EUROGRAPHICS 2002 / I. Navazo Alvaro and Ph. Slusallek
(Guest Editors)

Short Presentations

Frame-Coherent Stippling

Oscar E. Meruvia Pastor and Thomas Strothotte

Department of Simulation and Graphics, Otto-von-Guericke University of Magdeburg, Germany

Abstract

Stippling is an artistic rendering technique where shading and texture is given by placing points or stipples on the canvas until the desired darkness is achieved. Computer-generated stippling has focused on producing high quality 2D renditions for print media, while stippling of 3D models in animation has received little attention. In this paper we present a technique to produce frame-coherent animations of 3D models using stippling as a rendering style. The problem for which we provide a solution is how to obtain an even distribution of stipples and at the same time ensure frame-to-frame coherence while shading changes over time. In our approach, particles are placed on the surface of the 3D model and adaptively rendered as stipples using OpenGL point primitives and the input model as a canvas. At each frame, the density of particles can be increased if necessary. Selection of particles during rendering takes into account the screen space projection of the edges of a polygon fan that surrounds the particles and the desired tone at the position of the particles. The rendering technique presented here can be applied to arbitrary polygonal meshes and can be extended to include grey scale textures, bump mapping and custom illumination models.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation

1. Introduction

Stippling is an artistic rendering technique where shading and texture is given to the image by placing dots or stipples on the canvas until the desired darkness is achieved. In stippling, the artist must take care to judge the proper scale and spacing of individual stipple dots.[17]

Existing work in computer-generated stippling permits the creation of high-quality renditions of 2D images at high resolutions. These renditions are normally intended for printed media. Image based approaches such as that of Deussen et al.[4] and Secord[18] already provide interactive and automatic tools to obtain high quality renderings in the stippling style. However, stippling and most other artistic drawing styles cannot be introduced in an animation by simply putting together a sequence of independently stippled images without introducing noise in the transitions between frames. This happens because individual stipples or strokes in a given frame are not guaranteed to be placed in the same location at the next frame, or more precisely, in a *corresponding* location. In other words, even when the overall regions of an image remain properly shaded from frame to frame, an animation built in this way is not frame-coherent at the stipple level.

The main goal of this research is to produce animations of polygonal 3D models rendered using the stippling style which are also frame-coherent at the stipple level. Several aspects need to be considered when animating stippling: achieve frame-coherence at the stipple-level, control of stipple density according to changes in illumination, and control of stipple density as the projected size of the model in screen space changes.

We obtain frame coherence by using a view-dependent particle system. Particle systems, obtained by attaching particles to the surface of 3D models, are commonly used in non-photorealistic rendering (NPR) to achieve frame-coherence during animation. While most particle-based NPR systems use 2D primitives to render effects like 3d paint[14], hatching[21], or artistic representations of fur, grass and trees[9], we use points as our rendering primitive (see Figure 1). Because each point is attached to a specific location on the surface of the model (a vertex in a polygonal mesh), points move along with the model as the model is moved in the scene, which provides the frame-coherence effect.

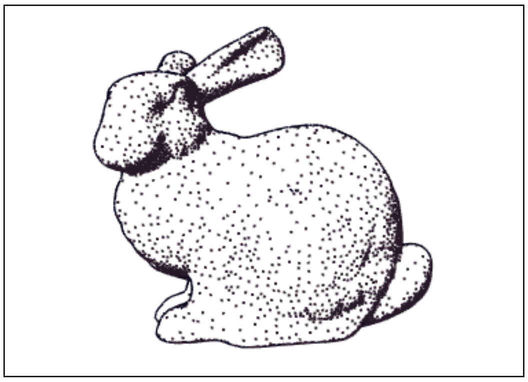

Figure 1: *The Stanford Bunny in the stippling style.*

In order to handle level-of-detail and changes in shading, we have adopted a hierarchical particle system that allows us to select which pixels are added or removed from a scene at a given frame. Detail hierarchies have been used in *artistic rendering*[12], where graphical elements called *graftals* are organized in hierarchies. For instance, a *tuff* has its own graphical representation at a certain level-of-detail, but can be decomposed in *graftals* if more detail is required. A *graftal* itself is defined at several levels of detail and can also be refined when necessary. We do not use this approach of grouping graphical elements because individual stipples do not only describe a shape (which is the case with *graftals*), but they also describe tone, as is the general case in pen-and-ink illustration[21]. When a darker tone is desired, the stipple density is increased. In painterly and artistic rendering, a darker tone is achieved by darkening the color of the 2D primitives being drawn, not by increasing the amount of primitives per area.

Our particle hierarchy is constructed from a mesh simplification and subdivision scheme (see Figure 2). Our approach is based on the work of Cornish et al.[3], who implemented a view-dependent particle system for real-time non-photorealistic rendering. Their solution works well for models at high resolutions, but it does not guarantee appropriate particle density when the model is close to the viewer or has a low resolution. We extended their approach by creating a surface subdivision scheme which can be used for arbitrary polygonal meshes and extends the hierarchical particle system obtained by mesh simplification. We use mesh subdivision when the resolution of the input model is not as high as desired.

Subdivision schemes allow NPR systems to obtain more particles on demand. Kaplan et al.[13] implemented an interactive rendering system for subdivision surfaces that uses *geograftals* as the drawing primitive. In their system, the user is able to arbitrarily add *geograftals* to subdivision surfaces. Each *geograftal* is assigned a random scaling function which lets some *geograftals* remain large in size, while others shrink as objects move away from the viewer. In a similar way, we assign to each particle a relevance function which lets some stipples remain in the image, while others vanish when needed. Kaplan et al. define the location and scaling of the *geograftals* using randomness. In our approach, we use the spatial hierarchy to determine the location and order of appearance of the stipples. Additionally, our system can process arbitrary polygonal meshes and includes constrained randomize and projection operators to improve the stipple distribution in the renditions.

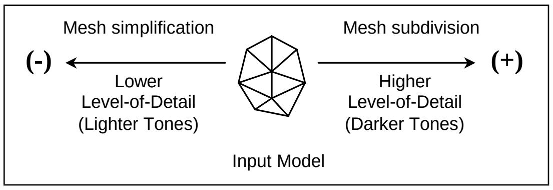
Mesh simplification
Mesh subdivision
(-)
(+)
Lower Level-of-Detail (Lighter Tones)
Higher Level-of-Detail (Darker Tones)
Input Model

Figure 2: *A continuous level of detail is created to achieve the desired shading at any given frame.*

In the stippling style, it is important to obtain even point distributions on the final rendition. In our system, we add and remove stipples from the surface of the models using a spatial criteria to meet this requirement as the animation occurs: new stipples, which are inserted lower in the particle hierarchy, are placed at locations roughly in the middle of existing stipples. Alternatively, when stipples are removed from the surface of a model, stipples at the bottom of the hierarchy are the ones that vanish first. Obtaining appropriate point distributions is a central topic for dithering techniques[20]. Most research on dithering focuses on improving point distributions on the image plane. Ostromoukhov[15] uses meshes with arbitrary connectivity on the image plane for pseudo-random dithering. Independently developed, our system works on meshes with arbitrary connectivity in 3D space, and resembles ordered dithering[20]: the particles on the surface of the polygonal mesh represent the locations where a stipple can be drawn, the particle-hierarchy represents the order of evaluation for drawing, and the geometric attributes of the surfaces and the illumination at the particle location determine the desired tone value.

The rest of this paper is organized as follows: In the next section we briefly review related work on computer-generated stippling. Section 2 explains the implementation of our stippling system. Section 3 presents some results and Section 4 gives our conclusions and future work.

1.1. Related Work

In the field of computer-generated stippling, Deussen et al.[4] and Secord[18] obtain high-quality stippled images using dithering and relaxation of Voronoi diagrams. However, we cannot rely on these approaches to build a noise-free animation sequence, because each frame is obtained by iterative

Voronoi relaxation of existing particles, and the particle distribution obtained in one frame are not guaranteed to correspond with the particle distributions obtained in the next frame. This reveals one of the questions this research attempts to address: whether it is possible to obtain even point distributions such as those obtained in image-based stippling while providing frame-coherence at the particle level.

Winkenbach and Salesin[21] presented an algorithm to render hatched and stippled images on parametric surfaces[22]. However, this work heavily relies on operations in the image plane and uses planar maps in 2D to divide regions of different stroke density, whereas our approach relies on 3D geometry, a local illumination model and a local region of interest around each vertex to place stipples.

Our approach is mainly related to the group of techniques in non-photorealistic rendering systems where particles[14], *graftals*[12], or *geograftals*[13] are placed on the surface of 3D models to achieve frame-coherence. In our case, we have chosen a particle system based on geometric mesh simplification similar to that of Cornish et al.[3]. Gooch and Gooch[7], and Strothotte and Schlechtweg[19] provide further insight into NPR techniques.

The reader can refer to the surveys of Garland and Luebke [6, 10] for information on view-dependent mesh simplification. Of particular importance to our stippling approach are mesh simplification and compression schemes which create evenly simplified meshes at different levels of detail.[8, 11, 2]

In point-based rendering, Alexa et al.[1] present a rendering technique where point sets without connectivity are used to render smooth (highly detailed) surfaces. Although point-based rendering techniques have not been used to emulate the stippling style, there is a natural connection between the approach presented in this work and their approach. To explore this connection is an area for future work.

Recent improvements in texture mapping hardware permit real-time frame-coherent hatching and other pen-and-ink styles using *stroke textures*[5, 22] and *tonal art maps*[16]. While it is clear that at an appropriate resolution stroke textures can be used to emulate stippling and adapt to changes in illumination at a given viewing distance, it is not clear that stroke textures behave appropriately for stippling during scaling. Because textures are parameterized to the surfaces, images on the texture scale according to the surface, but in stippling we need to maintain stipples with a constant or a maximum size (of a few pixels) at all times during an animation. Since in our system we use dots as primitives, keeping the correct stipple size is not a problem, because only the particle location is bound to the model and not its size.

2. System Implementation

2.1. Overview

Our rendering system does the following steps:

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 that appear when the vertices of the input mesh are taken as locations for the stipples.
3. Perform mesh simplification on the input mesh[8] and simultaneously create a bottom-up hierarchy for the vertices in the input mesh[3].
4. Render each frame of the animation using a series of key frames, interpolating values between each key frame to set the viewing parameters for each individual frame. These viewing parameters include the camera position and orientation, illumination and frame-rate among others, and determine whether more refinement is needed to fill in dark regions of the model.
5. Assemble the rendered frames in an animation file.

The information of each frame is passed to the renderer which performs the following steps:

1. Test the model for refinement.
   If the required level-of-detail does not satisfy the expected particle density, the model is refined until the appropriate level of detail is obtained. Simultaneously, existing vertices are tested for frustum and backface culling, and new vertices are assigned a set of neighbours creating a hierarchy for these vertices as well.
2. Test each potentially visible vertex for rendering.
   Each vertex has a list of relevant neighbors, previously defined either by refinement or simplification operations. Each edge departing from the vertex to a relevant neighbor vertex is compared against a threshold value. If all the vertices of the stipple exceed the threshold value, the stipple is set to be rendered, otherwise it is turned off. The threshold value is computed by using the vertex normal and the illumination parameters.

In the following sections we describe several aspects of our implementation in more detail.

2.2. Setup

After loading up the model a copy of the connectivity graph is created using the LEDA^{TM}library and the initial particle distribution is the collection of vertices of the input mesh. Input models contain arbitrary point distributions which sometimes are quite regular, depending on the algorithms or procedures used to create the models. To avoid the appearance of regular patterns, we first perform "randomize and project" operations on all the original vertices of the model. The result is a mesh which is topologically the same as that of the input model, but slightly different in geometry. The randomize and project operations are also applied after a new vertex is obtained by mesh subdivision. We describe them in more detail in section 2.4.

2.3. Creating the Point Hierarchy

According to Hodges[17], artists can produce stippled drawings by first placing some groups of dots in a region of interest and then filling in until the desired tone is achieved. In our system we follow the same strategy by creating a spatial hierarchy of vertices which represent stipple locations on the surface of the model. 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 stipples always come up to fill-in uncovered regions of the canvas until the desired tone is achieved.

The stipple particles are vertices of a polygonal mesh, used for simplification and subdivision. For input models at high resolution, we simplify the mesh and create a bottom-up hierarchy, as is done by Cornish et al.[3]. For input models with few vertices we usually need to generate more vertices to achieve the desired tone, because a stipple dot covers only a minimal number of pixels in the rendering area; this is done by mesh subdivision. To ensure that the appropriate level of detail is obtained at every frame, we mix mesh simplification (as a preprocessing stage) and mesh subdivision (performed at every frame if necessary) and provide seamless levels of detail regardless of the resolution of the input model (Figure 2). When the viewer gets closer to the object, additional stipples are required to keep the target shade. The rendering pipeline described above adaptively performs mesh subdivision whenever the stipple density falls below the desired level. If the user gets very close, the system generates a very large amount of geometry (which can easily reach levels of millions of vertices). This has an effect on overall system performance, and accounts for longer rendering time.

2.3.1. Mesh Simplification

In mesh simplification we create a bottom-up vertex hierarchy by applying a series of edge collapse operations until the model is simplified to a few vertices. The operator that we use for mesh refinement is the *edge collapse* (ecol), as defined by Hoppe[8], shown in Figure 3, top.

By performing mesh simplification we can take models of complex geometry (like horses, Stanford bunnies or dragons) and render them with few stipples by using the vertices at the top of the hierarchy (see Figure 1).

2.3.2. Hierarchical Subdivision

We generate a hierarchy by subdividing (refining) an input 3D model as follows. For each frame rendered we enforce the required level of refinement by comparing the screen-space projection of each edge against a varying screen space threshold set according to the desired darkness at the vertex (computed from either vertex of the edge); if the edge exceeds the threshold, the model is refined. At each refining step, the longest edge in object space is subdivided. The longest edge in the mesh is taken from a BSTree containing all the active edges sorted by their length in object space. The operator that we use for mesh subdivision is an edge-split that creates a point around the middle point between two vertices of an edge (see Figure 3, bottom).

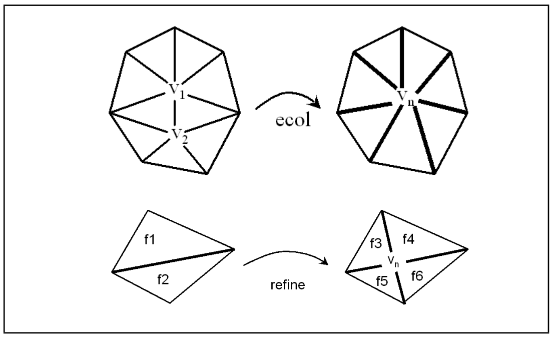

Figure 3: *Top: The edge collapse operation used in mesh simplification.*[8] *Bottom: The refinement operation used in mesh subdivision. The dark edges are saved in the list of relevant neighbours of the resulting vertex.*

Each vertex obtained by subdivision indicates the location of a new stipple, and its neighbours are included in the list of relevant neighbours for later use during rendering. The model refinement loop stops when the length of the edge that caused the model refinement does not exceed the threshold any more. We perform refinement in this order to avoid creating extremely thin triangles, which would appear if only a specific region of a model is refined. The algorithm checks at every frame that the level of detail necessary has been reached. This creates an overhead for rendering time, but is necessary to ensure that the required darkness is obtained at every frame. Figure 4 shows the mesh after refinement for the Utah Teapot.

2.3.3. Defining the Spatial Hierarchy

Since a stipple covers an area of influence delimited by the stipples in its neighborhood,[4, 18] we define a relevance function based on a list of neighbors to determine whether a pixel should appear or disappear during an animation. After applying either the edge-split or the edge-collapse operators we register the list of neighbouring vertices of the resulting vertex (the vertices of the polygon fan which surrounds a vertex) in a data-structure that is used during rendering to decide whether or not a vertex should be drawn as a stipple. Figure 3 shows the relevant edges (as thick lines) for a given node after a refine or simplify operation takes place. A particle is drawn depending on the desired darkness at the vertex and the length of the edges connected to the vertex in screen space (see section 2.5). This creates an implicit scaling function, which resembles the random scaling function in Kaplan et al.[13], but is more similar to the test for edge collapse presented by Hoppe[8].

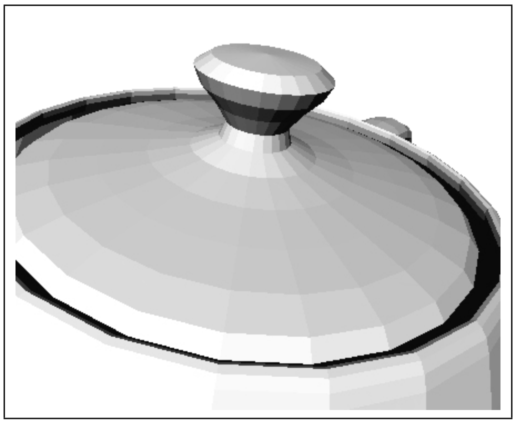
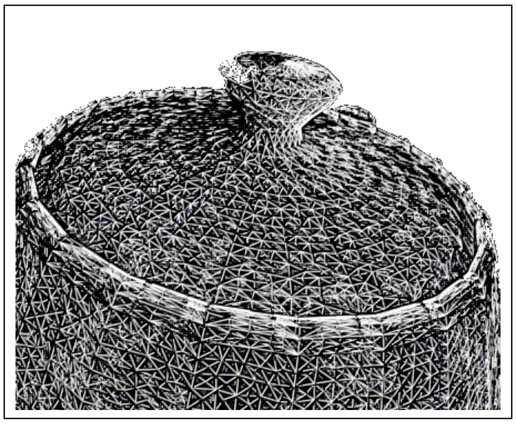

Figure 4: *Right: close-up of the teapot in flat shading rendering mode. Left: Hidden-lines view of the teapot after applying a series of refinement operations.*

Since each level of refinement has neighbours at different distances, we implicitly create a hierarchy of points by measuring the lengths of the edges around a vertex. Points with shorter edges are drawn only after points with larger edges have been tested, for an evenly shaded polygon. Additionally, points down the hierarchy appear or vanish between existing points.

2.4. Randomization and Projection

A problem that arises when the vertices of the original mesh are used to place stipples on the model is that 3D models exhibit significant regularity in the distribution of its vertices. This may happen for several reasons related to how 3D models are generated. This situation is more apparent under certain situations, for example when the shading tone set exactly the stipples that correspond to the vertices of the input mesh. Moreover, our subdivision operator also generates regular point distributions, since it creates vertices along existing lines on the model. For these reasons, we have a randomize phase applied to all vertices of the input model right as part of the set-up process and after each subdivision operation, so that new vertices do not align with existing ones.

2.4.1. The Randomize Operator

The randomize operator receives as input the vertex to be moved and its connectivity information. A face connected to the vertex is selected at random, and a new point is randomly generated on the surface of this face. We restrict the location of the new point so that the new vertex is located in a range of 1/8th to 1/4th of the distance between the original vertex and the two other vertices of the triangle. Finally, the location of the input vertex is updated to the new position (see Figure 5). The range values are chosen to ensure that the geometry of the input mesh is roughly kept (maximum displacement), and that the new point is not left at the original location (minimum displacement).

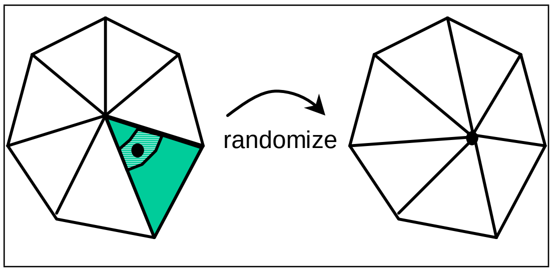

Figure 5: *The randomize operator. On the left, the original mesh and the new location of the input vertex are shown. On the right, the mesh after the operation.*

Whenever a randomize operation is performed on a vertex, the new location of the vertex is not guaranteed to lie on the surface of the input model, because the faces of the particle mesh do not necessarily fit to the surface of the input model, only its vertices. The only case where the new vertex is guaranteed to lie on the surface of the input model is when all the vertices if the polygon fan around it are coplanar. Otherwise, the vertex has to be projected back on the surface using the projection operator described in the next section.

Figure 6 illustrates the effect of the randomize operator on the overall stipple distribution. The image on the left shows a point distribution obtained by applying only the subdivision operator on a flat mesh with 7 vertices, where some stipples are arranged in linear distributions. The image on the right shows the stipple distribution obtained when the randomize operator is applied after each subdivision.

2.4.2. The Projection Operator

Since the edges of the particle mesh are not necessarily aligned to the surface of the input model, it is possible that

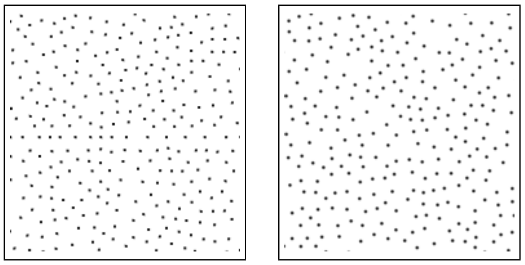

Figure 6: *Stippled patterns before (left) and after (right) applying the randomize operator on a flat shape.*

new vertices created by mesh subdivision do not lie on the surface of the model, and we need to project them back to this surface. The same can occur after a randomize operation, since a vertex may be displaced to a face which is not aligned to the surface of the model. To counter this, we use the projection operator shown in Figure 7. The operator first

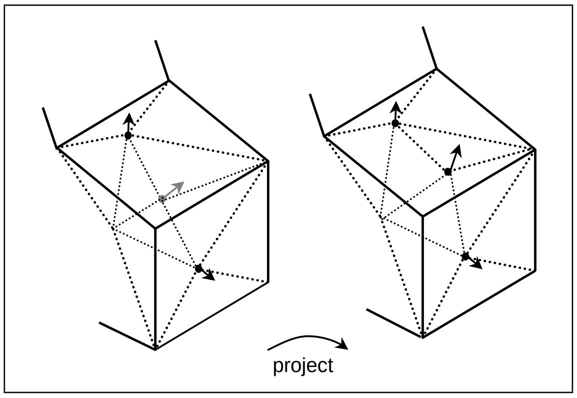

Figure 7: *The projection operator. Black lines belong to the input model, stippled lines belong to the particle mesh and dark stippled lines lie on the surface of the input model.*

defines a projection ray departing from the input vertex in direction of the normalized sum of normals of the vertices in the neighbourhood of the input vertex. After that, we test for intersections on the set of faces of the input model that are in the neighbourhood of the vertices in the polygon fan of the input vertex. When an intersection has been found the vertex is moved to the intersection point.

2.5. Rendering

After adaptive refinement has taken place, we proceed to render the potentially visible points that successfully pass a rendering test similar to the *edge collapse* test used by Hoppe for view-dependent mesh simplification[8].

The rendering test works as follows: The screen-space projection of every edge in the list of neighbouring nodes (computed for every node after mesh simplification or subdivision) is measured in pixels and compared against a dynamically computed threshold value. If a connected edge falls below the threshold value, the stipple is unset. If all edges exceed the threshold, the stipple is set to be drawn. The threshold value is computed for each particle as a function of the illumination model, the normal of the vertex and the viewing parameters.

In our rendering system we have stipples of variable point size. A stipple can have a value from 0 to 2.4 pixels of size. Stipples blend in the image when they are set, and grow in size at each frame, until they reach their maximum size. Conversely, a fade-out effect for the unset vertices is achieved in our system by smoothly decreasing their point size until it becomes zero. We use OpenGL smoothing for points, blending and multipass antialiasing to ensure that stipples are smoothly introduced in or removed from the images during animation.

3. Results

In Figure 8 we present frames taken from the stippling animations, where models are shown under different conditions of lighting, viewing and model orientation. The complete animations are available under http://isgwww.cs.uni-magdeburg.de/~oscar/. The rendering times for the initial frames of the animation (when most of the geometry is created) are 2 minutes for the sphere-model and 3 minutes for the teapot. Rendering each frame without refinement and saving into a file takes approximately 5 seconds. In average, 360 frames of animation are produced in one hour on an SGI Onyx2 Infinite Reality, with 2 195MHz MIPS R10000 processors and 900MB ram.

Better frame-rates can be obtained if we limit the amount of stipples available for a model, by only refining the model up to a maximum number of vertices. However, the drawback of such an approach is that dark areas loose darkness (i.e. the stipple density on the image decreases) when the viewer gets close to the model.

Due to time constraints, we were not able to produce objective metrics to evaluate the quality of the work presented here. In particular, it is possible to evaluate the quality of the stipple distribution following guidelines from the area of digital halftoning. With regards to frame-coherence, the animations presented show smooth transitions between frames, and how stipple dots emerge and disappear between existing dots, yielding an interesting visual effect, as if sand particles emerged or disappeared from the surface of the model, but on the other hand, they remain attached to the surfaces as the model moves.

To our knowledge, our approach is the first to solve the problem of frame-coherent stippling, but more work needs to be done to improve rendering times and the quality of the stipple distribution.

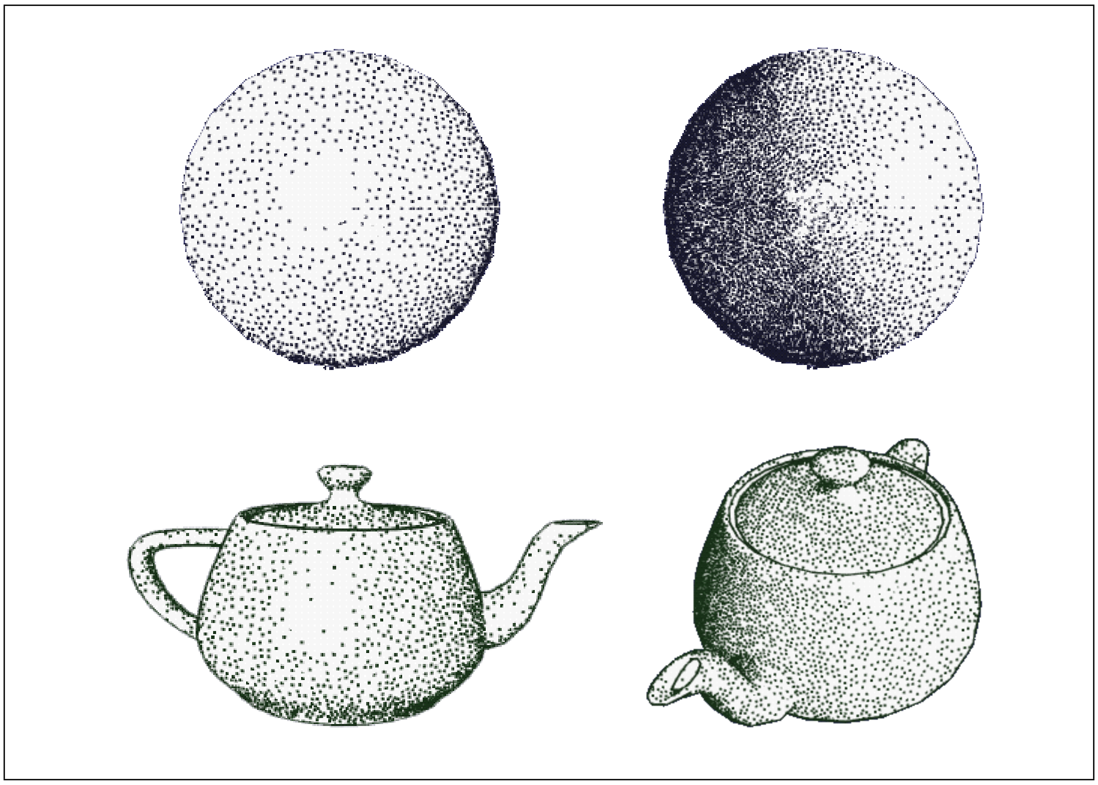

Figure 8: *Frames from our stippling animations, illustrating changes in the position of the light (top right) and changes in the position of the light and the model itself (bottom right).*

4. Conclusions and Future Work

In this paper we have presented the first system to produce frame-coherent stippled drawings of 3D polygonal models. Our system ensures frame-coherence at the stipple level, i.e. every stipple shown in a frame of a video sequence appears in a corresponding location on the next frame. We introduce a mix between mesh subdivision and mesh simplification to obtain seamless levels-of-detail on demand. Higher levels-of-detail, produced by mesh subdivision are used to fill-in the darker regions of the model. Lower levels of detail, produced by mesh simplification, permit removal of vertices at levels-of-detail lower than that of the input model and are used to stipple lighter shaded regions in the image. In the system presented, new stipples always fill-in the spaces between existing stipples, and smoothly blend-in or fade out of the image according to illumination and viewing parameters. Randomization and projection operators are used to improve the distribution of the stipples and ensure these are always mapped to the surface of the input model, respectively. Our algorithm is still in development, and work is under way to improve the rendering times by simplifying the evaluation for rendering.

Areas of extension for this work include:

- Optimization of the algorithms.
  The work presented here is still under development, and we have to improve the algorithms and operators presented here to achieve optimal rendering times.
- Stippling and point-based rendering.
  An interesting area of extension is the fusion of the work in point based rendering[1] and the work here presented. The parallels between both approaches indicates that it is possible to obtain frame-coherent stippling in real-time for static models.
- Objective evaluation of the particle distribution.
  We need to evaluate the quality of the particle distributions in the images, using guidelines from digital halftoning[20].
- Modification of the algorithms to allow 3D warping and morphing.
  One of the most interesting areas of extension for this work is the development of a hierarchical particle system for a model subject to morphing transformations, which can be obtained by allowing for relative coordinates definition, by using barycentric coordinates to define the stipple locations.
- Extending the system to take input models with grey scale textures and bump mapping.

Texture and bump mapping information can be used to determine the target tone of the surfaces and control shading with more detail.

Acknowledgments

The authors would like to thank Lourdes Peña Castillo for reviewing drafts, the members of the Institute for Simulation and Graphics at the University of Magdeburg, specially Nick Halper, for providing important feedback in the development of this technique. This work was supported by a scholarship of the state of Sachsen-Anhalt, Germany.

References

1. M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva. Point set surfaces. In *Proceedings of the IEEE Visualization Conference*, pages 21–28, 2001. http://www.igd.fhg.de/~alexa/paper/index.html. 3, 7
2. P. Alliez and M. Desbrun. Progressive compression for lossless transmission of triangle meshes. In *SIGGRAPH 2001 Conference Proceedings*, pages 195–202. ACM, ACM Press, 2001. http://doi.acm.org/10.1145/383259.383281. 3
3. A. R. Derek Cornish and D. Luebke. View-dependent particles for interactive non-photorealistic rendering. In *Proc. of Graphics Interface 2001*. Graphics Interface, 2001. http://www.graphicsinterface.org/proceedings/2001/158/. 2, 3, 4
4. O. Deussen, S. Hiller, C. van Overveld, and T. Strothotte. Floating points: A method for computing stipple drawings. In *Computer Graphics Forum, Vol. 19(3), S. 40-51*. Eurographics Conference Proc., 2000. http://www.eg.org/EG/CGF/volume19/issue3. 1, 2, 4
5. B. Freudenberg. Real-time stroke textures. In *SIGGRAPH 2001 Conference Abstracts and Applications*, page 252. ACM Press, 2001. http://isgwww.cs.uni-magdeburg.de/~bert/. 3
6. M. Garland. Developer's survey of polygonal simplification algorithms. In *Eurographics '99, State of the Art Report (STAR)*, 1999. http://graphics.cs.uiuc.edu/~garland/papers.html. 3
7. B. Gooch and A. Gooch. *Non-Photorealistic Rendering*. A. K. Peters, Ltd., July 2001. 3
8. H. Hoppe. Progressive meshes. In *SIGGRAPH 96 Conference Proceedings*, pages 99–108. ACM, 1996. http://doi.acm.org/10.1145/237170.237216. 3, 4, 6
9. M. A. Kowalski, L. Markosian, J. D. Northrup, L. Bourdev, R. Barzel, L. Holden, and J. F. Hughes. Art-based rendering of fur, grass, and trees. In *SIGGRAPH 99 Conference Proceedings*, pages 433 – 438. ACM, 1999. http://doi.acm.org/10.1145/311535.311607. 1
10. D. Luebke. Developer's survey of polygonal simplification algorithms. In *IEEE Computer Graphics & Applications/ (May 2001).*, May 2001. http://www.cs.virginia.edu/~luebke/publications.html. 3
11. D. Luebke and C. Erikson. View-dependent simplification of arbitrary polygonal environments. In *SIGGRAPH 97 Conference Proceedings*, pages 199 – 208, 1997. http://doi.acm.org/10.1145/258734.258847. 3
12. L. Markosian, B. J. Meier, M. A. Kowalski, L. S. Holden, J. Northrup, and J. F. Hughes. Art-based rendering with continuous levels of detail. In *Proc. of the NPAR Interational Conference 2000*, Annecy, France., 2000. Eurographics. http://doi.acm.org/10.1145/340916.340924. 2, 3
13. B. G. Matthew Kaplan and E. Cohen. Interactive artistic rendering. In *Proc. of the NPAR Interational Conference 2000*, Annecy, France., 2000. Eurographics. http://www.cs.utah.edu/npr/papers.html. 2, 3, 4
14. B. J. Meier. Painterly rendering for animation. In *SIGGRAPH 96 Conference Proceedings*, pages 477 – 484. ACM, 1996. http://doi.acm.org/10.1145/237170.237288. 1, 3
15. V. Ostromoukhov. Pseudo-random halftone screening for color and black&white printing. In *Proceedings of the 9th Congress on Advances in Non-Impact Printing Technologies*, pages 579–581, 1993. 2
16. E. Praun, H. Hoppe, M. Webb, and A. Finkelstein. Real-time hatching. In *SIGGGRAPH 2001 Conference Proceedings*, page 581. ACM Press, 2001. http://doi.acm.org/10.1145/383259.383328. 3
17. E. R.S. Hodges. *The Guild Handbook of Scientific Illustration*. Wiley Europe, 1988. 1, 4
18. A. J. Secord. Weighted voronoi stippling. In *Proc. of the NPAR Interational Sysmposium 2002*. Eurographics, 2002. 1, 2, 4
19. T. Strothotte and S. Schlechtweg. *Non-Photorealistic Computer Graphics: Modeling, Rendering, and Animation*. Morgan Kaufmann Publishers, April 2002. 3
20. R. Ulichney. *Digital Halftoning*. MIT Press, Cambridge, 1987. 2, 7
21. G. Winkenbach and D. H. Salesin. Computer-generated pen-and-ink illustration. In *SIGGRAPH 94 Conference Proceedings*, pages 91 – 100. ACM, 1996. http://doi.acm.org/10.1145/192161.192184/. 1, 2, 3
22. G. Winkenbach and D. H. Salesin. Rendering parametric surfaces in pen and ink. In *SIGGRAPH 96 Conference Proceedings*, pages 469 – 476. ACM, 1996. http://doi.acm.org/10.1145/237170.237287. 3