Possible Techniques for Three Dimensional Hatching

Vladimir Savchenko*, Hiroshi Unno*, and Nikita Kojekine**

*Hosei University, **Tokyo Institute of Technology

Abstract

This paper introduces a basic technique for drawing non-photo realistic images of volume models. The turtle graphics approach to imitate painting operations is implemented in 3D space. Objects are defined by sets of scattered range data. Analytically (that is; functionally represented) and procedurally defined geometric objects can also be used. The proposed technique is described in detail and painting examples that demonstrate its applicability to non-photo realistic rendering or hatching are provided.

Keywords: turtle graphics, volume modeling, computer art, hatching

1. Introduction

This paper presents work in progress, a project devoted to developing a system for visualization of implicitly defined objects with free shapes. The work was stimulated by previous investigations in the fields of computer graphics and shape modeling, and by recent significant interest in non-photo realistic painting, see for instance [1], [2]. The motivation for this project stems not only from applications such as computer art. Our intention is to develop a painter, which allows computer graphics (CG) novices to make sketches of three-dimensional objects. Thus, we can formulate our second goal as an educational one.

The 1990s saw a growing interest in the development of algorithms and systems for modeling and painting of constructive solid geometry (CSG), 3D parametric surfaces and polygonally defined objects (see [3], [4], [5], [6], [1], [7]). The key idea in these works is to use geometric characteristics such as curvature to provide trajectories of the particles or brushes. Our main focus is to try mimicking the motion of an artist's brushes.

Recently, virtual reality (VR) research has started to focus on the capabilities of humans with respect to the sensing of virtual objects. The Free Form Modeling system developed by SensAble Technology Company, which lets sculptors and designers use their sense of touch [8], is an encouraging example, for more references see also [9]. In presented work we give a brief report on the progress of our GUI, which allows the user to paint 2D strokes on 3D object by using mouse or tablet tools.

The object geometry and rendering used in this project are based on the so-called volume-modeling paradigm. By volume modeling, we mean that the system's architectural decisions are based on the dual volume representation by voxel data; scattered range three-dimensional coordinates converted into voxel data and by real-valued functions in particular.

From a geometrical point of view the problem considered can be stated in terms of allowable transformations or as a one-to-one mapping (see [10] and the references therein) of a portion of a surface S onto a portion of a surface S^*. Of particular practical interest are mappings that preserve certain geometric properties such as the length of every arc (in our case the length of line segments on the plane, for instance) on a surface and angle-preserving mappings, which preserve the angle between every pair of intersecting curves on a surface. Some fundamental theorems (see [10]) state that an allowable mapping of a portion of a surface S onto a portion of a surface S^* is possible only for developable surfaces. In this work, we shall try to find a practical way to preserve the above-mentioned geometric properties for arbitrary surfaces as far as possible, and follow a real painter's approach to painting a surface in order to obtain an "allowable" mapping. In addition, we present some experimental examples of painting volume objects.

Undoubtedly, the VR painter should be able to paint, see, and define an object as if she were painting a real object. Many authors tried to produce digital sketches, nevertheless simple geometric objects look too striped, see for example [6], while more sophisticated parametric surfaces attached to the 3D objects like human face do not produce desirable results. Our goal is very precise: reproduce a geometric object (e.g. a model of a human head, for instance) starting from scattered sets of range data. The resulting image should be visually pleasant, and the object (person) must be recognizable. One of the fundamental problems with non-photorealistic painting is computing visibility. In our

implementation we use a sufficiently simple approach: change of the normal is bounded by a fixed value, which cannot be negative. In practice, we have found that the algorithm performs well near silhouettes. Naturally, it cannot guarantee exact reconstruction of the silhouettes; an efficient algorithm would be useful.

Kunii and Shinagava in [11] proposed to model art in three steps:

- "The first step in scientific understanding of various phenomena is to observe them in reproducible ways."

- The second step is "the construction process."

- "The testing of the model against the hypothesis is the last step."

Actually, according to the "three-step model" we would like to raise questions, such as: is it absolutely clear that an artistic sketch should demonstrate strongly defined and closed silhouettes; should the rendered strokes follow isoperimetric curves. Thus, one more goal of the project is to provide a proof by visual inspection of the correctness of the model – painting a surface in order to obtain an "allowable" mapping –.

2. Configuration of the painter

The painter combines two processes:

- **Main process**
  - *-Scene creation*
  - *-Assignment of an initial motif or polyline segment, which will influence the line length, direction and width*
  - *-Turtle graphics tiling*
  - *-Redrawing of a specified area*
- **Core application**
  - *-Turtle graphics state update*
  - *-Turtle graphics control*

The speed limitations of hardware impose many restrictions on a developed painter, and compel us to consider some technical questions about model representation. This includes rendering, which exacts a high cost in computer performance.

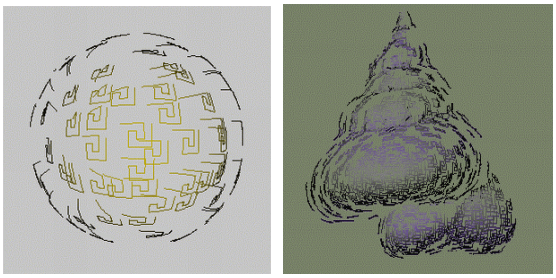
One of the primary goals is to develop functions that are able to calculate the positions of 3D points (locators) somewhere on the model being painted and allow data to be displayed in the main process. We have to satisfy major requirements such as fast locator calculations to maintain fast data rendering in a prescribed area.

3. Geometric model

Data used in this project are scattered range data. A volume is represented as an interpolated 3D data set of voxels and can be thought of as a subset of 3D space with additional scalar values given for processed points. Analytically (functionally represented) and procedurally defined geometric objects can also be used.

For visualization, interactive ray tracing is used. Ray tracing may take quite a long time, since it invokes the function evaluation for each ray that is cast from the observer position towards the object through each pixel. To accelerate this process, only parts (lines) affected by the most recent painting are redrawn.

Collision detection is a fundamental problem in 3D interactive applications. At this stage of the project we neglect the actual size of the painting tools and use a ray marching algorithm to search for the intersection of a straight line, which we traverse in the "forward" direction until an "entering" intersection is found. That is, the locator point defined in the vicinity of the surface is moving from the outside to the inside of the object according to the gradient vector direction. Each intersection point is added to the output "skeleton" list. This list is then used to paint the virtual object in the main process before rendering. Painting of a sub region of the object is made according to the "skeleton" list corresponding to the coordinates for the initial and destination points of the linear segment – for instance, a meander shown in Figure 1.



(a) (b)

Figure 1. (a) Imitation of meanders painted on a "pure" sphere. (b) Meanders on a "seashell".

4. Application programming, turtle graphics in 3D, and implementation

Application-programming C++ and Java libraries were developed to support 3D turtle graphics methods used for volume objects. The libraries contain classes that provide the functions, which handle the pixel operations and transmission of data to display an image; and classes that define GUI components to provide functionality for the turtle graphics; in other words, classes that make it possible

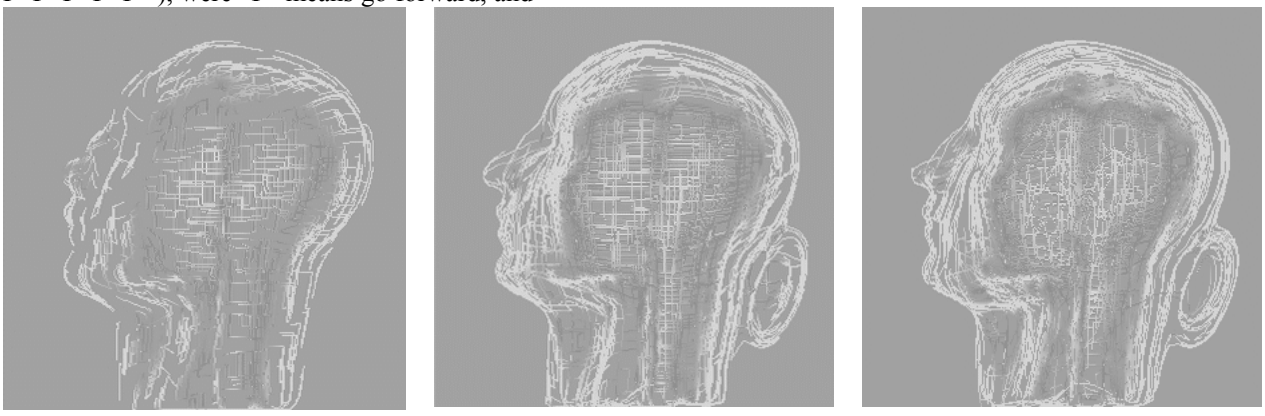
to draw images, to construct objects, and to modify them. Also there is a library, which supports converting of scattered data into volume representation. It is based on using compactly supported radial based functions proposed in [12]. This library [13] contains classes to construct variable-depth octal trees for space subdivision, to build and solve a system of linear algebraic equations with a banded matrix, and to produce a surface evaluation and extraction.

A large family of plane figures can be generated easily with the turtle graphics (see [14] and the references therein). A very simple graphical interface is based on Seymour Papert's concept of turtle graphics as a part of the LOGO language for teaching children some elements of geometry and programming [15]. The notion of "turtle" is similar to a turtle facing in a certain direction and migrating over the page. It leaves a trace on the paper.

To control the turtle we need four methods (see, for example, [14]): the turnTo() method, which turns the turtle to face the given direction; the turn() method, which turns the turtle through an angle; the forward() method, which moves the turtle forward in a straight line from the current position through a specified distance in the current direction; and the lineTo() method, which draws a line and updates the current position. Implementation of the turtle graphics provides a natural way to do painting in 3D, but is not quite straightforward from a geometrical point of view.

Let us consider painting of meanders. A meander is often made up of a line passing along some path according to a motif, which can be described by using symbol strings as follows: String ("F-F-F-F+F+F+F+F-"), were "F" means go forward, and "+" and "-" mean right and left turns, respectively, according to the angle, which controls the drawing of a line in the current direction. After the motif has been drawn we can repeat the drawing of the next motif. The example in Figure 1(a) shows meanders painted on "pure" sphere primitive. The model in Figure 1(a) demonstrates the possibility of reproducing "straight" painting furrows. An attempt to draw straight lines being in a "collision state" with noisy objects (see Figure 1(b)) leads to curved or noisy furrows that reflect surface roughness. Features of volume representations are very suitable for mimicking painting effects. There has been much recent interest in the development of techniques for creating artistic or non-photorealistic rendering methods for painting implicitly represented solids. The main idea is to use difference equations to provide the trajectories of the particles (brushes or cutting tools); see the pioneering work of Witkin and Heckbert [16]. Any vector perpendicular to the surface gradient vector will be on the surface, and it is possible to make the particles float over the surface. Akleman [3] simplified Witkin and Heckbert's differential equations and conducted experiments with various CSG objects for various functions describing the brush motion.

In the ongoing project we use a different approach, based on the motion of an artist's brushes.



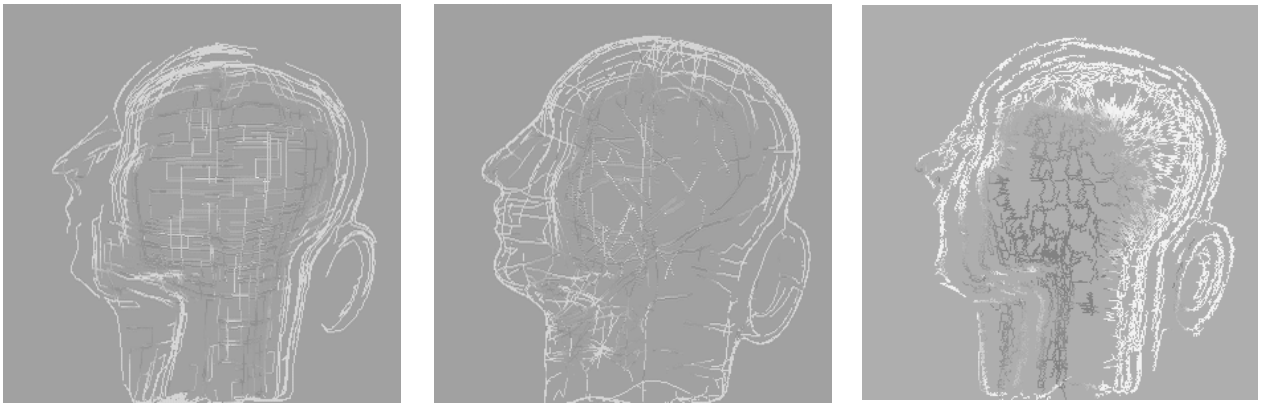


Figure 2. Examples of hatching fractal-like objects with different levels of details on a volume "head" represented by 1487 scattered 3D coordinates. For the first image of "meanders" hatching, model was slightly changed.

Our most important finding from observation of artists is the following: to produce the desired look over the boundaries of a geometric object such as a plate or a bowl; the painter rotates the object before drawing each instance or motif of the image, possibly with different sizes and orientations. In some sense, we imitate the approach mentioned above by adding in the turtle graphics method lineTo() a modeling transformation function in order to define a coordinate system called the master coordinate system. Consider painting of a meander pattern headed in 90 degrees direction. If the meander (motif) object is given, before starting to paint,

- We define a point on the object to be painted. On this stage of the project we define the points by the Sobol's quasi-random sequence [17]. The sample points in a quasi-random sequence are "maximally avoiding" each other, which allows us to obtain a sufficiently uniform display of arriving furrows. The point defines the origin of the master coordinate system in accordance to a gradient vector. In fact, another technique based on analyses of main curvatures, ridges and ravines of the object can be used. It was the main rule for creating facial sketches to define directions that should somehow follow facial features.
- A line segment is then scaled and x or y-step on the meander fragment is calculated. This step actually defines a floating step.
- The plane passing through the starting point and perpendicular to the gradient vector is then calculated.
- Next, the plane passing through the starting and the destination points on the line segment is calculated.
- The line of intersection of the planes defines the floating or motion of the locator and new skeleton points are calculated by using the ray marching method, taking into account the gradient direction and the floating step.
- The main process redraws the subimage. The new calculated brush position defines the new starting point, and the process is repeated for the next meander's segment.

Such a simple approach allows us to mimic the artist's way of working, and obtain painted pictures with a reasonable visual appearance in a reasonable time. The user can define a motif such as the meander; nevertheless, we illustrate the proposed approach by painting fractal-like objects.

We developed the painting algorithm within a system, which enables the user to interactively draw fractal-like curves on the surface by using the mouse cursor on the screen. The screenshot of the "Painter" interface is shown in Figure 3. With our system the user can also draw arbitrary curves on the surface by dragging pointing devices such as mouse and tablet. An example of hand drawing is shown in Figure 4. The color and the distribution of curves can be controlled according to the angle between the surface normal vector and the view vector that is defined as a vector from the surface position to the eye position. In addition to previously described drawing capability, our system has other functions as follows:

- Saving and loading the edited symbols of the string and/or a hand-drawn curve to reuse them later.
- Different color and size choices for the pen that is used for drawing the curve.
- Setting noise parameters for the step size and the current direction of the curve segment.
- Ray tracing the surface and synthesizing a background image as a model, in order to inspire users' imagination of curve drawing.

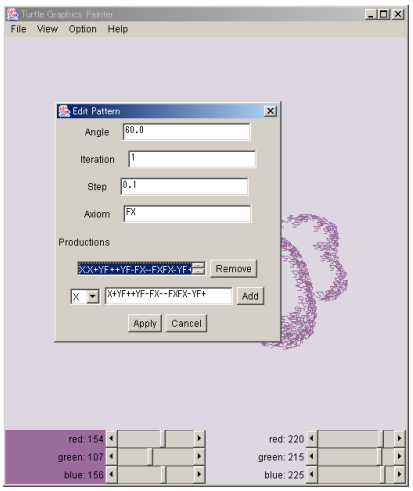


Figure 3. The "Painter" program interface.

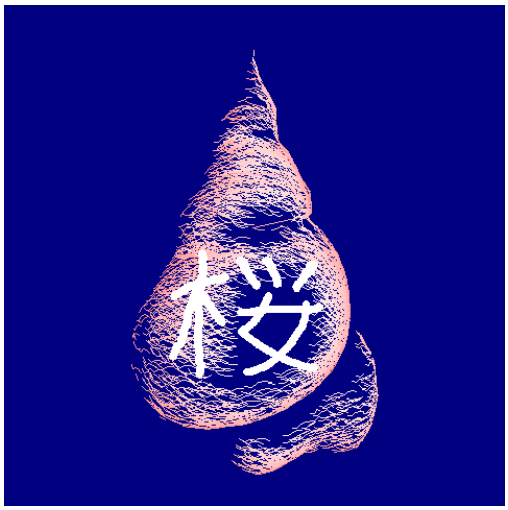


Figure 4. Illustration of "hand drawing" on the 3D "seashell", where we first drew 5,000 fractal-like curves on the seashell at quasi-randomly generated screen positions, and Kanji with meaning of cherry blossoms was drawn by hand.

Recently the CG community has acknowledged various applications of fractal-like objects and of the self-similarity demonstrated by nature (see Mandelbrot [18].) Many different approaches to generating fractals have been developed; in particular, one approach to drawing complex curves is based on a simple set of rules [19]. The string-production process produces the next highest-order string object and usually a recursive version of the string-production process is used to generate a new string object. In the fractal drawing example, there are two main applications: one that produces strings and another that draws an image of fractal-like objects on volume objects (see Figure 5).

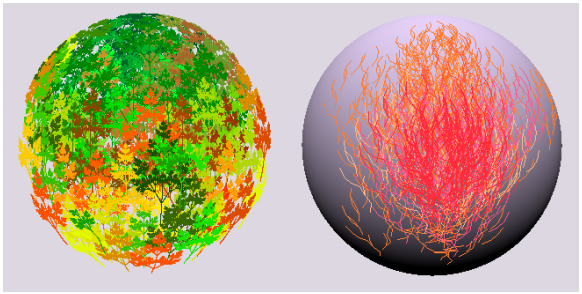


Figure 5. An example of hatching with fractal-like curves.

Figure 6. An example of hatching with Gosper curve.

As random examples, we used the so-called Dragon, Hilbert, Gosper curves and the Sierpienski arrowhead; see Figure 2. The first image shown in Figure 2 is a meander painted on a head model represented by 1487 scattered 3D points. The Next examples demonstrate "fractal-like" hatching, but for a slightly changed geometric model. The painted images shown in Figure 2 give us an important insight. The variety and the length of the furrows depend on a string producing process. Some pictures exhibit excessively long furrows as a result of the string producing process, especially for a relatively flat central part of the head. Nevertheless, all images are visually pleasant, and the object is recognizable.

Moreover, increasing a string-order allows the user to produce a reasonable sketch in few steps, after that the user can cancel the painting process. The system also allows the user to control the length of separated line segments for producing dash-like furrows (last image in Figure 2).

From our point of view, use of Gosper curves generated the most visually appealing image. It is shown in Figure 2 after few steps (first image in the second row) and in Figure 6 after many steps of calculations. Many artistic images are known to possess certain conventional symmetry element rules. Possibly, some local symmetry exhibited by Gosper curves might create an aesthetic impression.

5. Remarks

We have presented research in progress, in which we are considering to add advanced features of volume representation to simulate painting operations for 3D geometric objects. In particular, a prototype system was developed and tested for 3D painting operations that are considered as straightforward extensions of the turtle graphics approach.

The main goal of the ongoing project is to understand the aspects of visualization with regard to painting of implicitly defined object, by using turtle graphics. We hope that the proposed approach that produces realistic looking sketches and lessons that we have learned during the development of the painter, will help in the future design of computer art application systems. Although this paper uses fractal like patterns, one ambiguous goal would be to simulate an artist's hand motion.

References

[1] A. Hertzmann and D. Zorin, Illustrating Smooth Surfaces, *Proceedings of SIGGRAPH`2000*, 517-525, 2000.

[2] E. Praun, H. Hoppe, M. Webb, and A. Finkelstein, "Real-Time Hatching", *Proceedings of SIGGRAPH`2001*, .581-586, August, 2001.

[3] E. Akleman, Implicit Painting of CSG solids, *CSG'98 Set-Theoretic Solid Modeling: Techniques and Applications*, Ammerdown, 99-113, April 1998.

[4] S. Wang and A. Kaufman, Volume Sculpting, *The 1995 Symposium on Interactive Graphics, ACM SIGGRAPH*, 151-156, 1995.

[5] J.A. Baerentzen, Octree-based Volume Sculpting, *Proceedings of VIS'98 conference*, Eds. C.M. Wittenberg and A. Varsney, Research Triangle Park, North Carolina, 9-12, Oct. 21-23 1998.

[6] G. Elber, Line Art Illustrations of Parametric and Implicit Forms, *IEEE Transactions on Visualization and Computer Graphics*, **4**(1), 71-81, 1998.

[7] Y. Ohtake, M. Horikawa and A. Belyaev, Adaptive Smoothing Tangential Direction Fields on Polygonal Surfaces, *Proceedings of Pacific Graphics'2001*, Tokyo, 189-197, 2001.

[8] http://www.sensable.com/

[9] M. Kimura, K. Tanaka, N. Abe and H. Taki, 3-Dimensional Painting System with Tactile Sensation, *Proceedings of ICAT'99*, Waseda University, Tokyo, Japan, December 16-18, 137-141, 1999.

[10] E. Kreyszig, *Differential Geometry,* Dover Publications, Inc., New York, 1991.

[11] T. L. Kunii and Y. Shinagava, Science of Art – Sciences of Painting, Sports, Stage Arts and Beauty of Nature–, *Forma*, 9, 193-194, 1994.

[12] H. Wendland, Piecewise polynomial, positive defined and compactly supported radial functions of minimal degree, *AICM*, **4**, 389-396, 1995.

[13] http://www.karlson.ru/csrbf/

[14] F.S. Fill, JR., *Computer Graphics Using OpenGL*, Prentice Hall International, Inc., 2001.

[15] H. Abelson and A.A. Siessa, *Turtle Geometry*, Cambridge, MA, MIT Press, 1981.

[16] A.P. Witkin and P.S. Heckbert, Using Particles to Sample and Control Implicit Surfaces, *Computer Graphics*, **27**(3): 269-277, 1994.

[17] W.H. Press, S.A. Teulkolsky, W.T. Vettering, and B. P. Flannery, *Numerical Recipes in C*, Cambridge University Press, 1997.

[18] B. Mandelbrot, *The Fractal Geometry of Nature*, New York, Freeman, 1983.

[19] A.R. Smith, Plants, Fractals, and Formal Languages, *Proceedings of SIGGRAPH`84*, 1-10, 1984.