Computational Aesthetics in Graphics, Visualization, and Imaging
Paul Rosin (Editor)

EXPRESSIVE 2015

Painting with Flowsnakes

Brian Wyvill[1]

[1]University of Victoria, Dept. of Computer Science, BC, Canada

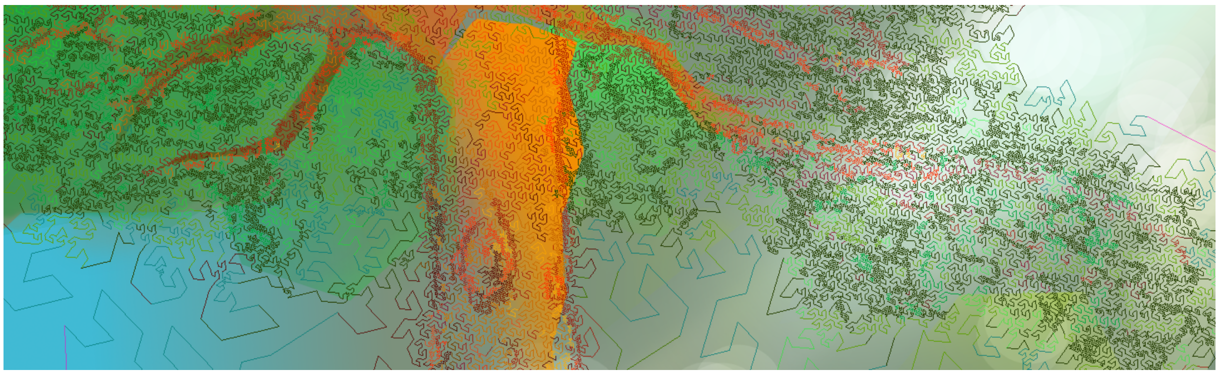

Figure 1: *Open Tree by Ruby Arnold*

Abstract

Space filling curves, invented by mathematicians in the 19th century, have long been a fascination for artists, however there are no interactive tools to allow an artist to create and explore various levels of recursion of the curve in different parts of the artwork. In this work a new type of painting tool for artists is introduced, which gives the artist control over the very base of a space filling curve, i.e recursive subdivision. Although there are many such curves that would lend themselves to this treatment, the Flowsnake (Gosper) curve has been chosen in this work, mainly for its aesthetics. The curve is based on a hexagonal grid, and in our system hexagons are subdivided at the artist's touch in a non-homogeneous manner, leaving a trail that forms the space filling curve. Some tools are introduced for controlling the painting, such as limiting the depth of recursion, and the 'slow brush', which interpolates slowly between subdivisions to allow the artist to stop at a chosen level. A set of space filling curve brush types provide different shapes and profiles, for giving the artist control of the non-homogeneous subdivision, including the ability to un-subdivide the hexagons. An algorithm for drawing the curve non-recursively is introduced in order to produce a polyline suitable for processing on the GPU to make the system function at interactive rates. An animated version of the image can be made by replaying the subdivisions from the first level. Some examples made by art students and graduates are shown, along with the artist's comments on the system.

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

1. Introduction

Mathematics has been a driver for many forms of art and in particular space filling curves (a special case of frac-

DOI: 10.2312/exp.20151190

tal curves) have fascinated both artists and mathematicians since Guiseppe Peano invented the concept in 1890 [Pea90]. A space filling curve is defined to pass through every point of a specific region. The first three levels of Peano's 'Nine' curve is shown in Figure 2. At each iteration the curve visits more points in the square. The curve is also self-similar, to iterate from one level to the next, each square is replaced by a scaled down version of itself at that level, in this case nine replacements.

Artists such as Susan Happersett [Hap0] have used space filling curves and there are many books and web sites devoted to their beauty [Man82, Ven15, Ven12]. Programming languages such as Logo (e.g. see [AD86, Wyv75]) and others designed to model such recursive structures as a space filling curve do not, in general, appeal to the majority of artists. To give access to procedural modelling, paint-like interfaces have been introduced to aid the design of objects such as landscapes [Epi15] or trees [LRBP12].

Despite advances to create better artist interaction and exploration, a paint-like interface for editing space-filling curves has not previously been devised. In this work, we seek a method that allows an artist to interact with space filling curves in such a way that the integrity of the curve is preserved, while giving the artist the freedom to explore the medium. To this end we have designed a method for painting space filling curves, along with a number of useful tools ('brushes') for helping an artist control the painting. We call the system, *FlowPaint*. Several art students and recent fine arts graduates have used the system and created some example artwork. These examples and the comments on the system given by the artists are reported in Section 6.

FlowPaint presents the artist with a procedural brush that leaves a connected trail forming a space filling curve. Since the trail is made of vectors not pixels, and the brush causes the space filling curve to subdivide wherever it touches, the system is technically a drawing system rather than painting, but the analogy to a paint system seems to work well for the artists who have so far used *FlowPaint*.

The contributions of the research described in this paper are summarized as follows:

- A method that allows an artist to 'draw' a space filling curve, and control the recursion depth interactively and non-homogeneously.
- Each drawing is created from a single polyline which operates similar to contour drawing, however as the curve is guided by the rules of the Flowsnake curve, control of the line is imperfect. (Imperfections in the medium characterize an art form - 'Wabi-Sabi'.)
- A series of 'brushes' that allow the artist to select cells to be subdivided similar to the way a paint system selects pixels.
- The use of the 'slow' brush, that gives the artist the ability to partially change the recursion level of a cell by interpolating the points of the space filling curve.
- A fast, non-recursive table driven drawing algorithm that exploits the vertex buffer on the GPU.

2. Background and Previous Work

Space filling curves have been used for many applications in science, such as for parallel domain decomposition in scientific computing applications [AS97], for dithering when using a bi-level display to represent continuous tone images [WN82], as a mathematical representation of nature [Man82] and for artistic effects [Ven12]. Although artists have used space filling curves as part of an artwork [Hap0] there has been no examples of attempts to build an interactive painting system based on controlling the recursion depth of a space filling curve.

2.1. Artistic Constraints

Artists have long worked with certain constraints to forge a new style. For example the pointillists [Ruh98] limited themselves to composing images with painted points and found something new in the gestalt. More recently image mosaics [FR98] are limited to creating an image from many smaller images, and Ascii art [Wik13] to using the Ascii character set to create an image. Outlines or contour lines of 3D shapes are often used to help teach art students about shading. These are examples of imposing constraints on an artist, where perhaps other methods would produce more realistic shading. An artist can become paralyzed by innumerable choices and imposing a constraint can free the artist in the hope that a style might emerge. In our work the mathematical constraint on a contour (polyline) is that it should form part of a space filling curve. To create shading an artist has to operate under this constraint. Each of these methods are characterized by their imperfections, a concept known in Japanese art as 'Wabi-Sabi' [Pow04].

2.2. Drawing Space Filling Curves

Prusinkiewicz in his book with Lindenmeyer classifies space filling curves as FASS curves (an acronym for space-filling, self-avoiding, simple and self-similar) [PL96]. One such curve is the Flowsnake curve, invented by mathematician, Bill Gosper in the early 1970s, was made popular by Martin Gardner in his Scientific American article in 1976, [Gar76], and more recently some variations were published by Fukuda et al in [HMG01].

The recursive construction can be done in several ways. Seymour Papert's Logo language [AD86] could be programmed to draw such a space filling curve, and he introduced the concept of Turtle graphics to draw the result. Prusinkiewicz [PL96], defines an L-system to represent the Flowsnake. A recursive string rewriting system, which for a particular recursion level, will produce a longer string whose symbols can be interpreted again by turtle graphics. Another approach is to traverse a variation on the scene graph, in which internal nodes are geometric transformation matrices, leaf nodes are primitives such as a line, and graph cycles represent recursive objects. In the work of [Wyv75], a recursive graph was constructed using a simple text input language. Examples were given of different space filling curves, in which a numerical limit was used to define the maximum level of recursion.

In the current work an interactive, table driven algorithm has been introduced to produce the Flowsnake curve at different levels of subdivision, depending on where the user moves the mouse. In this case speed and the ability to recursively subdivide a particular hexagon according to user input were the driving criteria for programming the curve this way, see section 5.1.

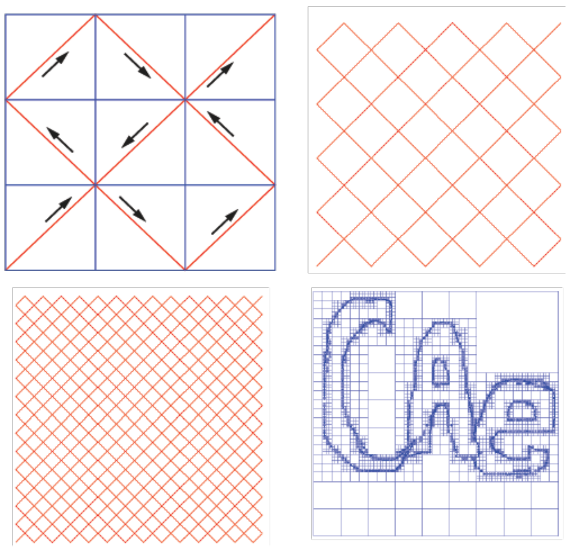

Figure 2: *Top left construction of Peano's Nine curve level 1. Top right shows level 2, bottom left; level 3 and bottom right a nine-curve image painted using FlowPaint.*

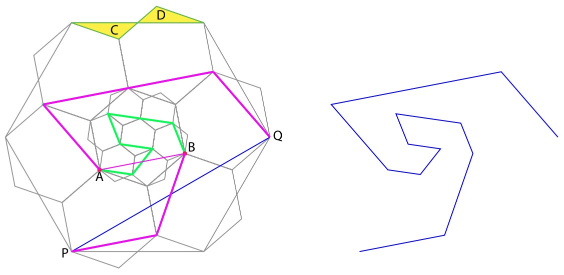

Figure 3: *Construction of Flowsnake curve. Left shows level 1 with the centre hex further subdivided to level 2. Right shows the result without the hexes.*

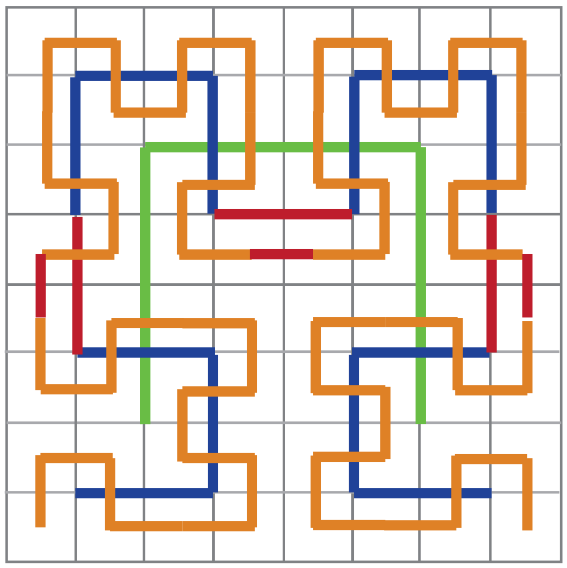

Figure 4: *Construction of Hilbert curve is not at first sight suitable for our FlowPaint system.*

3. Choosing a space filling curve

Space filling curves by definition tend to fill a defined area as the recursion depth is increased. Essentially there is a cellular structure to the curve. The quality that the curve must possess to be usable by *FlowPaint* is that the cell can be replaced by a subdivided cell at the next level of recursion, such that the points at which the curve enters and leaves the cell are identical with the subdivided curve at the next level. This can be seen for the square cell of Peano's Nine-curve in Figure 2 where each line will be replaced by a scaled down version of the curve in the direction indicated by the arrows; in all nine replacements. There is an option in the *FlowPaint* system to use the Nine-curve; the bottom right image of Figure 2 (the letters 'CAe') was painted with *FlowPaint*, but all who have used the system agree that the Flowsnake curve is the more aesthetic choice.

The Flowsnake curve is algorithmically similar to the Peano Nine-curve, except that it is based on a hexagonal grid. At level zero a line connects two vertices of the hex skipping a single vertex; blue line *PQ* in Figure 3. To progress from level zero to level one, the line is replaced by seven scaled down versions of itself, connecting points on the subdivided hexagonal grid (Figure 3). The line *PQ* is replaced by the 7 magenta Flowsnakes, the line *AB* in the figure has been further replaced by a level two Flowsnake shown in green. The condition that has to be satisfied is that each line in level $n-1$ is replaced by the curve at level n in such a way that the end points of the scaled down version exactly matches the end points of the line that it has replaced. Note that the level $n-1$ curve is unaffected by subdividing the neighbouring hexagon. Such a replacement is not immediately possible in all space filling curves, for example the Hilbert curve rescales the entire curve at each level, and introduces new segments as shown in Figure 4. Level zero is shown in green, level one in blue and level 2 in orange. The extra segments are shown in red. Even this curve should be possible to use for painting, due to recent work on half toning for images by Chung et al, which has led to a variation on the Hilbert curve, that allows non-homogeneous subdivision and connects the subdivided portions in an aesthetic manner, see [CHL07]. It should be possible to extend *FlowPaint* to include this type of curve in the future.

Despite other possible curves, the Flowsnake curve fulfills the requirements for *FlowPaint* and has an aesthetic appeal to the computer science and art students who have used the system. We have performed experiments on other space filling curves, but the tools developed in this research are demonstrated with the Flowsnake curve, although they could be easily extended to work with other curves.

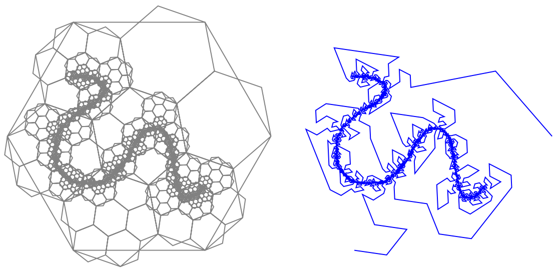
Figure 5: *A simple curved path drawn using a maximum depth of* 6*. The subdivided hexagons are shown on the left and the Flowsnake curve on the right.*

4. Painting with a space filling curve

In *FlowPaint* the artist usually uses a tablet stylus for input. The artist can set a maximum and a minimum recursion depth so that when a hex is entered the hex under the cursor will be subdivided to the current maximum level set by the user, see the curve in Figure 5. The maximum depth was set to 6, both the hexagons and the Flowsnake curve is shown. In *FlowPaint* the artist has the choice to see either representation or both. Hexagon colour, and Flowsnake colour can be set independently. The hexagons (and thus the Flowsnake curve) can also be unsubdivided by right clicking and painting. In this case a minimum depth can be set so that when an artist unsubdivides a hexagon, the hexagon under the cursor will be repeatedly unsibdivided until the minimum recursion depth is reached.

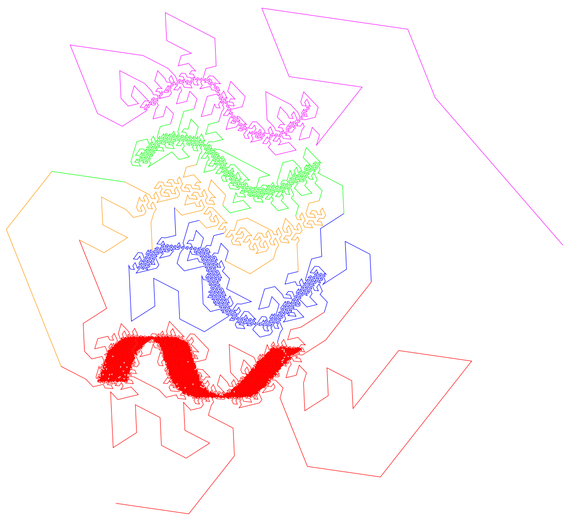
Figure 6: *Some examples of various brush types.*

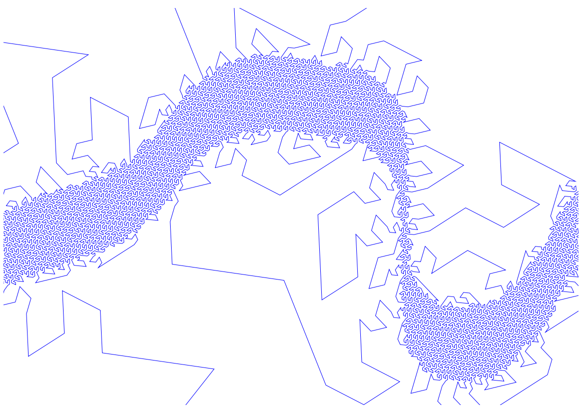
Figure 7: *Close up of a 30 wide depth* 6 *vertical brush.*

4.1. Brush Types

Several different brush types are available to the artist. The simple circular brush subdivides every hexagon it touches to a user set radius. Linear brushes, sample the hexagon grid in a horizontal or vertical line, and can produce interesting effects. The profile brush is similar to the linear brush but instead of simply subdividing hexagons to the maximum depth, the maximum depth of recursion will be set following a normal distribution along the length of the brush, with the maximum in the centre. The effects produced by any brush

Colour	Brush Type	Width	Max. Depth
magenta	circle brush	1	6
green	circle brush	4	6
orange	circle brush	4	5
blue	horizontal line brush	8	6
red	horizontal line brush	15	7

Table 1: *Various different brush types matching the strokes in Figure 6*

depends on its width and the maximum depth. Some examples are shown in Figure 6. Table 1 gives the details of the type, width and maximum depth of each of the examples in the figure. Figure 7 shows a detail of a vertical brush of width 30, where the maximum recursion depth was set to 6. Although the differences seem small in the figure, the artists found the different brushes useful in creating an overall effect.

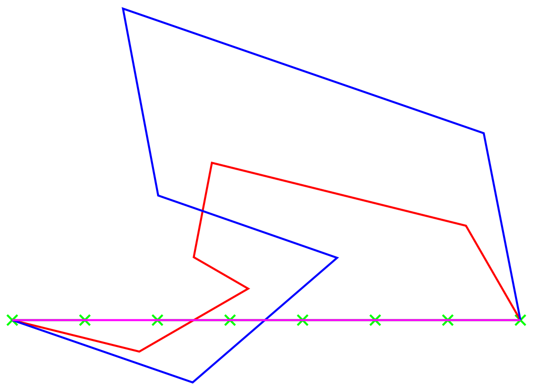
Figure 8: *Level 0 and level 1 with level 0.5 in between draw in red.*

4.2. The Slow Brush

Each line in the Flowsnake subdivides into 7. A simple way of making a level that is in-between two levels is to subdivide each line into 7 segments, then linearly interpolate each of the intermediate points to their correct position in the level+1 as shown in Figure 8. Seven segments are bounded by 2 vertices and 6 vertices equally spaced along the line are introduced. Each intermediate point is interpolated to the

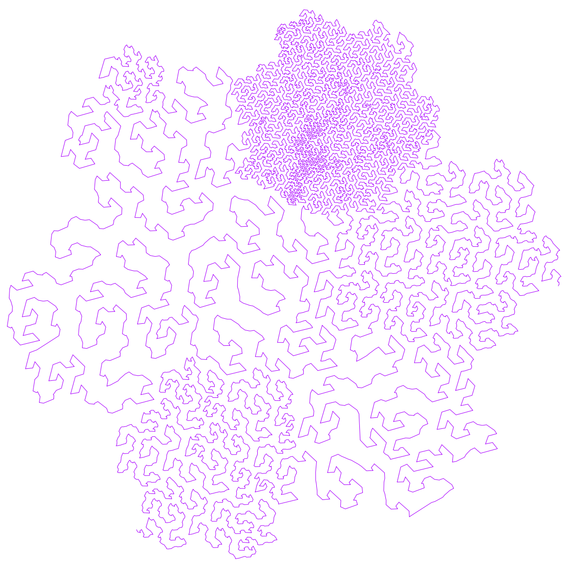
Figure 9: *Using the slow brush technique*

corresponding vertex in the next level. Irving et al use a similar technique to inflate a 2D fractal curve to 3D, see [IS13]. In *FlowPaint* the technique is offered to the user in an animated fashion. By selecting the slow brush each click initiates an animated subdivision of one level to the next. The animation proceeds slowly while the artist presses down the stylus, so that they can choose at what point in the interpolation to stop by relaxing the pressure. A slow 'unsubdivide' technique could be used to run the animation backwards in case the artists wants to return slowly to an earlier level, although this is not currently implemented. Figure 9 shows a variety of partially subdivided levels. The image was made by subdividing some areas more deeply than others, then going back over the levels using the slow brush technique.

5. Implementation

5.1. Linearising the FlowSnake

The Flowsnake can be viewed as seven connected line segments that suitably scaled and orientated replace each of the line segments at the level above see Figure 10. For faster drawing using the GPU, the drawing function was designed to output the Flowsnake as a series of vertices in the correct order. In this way a vertex buffer can be employed to make use of the GPU. To build the Flowsnake the initial level 0 parent hexagon (drawn in magenta) is subdivided into 7 smaller level 1 hexagons as indicated in the figure, and each of these hexagons is replaced by a line segment of the Flowsnake level 1 in black. When the artist's brush passes over a hexagon it is subdivided and the Flowsnake of the

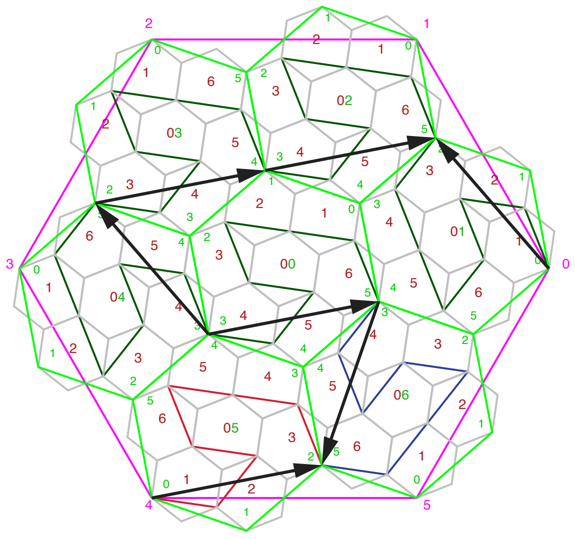

Figure 10: *Direction of the Flowsnake*

next level drawn. In the figure all the green hexagons have been subdivided into the grey level 2 hexagons. The red level 2 Flowsnake will replace the arrow of the level 1 snake, the blue Flowsnake to the right replaces the next arrow and so forth.

The way in which the hexagon and vertex numbering are defined is explained below in section 5.2. The lower left green level 1 hexagon is hexagon number 5 (number in green in the centre of the hexagon) the red number 0 beside it is the number of the centre grey level-2 hexagon.

There are two types of Flowsnake, forwards and backwards indicated by the black arrows in Figure 10. In the L-system definition in [PL96] these were given symbols F_l and F_r and the productions match the order of the arrows in the figure. If we think of these as simply direction 0 and 1 then the arrows identify the type of each of the seven line segments (Flowsnakes) in the the order: 0 1 1 0 0 0 1 in the forward direction and the order 1 0 0 0 1 1 0 in reverse. In *FlowPaint* instead of a string rewriting system, the Flowsnake is implemented through some tables that avoid the storage of very long strings and their conversion to turtle movements. This is done in order to make use of the GPU and save memory for more vertices (i.e. more complex Flowsnake paintings) the algorithm that follows will use a table drive approach to produce the curve rather than having to process each level of the L-system.

In Figure 10 it can be seen that each hexagon is characterized by an input vertex (*invtx*) and an output vertex (*outvtx*) see algorithm 1. For the number 5 green hexagon (hex-5) the input vertex (start vertex) is vertex 0 (vertex-0) and the output is vertex-2. The Flowsnake simply connects these two vertices. The input vertex of the next hexagon (green hex-6) is vertex-5 and the output is vertex-3; the Flowsnake is a backwards type as indicated by the arrow.

In its path from vertex-4 to vertex-0 of the magenta level 0 hexagon (the level 0 line is not shown to avoid cluttering the figure), the level 1 Flowsnake visits the level 1 (green) hexagons in the order: 5604321. As long as we know the first hexagon visited at a particular level and the direction type of the Flowsnake, the order of the hexagons can be calculated or in our implementation looked up in a table. Similarly the matching vertex numbers of the next hexagon can be calculated (also stored in a table) so that at any level the input and output vertices can be calculated and the Flowsnake drawn in the correct order. In overview:

1. subdivide the parent hexagon into seven hexagons
2. number the hexagons and their vertices, and build the h-table.
3. determine the input and output vertex of each of these hexagons.
4. draw the Flowsnake by connecting the input and output vertices sending the results to the vertex buffer.

Steps 1-3 are only executed on subdivision, but drawing can occur at any time.

5.2. Building and numbering the hexagons

In our system the data structure (see section 5.3) for a hexagon is simply called; *hex*. Each *hex* is constructed so that vertex-0 is the extreme rightmost vertex, and the vertices are numbered in counter clockwise order. When the *hex* is subdivided into 7 smaller hexagons (the data structure for this is a *hexring*) the hexagon in the centre is always 0 and the *hex*, which has a vertex in common with vertex-0 of the parent *hex* is number 1 and the hexagons are numbered sequentially in counterclockwise order, see the green hexagons in the figure. The relevant numbers in the centre of each hexagon are also in green. Now we have a consistent *hex* and vertex numbering system. The actual coordinates of the *hex* are stored in the *hex* data structure when the *hex* is created. Once the correct vertex numbers of the Flowsnake are calculated, a simple look-up produces the actual coordinates.

5.3. Data Structures

Two mutually recursive classes form the basis of *FlowPaint*. Each *hex* refers to its parent ring of hexagons (*hexring*), and also keeps a reference to its child *hexring*. The *hexring* keeps track of it's child *hexes* and it's parent *hex*. The *hexes* are generated by the parent *hexring*. They are assigned a Flowsnake type (forwards or backwards) and a set of tables are calculated to facilitate indexing the Flowsnake coordinates (details in Section 5.4). The process starts with an initial *hex*.

When working out the geometry of the subdivision as shown in Figure 3, the parent *hexring* is not precisely subdivided into *hexes* (hexagons do not pack as do squares), the triangle marked D is outside the parent *hex* but inside one of the child *hexes*. Fortunately there is an identical triangle (C in Figure 3) that is inside the parent hex but outside all the children. Since these triangles come in pairs the area of the parent hex is identical to the sum of the areas of the child *hexes*. Thus the ratio of the side of the parent hex to the side of a child hex is $\sqrt{7}$.

5.4. Calculating the Tables

The first table simply indicates the type of Flowsnake (forwards or backwards). When a *hex* is subdivided we know the parent type and can thus label each hexagon with the type of Flowsnake taken from the appropriate table: i.e. forwards (type 0 parent) is : 0 1 1 0 0 0 1 and backwards (type 1 parent) is: 0 1 1 1 0 0 1. When the smaller *hexes* become parent *hexes* their type is taken from the appropriate table.

A table, called the h-table, stores the hexagon order along the path of the Flowsnake for the *hexring*. This is calculated from knowing the type of the parent. At any level the parent *hex* is subdivided into 7 hexagons (a *hexring*). The child vertex numbering is taken from the parent. In Figure 10, it can be seen that vertex-0 of each of the green hexagons, is the vertex coincident (shared) with a vertex of the magenta parent. Thus the start and end points of each Flowsnake are given by the numbering of the parent's vertices. The black Flowsnake connects the vertices of the green coloured child hexes as follows: 02 35 35 35 24 35 02 taking into account the direction of the Flowsnake. By knowing the numbering of the vertices of the parent hex, the subdivided hexagons are appropriately numbered and the vertices connected using the correct direction definitions. Each of the child *hexes* is numbered (0 to 7 with 0 always in the centre) and the correct *hex* numbers on the path of the two different types of Flowsnake are held in tables. As a *hexring* is entered there is an offset (modulo 7) added to the table depending on the number that was assigned to the parent *hex* in it's own *hexring*.

The tables are set up once, when a *hexring* is initialized. When drawing occurs the tables simply supply the correct input vertex and output vertex (*outvtx*) for the *hex* that is being processed. The drawing function ensures that the vertices are sent to the vertex buffer in the right order (i.e. always drawing in the forward direction).

On generating a *hexring*, algorithm 1 is used to correctly order the hexagons in the h-table.

Having found the *hex* numbers along the Flowsnake path within a *hexring*, the next step is to find input and output vertices for the seven child *hexes*. This can be found easily from the parent input and output vertices. By choosing that every vertex on the parent corresponds to vertex-0 of the child, means that the numbering of shared vertices on neighbouring child *hexes* is constant. Vertex-0 and vertex-1 are not shared (see Figure 10), and the other 4 are numbered as follows: 2 5, 3 4, 4 3, 5 2. I.e. The 2 5 means that vertex-2 on *hex*-5 is the same vertex as vertex-5 on *hex*-6. As described above the correct *hex* numbers are in the h-table.

Algorithm 1 Finding the 7 Hexagons along the Flowsnake

```
if notset(parentHexNum) then
    h[0] ← 5          % level 0 hex
end if
if parentType == 0 then          % not level 0 hex
    if h[0]<0 then
        h[0] ← (parent->getInvtx() +1) MOD 7
    else
        h[0] ← (invtx+1) MOD 7
    end if
    h[1] ← (h[0] MOD 6) + 1
    h[2] ← 0
    for ($i = 3; i < 7; i++$) do
        h[i] ← ((h[0]+7-i) MOD 6) + 1)
    end for
else
    if parentHexNum ==0 then          % centre hex
        h[0] ← getgparentphex(0)          % same number as in the h-array of grandparent
    else
        h[0] ← (outvtx+3) MOD 7
    end if
    h[1] ← (h[0] MOD 6) + 1
    h[2] ← ((h[0]+1) MOD 6) + 1
    h[3] ← ((h[0]+2) MOD 6) + 1
    h[4] ← 0
    h[5] ← ((h[0]+4) MOD 6) + 1
    h[6] ← ((h[0]+3) MOD 6) + 1
end if
```

The centre *hex* has to be treated separately as unlike the other *hexes*, each vertex is shared with two other hexagons, so the correct vertex numbers depend on the next *hex* on the path. The input vertex of the centre *hex* is the same vertex as the output vertex of one of the surrounding *hexes*. All these surrounding vertices are numbered either 3 or 4. In practice only vertex-3 is ever chosen so it can be seen from the figure that the chosen vertex on the centre *hex* will be the number of the previous *hex* minus 1. This gives the input vertex. E.g. The Flowsnake goes from *hex*-6 to *hex*-0. So *hex*-6 (vertex-3) is the shared input vertex to the centre *hex* in Figure 10, so the input vertex of the centre *hex* will be $6-1=5$, (green vertex-5 in *hex*-0). The output vertex of the centre *hex* depends on the type of Flowsnake and is either two to the left or two to the right of the input vertex.

5.5. Brush and animation implementation

Each brush is represented by a set of point samples that are then tested against the top level hexagon mesh. If a point lies within a hexagon it is subdivided into a *hexring* and the point tested recursively in the child *hexes*. The process terminates when the current maximum limit is reached. For the slow brush the drawing routine is called while the mouse is down and each call increases the linear interpolation towards the next level. After some trial and error the increment at each *m*ousemove call was set to be 0.05, which the artists who have worked with the system find acceptable. The density of point samples in a brush affects performance (see future work section). In the future offering artist controlled brush density should enable more effects to be implemented.

Animation is also handled using the slow brush technique. To morph between a line and a completed Flowsnake is simply a matter of dividing the line successively into seven segments, interpolating the points towards the target points in each *hexring* and continuing recursively. It may be more efficient to store the vertex buffers of each frame, but would require a lot of memory for large Flowsnakes.

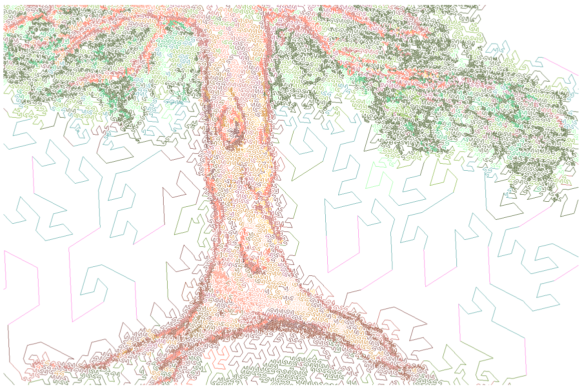

Figure 11: *Flow Tree by Ruby Arnold*

6. Case Studies and Results

Several art students and recent graduates of the University of Victoria were asked to use the system, to produce some artwork of their choice and to comment on the tools and facilities offered. Rhea Lonsdale graduated with a Bachelor of Fine Arts with Great Distinction with a major in Studio Art, and a minor in Digital Culture and New Media. Ruby Arnold from the Visual Arts department graduated in 2013 and holds a Bachelor of Science from the faculty of Fine Arts having done a combined major in Visual Arts and Computer Science. Hovey Eyres is a current student in the Faculty of Fine Arts.

Hovey Eyres produced Figure 14 in about half an hour in her second session with *FlowPaint*. Figure 15 is a detail of Marilyn's right eye. Rhea Lonsdale produced the vase in Figure 12 in a few minutes after using the system for an evening. She spent an hour producing Figure 13, but part of that time

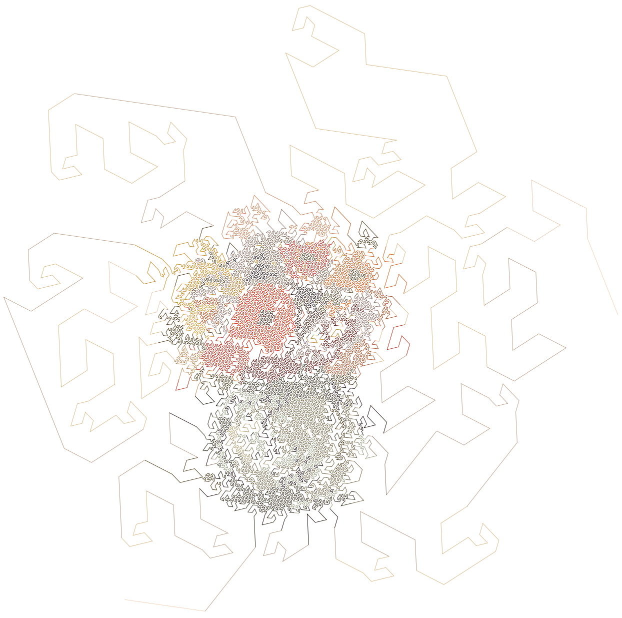

Figure 12: *Vase by Rhea Lonsdale*

was spent figuring out a workflow for doing hatching (see the comments). Ruby Arnold made the teaser image (see Figure 1). None of the students enhanced the images with other software, but in the teaser image the artist painted a background and imported it into *FlowPaint*. The tree without the background is shown in Figure 11.

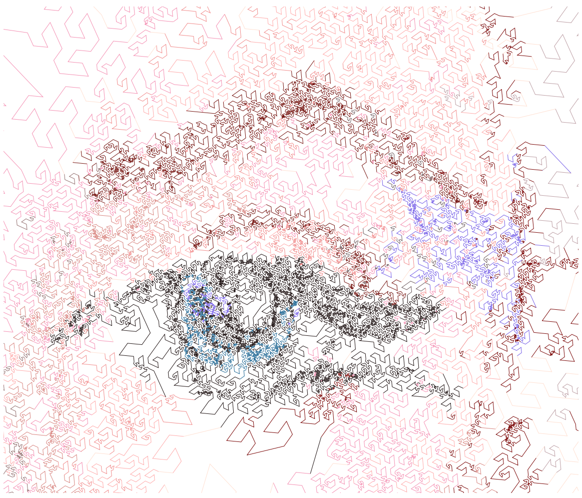

Figure 15: *Detail of Marilyn's right eye*

Various other students used the system and commented, but preferred to remain anonymous.

Quotes from the art students:

"*FlowPaint* is reminiscent of contour drawing, a technique in which an artist uses a single, unbroken line. Contour drawing is a common method for artists to gain proficiency in rendering from life without the use of value and shading. Despite constricting users to a similar unbroken line, *FlowPaint* lends itself to representation which uses value. The line itself is difficult to control to create complex representations without relying on value to create shapes, and value is created by means of the density of the folding line as it subdivides. The resulting drawing is thus not unlike pointillism or even ASCI art, wherein broken singular points and individual characters respectively are used to build value."

"I like working under the constraints of the system. What ever you do a continuous line is produced. You can always see the start point just left of the bottom of the image, and the end point on the extreme right. "

"The *FlowPaint* application is a unique experience in digital image creation. Most methods mirror the traditional aesthetics of pencil, pen and paint. The images created using the Flowsnake application are genuinely digital, which makes both the process and results quite interesting"

"*FlowPaint* as an art tool is interesting as it enables the making of geometrically complex drawings which have essential qualities, which divert from traditional drawing. Some of the limitations of control encourage exploration, such as the speed of the cursor as it draws when using high depth. In doing so the artist experiences a lag from cursor movement to lines drawn, and it's within this constraint that techniques such as scattering can be used to draw. By rapidly sketching with a high brush width vertically or horizontally, one makes marks more similar to those found in traditional drawing, such as crosshatching. Ultimately, the tool rewards creative approaches to its unique qualities as a medium."

Several students commented that hand movements using very wide brushes had to be slow as the system could not keep up and some drop out occurred. In other words the mouse movements were missed as too much processing was going on with multiple hexagons being subdivided. Meanwhile Rhea Lonsdale took advantage of the sparse effect created by the wide brush to obtain a cross hatching effect. She demonstrates this in the accompanying video but also it can be seen in her work shown in Figure 13.

The unsubdivide brush was quite popular, and although useful, does not replace 'undo' as was our original design intent. Students also wanted better colour control, particularly for the colour to be taken continuously from an underlying image, providing the ability to overpaint the image with the Flowsnake.

Nearly all the students liked the fact that the image is constructed from a single line that forms a space filling curve. Like other accepted artistic methods, zooming in reveals interesting details.

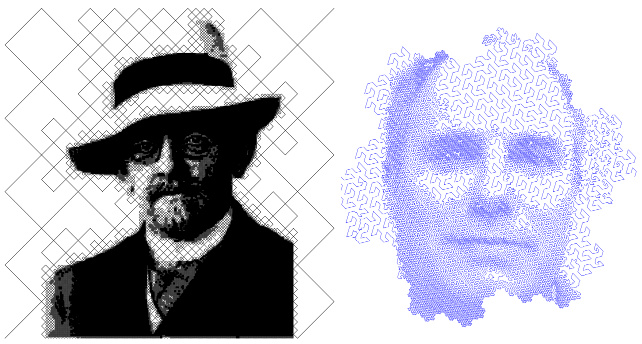

Figure 16: *Two space filling curves generated from images: Left a space filling version of Hilbert (software by Neil Bickford) and right Flowsnake created using FlowPaint. The procedure which replaces grey areas by a corresponding Flowsnake, using the slow brush technique to create grey between the fixed recursion levels.*

7. Conclusion and Future Work

In this work we have demonstrated that it is possible to use space filling curves, particularly the Gosper Flowsnake, to construct brushes that leave a trail that is a non-homogeneously subdivided space filling curve. Several different tools have been demonstrated to help produce artistic effects. Some artwork was produced by several artists and were generally enthusiastic about the technique, and thought that the results demonstrated a new style.

In the future, the range of brush types will be expanded to explore better colour control as well as the use of profile brushes at different scales with user defined profile curves that control the subdivision of the surrounding cells. It is a current limitation of the system that a wide circular profile brush, slows response time. This depends on both width and depth of subdivision.

Images can also be represented by non-homogeneous curves and work in this area inspired Neil Bickford (a high school student) to develop a method for a Hilbert Curve 16. The figures show subdivision according to some underlying feature such as the grey level of a particular area. Other metrics for saliency could also be explored. The ideas offered in the artist's comments will also be used to achieve better colour control, as well as exploring new ways to allow the artist to control subdivision.

Although artists can unsubdivide a hexagon, it does not completely replace the 'undo' command. Limited undo can be implemented by pushing references to each *hex* as it is subdivided, onto a stack and unsubdividing. Undoing an 'unsubdivision' would also be possible by subdividing. The amount of available memory would limit the stack size. All the recursion levels used are in the vertex buffer even if at the current zoom level they are not in view. Thus the artist will find some limitations in speed when the buffer can no longer be held on the GPU. A better approach would be to store on the GPU only what is actually displayed on screen to alleviate this problem. The second issue is that recursion levels that are deep relative to the current scale will result in vectors that are smaller than a pixel, and thus should not be stored in vertex memory. Implementing these improvements would be straightforward and again help relieve the GPU memory issue.

Acknowledgements

This project was made possible using by grants from the Natural Sciences and Engineering research Council, the GRAND NCE, and Intel corps. The project greatly benefitted from the many students who used and made comments on the system and particularly for the feedback from the three University of Victoria art students who used the software and took part in the case studies; Ruby Arnold, Hovey Eyres and Rhea Lonsdale. I would also like to thank all the graduate and undergraduate students in our lab. who helped in many ways, particularly for allowing me to free up enough time to do this project. I would also like to acknowledge the helpful discussions I had with Prof. Faramarz Samavati (University of Calgary) some years before this project came to fruition.

References

[AD86] ABELSON H., DISESSA A.: *Turtle Geometry: The Computer as a Medium for Exploring Mathematics*. Artificial Intelligence Series. AAAI Press, 1986. URL: http://books.google.ca/books?id=3geYp44hJVcC. 2

[AS97] ALURU S., SEVILGEN F.: Parallel domain decomposition and load balancing using space-filling curves. In *High-Performance Computing, 1997. Proceedings. Fourth International Conference on* (Dec 1997), pp. 230–235. doi:10.1109/HIPC.1997.634498. 2

[CHL07] CHUNG K.-L., HUANG Y.-L., LIU Y.-W.: Efficient algorithms for coding hilbert curve of arbitrary-sized image and application to window query. *Information Sciences 177*, 10 (2007), 2130 – 2151. Including Special Issue on Hybrid Intelligent Systems. URL: http://www.sciencedirect.com/science/article/pii/S0020025506003732, doi:http://dx.doi.org/10.1016/j.ins.2006.12.003. 4

[Epi15] EPIC GAMES: Unreal game engine. Software, 2015. http://www.fractalcurves.com/, retrieved March, 2015. 2

[FR98] FINKELSTEIN A., RANGE M.: *Image mosaics*, vol. 1375 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 1998. edited by Hersch, RogerD. and AndrÃľ, Jacques and Brown, Heather. URL: http://dx.doi.org/10.1007/BFb0053259, doi:10.1007/BFb0053259. 2

[Gar76] GARDNER M.: Mathematical games. *Scientific American 235*, 6 (Dec. 1976), 124–133. URL: http://www.nature.com/scientificamerican/journal/v235/n6/pdf/scientificamerican1276-124.pdf, doi:http://dx.doi.org/10.1038/scientificamerican1276-124. 2

[Hap0] HAPPERSETT S.: Artist interview: Irene rousseau. *Journal of Mathematics and the Arts 0*, 0 (0), 1–7. URL: http://dx.doi.org/10.1080/17513472.2015.1007409, arXiv:http://dx.doi.org/10.1080/17513472.2015.1007409, doi:10.1080/17513472.2015.1007409. 2

[HMG01] H. F., M. S., G N.: New gosper space filling curves. In *Proceedings of the International Conference on Computer Graphics and Imaging* (2001), vol. 34 of *CGIM2001*, pp. 38–48. 2

[IS13] IRVING G., SEGERMAN H.: Developing fractal curves. *Journal of Mathematics and the Arts 7*, 3-4 (2013), 103–121. URL: http://dx.doi.org/10.1080/17513472.2013.852399, arXiv:http://dx.doi.org/10.1080/17513472.2013.852399, doi:10.1080/17513472.2013.852399. 5

[LRBP12] LONGAY S., RUNIONS A., BOUDON F., PRUSINKIEWICZ P.: Treesketch: Interactive procedural modeling of trees on a tablet. In *Proceedings of the International Symposium on Sketch-Based Interfaces and Modeling* (Aire-la-Ville, Switzerland, Switzerland, 2012), SBIM '12, Eurographics Association, pp. 107–120. URL: http://dl.acm.org/citation.cfm?id=2331067.2331083. 2

[Man82] MANDELBROT B. B.: *The fractal geometry of nature*. W H FREEMAN, 1982. 2

[Pea90] PEANO G.: Sur une courbe, qui remplit toute une aire plane. *Mathematische Annalen 36*, 1 (1890), 157–160. URL: http://dx.doi.org/10.1007/BF01199438, doi:10.1007/BF01199438. 2

[PL96] PRUSINKIEWICZ P., LINDENMAYER A.: *The Algorithmic Beauty of Plants*. Springer-Verlag New York, Inc., New York, NY, USA, 1996. 2, 6

[Pow04] POWELL R.: *Wabi Sabi Simple*. Adams Media, 2004. 2

[Ruh98] RUHRBERG K.: *Seurat and the Neo-Impressionists". Art of the 20th Century*. Art of the 20th Century, Vol. 2. Koln: Benedikt Taschen Verlag, 1998. 2

[Ven12] VENTRELLA J.: *Brainfilling Curves - A Fractal Bestiary*. Eyebrain Books, 2012. URL: http://books.google.ca/books?id=Qj-zAwAAQBAJ. 2

[Ven15] VENTRELLA J.: Fractal curves. Software, March 2015. https://www.unrealengine.com/, retrieved March, 2015. 2

[WD10] WYVILL B., DODGSON N.: Recursive scene graphs for art and design. In *CAe '10: Proceedings of Computational Aesthetics, London, United Kingdom* (2010), Eurographics: European Association for Computer Graphics, pp. 33–40.

[Wik13] WIKIPEDIA S.: *Computer Art: Ascii Art, Fractal Art, Digital Art, Ars Electronica Center, Digital Media, Interactive Art, Demoscene, Desmond Paul Henry, New Media Ar*. University-Press Org, 2013. URL: https://books.google.ca/books?id=PQ9DngEACAAJ. 2

[WN82] WITTEN I. H., NEAL R.: Using peano curves for bilevel display of continuous-tone images. *Computer Graphics and Applications, IEEE 2*, 3 (May 1982), 47–52. doi:10.1109/MCG.1982.1674228. 2

[Wyv75] WYVILL B.: An interactive graphics language, 1975. PhD thesis, Bradford University Press. 2, 3

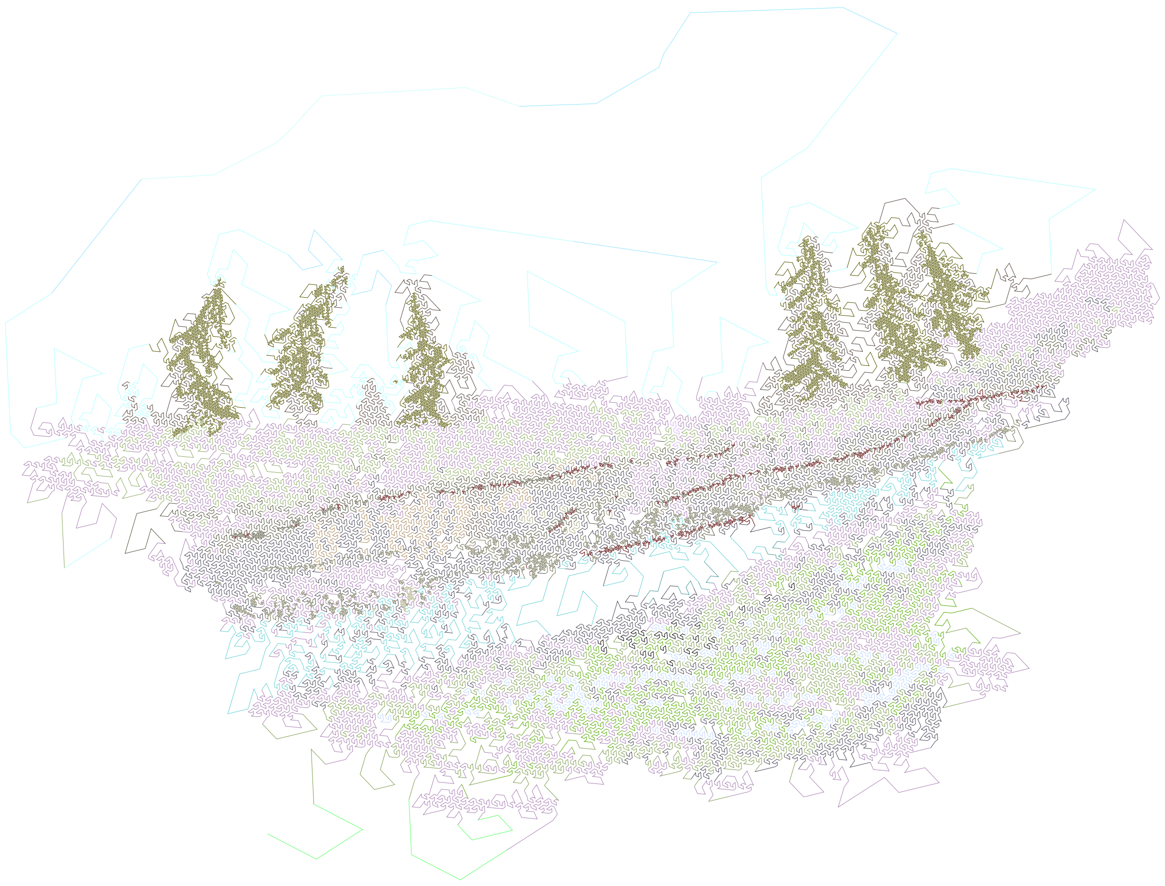

Figure 13: *Victoria by Rhea Lonsdale*

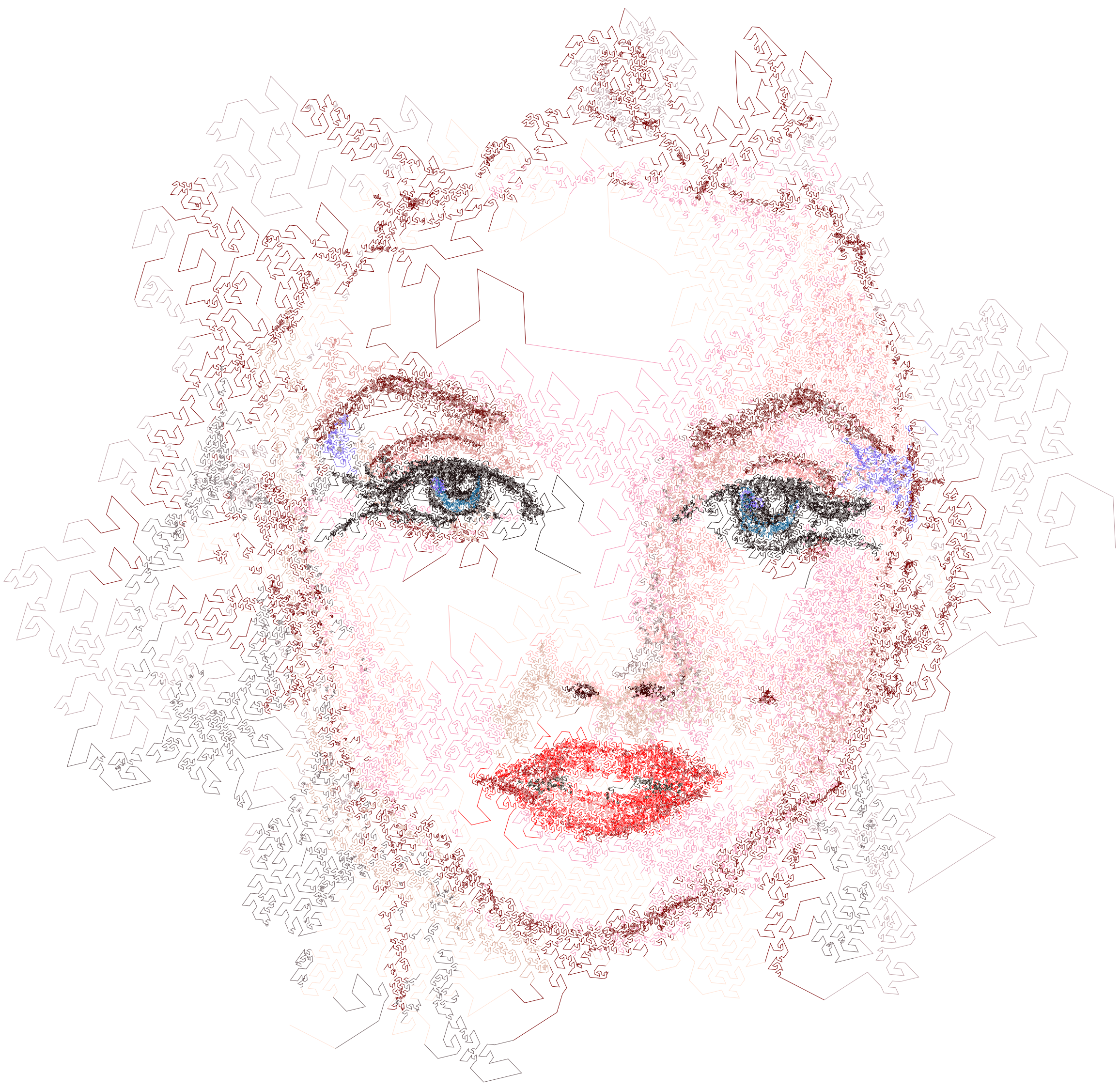

Figure 14: *Marilyn by Hovey Eyres*