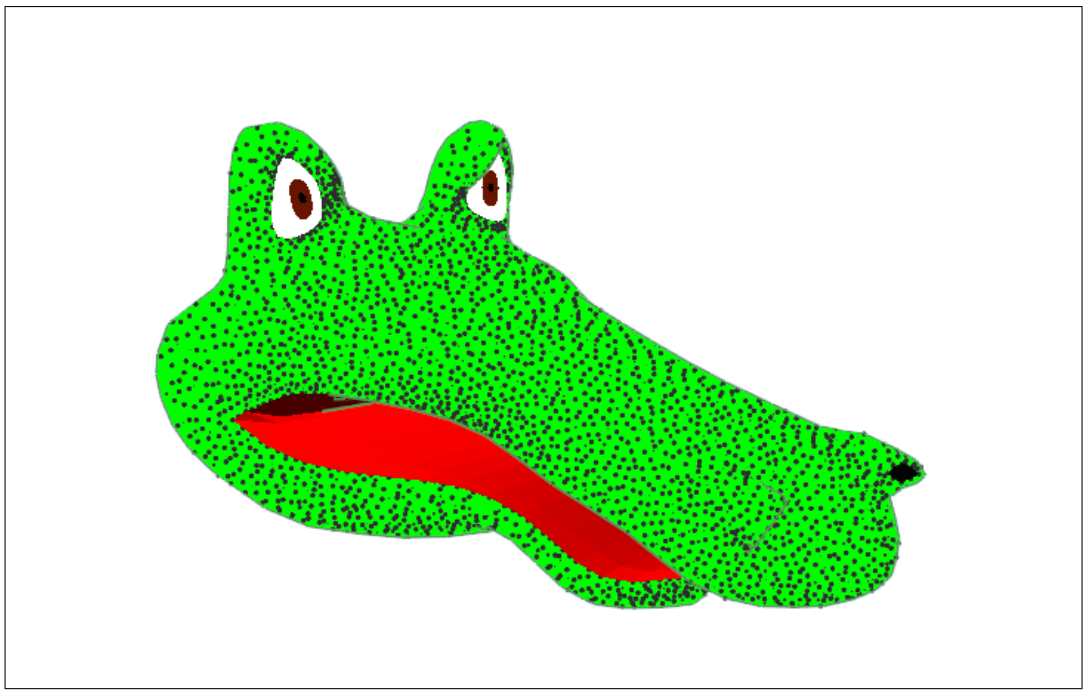

Figure 7.4: Stippling of a model in a toon style (original size: 700x450).

Some biological models, like the human brain or the human hand, convey a look which was not wellcome by all viewers. Some comments we received were that it is not normally expected to see the human brain covered with dots. On the other hand, bone models like the pelvis (Figure 7.5) and the hand-bones model turned out quite well (see Figure 5.13 in Chapter 5). In fact, bones are in most cases illustrated using the stippling style in archaeology (see Chapter 8).

Point size is a relevant factor: if points are very large (GL Point size 3.7 and larger), the model looks abrupt or coarse; on the other hand, when points are small, the model looks smooth. Figure 7.6 shows this. On the left side, the hand is shown using large stipples, and it has an unpolished look, even a look of sickness, while the hand shown on the right side (with GL Point size 2.5) has a smoother, finer aspect which corresponds to some extent with the smoothness of the skin.

Simple and flat objects do not benefit from stippling in a significant way from our point of view. For example we tried stippling the model of a cube, and it didn't look specially interesting. Other objects like satellites do not make sense when drawn in the stippling style, because stipples can hardly convey the look of metal or manufactured pieces.

7.3 Real-Time Rendering

A relatively recent advance in Graphics Hardware is the availability of Vertex Programs (also known as Vertex Shaders) and Pixel Shaders [Lindo1] as a new addition to the Graphics Pipeline. A vertex program is a piece of computer code which is programmed under certain restriction of memory available and possible length which has the advantage of being able to be executed in parallel for a large number of vertices within the GPU (Graphics Processing Unit). Because the point size now can be computed in parallel for hundreds of thousands of points, the graphics system can produce stippled renditions at interactive frame rates (see Table 7.1). Our results show that up to 100,000 points can be sent to the GPU and we can achieve frames rates close to 60 frames per second (fps). For a larger number of points, performance decreases to 30fps, and for the Kachel model, with around 400,000 points, it is not possible to hold real-time frame rates anymore, with the frame rate decreasing to 1.3 frames per second. Video 12-*realtime Stippling* shows the interaction with the real-time stippling system with the Bunny, Horse and Brain Models.

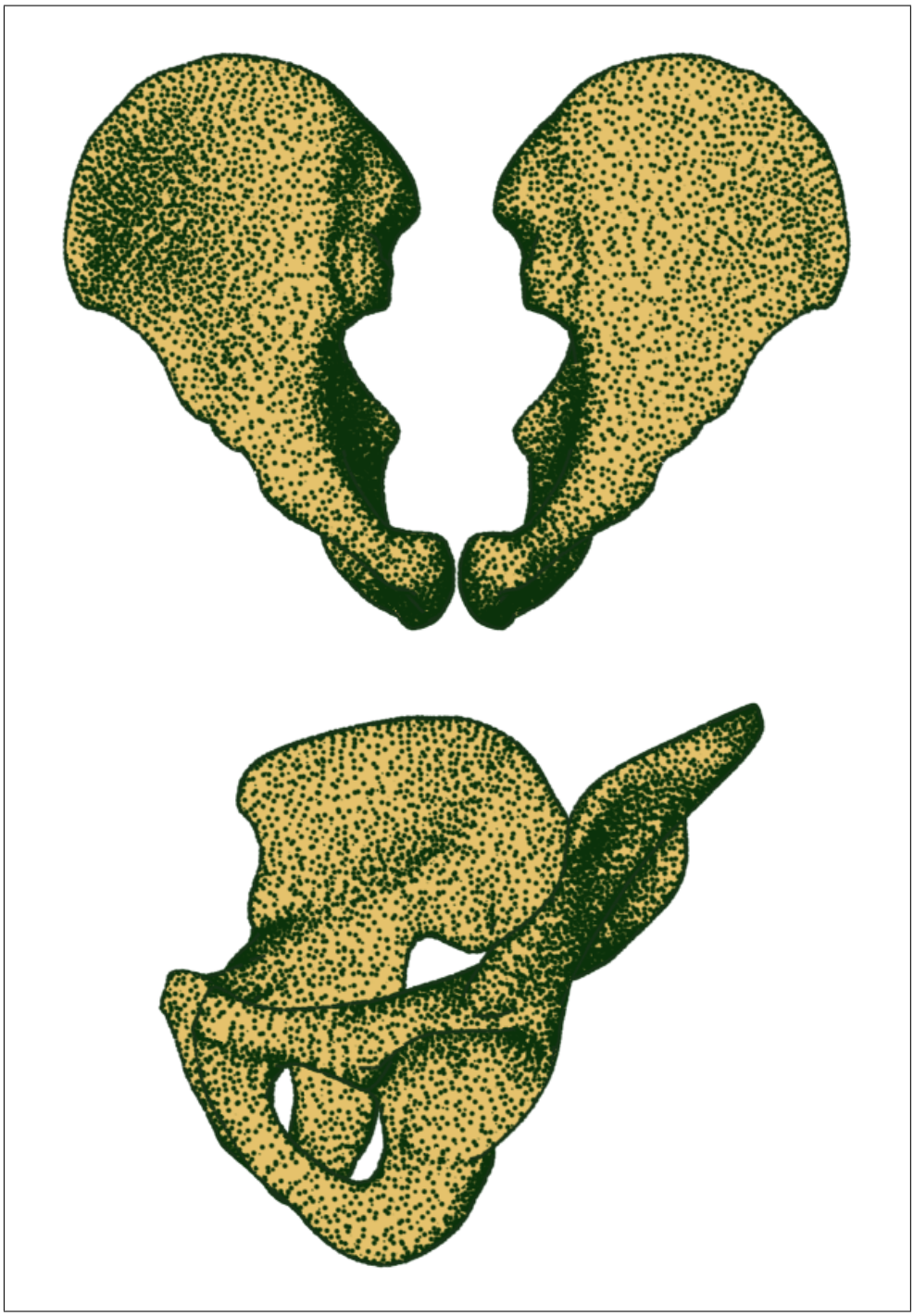

Figure 7.5: Stippled renditions of a pelvis bone model (original sizes: 604x456 & 467x436).

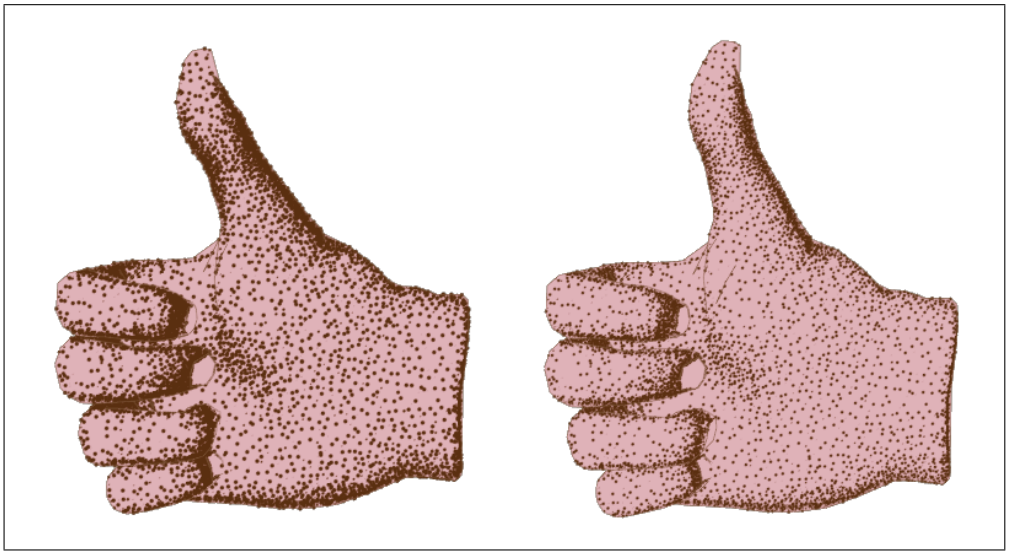

Figure 7.6: Effects of point size on the overall impression the image conveys. The hand model shown on the left has point of size 3.7, while the one on the right has point size 2.5 and looks smoother.

Model	Rendered Points	Frame Rate
Horse	96,966	59.0 fps
Bunny	69,451	56.0 fps
Brain	288,334	31.5 fps
Spar Pot L	172,078	26.0 fps
Mosaic	400,000	1.3 fps

Table 7.1: Rendering times for the real-time stippling system.

7.4 Transparency

Stipples are suitable for conveying transparency. In the work of [Lum02] stipples are used for volumetric rendering, where the volumes are rendered as semi-transparent objects. In our case, we have discovered that stippling is well suited for conveying transparent surfaces as well. The fact that a stipple covers a very small part of the surface of the model makes this possible. In Figure 7.7 we show an object rendered with transparency in our real-time system. The transparency was achieved simply by avoiding rendering the faces that correspond to the heart shell, and rendering only the stipples which correspond to those faces.

Impressive effects of stippling for semi-transparent surfaces are obtained by mixing opaque objects with semi-transparent stippled surfaces. In Figure 7.7 we show a frame from our beating heart animation with stipples replacing the outer shell of the heart for a transparency effect. This animation lets us take a look at the inner parts of a beat-

ing heart, while the stipples convey the outer surface of the heart. Not only that, stipples also illustrate how the outer layer stretches as the beating of the heart takes place. This reveals the potential of using this technique for producing animated medical illustrations. To our knowledge, this effect cannot be conveyed with conventional transparency techniques because semi-transparent surfaces have no distinguishing marks on them which could help us appreciate the stretching on such layers. To appreciate the transparency effect obtained using stipples, we have produced Video 13-*stippled heart with transparency*, which should be compared with Video 14-*beating heart with transparency*, where an animation of the beating heart with stipples is provided. In addition, a way to attach distinguishing marks to a transparent surface could be to put a texture on the surface, and then render it with a semi-transparency attribute, but this falls outside of the scope of this thesis.

In the case of the beating heart model, we can also observe that plain stippling of the heart surface does not significantly improve the aspect of the model. What is more, under some views, the shape of the heart conveys even an unintended look (that of a strawberry) for points placed over a opaque red surface. Most likely, the reason that some people might have this perception is that we have mental images of objects, and in this case, the resulting image matches the mental images that many persons have of a strawberry drawn in a cartoon-like style.

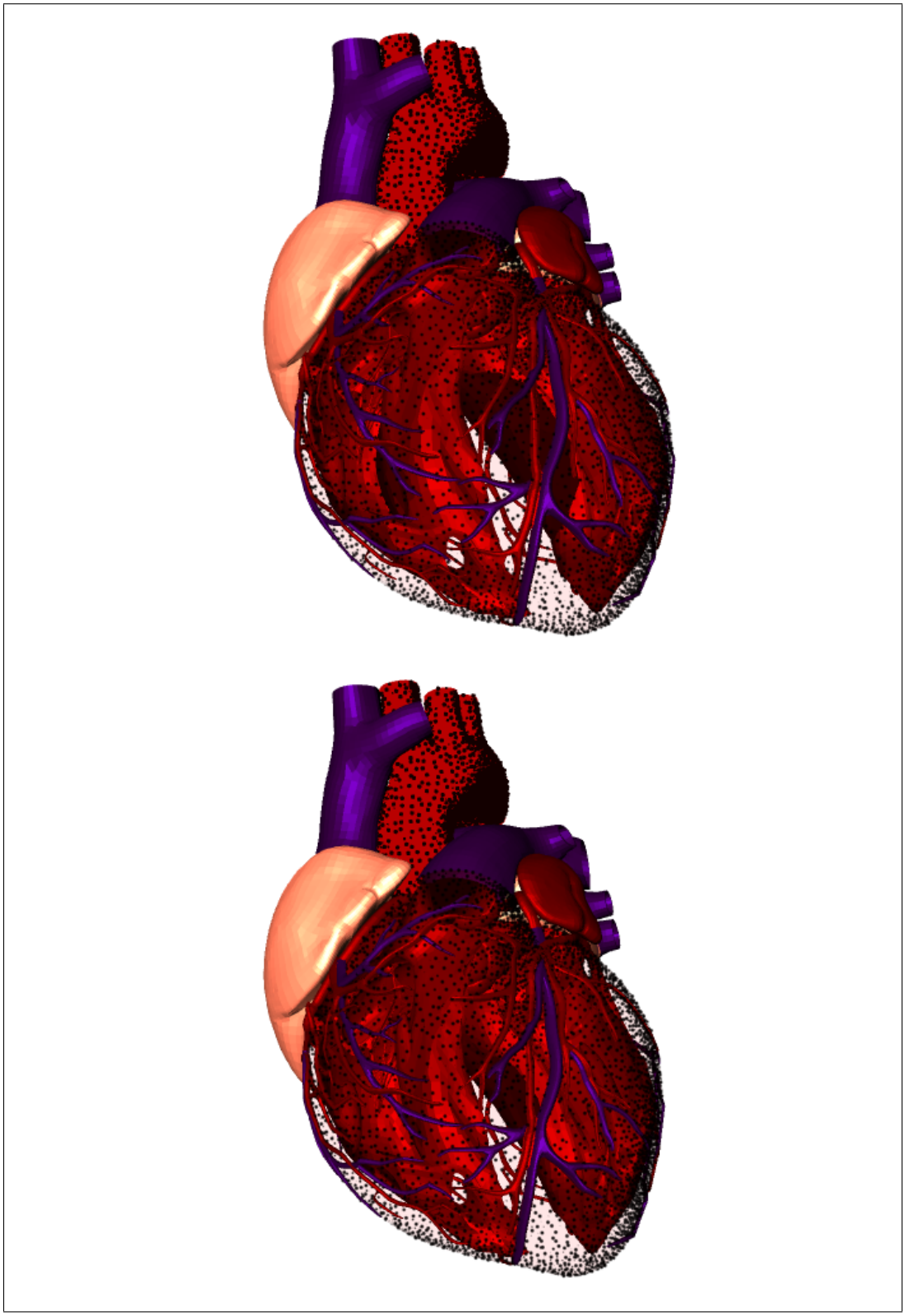

Figure 7.7: Frames from the animation "Transparent beating heart" using stipples to convey the shape of the exterior of the heart, while the user can look at the heart chambers (original sizes: 600x500).

8 Stippling in Archaeology

An important application for the stippling technique is found in archaeology, where stippling is used to illustrate the objects found during excavations. As part of our research, we wanted to explore the application of our rendering technique for actual objects which proceed from true archaeological findings. For this, we contacted the Head Archaeologist of the Bureau for Archaeology in Saxony-Anhalt (the Landesamt für Archäologie Sachsen Anhalt), Hr. Kuhn and his team, who currently work (2002-2003) at excavations taking place at the Dom Square in the city of Magdeburg.

In this Chapter we describe this collaboration and what we have learned from the interaction with the archaeologists and from the material they have provided us with.

We started by showing them some of our stippling animations applied on generic models like the teapot and the stanford bunny. The animations looked interesting to them, and made them willing to cooperate with us using objects from their excavation site.

The cooperation was then designed as follows: we received objects from their archaeological findings and we reproduced them as 3D models using a 3D scanner. After that, we applied our stippling techniques on the models and asked them for feedback about the results.

The 3D scanning was achieved with help from the Fraunhofer Institute for Factory Operations and Automatization in Magdeburg[1]. Two objects were 3D-scanned for us by the Fraunhofer Institute for Factory Operations and Automatization in Magdeburg. The first object is a mosaic piece (a glazed tile, known in german as Kachel) used as decoration of a heating oven commonly used in the rooms of old houses. This piece is dated from the 16th. century and its measures are 17cm wide, 19.5cm high and 4cm deep (see Figure 8.1). The second object is a piggy-bank (germ. Sparbüchse) from the middle ages (14-15th. century), which rather looks like a savings pot with a diameter of approximately 8cm and a height of 7cm (see Figure 8.2). Other objects were also

[1] Hr. Erik Trostman

considered for scanning, but since the surfaces have only smooth reliefs, the scanning was not done. This was rather a technical limitation which was due to the lack of dedicated resources for the project. In Figure 8.3 we show one of the objects that was not scanned due to this limitation. We introduce this Figure because we will discuss aspects related with this and similar objects later.

The mosaic model was scanned at a resolution of 400,000 polygons, while the piggy bank was scanned at a resolution of 200,000 polygons.

Since the mosaic model is colored, we took a picture of the model and attached it to its surface using a planar projection. To do this, we first had to distort the image to fit onto the surface of the model, because the original picture did not correspond to the proportions of the object. This is most likely due to the fact that the camera obtains images through perspective projection, and the mosaic model had significant changes in depth.

After the objects were scanned, we applied the point relaxation technique on the input mesh and obtained the point hierarchies for them. The time required to process these models are shown in Table 5.1.

At the beginning of April 2003, a visit from the archaeologists was arranged where they had the opportunity to observe the real-time implementation of both input models, where the real-time version of the mosaic was in the traditional black-and-white representation, and could give us some relevant feedback. The images the archaeologists observed are shown in Figures 8.6 through 8.8. We show some samples of the real-time renditions of the mosaic model that our users (the archaeologists) observed. We also presented them a hybrid rendering style where we put stipples on the surface of the model with the texture obtained previously. This images is shown in Figure 8.8.

Figure 8.1: Photograph of the glazed mosaic tile (original size: 17x19.5cm).

Figure 8.2: Photograph of the piggy bank (original diameter: 8cm, height: 7cm).

8.1 General Comments

The overall results is that they were impressed by the quality of the images and they found the work interesting. An initial thought that was expressed is that the results were impressive and that fulfilled their expectations.

To provide a point of reference for comparison, the Head Archaeologist provided us an illustration of the mosaic model done by a specialized technical illustrator[2]. This illustration is shown in Figure 8.4, and we will refer to this drawing as the 'scientific illustration'. It is important to notice that, within the context of this thesis, we want to reproduce stipple drawings in the way human artists do. Not to replace them, rather than that, an important motivation can be stated as the possibility of providing a new rendering style for 3D models for animations which is frame-coherent and complies with the restrictions derived from the stippling drawing style. Having said that, the discussion on how to produce similar results to the ones obtained by artists can continue.

As general information about the nature of the illustrators goals and performance we have been told by the archaeologists that an artist can produce the illustration of the

[2]Fr. Spring

Figure 8.3: Photograph of a pot fragment which was not scanned (original size: 7x6cm).

mosaic in one or two days and that in general, an illustrator is able to draw 30 to 40 pieces (fragments) each day, which is more than we had initially expected. We presume that the reason the mosaic drawing takes significantly more time than the small fragments is that it requires more dedication, since in such cases the artist produces preliminary versions or sketches before producing the final version. It is important also to notice that the technical illustrator has accumulated experience of several years (in the case of the medieval mosaic illustration, the artist had 3 years of experience at the time of production).

The process of making a stippled drawing includes several previous drawings until the final drawing is done, as is common in other artistic techniques. It is also interesting to discover that for the archaeologists the objects themselves are not of interest, what is important is the presence of decorations and the description of the form of the objects (the profile). These decorations and the form of the profile of the objects help archaeologists to determine the age of the archaeological site[3]. These findings are not the goal of the investigation per-se; rather, they indicate the approximate age of the exploration site.

[3]Hr. Kuhn, personal communication

Figure 8.4: A stippled drawing of the mosaic done by a technical illustrator (Saxony-Anhalt Office of Archaeology, original size 17x19.5cm).

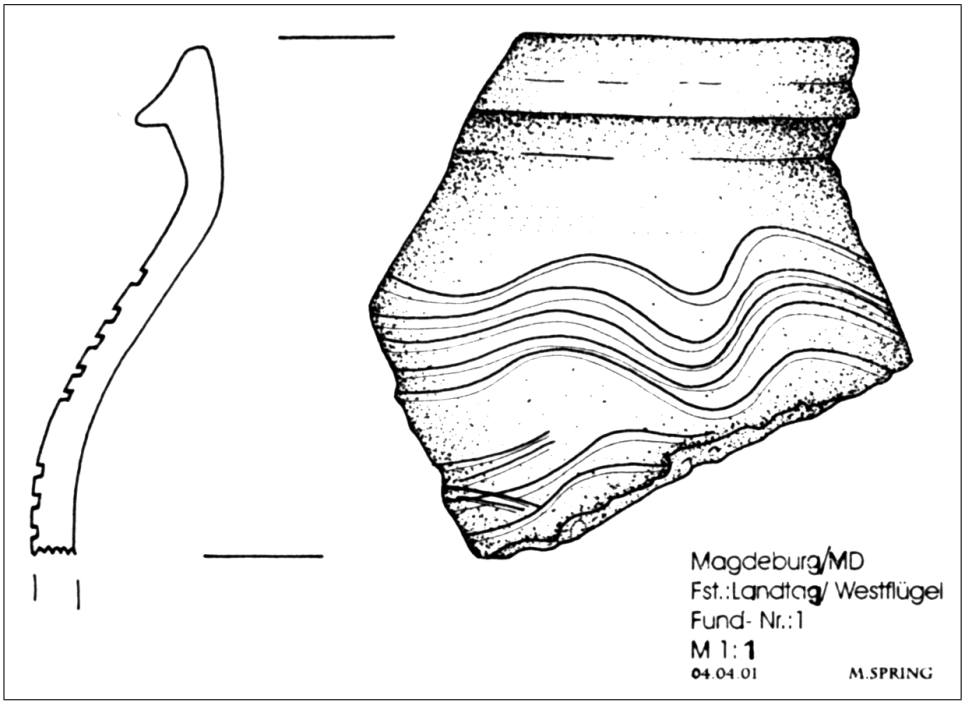

Figure 8.5: A stipple drawing of the object in Figure8.3 (original drawing size: 11 x 6.5cm).

8.2 Line Drawings on Top of the Stippled Renditions

The first issue that was commented was that the artistic drawing has an element of interpretation, which means that the illustrator explores the object and decides which information needs to be transmitted and how it is to be conveyed. This interpretation element is often expressed through the introduction of lines in the drawings, and sometimes by the increase of stipples to enhance shape in some regions of the illustration.

In addition, the artist pays much attention to detail in the object being drawn. Detail amplification is a strong element which was also acknowledged by the archaeologists themselves, who mentioned that the artist does not only perform visual inspection, but also they touch and feel the surfaces by tact to decide what to render. For example, this detail amplification is observed in the ornaments in the hat of the character shown in Figure 8.4, which are in reality incomplete and subtly noticeable in the original object. This effect is also shown in Figure 8.5, where we observe a stippled rendition of the object shown in Figure 8.3.

There are some options that can be used to overcome these problems, the first is to use specific algorithms to enhance some sharp edges and silhouette edges. In Figure

Figure 8.6: Snapshot from the real-time stippling system using the Kachel model (original size: 810x846).

8.9, we illustrate how sharp edges are used to emphasize parts of the model with high curvature. A side effect of introducing this feature lines is that the drawing has certain noise, which interferes with the stippling effect. These results are similar to those of enhancing features using gradient information, presented by Lum and Kwan-Liu [Lum02]. The other option is to encourage user-interaction. In a similar way as in the approach proposed by Buchanan and C. Sousa [Buch00], An artist could take the 3D model and determine lines which should always be present in the model. In the approach presented by Buchanan these lines would be placed on top of existing edges,

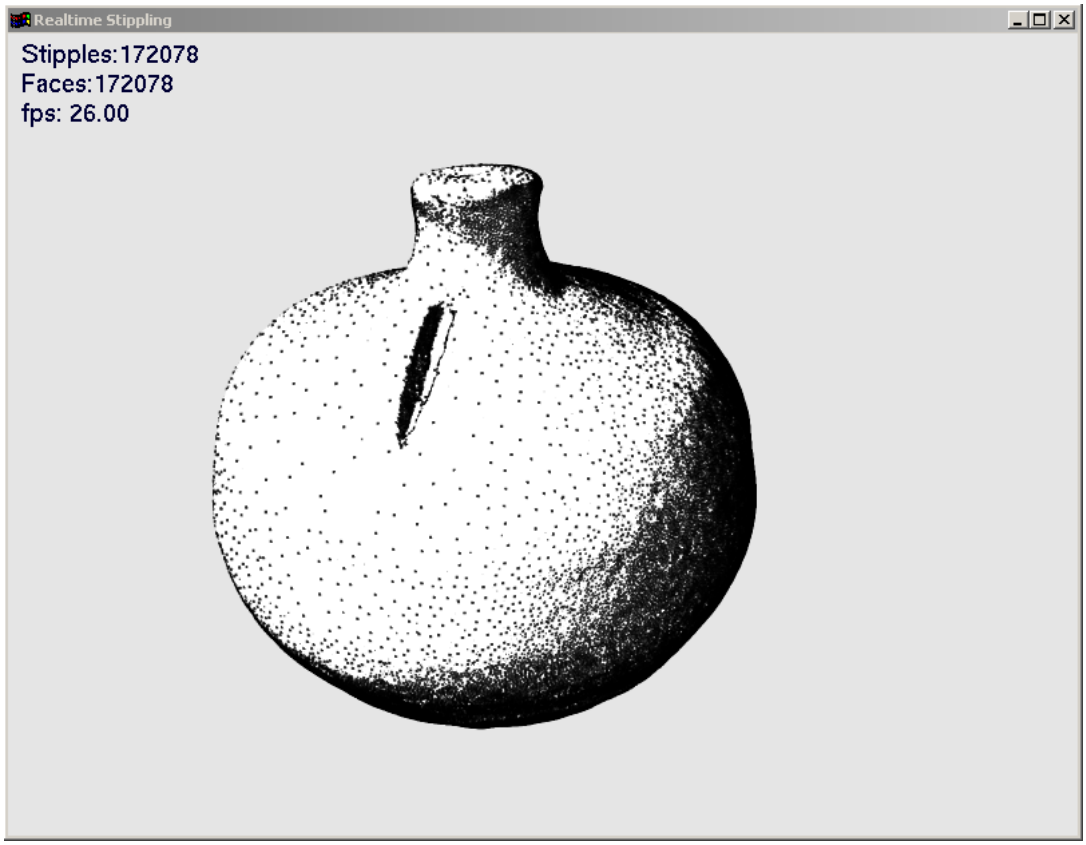

Figure 8.7: Snapshot from the real-time stippling system using the Piggy-bank model (original size: 807x626).

and are denominated as 'artists edges'. Such a system would be similar to existing 3D painting tools, such as Tarzan's Deep Canvas [Dani99, Igar01], but in general the solution of having to draw on the 3D model was not received with much enthusiasm. Another approach is to use curvature information and store it in the stipples. Areas with high curvature could be enhanced over areas of low curvature by increasing the relevance of points located in these areas.

8.3 Animation Versus Single Images

An interesting aspect noticed when talking to the archaeologists was that they were interested in specific views for still frames.

It was our perception that the archaeologists were not too excited about the fact that they could interact with the model in real time and change several parameters arbitrarily, or in the possibility of using the interactivity and playing with the model. Fur-

Figure 8.8: The Mosaic model with texture and stipples (original size: 563x640).

thermore, it was our perception that there were specific views which were interesting to them. They would go and ask for a specific point size in a certain moment and then say: "that's an image that fulfills my expectations". So they thought about the real time interaction more as a tool to determine specific images and views which are attractive for them. In addition, the archaeologists clearly stated that for the purpose of scientific exchange and publication, they definitely prefer the human-made technical illustrations.

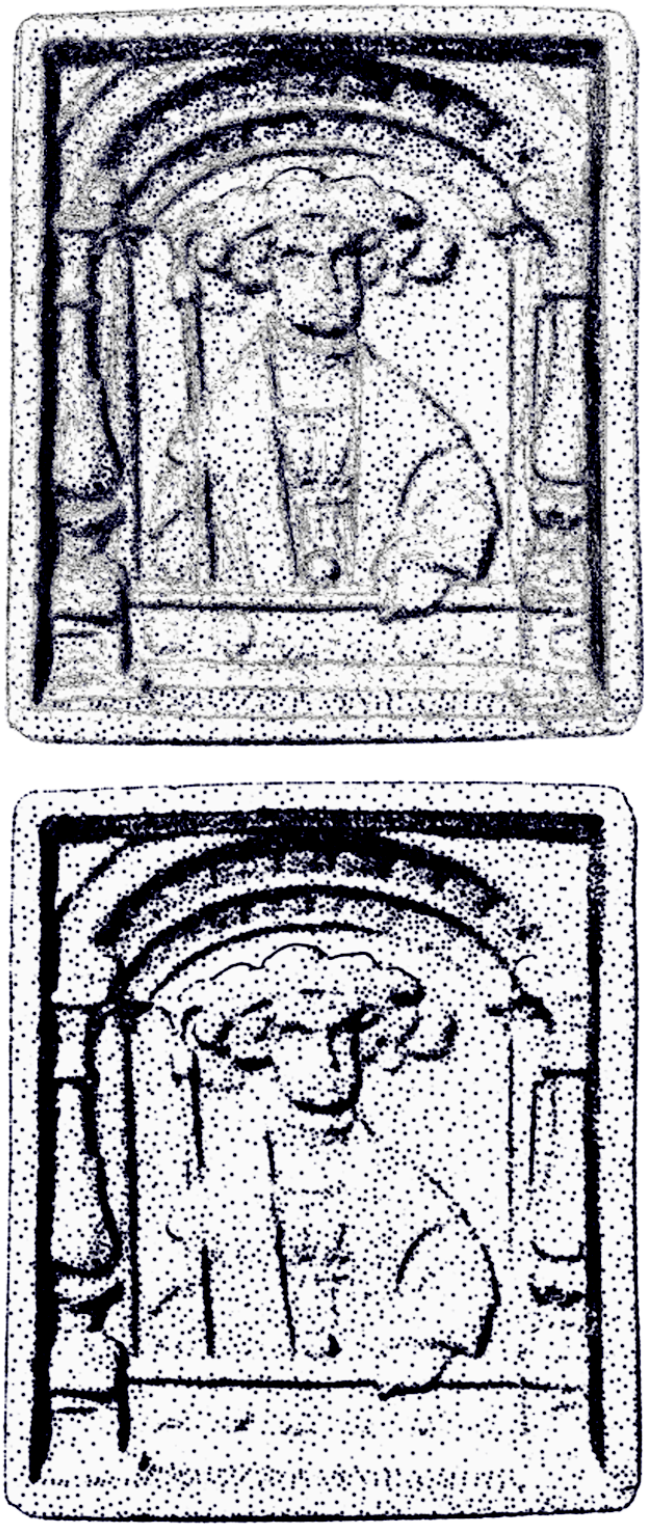

Figure 8.9: The Mosaic model with (top) and without (bottom) curvature enhancement using sharp edges, notice how more details related to the shape of the object appear in the image at the top, while the image at the bottom looks cleaner (original sizes: 630x736).

A reason for this preference might be that the archaeologists are used to communicate their findings using single drawings and not interactive 3D tools. It is the task of the illustrator to convey all the relevant features of the object in one front view and in some cases an additional profile view, and it is expected from the archaeologists to interpret these illustrations. Such illustrations constitute the language that they have been using for centuries (which in addition, gives them the possibility to analyze material from other times). Another factor that might influence the attitude of the archaeologists towards our system might be the fact that the availability of the technology is limited, such that nowadays it cannot be expected that most archaeologists have a powerful computer as part of their standard equipment, which in the end, also limits its attractiveness for common use. On the other hand, an immediate application they found for the system was the production of images for their internet website www.archlsa.de(2003).

8.4 Illumination and Shading

An interesting point indicated by our users is that dark areas should be avoided in general. This is regardless of whether the areas are in the shadowed regions or not. This is a requirement which is traditionally not considered in the graphics community, where the convention is that shading is strictly depending on illumination. A reason that dark areas are avoided by technical illustrators might be that dark areas provide no information for the scientist or the viewer. Regardless of whether an area should be lit or not from the point of view of the light models commonly used in 3D Graphics (flat and phong shading, radiosity), archaeologists expect to obtain information from the illustration. An area which is completely dark due to the light position may be correctly shaded from the graphics point of view, but is not interesting for them.

A first approach to reduce the saturation in the dark areas of our renditions is to set a limit for the darkest tone possible. This approach is shown in Figure 8.10. The illustration of the mosaic tile has lost contrast and some features of the character are harder to recognize due to the lack of contrast. On the other hand, the illustration of the piggy bank still looks dark in some areas, but is not completely black, achieving the desired effect.

Another approach is to have a static illumination model with multiple light sources, where the features which are important are enhanced for specific views of the model. Having several light sources is done in photography, where several light sources are projected on a static object to obtain an image for a magazine or for advertising. In addition, Hamel [Hame00] suggests the use of several light sources for producing non-photorealistic renditions. In our system, we have a unique light source, but the system could be extended by adding an additional lighting system with several light sources.

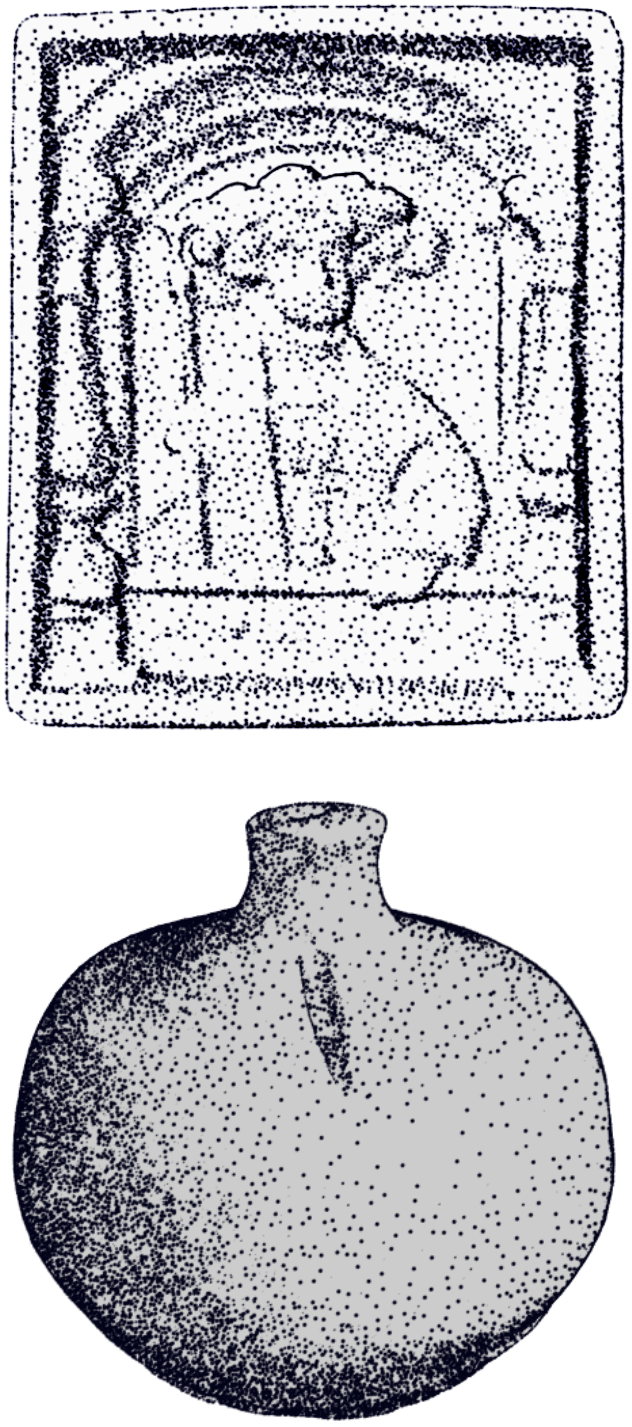

Figure 8.10: Setting a limit on the darkness of the stippled renditions: at the top, the mosaic model (to be compared with images in Figure 8.9). At the bottom, the Piggy bank model. Compare with the rendition in Figure 8.7 (original sizes: 630x745 and 483x510).

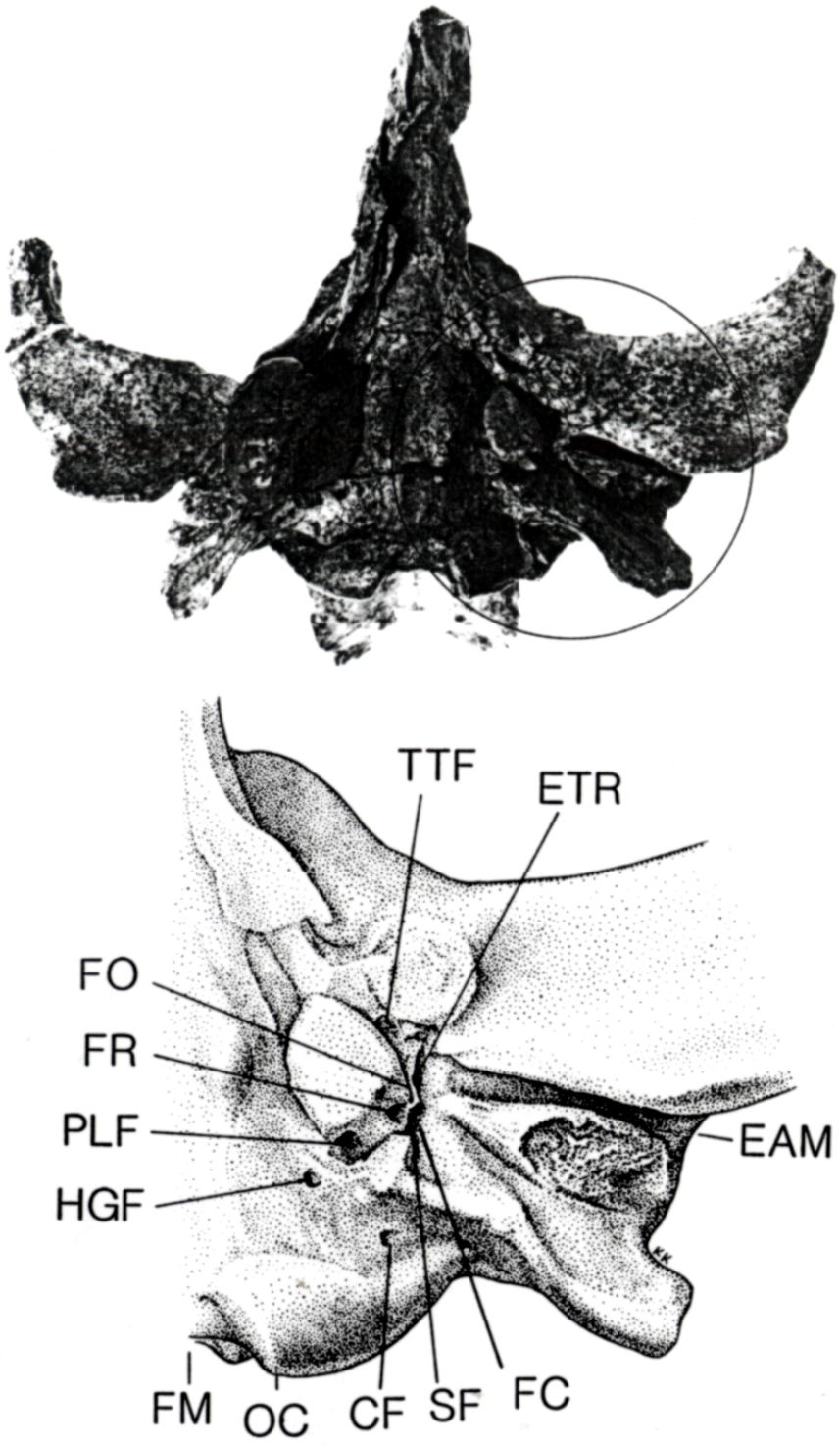

Figure 8.11: At the top, a photograph of the ear region of an Eocen whale from Pakistan. At the bottom, a human-made stipple drawing of enlarged area of interest, clearly showing morphological details without the discoloration and crack and chip distractions of the original specimen. Photo by George Junne. Drawing by Karen Klitz, courtesy of P.D. Gingerich [Hodg88] (original sizes: 826x655 and 800x715).

In Figure 8.11, bottom we observe the stippled rendition an artist made out of a region of the bone of a prehistoric whale (bone photograph shown at the top). Notice that there is not a unique light source in the drawing, instead, light sources are locally applied to specific pieces of the bone and dark areas are avoided, with the exception of the cavities illustrated at the center right of the illustation. Notice also, that this configuration of light sources is appropriate for this view of the model, which lead us to believe that the task of dynamically illuminating a moving model requires the combination of both local light sources and overall illumination criteria, such as that suggested by Hamel [Hameoo].

8.5 Depth Cues

Some stipple drawings make use of a depth cueing technique where parts of the model which correspond to holes or cavities are indicated through an increase in point density. Notice for example the cavities on the base of the columns on the left and right sides of the mosaic. This technique corresponds only partially to a physical illumination model, because the cavities are generally illustrated with a higher point density as other surfaces in general and are not entirely due to self-occluding geometry. A way to account for these changes in illumination could be to use radiosity equations to compute the illumination and reflection in a more exact way. We leave this as a recommendation for future work.

Another example where depth cues are significant is found in the difference between the character's background plane and its body, specially in the transition from the background to the upper body, specially the region that corresponds to the shoulders. In this case, the artist decided to leave the upper body part (with exception from the center of the body where some decorations appear) almost without stipples, which produces a strong differentiation between the background and the character. A suggestion to go in the direction of conveying such attributes in such an image, is the introduction of a relevance map which determines which parts of the model should have darker tonalities. Relevance maps are present in the literature on non-photorealistic rendering, and could be at least of partial help. An operator which determines shading according to the actual depth in a model could also be introduced, although it is very unlikely that a naive depth-dependent operator will work properly. Depth cues in this case are a mixture between color properties, and local depth differences. This means that the dot density is not dependent on the actual depth of a surface, but instead is dependent on the dot density around the cavity, and its purpose is to indicate that a local change in depth occurs and not to globally indicate the depth of the surface. As an example to illustrate our hypothesis, the shoulders of the character have a dot density which is almost as sparse as the frame of the image. In reality, both surfaces are not at

the same height, so the similarity in shading must be explained by other factors, such as color, as will be discussed in the next section.

8.6 Color Management

The piggy bank model did not originate as many comments as the mosaic tile model. There might be two reasons for this: one is the simplicity of the model with respect to the mosaic, and the second is that the object has a uniform color. It seems that for black and white objects with simple geometry (in the sense of a lack of decorations), the stippling system satisfied the archaeologists expectations. On the other hand, rendering a colored object using black-and-white tones is perhaps one of the situations where the interpretation of the artist plays a determinant role in the aspect of the final rendition. In the mosaic image, in some parts the geometry does not change, but the color does, so the artist decided to include more stipples for the darker colors. This has the risk of introducing some ambiguity in the image, since an observer may think that the presence of an increase of dot density indicates a cavity where in fact only a change in color is being illustrated. In the case of the upper body of the character, the artist even decided to change the color of the original object in order to enhance the difference in geometry. An appropriate color scheme that changes colors in the way an artist does is a task that could be suggested as common work between researchers in artificial intelligence and computer graphics, as a topic for the Smart Graphics community.

In addition, it is worth mentioning that for a colored model, we might try to use a pointillist style (see Chapter 1). This could be done if the texture information is mixed with the particle system. Finally, another possibilty to be explored is to use an alternative texture consisting only of bright colors and combine it with the stipples. This option would eliminate some of the darkness present in Figure 8.8, where stipples vanish in the colored areas.

8.7 Concluding Remarks

The collaboration with the archaeologists was useful and gave us an understanding of how they see our tools in their area of application. It was also very helpful for us to obtain new information with respect to the user expectations of a stippled drawing, the most interesting being that shading does not have to correspond to the conventions we assume for shading (the use of phong, gouraud or flat shading), and that it is more interesting to obtain information from the images, which suggests that more work needs to be done to consider the issue of user-oriented non-photorealistic illumination.

It is also interesting to notice that the real-time system presented at one moment was identified or considered as the preliminary work with respect to the mosaic with color. Instead, we as authors think of real-time rendering as a final product, and the stippling with colors as an additional product. But is interesting to see that a rendition with colors can be considered as a step ahead of a rendition in black and white per se.

In addition, there was no mention of artifacts like linear patterns or problems with flickering effects. This indicates that central technical challenges addressed in this thesis, namely frame-coherence and appropriate point distribution, were successfully dealt with.

The challenges that emerged from the collaboration with the archaeologists can be summarized as follows:

- Avoidance of dark areas: perform specific control of illumination, even when they are in the shadowed regions, as discussed in Section 8.4.
- Detail enhancement: perform silhouette detection and use sharp edges to improve the aspect of the stippled rendition Also, silhouettes permit enhancement of interesting features, which could be indicated by a technical illustrator or an archaeologist, such that these details are always part of the final rendition, in a similar way as suggested by Buchanan and Costa Sousa [Buch00].
- Develpoment of an interface for multiple light source defintion: currently, our real time application allows the user to control a number of parameters during interaction, like point size, scale, and spacing. However, the user should be able to have an interactive tool for placing additional light sources and control their position and orientation. In addition, the user should be able to indicate which features are most relevant, perhaps with help of a 3D painting tool [Igar01, Kaln02], or an indication map [Wink94].
- Depth-cue enhancement: several renditions focus on emphazising the overall shape of the model, and enhance features like holes or cavities in the drawing. Providing automatic software tools to identify such areas is challenging, and it is likely that the task of identifying interesting features could also be done by an illustrator who could make a judgment on which cavities are relevant for the purposes of the illustration. In this case, an interaction tool designed to indicate such areas might be also a viable solution for this problem.

9 Conclusions

The advent of non-photorealistic rendering in computer graphics has arisen several questions related to the creation of algorithms which bring traditional artistic and scientific drawing styles to the digital world by using them for enhancing 3D computer graphics. The fact that these styles are designed and meant as single prints, drawings or illustrations brings a number of challenges which appear as we want to use these styles in 3D.

We have identified three main issues that must be adressed when doing this transition from 2D to 3D graphics: frame-coherence, view-dependent scaling and projection, and possibility of doing morphing and animation using NPR styles. Frame-coherence is neccessary to produce smooth animations and to avoid flickering artifacts which are perceived as noise. View-dependent scaling and projection is necessary to maintain the intended style as objects move. Morphing and animation gives us the possibility of using non-photorealistic styles for animated models.

Our main motivation in this thesis was to deliver a solution to solve the problems which appear when trying to bring 2D drawing styles to 3D graphics. Our work shows that point hierarchies constitute a valuable tool for creating view-dependent, frame-coherent animations of static and animated 3D models in the stippling style.

By using point hierarchies, we ensure:

1. An adequate particle distribution and regular spacing over the surface of a model.

2. An appropriate behaviour of the particles under morphing and model animation.

3. Smooth particle insertion and removal during an animation when adapting to changes in scale and projection on the screen.

We have concentrated our efforts exploring animation in the stippling style, however, point hierarchies have a strong potential to be used in conjunction with other rendering styles. As one of the anonymous reviewers of our paper in the Computer Graphics & Applications Special Issue on Non-photorealistic Rendering [Meru03] puts it: "This technique complements a host of previous work in non-photorealistic systems including pencil sketching, pen and ink, charcoal, watercolor, and cartoon rendering. Each of these papers has contributed positively to the body of literature and collection of techniques available to computer graphics specialists and software developers creating NPR systems. Somebody out there is thinking about this very problem and this paper attempts to deliver a solution to the practitioner."

In addition, we have presented two approaches for producing point hierarchies. Both of these techniques are inspired in research related to level-of-detail, where vertex hierarchies are generated for 3D mesh simplification.

We would like to emphasize that it is the sum of all the parts which is unique to our work, our solution is in reality a modular one: a user can decide whether to use an alternative point hierarchy creation algorithm, add additional attributes to the points in the hierarchy, make a real-time implementation with additional special effects, or modify the behavior of the point hierarchy under deformations.

9.1 Future Directions

We see the following opportunities for extending this work:

9.1.1 Use of the Point Hierarchy for Adaptive Rendering

We create and make use of the point hierarchy to decide when and where to place stipples on the surface of a model. However, during rendering, the complete point hierarchy is traversed, where not all points within this hierarchy are rendered. A more efficient approach is to traverse only the sections of the point hierarchy which are necessary to produce the stippled rendition. This can be done in several ways: by view-frustum culling, backface culling, and screen-space or distance-based culling. These culling techniques are derived from the literature on level-of-detail, where the resolution of a model is dynamically controlled according to these and other heuristics. In [Dach03] an approach is presented which permits the use of hardware-acceleration to perform these optimizations in the Graphics Processing Unit and permits the rendition of point sets containing millions of points distributed in several models. Implementing similar optimizations in our real-time rendering system is possible and would improve significantly its performance.

9.1.2 Hardware-Accelerated Visibility Preprocessing

An additional way to reduce the number of points sent to the graphics engine is to determine a priori which parts of the model are visible and then send to the graphics engine only those particles which lie on the surface of the visible parts.

To achieve this, some preprocessing needs to be done. First, it is necessary to divide the model into parts. These parts can also be the surface patches, generated by patch fusion, as done for visibility-preprocessing based on ID-bitfields [Meru02a]. Second, it is necessary to organize the points of the patch hierarchy according to the parts of the model where the particles lie. This can be done using a sorting algorithm that traverses the point hierarchy. During rendering, two passes are needed. In the first rendering pass, the model is rendered with each part encoded in a unique color. After that, the color buffer is consulted and we determine which parts are visible. In the second rendering pass, we send the particles belonging to the visible parts to the graphics engine.

9.1.3 Use of Other Rendering Primitives

So far, we have done experiments rendering stipples. However, much work in NPR is based on pencil and paint strokes, or other rendering primitives. We believe that the approach here presented can also be used to generated drawings in other non-photorealistics styles using stroke or paint primitives, among others, which would allow for new effects in NPR animation.

One way to do this is to replace stipples with textured quad primitives, where the texture represents a paint stroke, as in the work done in painterly rendering [Meie96]. The quad primitives may contain paint strokes, brush strokes or other basic shapes.

Another way to use point hierarchies with other rendering styles is to use the points as the starting position for line strokes, as has been done for the case of image-plane frame coherence. In this case, the point determines the center of a line stroke, and the line stroke extends on the surface of the model according to a certain direction vector.

In this category, we can also include the use of colored point primitives. We could use the point hierarchy to emulate the pointillist painting style which was developed by the impressionists (see Section 1.3). In this case, however, the point density is handled in a different way than in the case of traditional stippling presented here. In the case of stippling, shading is given by the density of stipples. In the case of pointillism, images are completely saturated by dots, and shading and color is given by the hue, saturation and darkness of each dot in the image.

9.1.4 Point-Based Rendering for NPR

As mentioned in Chapter 2, little research has been done to bring together point-based rendering with nonphotorealistic rendering. With new developments in Graphics Hardware, it is now possible to render large amounts of points in real-time. In fact, current development in vertex and pixel shaders has shown that it is possible to obtain a variety of non-photorealistic rendering styles for polygonal-based models, and this gives us confidence to suggest that it should be also possible to generate point-based stippled drawing and other point-based NPR styles using point-hierarchies (as in the case of Q-splats [Rusi00]). This would open the field fur the use of stippling for massively large point sets. All the elements are present in the literature, and it could be a promising area for further development.

Bibliography

[Alex01] Marc Alexa. "Mesh Morphing STAR". *Eurographics 2001 State of The Art Reports; Computer Graphics Forum, 21(2)*, pp. 173–196, http://www.igd.fhg.de/~alexa/paper/index.html, 2001.

[Alli01] Pierre Alliez and Mathieu Desbrun. "Progressive Compression for Lossless Transmission of Triangle Meshes". *SIGGRAPH 2001 Conference Proceedings*, pp. 195–202, http://doi.acm.org/10.1145/383259.383281, 2001.

[Atta02] Dominique Attali and Jean-Daniel Boissonnat. "A Linear Bound on the Complexity of the Delaunay Triangulation of Points on Polyhedral Surfaces". Research Report 4453, INRIA, http://www.inria.fr/rrrt/rr-4453.html, 2002.

[Aure91] F. Aurenhammer. "Voronoi diagrams – a Survey of a Fundamental Geometric Data Structure". *ACM Computing Surveys*, Vol. 23, No. 3, pp. 345–405, Habilitationsschrift. [Report B 90-09, FU Berlin, Germany, 1990], 1991.

[Aure00] F. Aurenhammer and R. Klein. "Voronoi diagrams". *Handbook of Computational Geometry, Chapter V*, pp. 201–290. Elsevier Science Publishing, [SFB Report F003-092, TU Graz, Austria, 1996],url-http://www.igi.tugraz.at/auren/ , 2000.

[Berg00] Mark de Berg, Otfried Schwarzkopf, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*, chapter 7, pp. 145–161. Springer Verlag, http://www.cs.uu.nl/geobook/, 2000.

[Buch00] John W. Buchanan and Mario C. Sousa. "The Edge Buffer: a Data Structure for Easy Silhouette Rendering". *Proc. of the 1st. International Symposium on Non-photorealistic Animation and Rendering*, pp. 39–42, http://doi.acm.org/10.1145/340916.340921, 2000.

[Coop95] Philip Cooper. *Cubism*. Phaidon Press, http://tiger.towson.edu/users/kmclew1/, 1995.

[Corn01] Derek Cornish, Andrea Rowan, and David Luebke. "View-Dependent Particles for Interactive Non-Photorealistic Rendering". *GI 2001*, pp. 151–158, http://www.graphicsinterface.org/proceedings/2001/158/, June 2001.

[Dach03] Carsten Dachsbacher, Christian Vogelgsang, and Marc Stamminger. "Sequential Point Trees". *SIGGRAPH 2003 Conference Proceedings (to appear)*, http://www9.informatik.uni-erlangen.de/Research/Rendering/SPT, 2003.

[Dani99] Eric Daniels. "Deep canvas in Disney's Tarzan". *ACM SIGGRAPH 99 Electronic Art and Animation Catalog*, p. 124, http://doi.acm.org/10.1145/312379.312887, 1999.

[Deus00] O. Deussen, S. Hiller, C.W.A.M van Overveld, and T. Strothotte. "Floating Points: A method for Computing Stipple Drawings". *Computer Graphics Forum*, Vol. 19, No. 3, pp. 40–51, http://www.eg.org/EG/CGF/volume19/issue3, 2000.

[Elli99] S. Ellis, B.D. Adelstein, S. Baumeler, G.J. Jense, and R.H. Jacobi. "Sensor Spatial Distortion, Visual Latency, and Update Rate Effects on 3D Tracking in Virtual Environments". *Proceedings of the Virtual Reality '99 Conference*, pp. 218–221, http://human-factors.arc.nasa.gov/ihh/spatial/papers/pdfs_se/Ellis_1999_VR_3D_Tracking.pdf, 1999.

[Fili96] Mark Filipiak. "Report on Mesh Generation, 3.2.4 Delaunay Triangulation". Technical Watch Report 4453, Edinburgh Parallel Computing Centre (EPCC), http://www.epcc.ed.ac.uk/overview/publications/training_material/tech_watch/96_tw/tw-meshgen/MeshGeneration.book_1.html, 1996.

[Fole93] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice*, pp. 376–381. Addison-Wesley Systems Programming Series, Don Mills, Ontario, http://www.awprofessional.com/catalog/product.asp?product_id={25B96B4D-%59FA-4E13-BB7F-A1064EF55963}, 1993.

[Freu02] Bert Freudenberg, Maic Masuch, and Thomas Strothotte. "Real-Time Halftoning: A Primitive For Non-Photorealistic Shading". *Rendering Techniques 2002, Proceedings 13th Eurographics Workshop*, pp. 227–231, http://isgwww.cs.uni-magdeburg.de/~bert/, 2002.

[Gooc98] Amy A. Gooch, Bruce Gooch, Peter Shirley, and Elaine Cohen. "A Non-Photorealistic Lighting Model for Automatic Technical Illustration". *SIGGRAPH'98 Conference Proceedings*, Computer Graphics Proceedings, Annual Conference Series, pp. 447–452, http://doi.acm.org/10.1145/280814.280950, 1998.

[Gooc01] Bruce Gooch and Amy Gooch. *Non-Photorealistic Rendering*. A. K. Peters, Ltd., http://www.akpeters.com/book.asp?bID=131, July 2001.

[Haga01] Toshiyuki Haga, Henry Johan, and Tomoyuki Nishita. "Animation Method for Pen-And-Ink Illustrations Using Stroke Coherency". *CAD / Graphics'2001*, http://nis-lab.is.s.u-tokyo.ac.jp/~haga/study.html, August 2001.

[Hame00] Jörg Hamel. *A New Lighting Model for Computer Generated Line Drawings*. Ph.D. thesis, Otto-von-Guericke-Universität Magdeburg, Magdeburg, Germany, 2000.

[Hodg88] Elaine R.S. Hodges. *The Guild Handbook of Scientific Illustration*. Wiley Europe, http://he-cda.wiley.com/WileyCDA/HigherEdTitle/productCd-0471360112.html, 1988.

[Hopp96] Hugues Hoppe. "Progressive Meshes". *SIGGRAPH 96 Conference Proceedings*, pp. 99–108, http://doi.acm.org/10.1145/237170.237216, 1996.

[Hopp97] Hugues Hoppe. "View-dependent Refinement of Progressive Meshes". *SIGGRAPH 97 Conference Proceedings*, pp. 189–198, http://doi.acm.org/10.1145/258734.258843, 1997.

[Igar01] Takeo Igarashi and Dennis Cosgrove. "Adaptive Unwrapping for Interactive Texture Painting". *Proceedings of the 2001 Symposium on Interactive 3D Graphics*, pp. 209–216, http://doi.acm.org/10.1145/364338.364404, 2001.

[Isen03] Tobias Isenberg, Bert Freudenberg, Nick Halper, Stefan Schlechtweg, and Thomas Strothotte. "A Developer's Guide to Silhouette Algorithms for Polygonal Models". *IEEE Computer Graphics and Applications Special Issue on Non-photorealistic Rendering*, http://csdl.computer.org/comp/proceedings/pg/2003/2028/00/20280424abs.htm, July /August 2003.

[Kaln02] Robert D. Kalnins, Lee Markosian, Barbara J. Meier, Michael A. Kowalski, Joseph C. Lee, Philip L. Davidson, Matthew Webb, John F. Hughes, and Adam Finkelstein. "WYSIWYG NPR: Drawing Strokes Directly on 3D Models". *SIGGRAPH 2002 Conference Proceedings*, pp. 755–762, http://doi.acm.org/10.1145/566570.566648, 2002.

[Kapl00] Matthew Kaplan, Bruce Gooch, and Elaine Cohen. "Interactive Artistic Rendering". *Proceedings of the First International Symposium on Non-photorealistic Animation and Rendering*, pp. 67–74, http://www.cs.utah.edu/npr/papers.html, 2000.

[Laer03] Kristof Van Laerhoven. *Voronoi Diagrams & Delaunay Triangulation*. Computing Department, Lancaster University, UK, http://www.comp.lancs.ac.uk/~kristof/research/notes/voronoi, 2003.

[Lindo1] Erik Lindholm, Mark J. Kilgard, and Henry Moreton. "A User-Programmable Vertex Engine". *SIGGRAPH 2001 Conference Proceedings*, pp. 149–158, http://doi.acm.org/10.1145/383259.383274, 2001.

[Luo2] Aidong Lu, Christopher Morris, David Ebert, Penny Rheingans, and Charles Hansen. "Non-photorealistic Volume Rendering Using Stippling Techniques". *IEEE Visualization 2002 Conference Procedings*, http://www.cs.umbc.edu/~rheingan, 2002.

[Lueb97] David Luebke and Carl Erikson. "View-Dependent Simplification of Arbitrary Polygonal Environments". *SIGGRAPH 97 Conference Proceedings*, pp. 199–208, http://doi.acm.org/10.1145/258734.258847, 1997.

[Lum02] Eric B. Lum and Kwan-Liu Ma. "Hardware-Accelerated Parallel Non-Photorealistic Volume Rendering". *Proc. of the 2nd. International Symposium on Non-photorealistic Animation and Rendering*, pp. 67–ff, = http://doi.acm.org/10.1145/508530.508542, 2002.

[Mack93] I. Scott MacKenzie and Colin Ware. "Lag as a Determinant of Human Performance in Interactive Systems". *Proceedings of ACM INTERCHI'93 Conference on Human Factors in Computing Systems*, pp. 488–493, http://doi.acm.org/10.1145/169059.169431, 1993.

[Mark00] Lee Markosian, Barbara J. Meier, Michael A. Kowalski, Loring S. Holden, J. D. Northrup, and John F. Hughes. "Art-Based Rendering with Continuous Levels of Detail". *Proc. of the 1st. International Symposium on Non-Photorealistic Animation and Rendering*, pp. 59–66, http://doi.acm.org/10.1145/340916.340924, 2000.

[Mehl00] Kurt Mehlhorn and Stefan Näher. *LEDA : A Platform for Combinatorial and Geometric Computing*. Publisher: Cambridge University Press, http://www.algorithmic-solutions.com, February 2000.

[Meie96] Barbara J. Meier. "Painterly Rendering for Animation". *SIGGRAPH 96 Conference Proceedings*, pp. 477–484, http://doi.acm.org/10.1145/237170.237288, 1996.

[Meru02a] Oscar Meruvia. "Visibility Preprocessing Using Spherical Sampling of Polygonal Patches". *Eurographics'2002 Short Paper Proceedings*, http://isgwww.cs.uni-magdeburg.de/~oscar/, 2002.

[Meru02b] Oscar Meruvia and Thomas Strothotte. "Frame-Coherent Stippling". *Eurographics'2002 Short Paper Proceedings*, http://isgwww.cs.uni-magdeburg.de/~oscar/, 2002.

[Meru03] Oscar Meruvia, Bert Freudenberg, and Thomas Strothotte. "Real-Time, Animated Stippling". *IEEE Computer Graphics and Applications Special Is-*

sue on Non-photorealistic Rendering, http://isgwww.cs.uni-magdeburg.de/~oscar/, July / August 2003.

[Mitz03] Michael Mitzenmacher. *Lecture Notes on Data Structures and Algorithms: BFS and Shortest Paths*. Computer Science, Harvard University, USA, http://www.fas.harvard.edu/~libcs124/cs124/intro.htm, 2003.

[New 01] State University of New York. *The Stony Brook Algorithm Repository: Voronoi Diagrams & Delaunay Triangulation*. Department of Computer Science State University of New York, Stony Brook, NY 11794-4400, http://www.cs.sunysb.edu/~algorith/files/voronoi-diagrams.shtml, 2001.

[Pfis00] Hanspeter Pfister, Matthias Zwicker, Jeroen van Baar, and Markus Gross. "Surfels: surface elements as rendering primitives". *SIGGRAPH 2000 Conference Proceedings*, pp. 335–342, http://doi.acm.org/10.1145/344779.344936, 2000.

[Prau01] Emil Praun, Hugues Hoppe, Matthew Webb, and Adam Finkelstein. "Real-Time Hatching". *SIGGGRAPH 2001 Conference Proceedings*, pp. 581–586, http://doi.acm.org/10.1145/383259.383328, 2001.

[Rigg99] Heather Riggs. *Tales to Tell - Kent Goodliffe*. Springville Museum of Art & Springville High School, Utah, http://www.shs.nebo.edu/museum/swap/ttgoodliffeact.html, 1999.

[Rusi00] Szymon Rusinkiewicz and Marc Levoy. "QSplat: a multiresolution point rendering system for large meshes". *SIGGRAPH 2000 Conference Proceedings*, pp. 343–352, http://doi.acm.org/10.1145/344779.344940, 2000.

[Sali96] Mike Salisbury, Corin Anderson, Dani Lischinski, and David H. Salesin. "Scale-Dependent Reproduction of Pen-and-Ink Illustrations". *SIGGRAPH 96 Conference Proceedings*, pp. 461–468, http://doi.acm.org/10.1145/237170.237286, 1996.

[Seco02a] Adrian J. Secord. "Weighted Voronoi Stippling". *Proc. of the 2nd International Symposium on Non-Photorealistic Animation and Rendering*, pp. 37–43, http://www.cs.ubc.ca/~ajsecord/publications.html, 2002.

[Seco02b] Adrian J. Secord, Wolfgang Heidrich, and Lisa Streit. "Fast Primitive Distribution for Illustration". *Proc. of the 13th Eurographics Workshop on Rendering*, pp. 215–226, http://www.cs.ubc.ca/~ajsecord/publications.html, 2002.

[Stro02] Thomas Strothotte and Stefan Schlechtweg. *Non-Photorealistic Computer Graphics: Modeling, Rendering, and Animation*. Morgan Kaufmann Publishers, http://www.mkp.com/books_catalog/catalog.asp?ISBN=1-55860-787-0, April 2002.

[Turk91] Greg Turk. "Generating Textures on Arbitrary Surfaces Using Reaction-Diffusion". *SIGGRAPH 91 Conference Proceedings*, pp. 289–298, http://doi.acm.org/10.1145/122718.122749, 1991.

[Turk92] Greg Turk. "Re-Tiling Polygonal Surfaces". *SIGGRAPH 92 Conference Proceedings*, pp. 55–64, http://doi.acm.org/10.1145/133994.134008, 1992.

[Ware94] Colin Ware and Ravin Balakrishnan. "Reaching for Objects in VR Displays: Lag and Frame Rate". *ACM Transactions on Computer-Human Interaction*, Vol. 1, No. 4, pp. 331–356, http://www.acm.org/pubs/articles/journals/tochi/1994-1-4/p331-ware/p331-ware.pdf, 1994.

[Wink94] Georges Winkenbach and David H. Salesin. "Computer-Generated Pen-and-Ink Illustration". *SIGGRAPH 94 Conference Proceedings*, pp. 91–100, http://doi.acm.org/10.1145/192161.192184, 1994.

[Wink96] Georges Winkenbach and David H. Salesin. "Rendering Parametric Surfaces in Pen-and-Ink". *SIGGRAPH 96 Conference Proceedings*, pp. 469–476, http://doi.acm.org/10.1145/237170.237287, 1996.

[Zwic01] Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, and Markus Gross. "Surface Splatting". *SIGGRAPH 2001 Conference Proceedings*, pp. 371–378, http://www.cg.inf.ethz.ch/Downloads/Publications/Papers/2001/p_Zwi01b.pdf, 2001.

List of Algorithms

List of Tables

List of Figures