Coordinated Particle Tracing

by

Chujia Wei

A project report submitted to

the Faculty of Graduate Studies and Research

in partial fulfilment of

the requirements for the degree of

Master of Computer Science

in

School of Computer Science

Carleton University

Ottawa, Ontario, Canada

September 2013

Abstract

It is common today to see people post their photographs with a cartoon effect or a sketching style. Although they seem to have been created through artists' hands, we can produce such stylized images algorithmically. However, no applications can create images in an engraving or line-drawing style such that parallel lines are growing steadily and smoothly changing directions and thicknesses together. Our project provides a new way to generate a line-drawing effect for input images. When particles perform coordinated movement, their traces are parallel. To reproduce artistic works, we also added many rules for particles so that they are able to move, create new particles, communicate with neighbors and terminate automatically. Our approach adjusts the spacing and the thickness of drawing lines with the tones in real images thereby expressing bright and dark areas. By giving different distributions of particles, we can also generate smooth abstract patterns.

Acknowledgments

I would like to thank my supervisor Dr. David Mould, who helped me to start this project and guided me when obstacles came.

I also want to thank my parents and friends for supporting me all the time and giving me the confidence and courage to hold on to my dreams.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

With just simple black and white lines, artists can create stylized images which depict objects clearly. As shown in figure 1, artists can use black lines on white canvas to draw features. The darker parts of the image can be seen as using more strokes, while the brighter parts use less lines. Interestingly, the objects are shaped using different types of curves (strokes with various directions) without actually drawing out the edges.

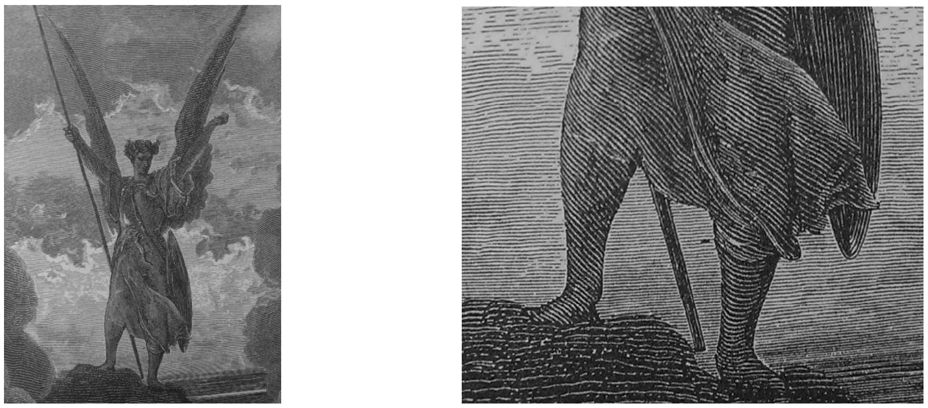


Figure 1: An artistic drawing with details shown on the right. Engraving by Gustave Doré [2]

If we look closely at the details of the drawing, we will notice three main characteristics:

- First, strokes are almost parallel to their neighbors.
- Second, strokes have different individual directions and thicknesses, with some randomness as well.
- Third, strokes change their directions, thicknesses and spacings smoothly without sharp or sudden turns. This is especially noticeable on the clothes.

Images with these properties are natural and attractive, which leads to the question: "Can we algorithmically create such a style?"

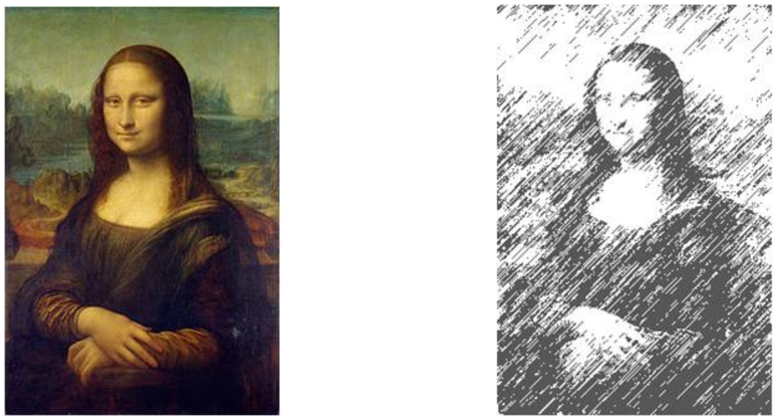

(a) Mona Lisa **(b)** Reproduced by Photoshop

Figure 2: Stylization with softwares

A lot of software is able to produce stylized images simulating the activities of artists. In Figure 2, the Mona Lisa reproduced by Photoshop consists of only black lines. In the areas that have darker tones, lines tend to be close to their neighbors and stretch longer. Compared to the original image, the result generated by software has similar properties to the artistic style mentioned above, such as parallel black and white lines and changing line-spacing to match tones. However, it does not have the irregularity of an artist's drawings. To make a natural yet irregular pattern, software should not only be able to draw objects clearly but also make strokes satisfying the three characteristics mentioned above.

1.2 Contributions and Organization

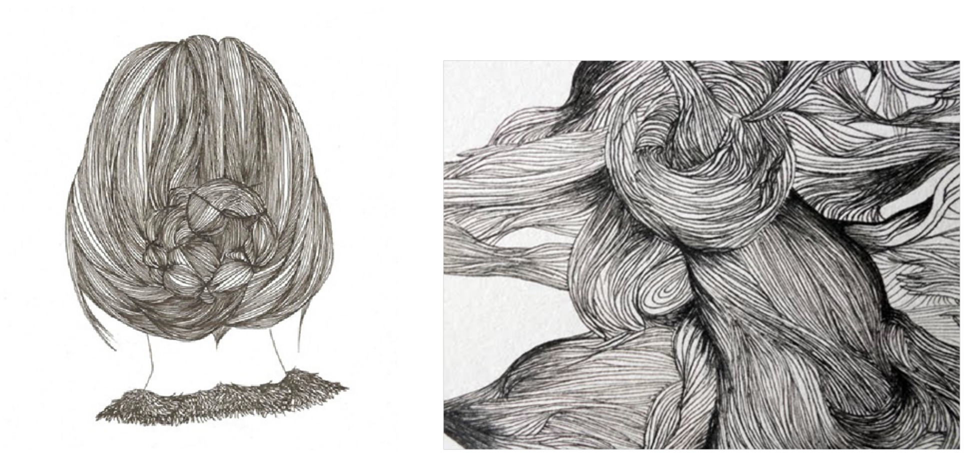


Figure 3: Left: A hair style created by Maria Gil Ulldemolins. Right: An abstract hair style drawn by Rachael Bartram. Images from Flickr.

The method described in this report is inspired by artistic pen-and-ink drawings, where strokes are mostly parallel with tones represented by different line spacings and thicknesses. Figure 3 shows an example of this kind of pattern: several groups of lines are used to draw the flow of hair; each group of lines has a main direction, which allows lines to move in parallel generating a smooth flow; lines change their directions and spacings gradually to depict the orientation and density of the hair flow.

This project presents a new way to create stylized images which have the line-drawing effect. The density of curves decides the brightness of the image and the direction of curves shapes the figures. Our process is also highly self-managing – the sketching lines generate, grow and terminate all by themselves. Most of all, the outcome of our project fully satisfies the three requests mentioned in section 1.1, and is able to intelligently manage parallel, irregular, and smooth curves. Specifically, we use the basic structure of a particle system to create a system where the traces of particles draw objects and leave a natural and irregular pattern.

First, we generate a “feature map” for an input image. The information of different features can be obtained through segmentation or other approaches, but here we simply use thresholding to separate features with black and white tones.

We then build a particle system, and define a set of rules to guide the behavior of particles. All particles move and leave traces in a coordinated way, communicate with other members, and obtain information from an input image. Under the rules, each particle keeps tracking the distance with neighbors thereby modifying its direction, so that all particles can remain parallel with each other, forming an interesting pattern. See figure 4 for an example.

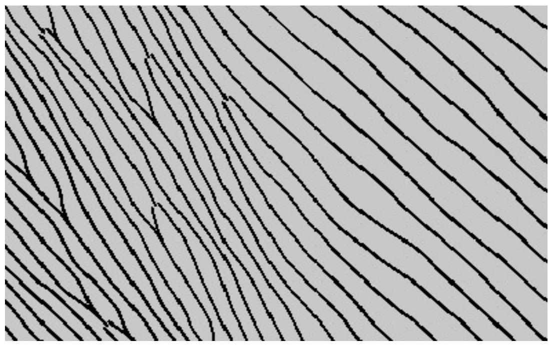


Figure 4: Particles move and leave parallel curves which can change spacings and thicknesses smoothly.

Some natural patterns have a texture similar to the result generated by our system. For example, in a leaf texture as shown in figure 5, no lines cross each other and many small lines are in parallel. To simulate this kind of texture, particles should detect collisions with each other and decide to be terminated or to change directions.

Figure 6 shows an application of creating a real texture of mushroom gills with our system. Real mushroom gills are full of almost parallel straight lines, with new

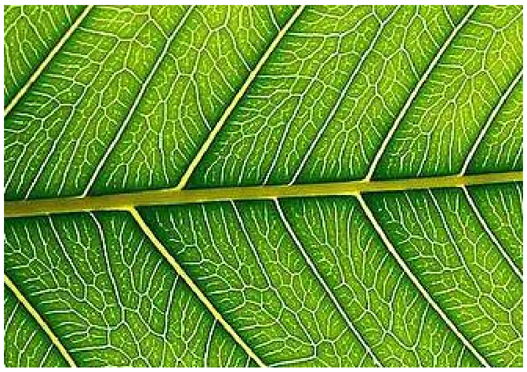


Figure 5: Texture of a leaf.

ones spreading out from the middle of two lines. Our result nicely reproduces this feature and also uses some randomness to make it more natural. See how the lines turn and leave curving traces in the bottom area.

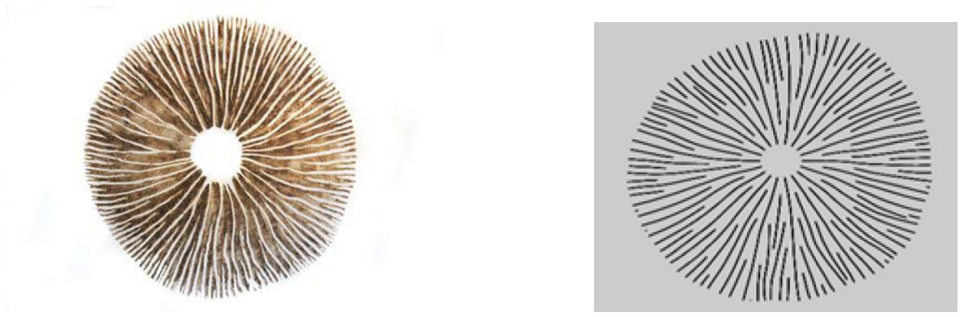


Figure 6: Left: Real mushroom gills. Right: An image created with our coordinated particle system

This report is organized as follows: In chapter 2 we introduce some related work in the area of non-photorealistic rendering such as edge flow, line drawing, and painting. Some background information including particle system, intelligent agents, and particle tracing is also discussed.

The main algorithm of our method is introduced in chapter 3. We first give an overview of the whole structure of our coordinated particle system. The approach to obtain the behavior of our particles is then explained. We also define important rules for better performance, and assistant maps generated from the input image to guide our particles' behavior. Pseudocode is provided at the end.

Chapter 4 presents some results from our work, where we discuss the effects with different parameter values. To evaluate our system, we compare the results with other related work in chapter 4. Finally we conclude by summarizing our work in chapter 5, with a discussion of possible future work.

Chapter 2

Background and Previous Work

2.1 Non-photorealistic Rendering (NPR)

Non-photorealistic Rendering is the sub-area of computer graphics concentrating on creating stylized patterns and images with computers. Unlike photorealistic approaches, NPR focuses on creating images in an artistic way, for example, producing a cartoon flame instead of a figure picturing real fire. NPR is also used in applying different styles like watercolor, mosaic and hand-drawing to photographs, and is even able to make stylized animations. There are many styles of NPR, such as line art drawing, painterly rendering, and stippling, that have been studied already to create artistic patterns.

As a kind of line art drawing, pen-and-ink illustration consists of only black lines. In the traditional way, it is tricky and important to form proper shadows by managing the position and proportion of black ink on white paper. For digital rendering, matching the tone is also essential to sketch out features. A considerable amount of work has been done in this area. In 1994, Winkenbach and Salesin introduced a system to produce pen-and-ink style images [19], which is capable of rendering with different stroke styles and various outlines and tones. Although the algorithms in the system are proposed originally by others, and user control is needed for creating textures, it shows the possibility of digital line drawing rendering. Another contribution of this paper is the summarization of the principles of traditional pen-and-ink illustrations. For example, the author pointed out that even-weighted lines are lifeless; lively artistic lines should be able to change thicknesses. In addition, lines should have roughly equal weights and spacings.

Two years later, Winkenbach and Salesin presented a new technique of “controlled-density hatching” [20], which allows lines to be created in dark areas or the places where lines are far from each other, and makes lines disappear when the area is bright or has too many lines close together. With the use of planar maps and traditional texture mapping techniques, this method can also create pen-and-ink illustrations with different texture styles such as wood, stippling and crosshatching, and cast curved shadows onto curved objects. One advantage of this work is that lines can appear and disappear gradually to convey tone and shape, and generate various textures.

Salisbury et al. also proposed a method to produce pen-and-ink illustrations [16]. However, unlike the work mentioned above, this method requires user input for choosing and placing stroke textures. Users will paint with a type of texture, and let the computer produce individual strokes, since the process of creating individual lines is tedious. Although user interaction is required, this method is able to use limited input to produce quality illustrations with various texture styles.

Hertzmann divided the schemes for NPR into two directions [8]: image-based and object-based. Rendering based on images usually takes an input image and generates stylized textures on the surface. Object-based techniques, on the other hand, work on untextured object models without actual images.

Many methods are image-based, as is our work. For example, the technique proposed by Ostromoukhov [14] to produce engraving style images shares some similarities with our work, such as roughly parallel lines, and different groups of lines for different objects. With this method, photos can be stylized with an engraving effect which applies a set of lines with a consistent main direction for each engraving layer. However, there are downsides of this method: first, manual control is needed to create each engraving layer. Since engraving layers are related to objects in an input image, user interaction can be time consuming for a complex image. Second, the output has highly parallel lines without irregularities, which makes the result lifeless.

Kang et al. [11] explored a method to create a coherent line drawing style. First, a vector field is computed from an input image. The vectors are perpendicular to the image gradient, which gives us a smooth edge tangent flow (ETF) with the ability of preventing a "swirling" effect. After that, line illustrations are produced by applying a flow-based DoG filter (FDOG) to the vector field. The ETF smooths out noise and enhances fine features in the input image. The lines created with FDOG are continuous with the capability of connecting isolated edge points, which allows us to generate stylized images with the outlines of features nicely depicted.

In 1998, Curtis [1] proposed a rendering method which only requires a depth map of a 3D model and several parameters to produce a hand-drawn sketchy animation. The method first generates an edge map and a forcefield vector map from the given depth map. Particles are then released in the areas that need ink, keep tracking edges with the help of the vector map, and finally die when they enter areas with no need of ink. This loose and sketchy rendering is simple and effective. With limited input, it can generate expressive digital line drawings of the silhouette edges of an input image, and create animations as well.

The stroke brush rendering technique introduced by Haeberli [6] is also an example of image-based rendering. The size, color, shape and direction of strokes are controlled to create abstract paintings from real photos. Unlike many modern NPR techniques which operate automatically, this method works more like drawing on a digital canvas with a mouse. As the user moves the cursor, several brush strokes are generated along the cursor's path. This system provides different brush shapes such as circles, lines and rectangles to create various styles. Moreover, the size of strokes can be controlled by the speed of the cursor, which allows users to create rough scenes with large strokes and depict fine details with small ones. The variety of choices allows users to improvise with creativity, though this technique did not give new or complex algorithms.

With the same idea of depicting features with multiple size brushes, Hertzmann also presented a painterly rendering method [7]. Like Haeberli, Hertzmann applied large brushes for a rough representation and small ones for fine details, only this technique is an automatic approach for a digital painterly rendering. The locations of brushes are decided by comparing the current canvas with a reference layer and finding the places that have large enough differences. The reference layer for each size of brush can be obtained by applying a Gaussian blur to the source image. Using this method, large brushes are placed throughout the canvas at first, and as the process goes on, less and less places need strokes, and smaller and smaller brushes are used for finer detail. Compared with Haeberli's system, this method needs much less user interaction and reasonably decides brush locations, which then nicely creates outcomes with an impressive painterly effect that is close to real artistic works.

Inglis et al. proposed three algorithms to create images with an Optical Art effect (Op Art) [9, 10]. Op Art rendering makes use of visual perception to create exaggerated abstract images with black and white lines. The first algorithm is for creating straight parallel lines with two directions without any artifacts such as crossing or line breaking; the second one is for creating straight parallel lines with more than two directions and the minimum artifacts; and the third algorithm explored Op Art with curving parallel lines. In general, this approach first obtains a suitable input map by thresholding the original image, and puts it into a grid. The grid is then filled with different oriented lines corresponding to the color. By also drawing alternate lines to connect regions of different color sections, the final result is rendered without line breaks or crosses.

There are also many studies in the area of object-based rendering. Often line drawing styles like sketching and engraving are used. In 1993, Elber and Cohen presented a scheme to cover free-form surface with isoparametric curves [5]. A few years later, based on this work, Elber enhanced the algorithm with the capability of changing the density of curves [3]. Taking an input model, this algorithm generates isoparametric curves on the model's surface taking into account the light source and viewing direction to control the density of the coverage. A real-time line rendering method is also presented by Elber [4], which generates different styles of aesthetic line-art textures on free-form surfaces in seconds. In 2000, Hertzmann mentioned that the computation of directions for hatching in Elber's work is not reliable, and proposed a line-art rendering method with a more robust technique [8] to find the directions of hatching curves. With the input of a polygonal mesh, this method first computes a direction field, and then draws out smooth silhouette lines; finally, the surface is covered with hatching lines with different densities. The disadvantage of this work is the rendering time of hatching, which can be minutes depending on the density and the complexity.

2.2 Particle System and Intelligent Agents

In 1983, Reeves put forward the concept of a "particle system" [15] – a technique to simulate fuzzy objects using a large number of particles. In a particle system, all the particles are bound by a set of rules which control the way that particles are born, move, and die. Typically, particle systems are used for simulations such as fire, smoke, and waterfalls (see figure 7), which are commonly associated with physical laws – a set of rules. As an example, we created the butterfly in figure 7 with 2000 particles. Each particle is released inside a shape of butterfly and travels randomly. The further particles move into the screen, the smaller they will be, and if particles move closer to the screen, they become larger. Particle systems can be very different, but usually the structures of particles and a emitter are included: the particle structure stores all the individual information such as color, location, speed; and the emitter decides other information such as the distribution of particles and the logic for creating particles. A particle can change its properties, such as color, transparency and size, depending on given rules.

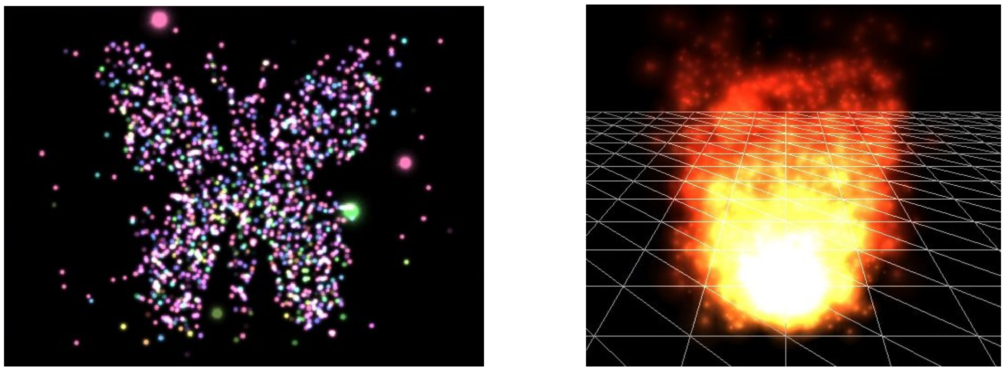


Figure 7: Objects simulated with particle system. Left: butterfly; right: fire

Unlike a standard particle system, particles in our project act like intelligent agents. As defined by Weiss [18], an intelligent agent is autonomous, goal-oriented, and has the ability to respond to different environments. When the environment changes, the intelligent agent adapts its behavior to continue completing its goal. It is simple to build a system with goal-directed agents, but hard to balance an agent's behavior between achieving goals and reacting to the environment. An intelligent agent should be able to stop when its behavior cannot lead it to a goal. In addition, when the environment is changing continuously, the agent needs to maintain progress towards its goal. Sometimes, it is difficult for one intelligent agent to sense the environment and make decisions, so a multi-agent system is used. To cooperate, agents can exchange information thereby affecting each other's behavior.

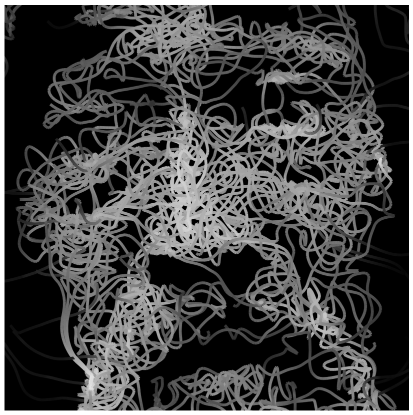
Similar to our work, Schlechtweg et al. proposed an approach [17] to render stroke-based non-photorealistic images with a multi-agent particle system. In their method, RenderBots are designed to produce different effects, such as edging, hatching, and painting, simultaneously in one image. Bots in different styles have different objectives and are capable of exchanging information, adjusting their behavior with others. The theory of this framework is based on a particle system, where a set of predefined rules are used to guide particles to complete their objectives. In addition, to add the ability of self-organization, the concept of "agents" is used also. Being capable of interacting with environments and communicating with others, each RenderBot can also be seen as an intelligent agent.

There are three main activities of RenderBots. First, the direction and velocity are computed with the information obtained from the environment and neighbors. This step controls the basic behavior of bots. In the second step, a new position is then calculated with the new direction and velocity. The last step is to draw the traces. Since the styles of bots are different, the traces can be painted as lines, stipples, or strokes. Among different styles such as "EdgeBots", "StipplingBots", and "MosaicBots", the model type "HatchingBots" is similar to our method, where bots draw almost parallel lines and stop when they are too close to other bots or edges.

2.3 Particle Tracing

Particle tracing uses the particle system structure, but focuses on the trajectories of particles, since the traces can give us vivid and interesting patterns. See figure 8 for examples. Instead of studying how to simulate fire with a large number of particles, we are concerned more about how to organize the movement of just a small number of particles shaping the fire with their traces.

Often when creating non-photorealistic images with particle tracing, particles do not detect collisions with each other and each particle is simulated separately. As in the example in figure 8a shows, the final pattern is shaped by different densities


(a) An old man's face

(b) Cartoon flames [21]

Figure 8: Figures created by particle tracing

of traces with overlapping lines. However, if particle tracing is used to generate a pattern where lines will grow and respond to the distributions of their neighbors while avoiding overlapping, collision detection is required.

The method that we are proposing is able to create patterns with roughly parallel lines that change directions and thicknesses smoothly. In general, our approach to tracing particles is based on the idea that we can let them grow towards different directions, adjusting their movements to their neighbors and the environment. Some rules are used thereby controlling their behavior. A new particle is added between two neighbors whenever the distance between them gets too far; and a particle will be terminated if the distance gets too short. Other rules will be explained in chapter 3.

2.4 Growing Curves With Particle Tracing

Various algorithms have been studied to create stylized images with particle tracing. Xu and Mould [21] proposed a method using the simulation of changed particles in a magnetic field for creating artistic curves with continuously varying curvature.

A particle with a constant charge will move in a circle if we put it into a constant magnetic field. When the magnetic field and the charge change, the particle will trace out different curves. In this method, a particle is released as a parent seed, where new particles can be generated from its path creating various curves under the effect of random changing charges. Each particle will be traced until it crosses another path or moves beyond the screen. When a particle is terminated, it will be released again at its initial location with a new set of variables to trace out a shorter curve. Figure 9 shows the process of filling a space with magnetic curves: a parent particle is released in the middle, then more branches are growing out from its path. After recursively releasing particles and spawning new ones, a stylized tree is formed in the bottom-right corner.

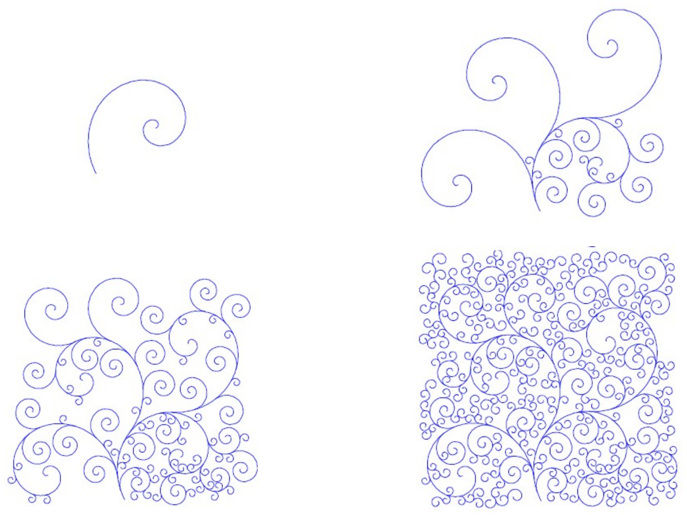


Figure 9: Creating curves by recursively releasing particles

Similarly, Li and Mould used particle tracing to track growing curves, thereby tessellating a region [12]. The main idea of this method is to release particles with different positions and directions, and then track the path of each particle until it reaches another curve or a region boundary. Some principles are used to produce natural partitions. For example, particles share similar properties in order to grow curves with similar shapes. Also, the spacing between particles and the length of paths are controlled so that no narrow or short curves can be drawn. Moreover, some randomness is added to particles' properties thereby obtaining irregular tessellations. Figure 10 shows some artistic tessellations generated by this algorithm. We notice that the curves are about parallel, which is also one of the most important features in our work; however, these parallel lines are not shaped by intelligently checking spacings but produced by proper initial distributions.

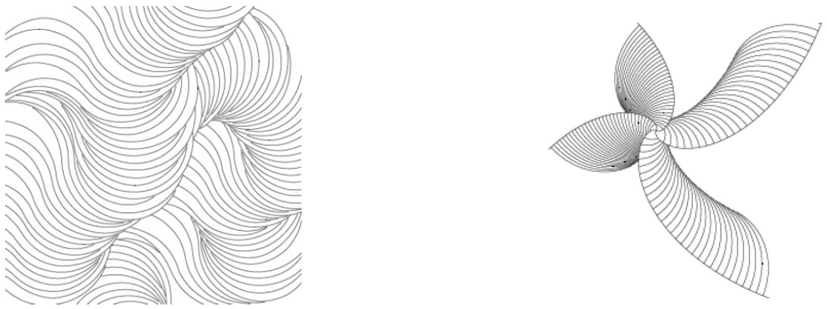


Figure 10: Tessellation created with growing curves

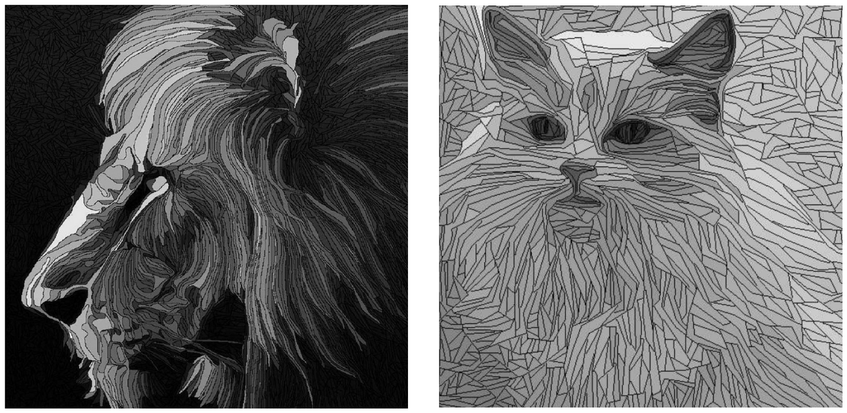


Figure 11: Reproducing images in mosaics style with growing curves

With the technique of magnetic curves mentioned above, and a vector field computed with the "edge tangent flow"(ETF) [11], each particle is able to update the force that affects its movement every timestep. This approach is able to reproduce an input image into an artistic stylization which also preserves texture information. Figure 11 shows two examples in a mosaic style. First, particles are released with a initial distribution computed with a structure-aware stippling technique [13]. Second, with the tessellation method, the movement of particles depends on the force from a vector field generated from the ETF. Several rules are applied to prevent particles from suddenly changing directions greatly and to help them trace out texture features without noise. Finally, the tiles of the tessellation are colored with the average colors in the input image. The furry texture of the lion, and the hair of the cat, are nicely shown in the example.

2.5 Summary

Overall, previous work in the area of line-drawing rendering is able to produce quality illustrations. However, none of the work can create such a style that almost parallel lines are about evenly spaced, tracing out features with different densities and leaving smooth turns.

The multi-agent particle system we propose uses the basic idea of particle system with intelligent agents which allow our particles to communicate with others as well as the environment. Our system is able to produce images in a line-drawing style with almost parallel lines. The main algorithms will be explained in the following chapter.

Chapter 3

Algorithms

We proposed a coordinated particle system which can release particles, trace out parallel lines and create smooth natural patterns. The main idea is to leave parallel traces by coordinating the movement of the particles. As we discussed in chapter 2, previous methods cannot create irregular, parallel, natural lines with the capability of changing directions and thickness smoothly.

In the coordinated particle system, particles store their neighbors' IDs – particles are stored in a linked list and each particle stores its previous and next ones. A particle will move alongside its two neighbors. During the simulation, some particles will be killed and some new ones will be generated and inserted into the list, but for each particle, its previous and next particles will always be its neighbors. In addition, we have two special particles: the head and the tail, which are the first and the last in the list respectively. At every moment, each particle will check its distance with its two neighbors and try to stay in the middle. Therefore, as the particles move, the traces are drawn with each pair of neighbors almost parallel.

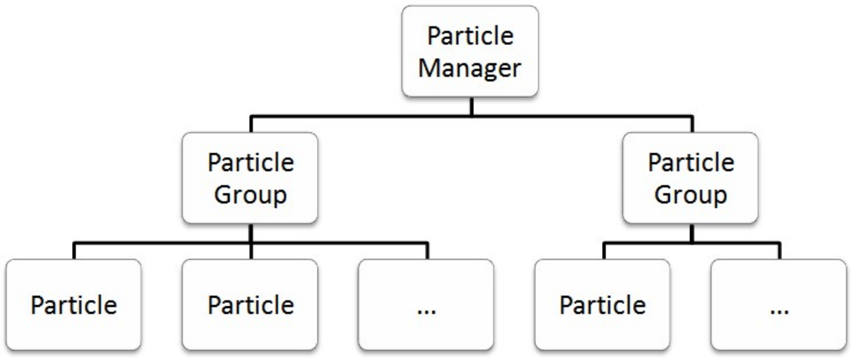


Figure 12: The structure of our particle system.

3.1 Overview

The main structure of our software is shown in figure 3.1. Particles can be divided into different groups, and controlled by a central manager. In each group, particles will have similar directions and use the same feature map used to indicate to particles if they are inside specific areas. This hierarchical structure makes sure that particles from two groups will have different independent behavior. The purpose of having multiple groups is to create the effect of shaping objects without actually drawing edges. For example, in figure 1, we are able to distinguish the man from the background not only because of some visible edges and intensity differences, but also because the two groups of lines have different directions. Figure 13 shows one of our results from using multiple groups of particles. We recognize a star in the middle despite not drawing any edges.

The main steps of creating patterns are performed by particles – the lowest level in the hierarchy. Each particle will go through three main stages: birth, coordinated movement, and termination.

There are two kinds of birth in this system: initial releasing and being created by parent particles. The initial positions for distributing particles can be different. For

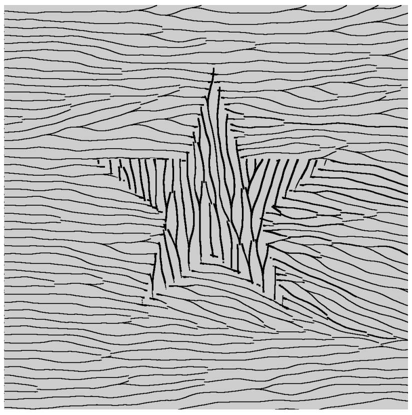


Figure 13: Two groups of particles with different directions form a star.

example, we can release particles on the edges of objects; however, particles tend to leave smoother traces if they are released at the boundaries of the canvas, as we will explain later. As for the second type of birth, we set a maximum spacing for each pair of neighboring particles. If the distance between them exceeds the maximum, a new particle will be created from one of its parents (the new particle separates from an old one's path).

Coordinated movement is the actual step of forming interesting patterns. Each particle's spacing with its two neighbors is checked throughout the whole simulation, and an additional direction obtained by applying sigmoid movement is added to the particle's direction to help it move to the middle of its two neighbors. In our project, the coordinated movement uses sigmoid curves – whenever a particle needs to move to the right, instead of turning towards the destination sharply, it follows the curve of a tangent function; conversely, if it moves to the left, a cotangent curve will be applied. By configuring the variables, particles are able to follow sigmoid paths and change directions smoothly. The update of a particle's position vector $\overrightarrow{pos}$ can then be achieved by Euler integration, as shown in equation 1, where $\overrightarrow{d}$ indicates the direction vector, and the timestep Δt is 0.01.

$$\overrightarrow{pos}_{new} = \overrightarrow{pos}_{old} + \Delta t \times \overrightarrow{d} \tag{1}$$

A particle is terminated when it moves too close to particles of its group. If a particle is going to cross a neighbor, then after terminating this particle, the neighbor is relocated to the midpoint between their positions. Otherwise, both of the two colliding particles are killed. More details are given in section 3.3.

In order to reproduce an input image with our line-drawing style, we apply tone-matching. A spacing map A is built associated with the tone value in the image using the principle that dark areas have small spacings and light areas have wide spacings. At each position x, a desired spacing $A(x)$ can be obtained from A. In general, particles will change their directions all the time to stay at the distances indicated by A. In particular, a new particle is created if two neighboring particles are too far from each other, and if the two neighbors are too close to each other, one of them will be killed.

A feature map B is built to inform particles where to draw. A particle p will be told if its current position pos belongs to an area that has features needed to be drawn by checking the boolean variable stored in $B(p.pos)$. If $B(p.pos) == true$, then p is inside the area. With the knowledge of its location relative to features, a particle can then choose to draw or just pass through the areas without leaving a trace.

3.2 Coordinated and Sigmoid Movements

The coordinated and sigmoid movements are the two main behavior that particles perform. In general, every particle acts in a coordinated way with neighbors in order to have almost parallel traces.

Figure 14 is an example of the desired behavior of particles. Suppose we have particles A, B, C and D going towards the direction of the arrow. At the beginning, B is not in the middle of its neighbors A and C, so the particle B should move maintaining the main direction as others while turning to C slightly, and A should move away from B as well. By the end, we want the curves about evenly spaced.

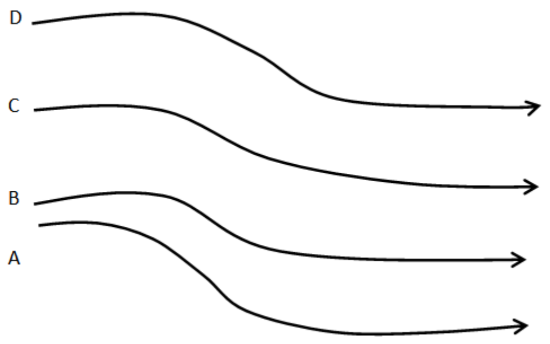


Figure 14: An example of desired coordinated movements

The direction for each particle is affected by three factors: its own current direction, the average bearing of its two neighbors, and the direction that points towards the middle of the neighboring lines. At every step, a particle will compute a new direction taking into account the three factors. The equation of computing a new direction vector $\overrightarrow{d}_{new}$ for particles at each moment is the following:

$$\overrightarrow{d}_{new} = (1-\alpha)\overrightarrow{d}_{old} + \alpha(\overrightarrow{d}_{aver} + \overrightarrow{d}_{sig}), \tag{2}$$

where $\overrightarrow{d}_{old}$ represents the current direction, $\overrightarrow{d}_{aver}$ is an average of the two neighbors' directions, and $\overrightarrow{d}_{sig}$ stands for the bearing computed for sigmoid movement. Since we want particles to move along with their neighbors, the $\overrightarrow{d}_{aver}$ is used to make particles all move in a similar direction. To realize the sigmoid movement, we apply $\overrightarrow{d}_{sig}$ to add a direction which depends on the location on a tangent- or cotangent-like curve that the particle is currently in. Since we want particles to change their directions slowly despite the influence of the neighbors and the sigmoid movements, thus avoiding sharp turns, we use $\overrightarrow{d}_{old}$ as a base direction and change it slightly at each timestep. Note that α is a small value less than 1, which is multiplied with the sum of $\overrightarrow{d}_{aver}$ and $\overrightarrow{d}_{sig}$. Correspondingly, $1 - \alpha$ is multiplied with the base direction $\overrightarrow{d}_{old}$. With α, we ensure that only a small amount of new value is added to one particle's old direction, so that the particle will not turn sharply.

Figure 15 shows some examples using different α values. When $\alpha = 0.0001$, particles collide excessively since they barely change directions with neighbors and almost no sigmoid movement is performed. As α goes up, particles become more and more sensitive to their neighbors' behavior. In the two images at the bottom, particles cannot perform normal behavior because almost all of them collide with neighbors – if one particle suddenly turn because of some randomness added to the direction, all other particles will be affected and turn with it immediately. Therefore, we set $\alpha = 0.001$ as default to achieve an output with smooth lines as shown in the top-right image.

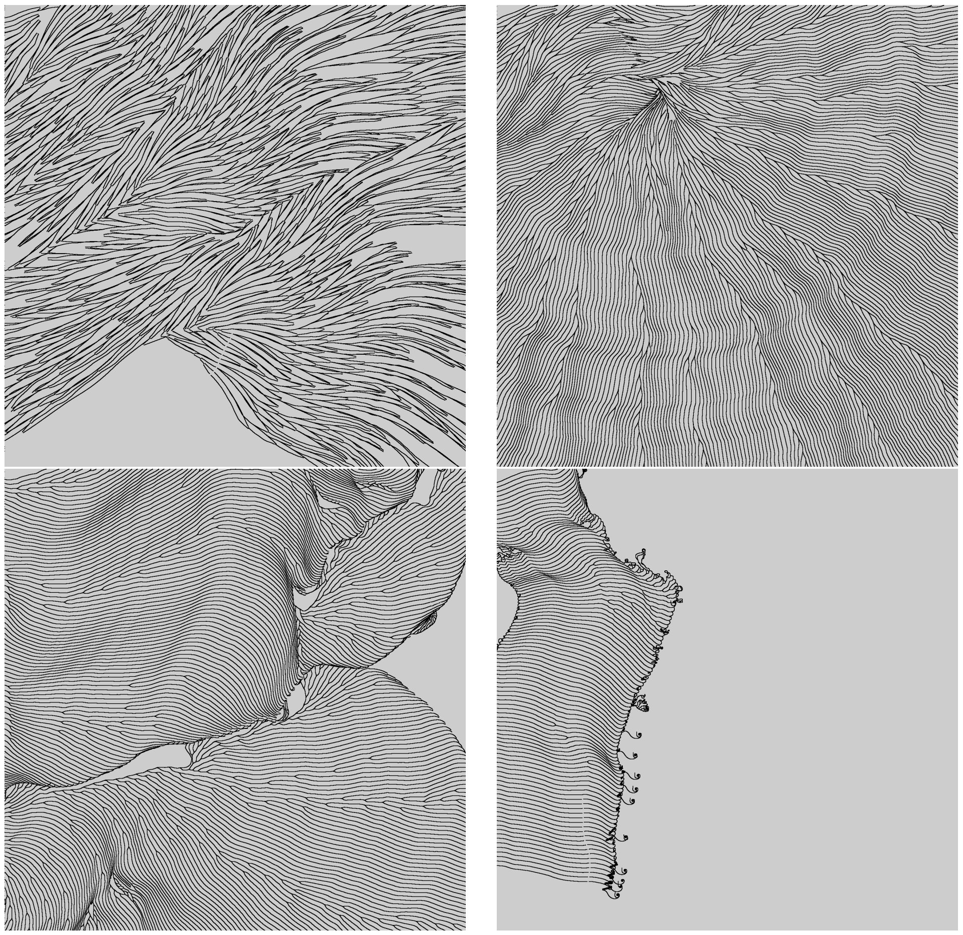


Figure 15: Top left: $\alpha = 0.0001$; top right: $\alpha = 0.001$; bottom left: $\alpha = 0.01$; bottom right:$\alpha = 0.1$.

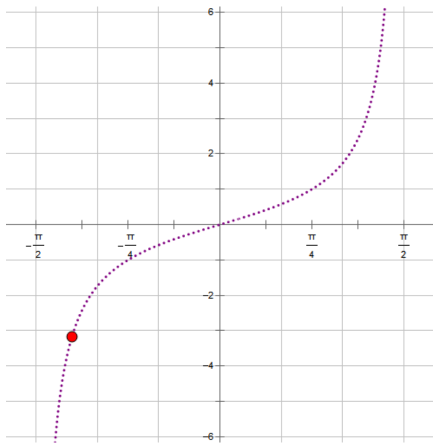


Figure 16: A curve of tangent function from $-\pi/2$ to $\pi/2$. A particle moves towards $\pi/2$.

To produce particles' coordinated movement without making sharp turns, we use sigmoid curves to guide our particles. Since the shape of sigmoid curves is like a letter "S", by following this kind of curves, particles can smoothly change directions. To be specific, we apply a tangent function to approximate a sigmoid curve. For example, in figure 16, suppose there is a particle moving on the curve of a tangent function from left to right; instead of moving straight to the right side, the particle follows the curve which allows it to leave a smooth trace as we desired.

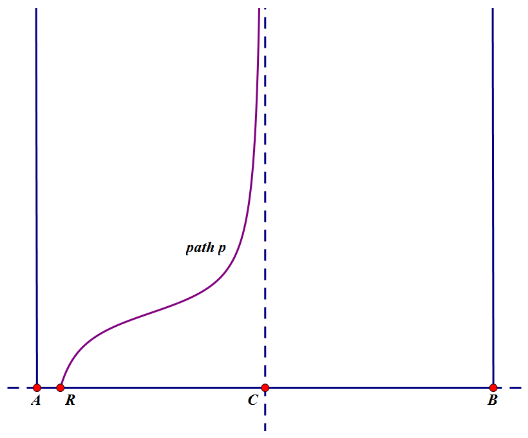


Figure 17: The path p of a tangent-like curve which leads particle R to a middle line between A and B.

To compute $\overrightarrow{d}_{sig}$, we use the tangent function. For example, as shown in figure 17, particle R moves in cooperation with the two neighboring particles A and B. The distance between A and B is known, and C is the middle line. We want R to move away from A to the middle line; p is the desired path. The vector $\overrightarrow{d}_{sig}$ for R is computed:

$$\overrightarrow{d}_{sig} = (\sin(\varphi_{sig}), \cos(\varphi_{sig})) \tag{3}$$

where φ_{sig} is the corresponding bearing of vector $\overrightarrow{d}_{sig}$. The equation of finding φ_{sig} is the following:

$$\varphi_{sig} = \arctan(slope) \tag{4}$$

$$slope = d\tan(\theta)/d\theta = \sec^2(\theta) \tag{5}$$

$$\theta = \pi \times (S_{AR}/S_{AC}) - \pi/2 \tag{6}$$

Note that here we only gave the example when particles need to perform a tangent-like movement (move to the right-hand neighbor). In fact, the sigmoid movement is decided by the value of S_{AR}/S_{AC} where S_{XY} denotes the distance between X and Y. If $S_{AR}/S_{AC} < 1$, a particle will follow a tangent-like curve moving to its right side; a cotangent-like curve will be used if $S_{AR}/S_{AC} > 1$ for moving to the left. For handling the cotangent movement, we simply apply equation 7 instead of 6.

$$\theta = \pi \times (2 - S_{AR}/S_{AC}) - \pi/2 \tag{7}$$

In the example, A, R and B are on the same line. However, in practice they are usually not, which requires us to use $(S_{AR} + S_{RB})/2$ instead of S_{AC}.

As equations 4 and 6 suggest, there are three steps for computing the direction of particle R at each moment. At the first step, we use equation 6 to get the corresponding θ – the angle in the tangent function corresponding to the location of particle R. Since each particle has a coordinate vector indicating its position, and it stores the IDs of the two neighbors beside it, we can easily get S_{AR} and S_{AC}.

At the second step, we compute the slope value of R on the desired path p. In figure 18, we use a solid line to represent a tangent function curve, which is associated with the tangent-like path p in figure 17, and a dashed line to indicate its derivative. At any point X on the tangent curve with θ denoting its corresponding angle, we can find its *slope*, equal to the derivative of $\tan(\theta)$.

Finally, the bearing φ_{sig} for changing a particle's direction towards the middle of its neighbors is obtained by getting the angle of arctan($slope$): see equation 4.

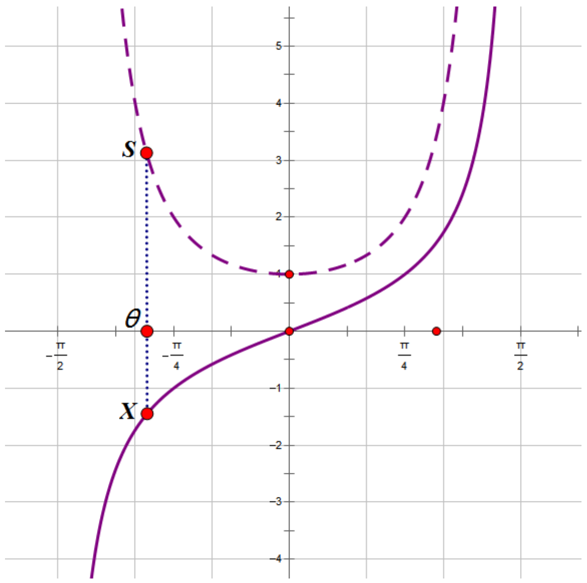


Figure 18: X on a tangent curve and its corresponding point S on the curve of the derivative of $\tan(x)$

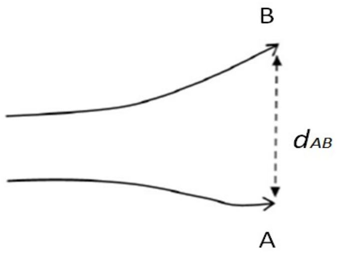
3.3 Birth and Death Control

In a line-drawing illustration, a bright area will be drawn with less strokes, or even left blank, but for a dark area, more strokes will be used. Similar to the artist's method, our system is capable of changing line thickness, and more important, the amount of particles to place more or less strokes, thereby matching the tones. Our method manages the number of particles with birth and death control – when two neighboring particles get too close, one of them will be terminated, and if they move too far away from each other, a new particle will be generated.

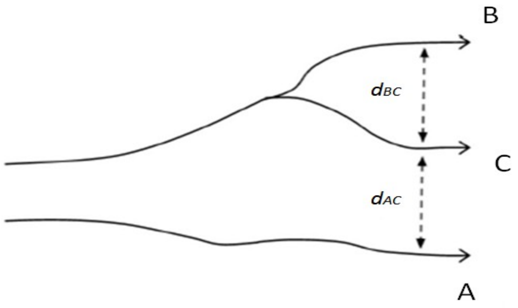
To be specific, each pair of neighboring particles is checking their distance constantly. Once the distance becomes larger than a threshold T_b, the birth control is triggered; if the distance is smaller than a threshold T_d, the death control is applied. The thresholds T_b and T_d are computed as shown in equations 8 and 9.

Figure 19 shows the procedure of birth control. The distance $S(A, B)$ of neighboring particles A and B is checked. When $S(A, B) > T_b$, a new particle C is created from the path of B with a direction which is the average of its parents. In figure 19b, we see how B and C spread out smoothly and all three particles form a new structure of parallel lines again with almost equal spacings. Note that with the technique of coordinated movement, C approaches the middle of A and B in a beautiful sigmoid curve.

Although all particles move in a coordinated way, sometimes it is hard to balance the distance when two particles are too close and about to collide. Also, when particles enter a lighter area where a larger spacing is required, it is difficult or impossible for them to push their neighbors away in order to match the required spacings. In these situations, death control is triggered. As shown in figure 20, particle B detects a small distance $S(B, C)$ with neighbor C, $S(B, C) < T_d$ where T_d is a death threshold; immediately it kills C and jumps to the middle of its new pair of neighbors A and D. Note that in practice we can kill either B or C. Along with the change of location, B's direction is also modified to the average of C and itself. After the coordinated movement, the three particles finally obtained almost the same spacings and directions. Figure 21 shows a real case of birth and death control, where particles behave just like we described.



(a) Particles A and B move too far from each other



(b) A new particle C spawns from B adding one more line to the image

Figure 19: The procedure of birth control

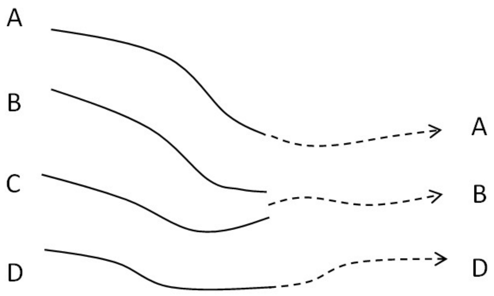


Figure 20: Death control: kill C and move B to the middle

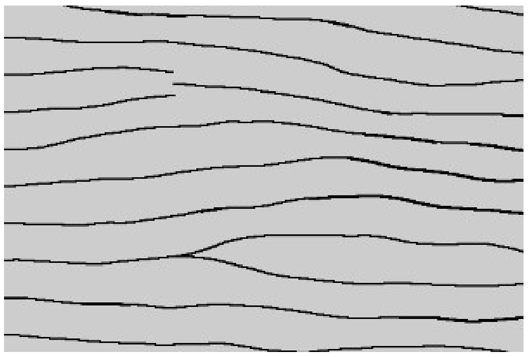


Figure 21: A real case of birth and death control

We store a base threshold for each particle, and update it every timestep according to the spacing map. The birth and death thresholds are obtained from two neighbors' base thresholds. By dynamically updating the base threshold, we are able to avoid a burst of birth or death. It is easy to set up the birth and death thresholds for two neighboring particles A and B. Equation 8 shows how to compute birth threshold T_b and equation 9 the death threshold T_d:

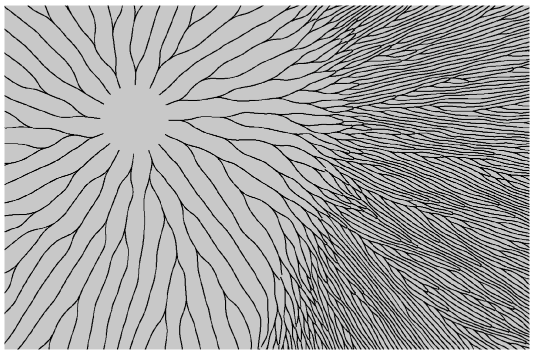
$$T_b = (T_A + T_B)/2 * \delta_b \tag{8}$$

$$T_d = (T_A + T_B)/2 * \delta_d \tag{9}$$

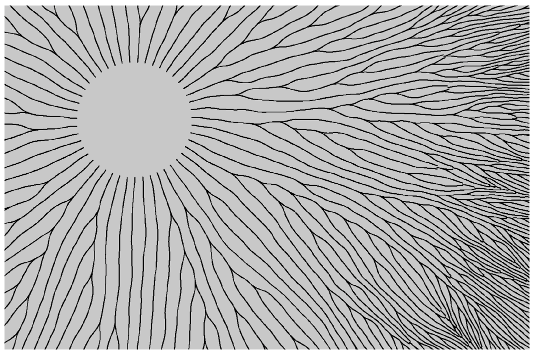
where T_A and T_B indicate the two base thresholds stored in particles A and B. Initially, we set the base threshold value T_x for a particle x to the value stored in the spacing map A at the position $x.pos$. At every moment, when the particle x moves to a new location pos, it will compare its own threshold T_x with the threshold $A(T_x.pos)$ stored in the spacing map. If $T_x > A(T_x.pos)$, subtract a small value β from T_x; if $T_x < A(T_x.pos)$, add β to T_x. The value β is used to control the magnitude of T_x updates. The bigger β is set, the more quickly a particle reaches the threshold stored in the map. In practice, users can define β depending on their preferences. Here we use $\beta = 0.001$ for all the results shown in this report.

After taking the average of two neighboring particles' base thresholds, we multiply the average with a birth or death offset δ. Here we set $\delta_b = 1.5$ and $\delta_d = 0.5$ respectively. These values were chosen empirically, with respect to a standard spacing value equal to one. By dynamically computing the birth and death thresholds, the amount of strokes can be changed according to the intensity of an input image, in a smooth way that will not cause a burst of birth or death.

Consider the example in figure 22 where we set a much smaller default spacing on the right half of the screen. Both figure 22a and 22b show the ability of creating new particles as demanded. In the top image, we update a particle x's threshold by applying $T_x = A(T_x.pos)$ directly; in the bottom image, we use β to control the update of T_x. Therefore, the top image shows a burst of birth when particles enter the right half of the canvas immediately while in the bottom image, new particles are generated gradually.



(a) Particles burst into birth immediately



(b) The number of new particles increases gradually

Figure 22: In each image, a smaller spacing is applied on the right half side of the canvas.

The burst of birth or death can cause problems. Figure 23 shows an example. When particles A, B and C enter the area with a smaller spacing, they start to give birth to enforce the small spacing. Two birth events occur from time T_1 to T_3. Some of the newborns of C are outside of the smaller spacing area, and they try to move into the bottom area of the canvas thereby obtaining a larger distance with neighbors.

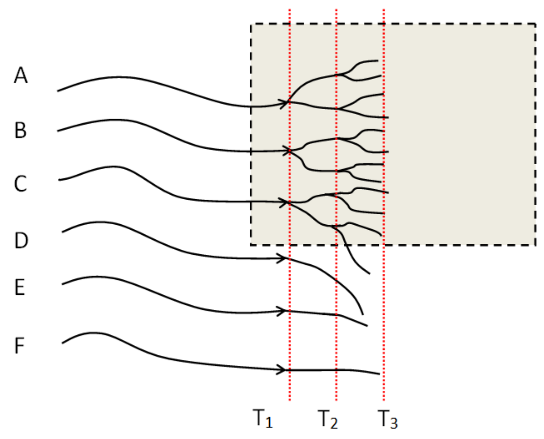


Figure 23: Particles A to F move in parallel. An area with a smaller spacing is shown in a gray box. Three dashed lines represent time T_1, T_2 and T_3.

However, D, E and F do not need to respond to the smaller spacing, which means they do not have sudden changes of directions because of the birth control. Although D, E, and F still change directions and keep even distances with neighbors, the changes are small since we still want particles travel on their old paths. As we mentioned in equation 2, the average of neighbors' directions is applied, as well as a small value α to control the amount added to one's direction. However, the changes given by a newborn's direction affect D continuously even though they are small, since D is directly next to the newborn. E is affected by its neighbor D's changes as well, but not as strongly as D was affected by its newborn neighbor. After taking the average of neighbors' directions, E changes less than D, and F is barely affected. The big change of direction causes D to turn towards E while E does not have time to turn with it, which is highly likely to trigger a death control. This case also appears in the middle of the bottom of figure 22a, where we see a death event.

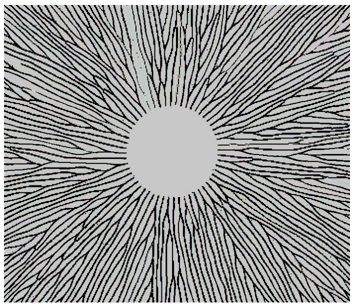
Birth control is triggered when the distance between two neighboring particles is larger than T_b, and a death control is triggered when the distance is smaller than T_d. By updating particles' thresholds with β, particles take longer to make T_b small enough to trigger birth control, or make T_d large enough to trigger death control, since each timestep particles will only add or subtract a small amount to their thresholds. Therefore, less birth or death will be triggered within a certain time. For example, in figure 23, if particles only update their birth thresholds with a small amount every time. The each of A, B and C may only give birth once from T_1 to T_3, and so D will not need to greatly change its direction since the newborn beside it will have time to adjust itself to stay parallel with D before the next birth.

3.4 Distribution of Particles

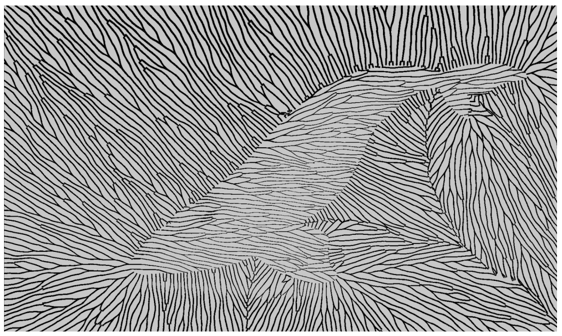
As we mentioned in section 3.1, we can release particles in different manners, for example, at the edges of objects or simply at the boundaries of the canvas. For convex polygons, a distribution on edges gives us nice curves spreading out. In figure 24a, the distribution is on a circle, and all particles have initial directions perpendicular to the edge. The traces of particles are almost parallel. However, when the object is a concave polygon, particles are likely to collide with others on the opposite side and to be terminated at the same time, which leaves a noticeable "death line" as seen in figure 24b – an irregular "line" formed when particles that come from two sides collide and die.

In fact, since all particles are able to change the direction in a coordinated manner, particles should not suddenly turn towards others and collide. The direct cause is a large sudden change of direction. For example, if one particle changes direction greatly, then its neighbors will change along with it so they can stay parallel. However, if the particle changes too fast for its neighbors to catch up, it is highly likely for it to collide with a neighbor that has not yet turned (see the problem in figure 23). When particles collide with neighbors one by one, then a "death line" appears.

To prevent the burst of birth or death, we apply β to control the magnitude of updating thresholds T_b and T_d as we described in section 3.3, which therefore decrease



(a) Particles spread out from a circle smoothly



(b) Initial distribution on a whale's edges. A "death line" is formed on the tail of the whale when particles collide

Figure 24: The distribution on convex and concave objects

the appearance of "death lines". Here we give two more suggestions for minimizing the "death lines".

One approach is to use high resolution input images so that the tones can change gradually, and particles do not need to give birth or death often since the tones changes less in the same spatial distance. Another solution is to release particles at the boundaries of the window – doing this always avoids the distribution on a concave shape edge, which will also reduce the appearance of "death lines". Moreover, distribution on a straight line instead of a concave curve helps improve the effect and achieve smoother traces.

In our project, we assign different initial directions to particle groups in order to achieve various texture effects. Figure 25 shows an example where we used three groups of particles. The first group is released on the left boundary of the canvas taking care of the background and the face of the panda. The second group is distributed along the bottom boundary drawing all dark parts such as eyes, ears and the body. The third group fills the light areas on the face, and is created on a line which is about 45 degrees to the x axis. Since different groups can be released with different initial directions, the more groups we have, the more possibilities for particles' orientations. However, more groups also means more manual configurations; therefore, we usually use two particle groups only, which is enough to generate a nice result.

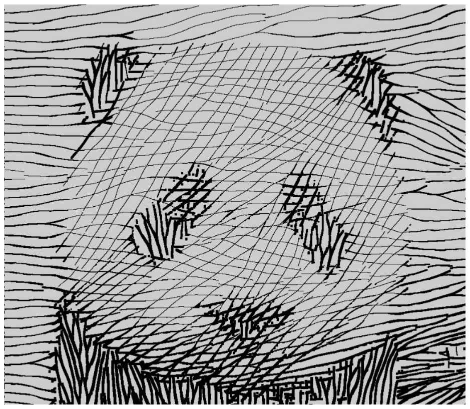


Figure 25: A panda formed by three groups of particles with different directions

3.5 Assistant Maps

To help particles perform coordinated movements, two assistant maps are computed beforehand: a spacing map and a feature map. The spacing map is used to indicate the desired spacings between particles and the feature map informs particles if they are inside feature areas that need to be drawn.

3.5.1 Spacing map

To depict features in an input image clearly and smoothly, particles need to match the tone of the image. In the right image of figure 1, the left-top corner is drawn by an artist with less strokes so that a lighter area is shown. Similarly, we assign narrow spacings (more particles) to darker areas and large spacings (less particles) to lighter ones.

We compute a spacing map from the intensity of a grayscale input image to indicate the required distance for particles. At each pixel, there are two factors affecting the spacing λ: a variable μ and a base-distance σ. λ is obtained as $\lambda = \mu * \sigma$. The variable μ is related to the scale of the overall spacing. Given a minimum min and a maximum max, we rescale the intensity level of an input image into the new range from min to max. The new value of the intensity in each pixel is then assigned to the corresponding μ. Note that σ can be set by the user to any value to give a desired spacing range.

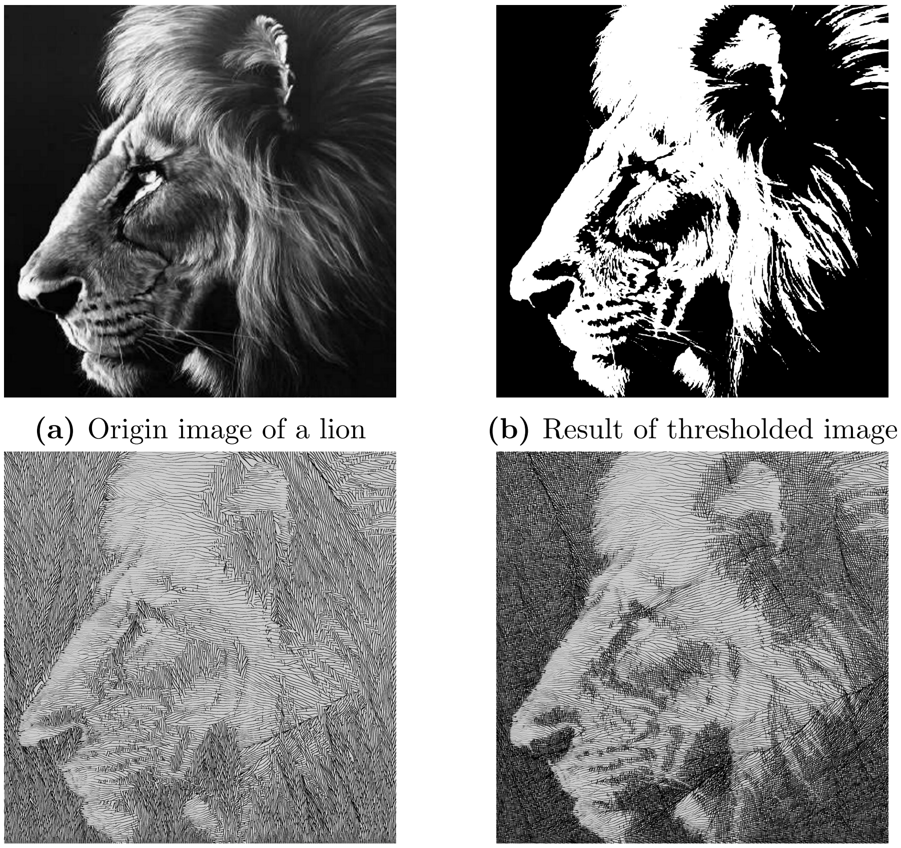
3.5.2 Feature map

To put heavier strokes in desired areas, and to give a unique texture to a certain place, we associate areas with different groups of particles. To be specific, we generate a feature map and store a boolean variable for each pixel in the map: if the pixel belongs to a feature area, set the variable to $true$; otherwise, set it to $false$. One or more groups of particles can trace out main features with this feature map.

The map can be built by segmentation, and each segment can be associated with a particle group. We allow one group to trace out multiple segments, so one feature area which is segmented into chunks can still have particles with about the same orientation. We also allow one segment to be painted by several groups, so the result can have a heavy tone as more than one group of particles move through.

Although more groups means the image is traced out by more particles with more main directions, we found that two groups can already produce a nice result; figure 26d is an example of using only two groups. Therefore, to obtain proper segments for only two particle groups, we simply use thresholding to compute the feature map of an input image. One group of particles traces out only the dark part of the map, and the other group can trace out either only the light places or the entire image. The differences are shown in the following.

For example, figure 26b is the result after thresholding an image. This image was also used by Li and Mould [12]; see their result in figure 11. We used two particle groups to trace out the black and white areas in the image 26b separately – the two groups did not cross, and the result is shown in figure 26c. If we only assign the black area to one group while the other group travels through the whole canvas, we obtain an effect where the main features become more obvious due to increased contrast. The result is shown in figure 26d.



(a) Origin image of a lion **(b)** Result of thresholded image

(c) Two particle groups trace out features separately **(d)** The crossing of two groups makes the tone heavier

Figure 26: Two particle groups tracing out objects with a feature map

3.6 Collision Detection

Collisions of particles from within a group are not allowed. There are three types of collisions: colliding with neighbors, colliding with particles other than neighbors, and crossing with the traces of others. Our death control will prevent the first case, while the second one is detected and stopped by grid collision detection. The third one is prevented by checking sensors of particles, which will be explained next.

The whole canvas is split into grid cells. For example, if we divide a canvas with a resolution of $1000 * 1000$ with a $10 * 10$ grid, then each grid cell contains $100 * 100$

pixels. When a particle enters a grid cell C, cell C will store its ID; when it leaves the cell, its ID will be deleted. When a particle moves to a new grid cell, it first requires a list of all the particles within the same cell. Death control is then used by checking the distance between this particle and all others on the list. Unlike the normal death control which moves a parent to the middle, all particles that are too close together are killed, so as to leave a clean death area – particles may collide with others coming from various directions at the same time, and killing all the involved particles will prevent them crossing others' traces.

The grid collision detection prevents particles from colliding with the ones which are currently inside the grid cell. Since each cell will remove particles' IDs when they move out of the cell, the traces that belong to those ones cannot be detected by simply requiring a list of particles currently inside the cell. Therefore we let each particle mark its traces at intervals. A particle x uses a counter Z to accumulate the distance between its current position *x.pos* and the last one *x.lastpos* all the time:

$$Z+ = x.pos + x.lastpos \tag{10}$$

We define a threshold T_Z for dropping sensors: whenever $Z > T_Z$, the particle x will leave its ID in the current location, and reset $Z = 0$. In the meantime, the 8-neighborhood of x is constantly being checked. Whenever an ID of others is detected, kill particle x. Note that here we only kill the particle that is currently checking collisions. Since the owners of the traces that the current particle will cross can be far away and will not affect the current particle, we do not need to kill them as well.

Crossing the screen boundaries is not considered a collision. In some cases, after particles left the window, there are still places that have not been drawn. However, since particles can be affected by neighbors under coordinated movement, they can be dragged back, which will give us a result image with traces returning from the outside of the screen. In fact, in our system, particles who move out of window will be set to *invisible* – no traces will be left – to be drawn again once they move back. By doing this, we ensure that the result image will not be affected by the actual size of the screen. After all particles in a group have left the window, the simulation for this group can be terminated.

3.7 Additional Controls

The basic theory of our coordinated particle system is introduced in the previous sections. More rules are applied in order to make a more natural and irregular result, and we describe these additions here.

First, we apply some randomness to the directions of particles. Due to the coordinated and sigmoid movement, particles move and stay in the middle of their adjacent neighbors automatically, and the spacings between them tends to be perfectly equal. However, we do not find this "perfectly equal" spacing in real hand-drawn images, and that is why we use random factors to make results seem more natural.

In addition, we use *tilt control* to manage the rate of adding new randomness since we do not want particles to change directions too much nor too often. Specifically, we assign a variable τ for each particle to indicate its current tilt. Initially $\tau = b$ where b denotes the particle's bearing (note that direction $\overrightarrow{d} = (\sin(b), \cos(b))$). Every time a particle updates $\overrightarrow{d}$, we compute its bearing b from $\overrightarrow{d}$. We then create a Δr randomly and compare the sign of τ and Δr:

$$\Delta r = \begin{cases} -\Delta r & \text{if } \Delta r * \tau < 0 \\ \Delta r & \text{otherwise} \end{cases}$$

and add the computed Δr to b:

$$b = b + \Delta r; \tag{11}$$

and then we update $\tau = \Delta r$ for the next time.

Overall, Δr is the direct factor that gives particles randomness, but the sign is decided by τ. Δr should be small so that it will not change a particle's direction too much. To make particles smoothly affected by randomness, we also define a random tilt limit lim for each particle. When a counter I of timesteps accumulates to the amount of lim, we flip the sign of τ and reset the counter I and assign a new random lim. By doing so, we are actually using two random factors to add irregularities to particles' behavior – the first factor Δr directly controls the bearing while the second factor τ applies limits to the first one.

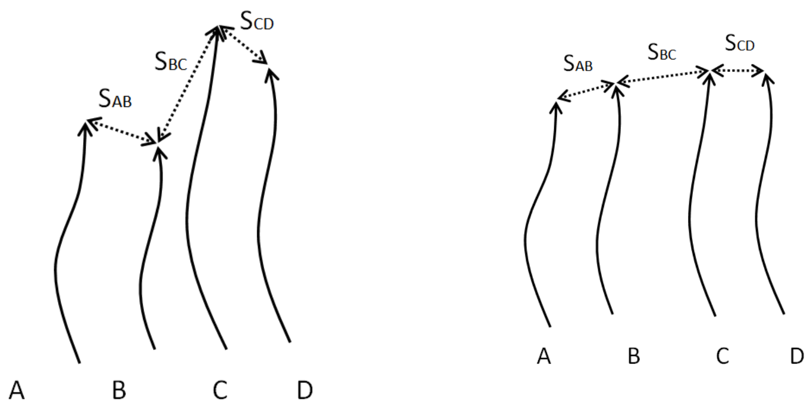


Figure 27: Particles A, B, C and D move together with distance $S(A, B) < S_(B, C)$ and $S(B, C) > S(C, D)$. Left: according to the rules applied in coordinated movement, particle B and C will move towards each other and B is very likely to cross with C's trace; right: B and C will tilt to each other without collisions

It is also important to move all particles in a line, which means no particle will be too far ahead or behind compared to its neighbors. For example, in figure 27, particles A, B, C and D are moving together with the distance $S(A, B) < S_(B, C)$ and $S(B, C) > S(C, D)$. According to our coordinated movement, particles will always move to the middle of their neighbors; and the middle point is computed by taking the average of the sum of distances with two neighbors (see section 3.2). In the example, we can know that particle B is going to move to the right and C to the left. In the left image, since particles are not moving in a line, there is a high probability for B to cross the trace left by C. However, in the case that all particles maintain almost a line, there is no chance that B and C will collide.

We make particles maintain a line by pushing particles which are left behind a bit forward, and dragging back those running too fast. First, for each pair of neighboring particles A and B, a vector $\overrightarrow{AB}$ is defined and normalized. We then calculate A's direction vector $\overrightarrow{d}$'s projection on $\overrightarrow{AB}$ by applying dot product, and the value ω is $\omega = \overrightarrow{d} \cdot \overrightarrow{AB}$.

If $\omega > 0$, then A is left behind by B, while if $\omega < 0$, A is in front of B. We compute a vector $\Delta\overrightarrow{AB}$ as:

$$\Delta\overrightarrow{AB} = \phi * \overrightarrow{A} * \omega/2; \tag{12}$$

where ϕ is a small value that controls the change rate.

$$A.pos+ = \Delta\overrightarrow{AB}; \tag{13}$$

$$B.pos- = \Delta\overrightarrow{AB}. \tag{14}$$

The equation 12 to 14 can be seen as a small update of A and B's positions, where the variable ω is used for deciding whether push or drag behavior should be performed. The amount of ω indicates the degree to which A is departing from a line with B. When $\omega > 0$, the larger $\Delta\overrightarrow{AB}$ is, the more value is added to A (push it forward), and the more is subtracted from B (drag it backward); when $\omega < 0$, $\Delta\overrightarrow{AB}$

is subtracted from A to drag it back, and added to B to push it forward.

3.8 Summary and Pseudocode

In this chapter, we explained the main algorithm of coordinated movement. We also defined some additional rules such as birth and death control, grid collision detection. In section 3.1, we presented our system structure where a manager is used to control all the particle groups. Actually, this manager decides how many groups should be applied and which group takes care of what areas. The main simulation of each group is looped inside the manager. Although the main algorithm for simulation is the same, different groups can use a different feature map to indicate where to draw traces. For example, a particle group can allow particles to be "visible" only when they are inside the feature area; only the visible ones will be drawn on the canvas.

In order to give an explicit expression of our method, we also provided the pseudocode in Algorithm 1. The related particle group in this code is responsible for the areas in a feature map – it will only trace out the features in the map. We pass some input information to the algorithm: a spacing map computed as in section 3.5.1, a feature map obtained as in 3.5.2, a grid for collision detection, and the particle group. Algorithm 1 relies on subroutines for collision detection and for particle updating, which are given in Algorithm 2 and 3.

Algorithm 1 Pseudocode for main simulation

Input: spacing map A; feature map B; grid G; particle group W, consisting of particles P. Each particle P_i has properties: pointers $prev$ and $next$, a position vector pos, a direction variable d, and status variables $IsNewBorn$ and $IsVisible$. We denote the distance between P_i and P_{i-1} as $S(P_i, P_{i-1})$, and the total number of P as $numpt$. $IsNewBorn \leftarrow false$ and $IsVisible \leftarrow false$ for each particle P_i.
Output: a canvas with all particles drawn.

```
1. For each particle $P_i$:
1A. take the value at location $P_i.pos$ from map $B$, say $B(P_i.pos)$;
1B. $P_i.IsVisible \leftarrow B(P_i.pos)$; [see section 3.5.2 for setting up map B]
1C. if $P_i.IsVisible == true$, insert $P_i$ into $G$; otherwise, delete $P_i$ from $G$.
2. For each particle $P_m$:
2A. Take the spacing value at $P_m.pos$ from map $A$, say $A(P_m.pos)$;
2B. denote the spacing threshold stored in $P_m$ as $P_m.T$
2C. If $P_m.T > A(P_m.pos)$, subtract a small value $\beta$ from $P_m.T$;
2D. else if $P_m.T < A(P_m.pos)$, add $\beta$ to $P_m.T$.
2E. $P_n \leftarrow P_m.next$;
2F. $T_b \leftarrow (P_m.T + P_n.T)/2 \times \delta_b$; $T_d \leftarrow (P_m.T + P_n.T)/2 \times \delta_d$. [see equation 8 and 9]
2G. If $P_m.IsNewBorn == false$ and $P_n.IsNewBorn == false$, then:
2Ga. if $S(P_m, Pn) > T_b$, create a $P_{numpt}$; $P_{numpt}.pos \leftarrow P_m.pos$;
2Gb. $P_{numpt}.IsNewBorn \leftarrow true$; insert $P_{numpt}$ into $G$; $numpt+ = 1$;
2Gc. else if $S(P_m, P_n) < T_d$, kill $P_n$; $numpt- = 1$;
3. For each $P_i$:
3A. get a list $L_i$ using $CollisionDetection()$ [see Algorithm 2];
3B. kill all the particles in $L_i$; and delete them from $G$.
3C. if $P_i$ enters a new grid cell, delete it from the old one; insert it in the new one.
3D. drop $P_i$'s sensor. [see section 3.6]
3E. draw $P_i$ on canvas.
4. Update each particle's direction $d$ and position $pos$. [see Algorithm 3]
```

Algorithm 2 Pseudocode for collision detection

Input: a particle P_i.
Output: a list L_i of particles which will collide with P_i.

```
1. Create a list $L_i$, initially empty; take the grid cell that $P_i$ is in, say cell $C$;
2A. for each particle $q \in C$:
2B. $T_d \leftarrow (q.T + P_i.T)/2 \times \delta_d$; [see equation 9]
2C. if $S(q, P_i) < T_d$, add $q$ into $L_i$.
2D. For each pixel $x$ in $P_i$'s 8-neighborhood:
2E. if the sensor of a particle $e$ in $x$ is found where $e \neq P_i$, add $P_i$ into $L_i$.
3. Return $L_i$;
```

Algorithm 3 Pseudocode for direction and position update

Input: particle group R, consisting of particles P.
Output: particle group R with all particles' direction updated.

```
1. For each particle $P_m$, $P_n \leftarrow P_m.next$:
1A. $P_m.pos + = \Delta t \times P_m.d_{old}$. [See equation 1]
1B. If $P_m$ is ahead of $P_n$, push $P_n$ forward and drag $P_m$ backward;
1C. else if $P_m$ is behind of $P_n$, push $P_m$ and drag $P_n$. [See section 3.7]
1D. Add randomness to $P_m.d_{old}$. [See section 3.7]
1E. Compute $P_m.d_{new}$ [see equation 2];
1F. $P_m.d_{old} \leftarrow P_m.d_{new}$.
```

Chapter 4

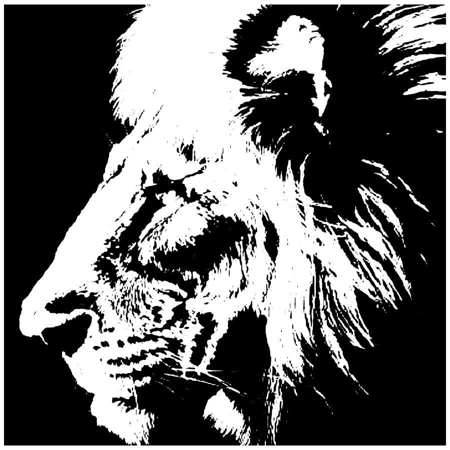
Discussion and Evaluation

We introduced some related work and explained our algorithms in the previous chapters. In this chapter, we present more results from our coordinated particle system. In particular, we provide the process of generating images with our system, as well as some results showing the effects of different parameter settings. We will also compare our results with some previous work, and then discuss the differences.

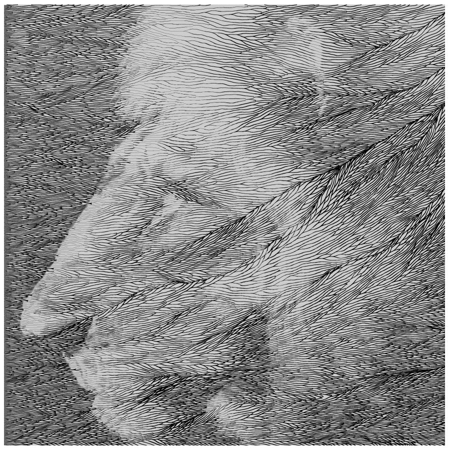
4.1 Complete Procedure of Rendering

Figure 28 shows the stages of image creation. We first take an input image, and compute a feature map by thresholding it. Two particle groups are then released. The first group traces out the features on the whole canvas, while the second group only draw the black areas in the feature map. Finally, the combination of the two groups forms our final result.

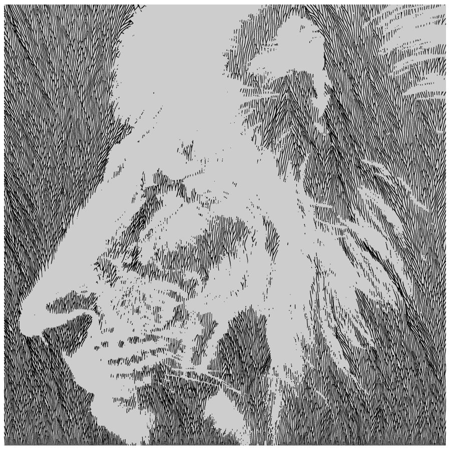
(a) Original image



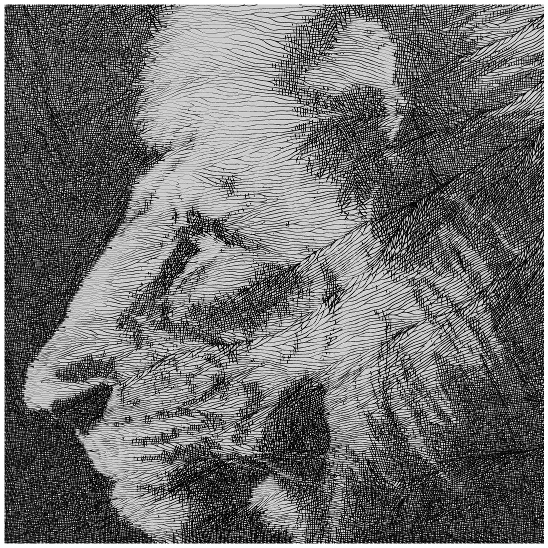
(b) Thresholding image



(c) First group of particles



(d) Second group of particles



(e) Final result

Figure 28: Two groups of particles are released separately, move according to the thresholding image. The final result is the combination of *c* and *d*.

A full sequence of rendering is presented in figure 29. We generate the first group as shown in the top two images: particles are released on the boundary of canvas and move from left to right. After the first group result is finished, the second group of particles is then released, growing from bottom to top, filling the areas that need more strokes according to the feature map.

Note that the same input image was used by Li and Mould [12], whose work was demonstrated in figure 11. We will compare and discuss more about the results later.

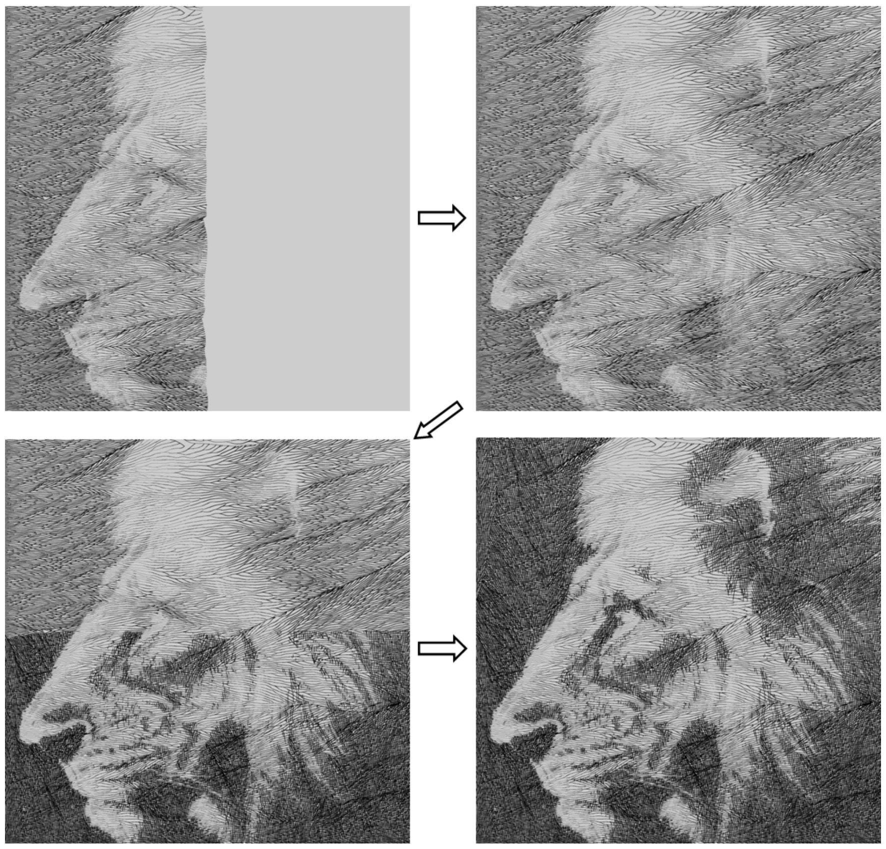


Figure 29: The procedure of creating a stylized image with our coordinated particle system

4.2 Parameters

4.2.1 Spacing and line-width setup

The quality of our system, to a large extent, relies on the spacing map, which guides the birth and death control and affects the coordinated movement. Generally, with a wider range of spacings, lines are smoother, since it takes longer to achieve a large change in the spacing. As a result, a burst of birth or death is prevented. See figure 31 where the middle and right images show long and smooth curves while the left one has a lot of birth and death of particles.

However, with a larger upper limit of the spacing, there will be more areas with a light tone in the result – particles spread out to keep a large spacing with others. For example, the background and the stripe of tiger in the middle image of figure 30 appear to be pale and lack ink, compared with the result in the top image. The reason is that, when the upper limit of the spacing is larger, the range of the spacing is bigger, which means the specific spacing corresponding to a particular intensity becomes larger as well and leads to a wider separation between lines. To achieve a darker tone in areas that need more ink while the spacing maintains a larger upper limit, we enlarge the line-width range. See the bottom image in figure 30 which we use the same spacing range as in the middle image while the line-width range is higher. As a result, the dark areas is then clearer with a higher contrast. For example, the strokes in the background and the face of the tiger are darker and convey features more clearly.

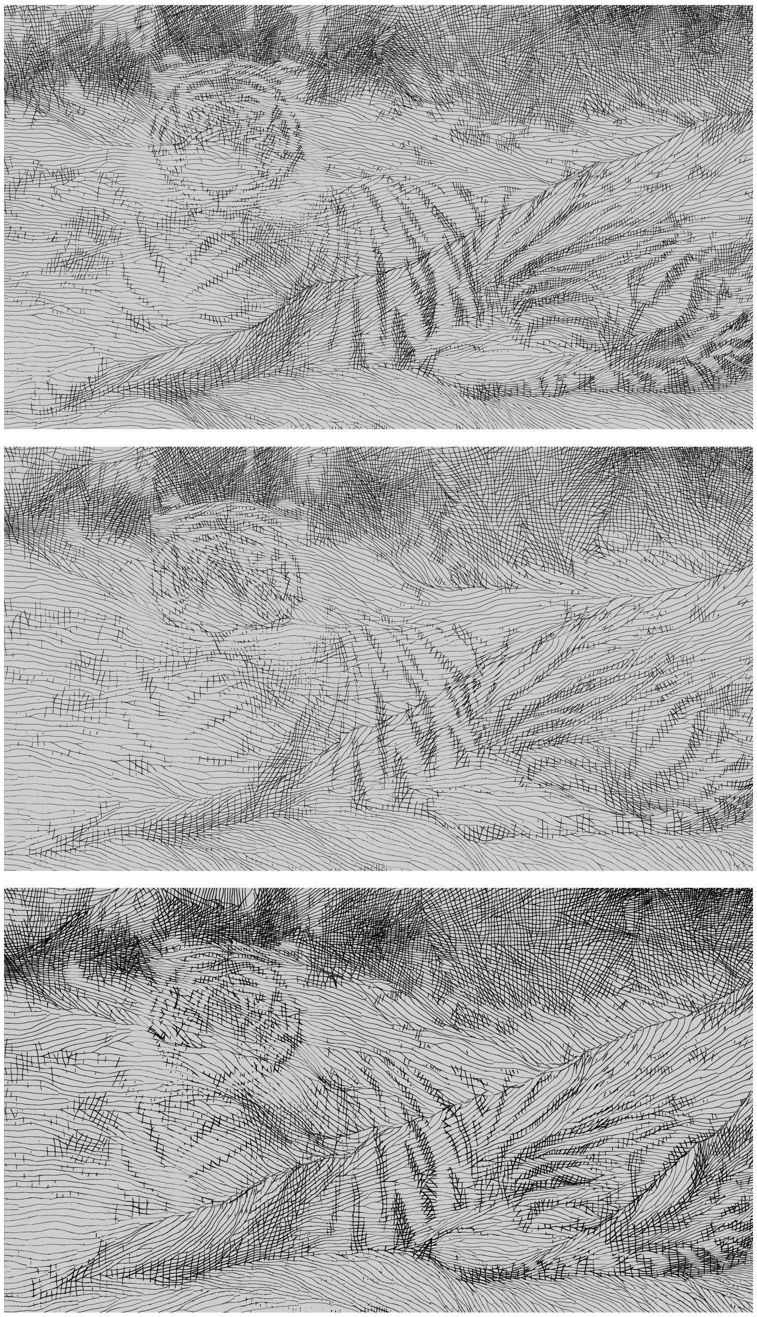


Figure 30: Top: spacing range from 10 to 50, with line-width from 0 to 3. Middle: Spacing range from 10 to 70, with line-width from 0 to 3. Bottom: Spacing range from 10 to 70, with line-width from 0 to 5.

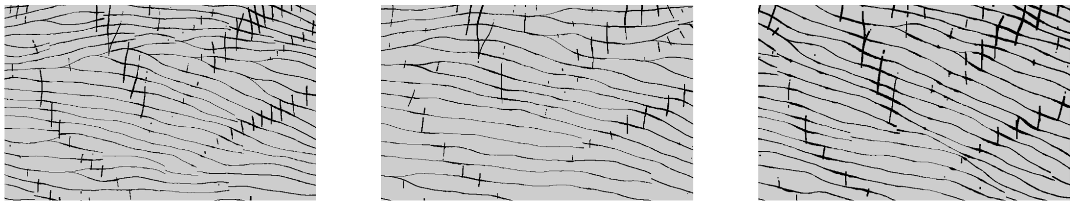


Figure 31: From left to right: the closer view of results in figure 30 from top to bottom respectively.

However, it is not always good to have a high range of spacing. In fact, in some cases, a lower spacing range performs better. Figure 32 and 33 are the examples. It is obvious that the result in figure 32 nicely handled the shadow and scales, while the other one could not even trace out the head completely.

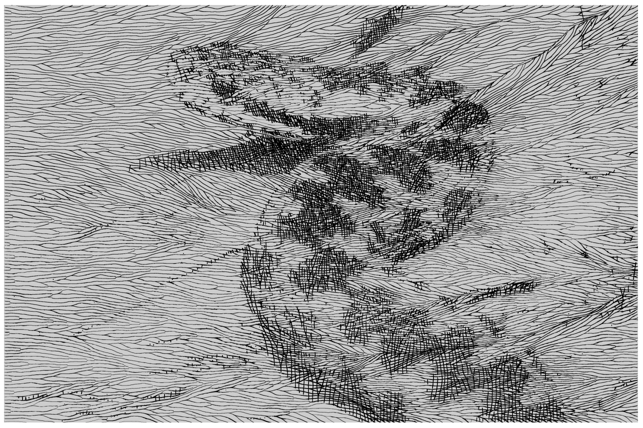


Figure 32: Spacing ranges from 10 to 30, with line width from 0 to 5

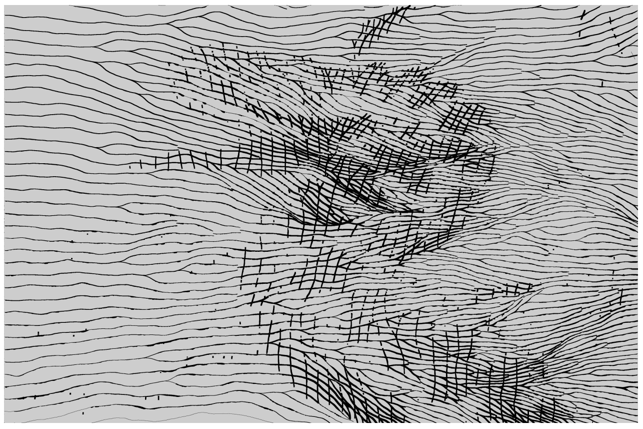


Figure 33: Spacing ranges from 10 to 90, with line width from 0 to 9

Since birth control depends on the distances with particles' neighbors as explained in section 3.3, if the distance between two neighbors is not as large as the birth threshold associated with the current spacing in the image, a particle will consider this area does not need more ink. When the spacing range is higher, the birth threshold goes up as well. In addition, since particles maintain a distance with their neighbors as indicated by the spacing value, a higher range of spacing will make particles stay further from their neighbors in general, which results in a lot of empty space. Therefore, the drawback of having a large spacing range is that particles cannot place more strokes and stay close with neighbors to depict detail features from an input image.

4.3 Tone Matching Images

With proper setting of parameters, our system is able to reproduce images with a nice hand-drawn style with smooth long lines. To match the tone in an original image, more particles will be automatically created in a dark area leaving thicker traces. Some results are shown as follows.

Overall, our system is able to reproduce the tones from an original image. For example, in figure 34, the branches and the snow are nicely presented with particle groups overlapping and generating darker tones; the bottom area has only one group of particles traveling through to form a lighter tone, and the particle traces shape a smooth, waving texture for the snow on the ground. Other results also nicely depict the input images. However, there are some "death lines" which are distracting, especially in the result in figure 37 where an obvious "death line" crosses from the bottom left to the top right corner splitting the canvas. In addition, some fine details are missed while tracing out features. For example, the face of Mona Lisa in figure 35 is drawn but the shapes of nose and mouth are not clear.

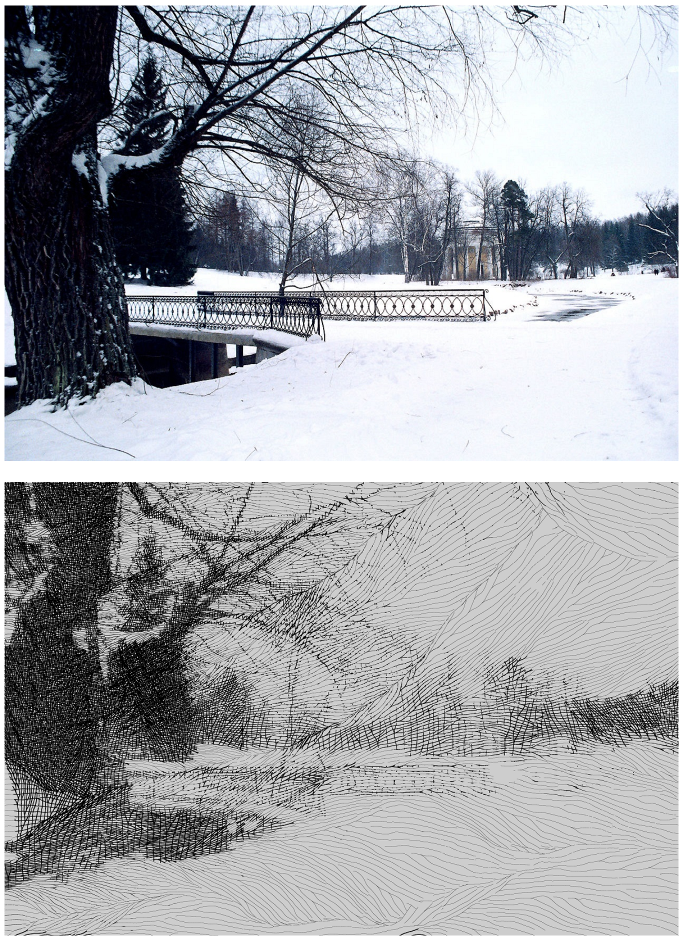


Figure 34: Original image: Pavlovsk railing of bridge yellow palace winter

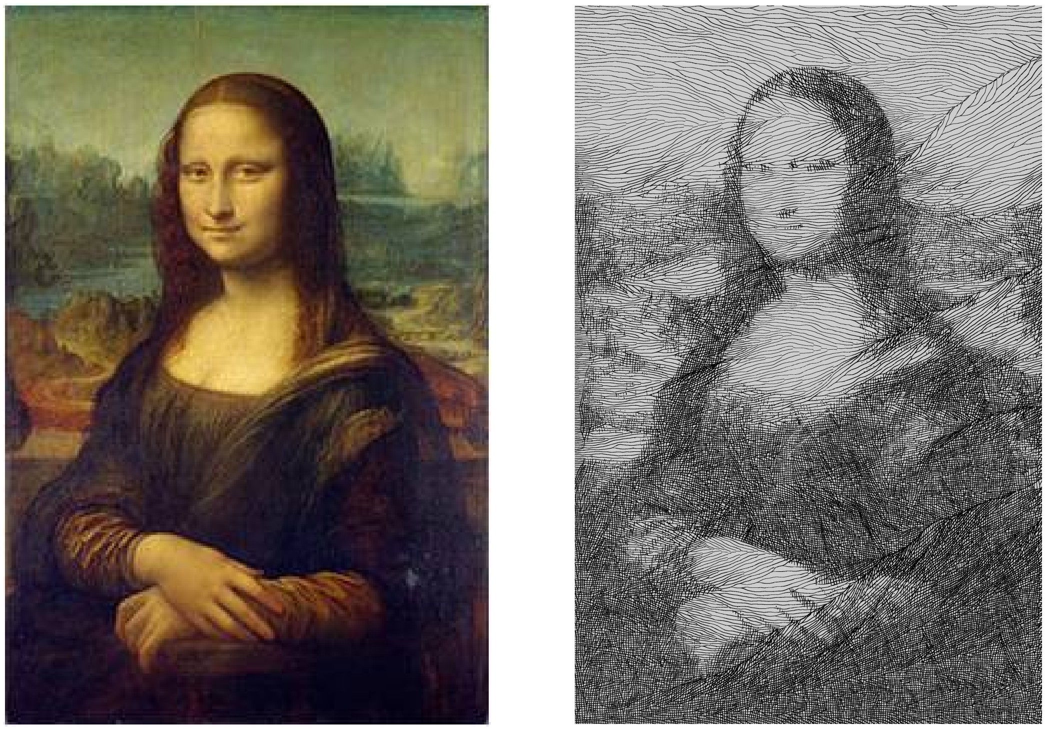


Figure 35: Original image: Mona Lisa by Leonardo da Vinci

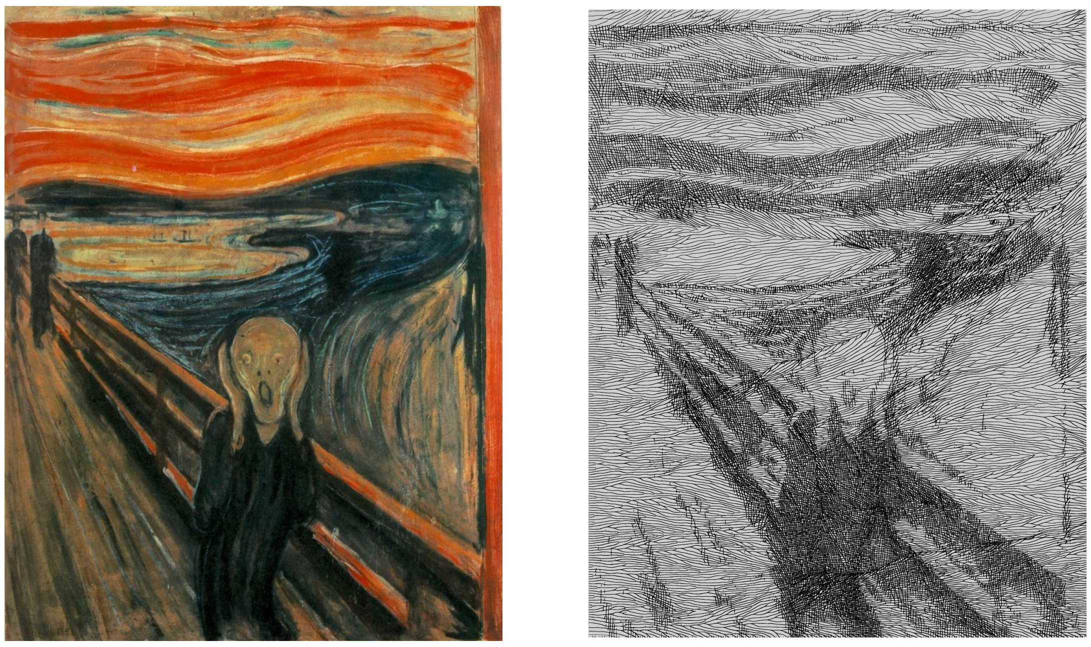


Figure 36: Original image: The Scream by Edvard Munch

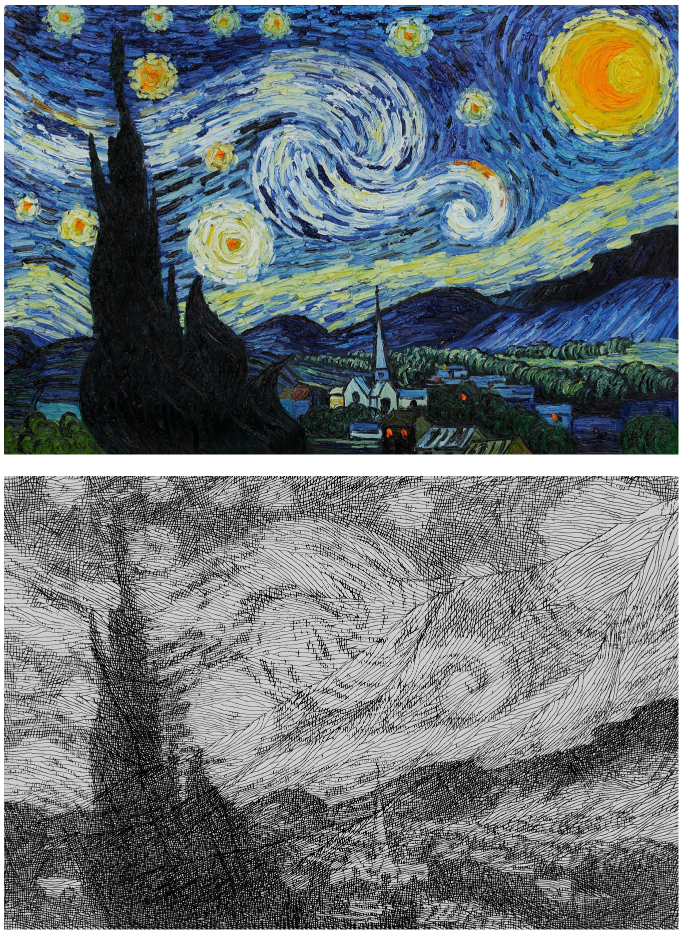


Figure 37: Original image: The Starry Night by Vincent van Gogh

4.3.1 Resolution setup

The results shown in figure 34 to 37 are high resolution images. We found it hard to trace out all fine features with a low resolution input image. The reason is similar to the one that is responsible for the drawbacks in figure 33. Since particles try to maintain distance indicated by the spacing values from the image, each time they will only update their thresholds a small amount. If we use a higher resolution input image, then more pixels will be used for one object, which means particles will have to travel longer through features with more space available to settle to the correct separation. Therefore, particles are less likely to pass through a feature without adapting themselves enough to give birth or death responding to the need.

For instance, the bottom image in figure 38 is at a resolution of $4000 * 2300$ pixels; and it successfully depicted the white spot on the whale's head and the shadow beneath it. In comparison, the top image used only $1600 * 930$ pixels and the whale is just an outline without any obvious details.

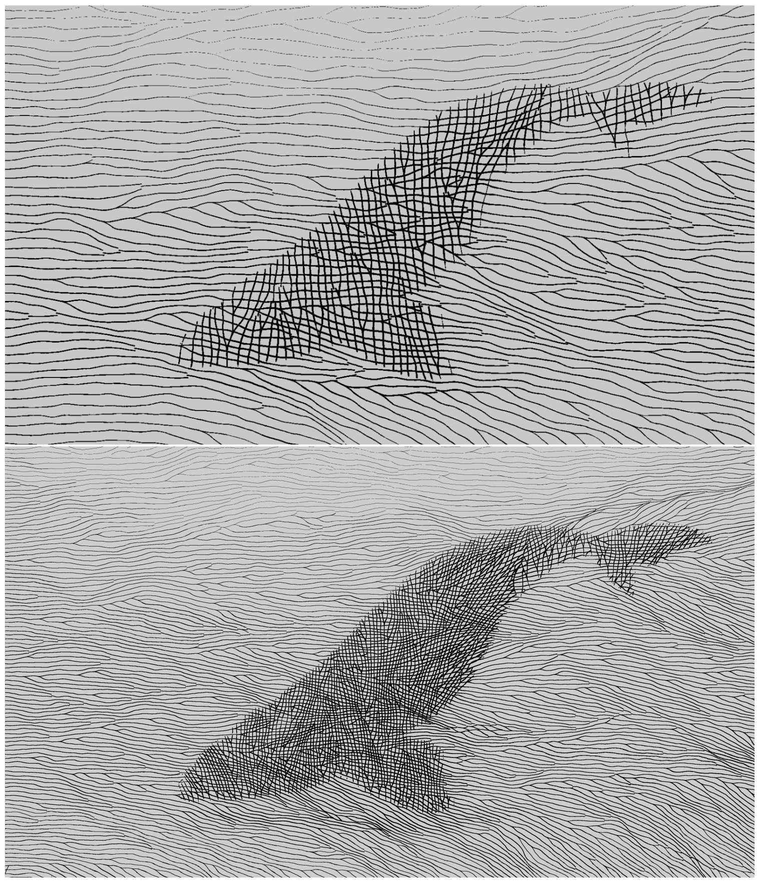


Figure 38: Top: low resolution; bottom: high resolution.

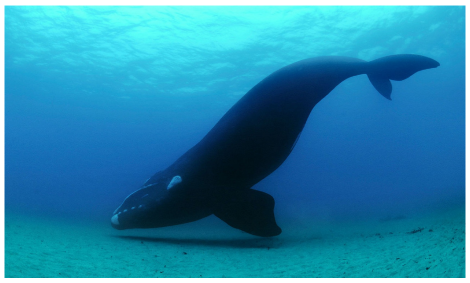


Figure 39: Original image

4.4 Abstract Patterns

Our system is capable of reproducing images with a natural smooth line-drawing style. It can also create abstract images with nice patterns. See the example in figure 40 where particles in multiple groups are used traveling with different directions and being terminated when collisions occur. We first create several lines on the canvas, and then release particle groups perpendicular to the lines.

Note that particles will run through the whole canvas until they move out of the screen or are stopped by collisions with other groups of particles. Therefore, if we release particles from both sides of one given line first and wait till them move out of the screen, there will not be much space for other groups to travel. Moreover, if we release particles from every line at the same time, particles may collide soon leaving many empty areas. Therefore, to produce a natural texture where particle lines can be covered by another group of lines or even twist with other groups, we run one particle group at a time and release particles on each line in turn.

Since here we have no input images for tone matching, particles spread out through the canvas with a constant spacing value, which gives us no line-width adjustment. The distance between neighboring particles is still being checked all the time, but not much adaption is needed since there are no different spacings for particles to update to. Therefore, our abstract images show lively lines which move in a coordinated way and even twist with others turning together.

Various random abstract patterns are shown in figure 41. By changing the randomness values that modify particles' directions, we can obtain curving lines which give an illusion of small tiles as shown in the second image from the left. We can also create abstract images in a higher resolution. See the rightmost image where particles spread out smoothly like cloth. Note that the leftmost image shows some similarities with the hair style drawing in figure 3: lines change directions gradually and show no sharp turns; multiple groups of lines with different directions collide with each other and create interesting illusions, such as moving beneath other groups and twisting with others.

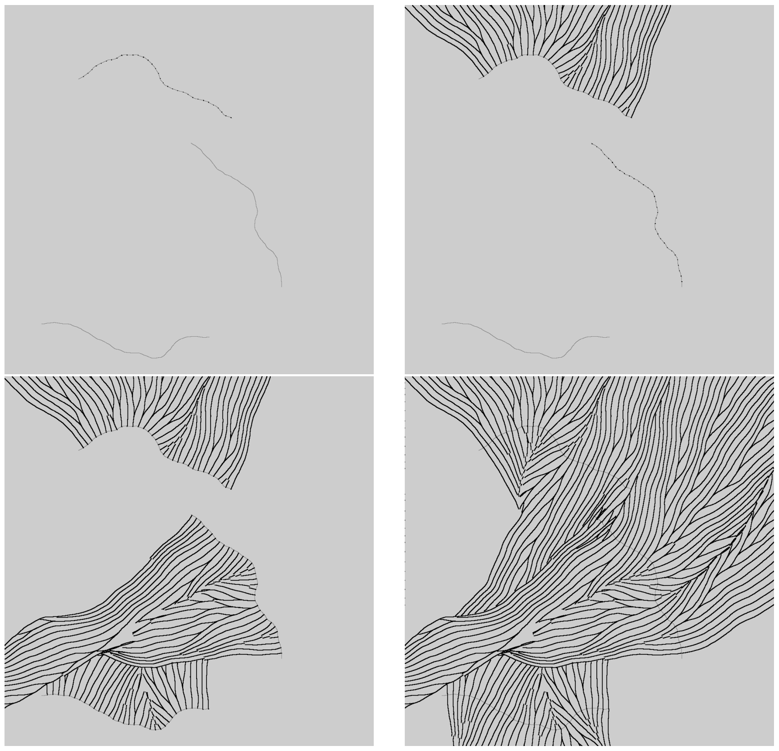


Figure 40: Procedure of creating abstract patterns

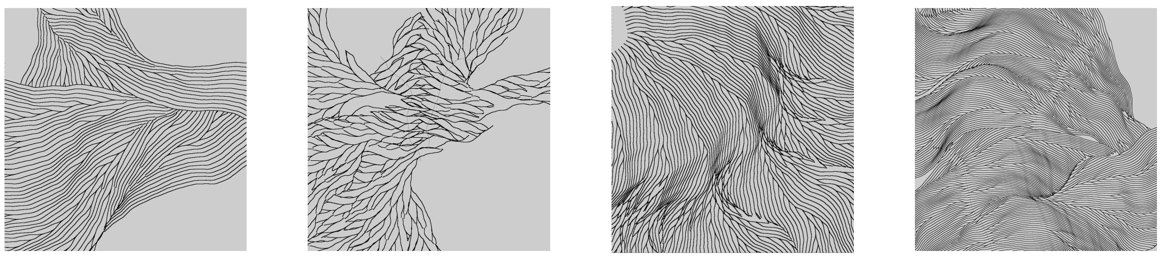


Figure 41: More results of random abstract patterns

4.5 Comparison And Discussion

So far we have presented various results using coordinated particle system, and provided some suggestions and solutions for possible problems. In this section, we will compare our results with some previous work.

4.5.1 Our system and artistic tessellation

As we mentioned in chapter 2, Li and Mould proposed a method [12] to tessellate images and trace out features, which also used a particle system to create the strokes. Figure 42 shows a comparison with our result. In Li and Mould's work, the growing curves successfully sketch the outline of the lotus and create a smooth texture with long continuous lines. The area below the lotus is drawn with almost straight lines changing directions together, which allows us to distinguish the background and the flower. Our result focuses both on the texture and the tone. Two particle groups cross in the dark areas so that fine details can be shown with the high contrast. For example, the shadow separates the petals nicely in our result.

Figure 43 is another example. Different from the mosaic effect Li and Mould presented, we reproduced the eagle with black-and-white line-drawing style. The light and shadow are well managed, especially around the eye. We can also notice that there is an area below the eagle's beak, which is just slightly lighter than the background but cannot be well separated and depicted. The cause for this defect is the imperfect map obtained by image thresholding – it is hard to find a proper threshold to separate areas that have similar tones. Since particles rely on the feature map to outline features and fine details, the way of generating feature maps affects the quality of our system. Segmentation is a possible approach to achieve nice feature maps, and we believe that the results will be clearer and more detailed with a better map.

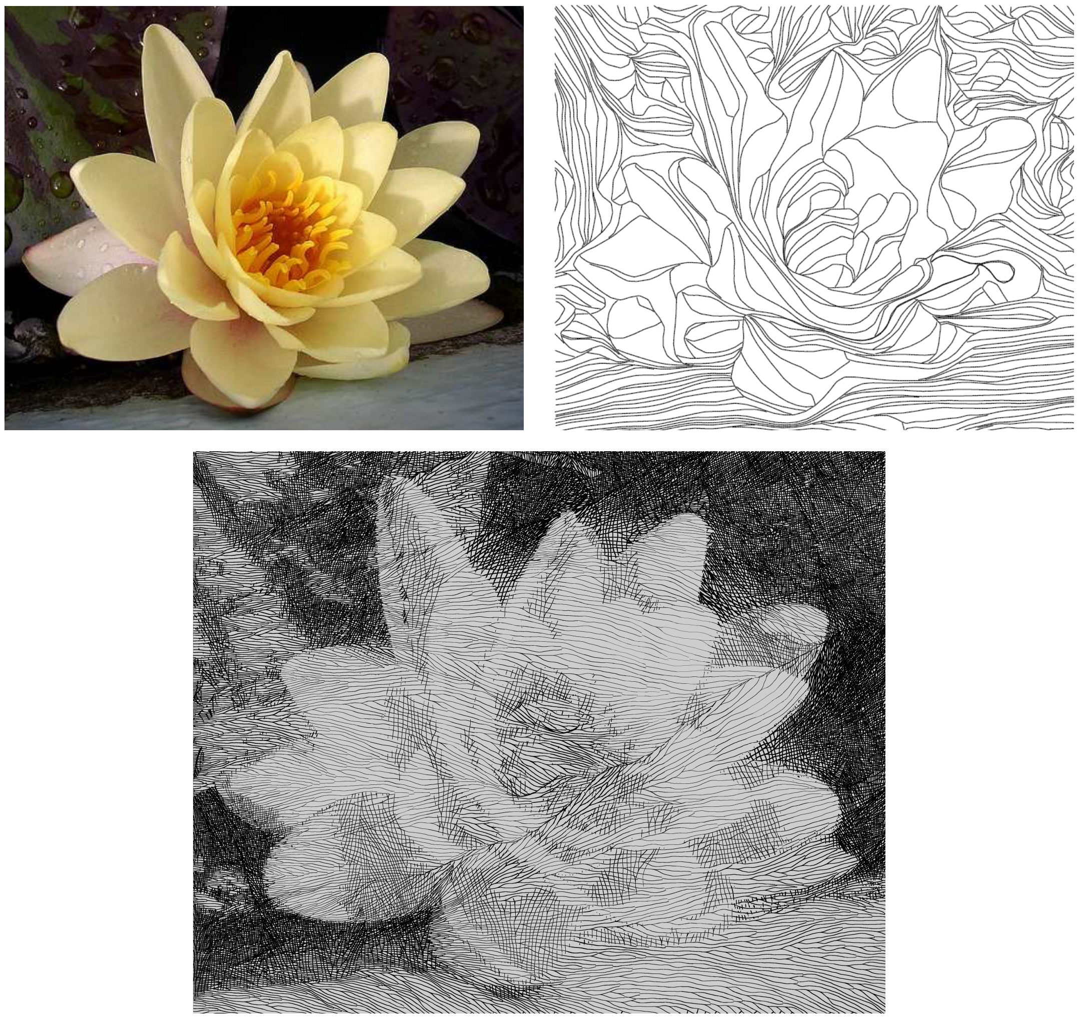


Figure 42: Top left: original image – lotus; top right: reproduced with tessellations [12]; bottom: our result.

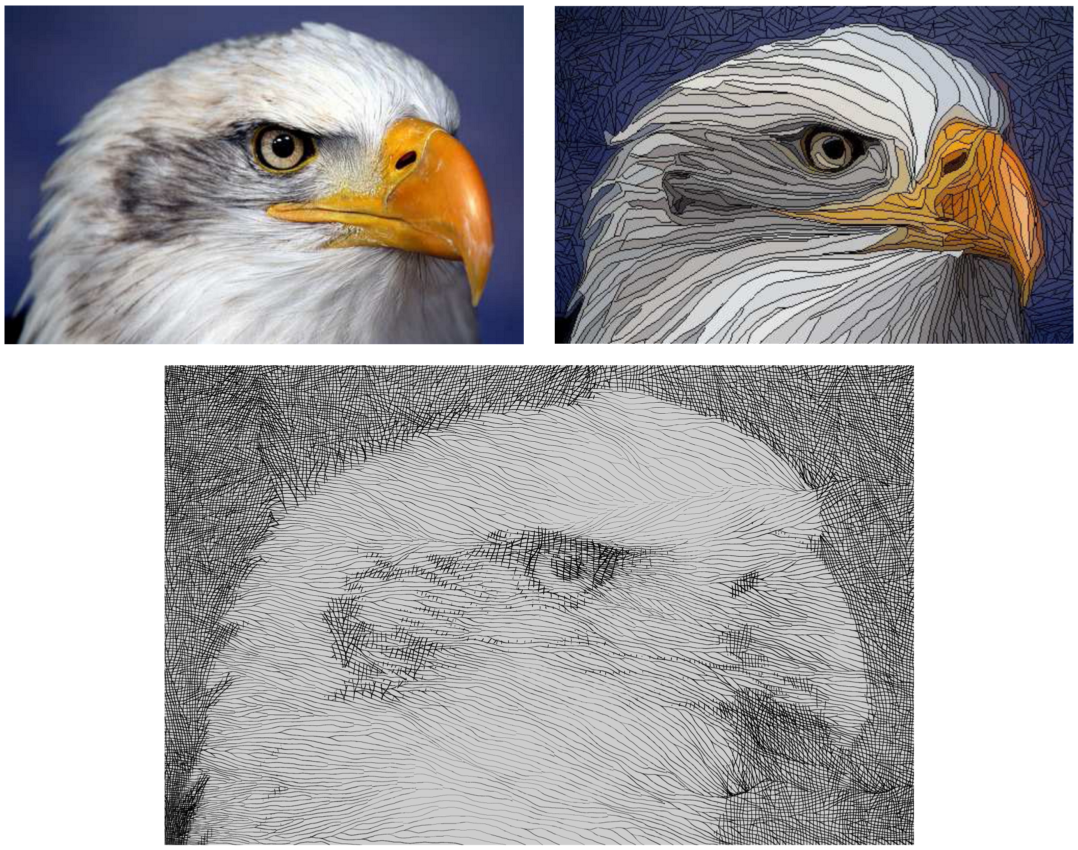


Figure 43: Top left: original image – eagle; top right: created by Flow-preserving mosaic effect with S-Method [12]; bottom: our result

4.5.2 Our system and coherent line drawing

The Edge Tangent Flow (ETF) and Flow-based Difference-of-Gaussians (FDOG) proposed by Kang et al. [11] are used to produce a coherent line-drawing style. Compared with our work, this style has an outline effect which clearly traces out edges and fine details with continuous lines, while our system uses tone matching and alignment control to express features. For example, in figure 44, we can see that the coordinated particle system produced a pen-and-ink drawing which is quite similar to the original picture (see how the particles manage the mustache and eyes). While performing the tone matching, as a result of the low contrast issue mentioned before, our system can miss some fine texture details such as the wrinkles on the forehead and the lines on the clothes. However, as a line-drawing style, it is not necessary to keep all the non-obvious details. As shown in figure 45, the FDOG method creates a lot of lines indicating the subtle features around the pillars, while our result shows a clean shadow effect.

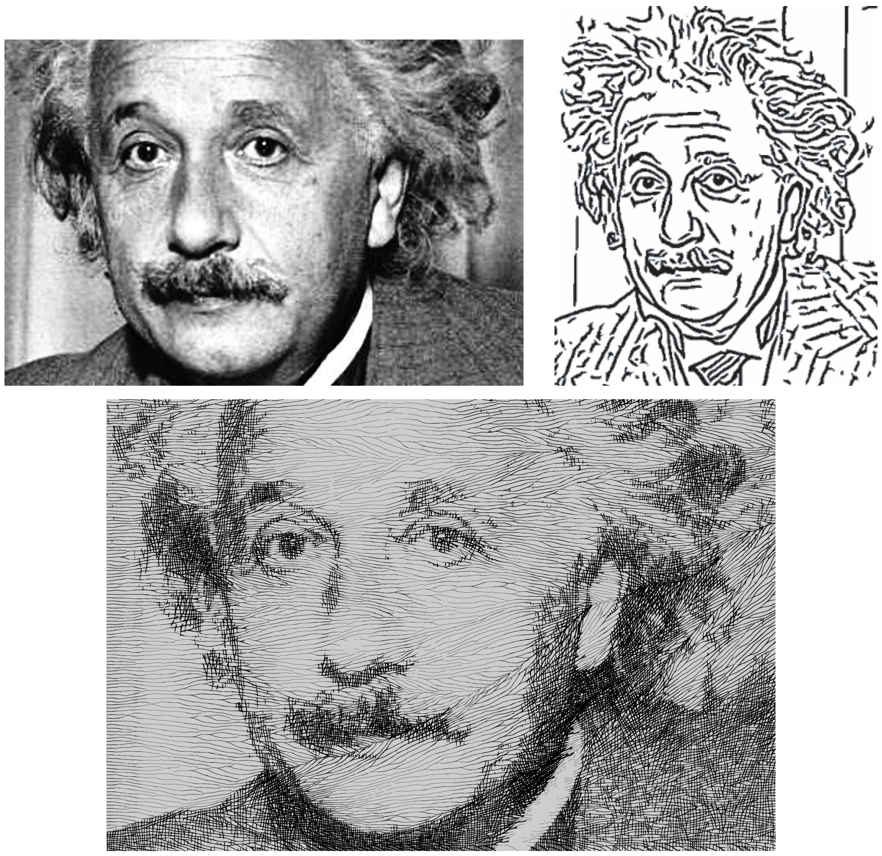


Figure 44: Top left: original image – Einstein; top right: reproduced with FDOG [11]; bottom: our result

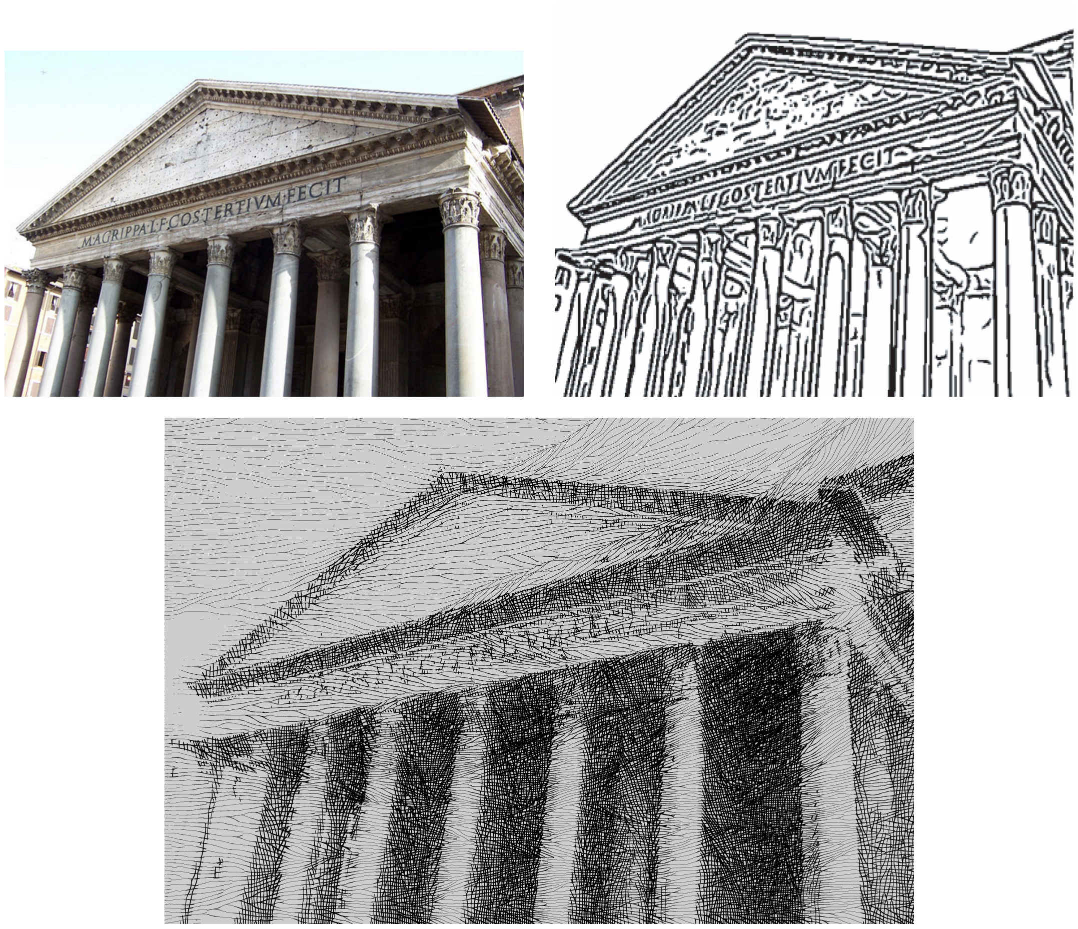


Figure 45: Top left: original image – pantheon; top right: reproduced with FDOG [11]; bottom: our result

Although Kang et al.'s algorithm offers a different style from ours, we find the ETF interesting and we think that it can even help improve our method in the future. In our system, particles do not respond to the image texture in terms of the direction update. However, we can use ETF to guide our particles moving along edges, as Li and Mould did [12], thereby drawing the texture with similar orientations.

Figure 46 shows a preliminary attempt to apply ETF to a particle system. Particles are able to trace out main feature outlines. We first use the method of creating ETF to obtain a vector map with vectors perpendicular to the image gradients. Particles are released randomly and travel following the direction obtained from the vectors. Whenever a particle moves to a pixel that is drawn previously, it will be terminated and released at a random place again. Note that particles are not moving in a coordinated way since we only focus on finding the flow of features here and do not need particles to keep certain spacings or directions with neighbors.

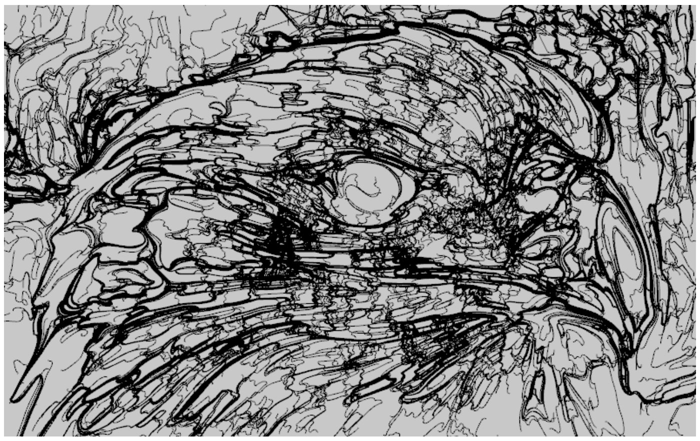


Figure 46: An eagle created by releasing particles and changing direction according to the ETF

4.5.3 Our system and loose and sketchy method

The loose and sketchy rendering [1] is able to create simple and expressive line-drawing style images. With just a depth map of the input model, it generates an edge map and a force-field vector map, and releases particles from areas that need ink to trace out edges. Since particles will be stopped when they enter empty areas, the lines can be discontinuous. In addition, this method can also be used to create animations.

Unlike this scheme, our system concentrates on creating images with brightness information, which allows particles to depict tones. Figure 47 shows a pair of examples. We use long continuous curves to trace out features, and more strokes to express the darker areas – for example, the left leg. Curtis also handled the areas which need more ink by distributing more particles. For example, the left leg is very dark and has more lines along the edges than other places.

Although we focus on different line-drawing effects, Curtis' work gives us some inspiration. For example, the use of vector map forces particles to follow the edges, which may also give us a nice effect of coordinated particles moving along silhouette lines.

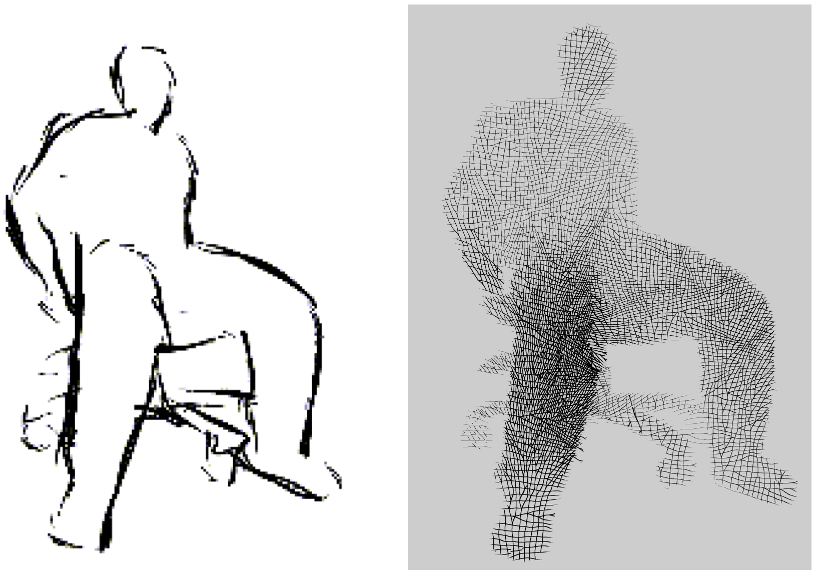


Figure 47: Left: Created in a loose and sketchy style [1]. Right: our result

4.5.4 Our system and Op Art rendering

Like the Op Art method proposed by Inglis et al. [9], our system is also able to convey different color sections with parallel lines moving in different directions. In general, we use feature maps to indicate the positions of features for particle groups; each group is initialized with one direction and collisions with particles from other groups are not allowed.

As shown in Figure 48, the bottom left image of our result gives a smooth illusion effect which is similar to the result generated by Op Art rendering. Both Op Art and our results applied parallel lines with different directions, and the features are successfully depicted without drawing the outlines.

The differences between Op Art and our method are obvious as well. In our system, the two particle groups trace out features separately, and two lines in different groups do not link with each other to form a single line. In addition, our results show lines with the ability of changing directions with a smooth tangent-like path. However, in the Op Art method, after the two parts of lines are drawn out, some additional lines are used to connect the two parts, which gives their result long continuous lines spreading out from one section to another. Moreover, instead of creating highly abstract illusion images with strictly parallel lines, our system reproduced the heart in the input image with some irregularities. We are also able to control the brightness by changing the line spacing and thickness – more lines appear in darker areas and less are shown in light areas.

Since we are interested in managing particles to create natural, irregular lines, we generated another result with more randomness values added to particles' directions. See the bottom right image where a lot of birth and death are triggered. With the lines being affected by high randomness, the heart looks "furry" and lively, compared to the one in the bottom right example.

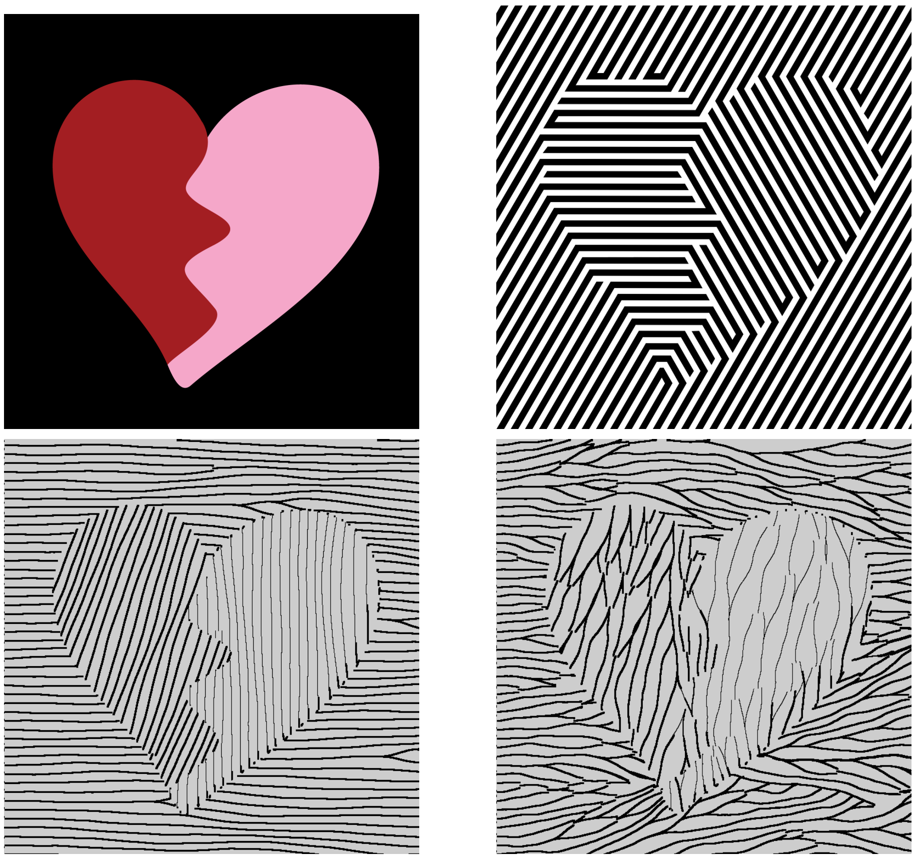


Figure 48: Top left: input image; top right: created with Op Art [9]; bottom left: our result with a small randomness; bottom right: our result with a high randomness.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

There are many topics in the area of NPR, such as reproducing images in a stylized way and creating textures to a model. Line-drawing usually focuses on creating back-and-white lines which are able to trace out objects by outlining features or matching tones. However, none of the algorithms proposed before can produce parallel lines that change spacings and thicknesses with neighbors automatically. Some of the previous work can generate images with parallel lines, such as Ostromoukhov's engraving style rendering [14], the "HatchingBots" proposed by Schlechtweg et al [17], and the tessellation method by Li and Mould [12], but the lines created by their methods are highly parallel without gradually and directly adapting the spacings with neighbors. Therefore, their results still cannot produce smooth and natural parallel lines as ours.

We proposed a coordinated particle system which is capable of tracing out features and details with parallel lines changing directions and thicknesses automatically. To match tones and generate parallel lines, we allow new particles to be created in areas that have small spacings and let particles be terminated when collisions occur or when the desired spacings become larger. Natural yet irregular patterns are achieved by adding some randomness to particles' directions.

The structure of our system works in a hierarchy – a manager is used to control multiple particle groups, and each group releases particles tracing out objects in its own areas. In the end, all the areas that are drawn by different groups combine to form the output image. Since we allow user interaction to give specific initial directions, more particle groups means more choices of orientations and more features can be distinguished. However, more groups requires more manual input to indicate initial directions and emitting locations, which is why we recommend using only two particle groups.

Our system is able to reproduce an input image with line-drawing style. The output image consists of almost parallel lines which automatically change their spacings and directions with neighbors to stay required distances with each other. With some randomness added to particles' directions, the lines are also irregular and natural.

One of the disadvantages of our system is that the quality of results depends on the original image. With a high contrast input image, we are able to nicely separate features by thresholding and to generate a feature map as explained in section 3.5.2, which can give particles positions of the places that need strokes. However, if we use a low contrast input image, some features may be lost during thresholding, which leads to the result that particles miss details. Another drawback is that we cannot prevent the appearance of "death lines". As we seen in figure 34 to 37, a lot of "death lines" appear which distract viewers' attention from main objects. In addition, sometimes fine details cannot be portrayed because of the limitation of particles' spacings. Some details need to be drawn with very small spacings, but when the spacings narrow down, the values of birth and death thresholds will become closer and closer. When the distance of two neighboring particles is between the birth and death thresholds, no birth or death control will be triggered. Therefore, with the two thresholds getting closer to each other and the range between them becoming smaller, more birth or death will be applied, which makes it difficult for particles to maintain smooth parallel lines.

5.2 Future Work

One possible direction for continuing our work will be adding a force to a particle's direction which is tangent to edges. By doing so, particles will be able to leave traces along with the flow of textures. For example, particles might trace out the texture of feathers. The ETF proposed by Kang et al. [11] will be a good start to build the required vector field.

As we described before, particles in a group all start with a same direction. No doubt user interaction is one possibility to obtain textures with desired directions; however, we also want to have an option to find the best orientation for each group automatically. Studying the original texture of images and summarizing the main directions will help this future work. Furthermore, with the ability to classify the texture orientations, we can have a new way to create feature maps by segmenting the image according to different directions. As a result, particles in one same group will trace out features which have same orientations only.

Another future work is to make animations with our system. The performance of our system depends on the resolution and the features of an input image, but overall it takes minutes to reproduce images. Note that abstract patterns without input images can be created in seconds. Besides the run time, another problem is that the feature map for informing particles the positions to draw cannot follow the features in the images if objects move between frames.

Moreover, we would like to explore more in the area of abstract patterns. As discussed in section 4.4, we can generate abstract images by creating random initial thread lines to release particles. If we can arrange the threads in a proper way, particles will leave traces following a designed pattern such as waves and smoke. For example, by placing thread lines carefully, we will be able to generate hair textures as shown in figure 3. However, we also want to create abstract patterns without manual control – the thread lines can be placed automatically.

We also want to control the "death lines". Sometimes, the "death line" in an abstract image is not as distracting as in the result of reproducing input images. For example, there is some "death lines" in the final result in figure 40, but they do not disturb the aesthetic of the image since it becomes a part of the features. Therefore, by controlling them, we can produce patterns with more features.

The resolution of image affects the quality of our results. Therefore, creating lines based on vector map instead of drawing them directly will give us clearer results. For example, when particles move, we can store their directions as vectors in a map, and when all particles finish drawing, we generate the result by using the vectors with spline functions, which can give us smooth and clear lines.

List of References

[1] Cassidy J. Curtis. Loose and sketchy animation. In *ACM SIGGRAPH 98 Electronic art and animation catalog*, SIGGRAPH '98, page 145, New York, NY, USA, 1998. ACM.

[2] George Davidson. *The Drawings of Gustave Doré: Illustrations to the Great Classics*. Metro Books, 2008.

[3] Gershon Elber. Line art rendering via a coverage of isoparametric curves. *IEEE Transactions on Visualization and Computer Graphics*, 1(3):231–239, September 1995.

[4] Gershon Elber. Interactive Line Art Rendering of Freeform Surfaces. *Eurographics 1999*, 18:1–12, 1999.

[5] Gershon Elber and Elaine Cohen. Tool path generation for freeform surface models. In *Proceedings of the second ACM symposium on Solid modeling and applications*, SMA '93, pages 419–428, New York, NY, USA, 1993. ACM.

[6] Paul Haeberli. Paint by numbers: abstract image representations. *SIGGRAPH Comput. Graph.*, 24(4):207–214, September 1990.

[7] Aaron Hertzmann. Painterly rendering with curved brush strokes of multiple sizes. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '98, pages 453–460, New York, NY, USA, 1998. ACM.

[8] Aaron Hertzmann and Denis Zorin. Illustrating smooth surfaces. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '00, pages 517–526, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[9] Tiffany Inglis, Stephen Inglis, and Craig S. Kaplan. Op art rendering with lines and curves. *Computers and Graphics*, 36(6):607–621, 2012.

[10] Tiffany C. Inglis and Craig S. Kaplan. Generating op art lines. In *Proceedings of the International Symposium on Computational Aesthetics in Graphics, Visualization, and Imaging*, CAe '11, pages 25–32, New York, NY, USA, 2011. ACM.

[11] Henry Kang, Seungyong Lee, and Charles K. Chui. Coherent line drawing. In *Proceedings of the 5th international symposium on Non-photorealistic animation and rendering*, NPAR '07, pages 43–50, New York, NY, USA, 2007. ACM.

[12] Hua Li and David Mould. Artistic tessellations by growing curves. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Non-Photorealistic Animation and Rendering*, NPAR '11, pages 125–134, New York, NY, USA, 2011. ACM.

[13] David Mould. Stipple placement using distance in a weighted graph. In *Proceedings of the Third Eurographics conference on Computational Aesthetics in Graphics, Visualization and Imaging*, Computational Aesthetics'07, pages 45–52, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[14] Victor Ostromoukhov. Digital facial engraving. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, pages 417–424, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[15] W. T. Reeves. Particle systems - a technique for modeling a class of fuzzy objects. *ACM Trans. Graph.*, 2(2):91–108, April 1983.

[16] Michael P. Salisbury, Sean E. Anderson, Ronen Barzel, and David H. Salesin. Interactive pen-and-ink illustration. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, SIGGRAPH '94, pages 101–108, New York, NY, USA, 1994. ACM.

[17] Stefan Schlechtweg, Tobias Germer, and Thomas Strothotte. Renderbots – multi agent systems for direct image generation. *Computer Graphics Forum*, 24(2):283–290, 2005.

[18] Gerhard Weiss, editor. *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT Press, Cambridge, MA, USA, 1999.

[19] Georges Winkenbach and David H. Salesin. Computer-generated pen-and-ink illustration. In *Proceedings of the 21st annual conference on Computer graphics*

and interactive techniques, SIGGRAPH '94, pages 91–100, New York, NY, USA, 1994. ACM.

[20] Georges Winkenbach and David H. Salesin. Rendering parametric surfaces in pen and ink. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96, pages 469–476, New York, NY, USA, 1996. ACM.

[21] Ling Xu and David Mould. Magnetic curves: curvature-controlled aesthetic curves using magnetic fields. In *Proceedings of the Fifth Eurographics conference on Computational Aesthetics in Graphics, Visualization and Imaging*, Computational Aesthetics'09, pages 1–8, Aire-la-Ville, Switzerland, Switzerland, 2009. Eurographics Association.