Coordinated Particle Systems for Image Stylization

Chujia Wei*
Carleton University

David Mould†
Carleton University

Abstract

Our paper provides an approach to create line-drawing stylizations of input images. The main idea is to use particle tracing with interaction between nearby particles: the particles coordinate their movements so as to produce varied but roughly parallel traces. The particle density varies according to the tone in the input images, thereby expressing bright and dark areas. Using procedural distributions of particles, we can also generate smooth abstract patterns.

Index Terms:
I.3.3 [Computer Graphics]: Picture/Image generation—Line and curve generation

1 Introduction

Using only black and white lines, artists can create stylized images which depict objects clearly; Figure 1 shows an example. The darker parts of the image use more strokes, while the brighter parts use fewer. The shape of the objects is communicated in part by the local orientation of the strokes.

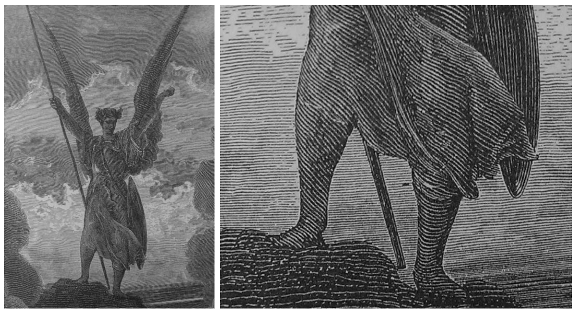
Figure 1: An artistic drawing with details shown on the right. Engraving by Gustave Doré [2].

We identify three main characteristics of this drawing:

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

Many algorithms can produce stylized images akin to those created by artists. We present an algorithm inspired by the natural yet irregular style illustrated in the figure; our goal was to create strokes possessing the three characteristics listed above. In our line drawing approach, individual lines are smoothly curved and the collection of lines fills space.

*e-mail: chujiawei@cmail.carleton.ca
†e-mail: mould@scs.carleton.ca

The method we describe is further inspired by a particular style of artistic pen-and-ink drawings, where strokes are mostly parallel and tones are produced by different line spacings. Figure 2 shows an example of this kind of pattern: several groups of lines are used to draw a head of hair. The lines in each group are approximately parallel, generating a sense of unity within a group; lines change their directions and spacings gradually, except at the intersections between two groups, and the visible intersections provide a sense of the 3D structure of the surface. We would like to create both representational images based on input photographs and more abstract images based on procedural or user-designated arrangements of curves. Our image synthesis method is based on a particle system [14], where the particles' trajectories are the curves drawn in the final image. Our process is automatic: the logic of the particle system creates the image without human intervention. The method creates smooth, parallel curves with added irregularity for a more natural look. The density of curves matches the tone in the input image.

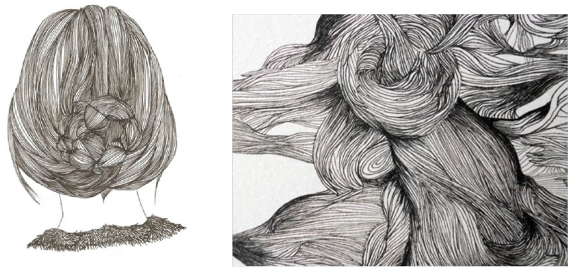
Figure 2: Left: A hairstyle created by Maria Gil Ulldemolins. Right: An abstract hair style drawn by Rachael Bartram. Images from Flickr.

A set of rules guides the behaviour of particles. All particles move and leave traces in a coordinated way, obtaining information from an input image. Each particle tracks the distance to its neighbors and adjusts its direction so that all particles can remain approximately evenly spaced and parallel. See Figure 3 for an example. The approach is reminiscent of flocking [15], but used here for drawing rather than animation. We also add small random perturbations to particle orientation to increase the variety and naturalness of the trajectories.

A local target particle density is computed from an input image, so that dark areas have more closely packed lines and lighter areas have wider spacing. New particles can be spawned to increase density, and similarly, particles can be terminated when the density is too high. Guptill [10] advises that textures of parallel lines look better when the line endings match up with other lines, rather than having lines terminate in empty space; we take some trouble to ensure that particle birth and death events are matched up with line intersections.

Our minimum line density is not very low, so it is difficult to portray very dark tones. To increase contrast, we use a second layer of particles to create a cross-hatching effect with the first; we construct a foreground map that governs where this second layer operates. The foreground map can be created by various means, but thresholding the input image is one simple and moderately effective mechanism.

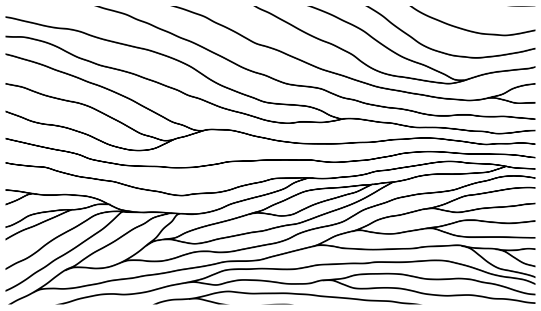
Figure 3: Particles leave parallel curves which can smoothly change spacing.

This paper is organized as follows. Having mentioned the problem of line drawing with coordinated particle systems in this introduction, we give more background and discuss related work in the next section. Next, we provide details of our method. A gallery of results and comparisons to similar methods follows; finally, we conclude and give some suggestions for possible future directions.

2 Previous Work

Over the years, there has been considerable attention paid to algorithms for creating line art and for image stylization with line and curve primitives. In the following, we describe some highlights of previous work in the area.

Winkenbach and Salesin introduced a system to produce pen-and-ink style images [18], capable of rendering with different stroke styles and various outlines and tones. They also presented a technique of "controlled-density hatching" [19] to generate lines that can appear and disappear gradually to convey tone and shape, and generate various textures. Salisbury et al. also proposed a method to produce quality pen-and-ink illustrations with various texture styles [16]. Some user interaction is needed for choosing and placing stroke textures.

Kang et al. [9] explored a method to create a coherent line drawing style. They compute a smooth edge tangent flow (ETF) and then apply a flow-based difference of Gaussians filter to produce stylized images in a line drawing style. Aiming at a different visual effect, Curtis [1] proposed a method to produce loose and sketchy animations. With limited user intervention, it can create expressive line drawings and animations showing the silhouette edges of the input. Curtis used a particle system but the particles did not coordinate except indirectly through using the same input edge map.

Ostromoukhov [12] proposed a technique to produce engraving-style images. This method shares some similarities with our work, such as parallel lines and different groups of lines for different objects. Inglis et al. proposed algorithms to create an Op Art effect with groups of parallel lines [7, 8]. The technique of Xu and Kaplan [20] shows mazes built with almost parallel lines based on input images. Pedersen and Singh [13] synthesize labyrinths and mazes with a set of curves which evolve through simulated forces. By controlling solution paths, boundaries, and other factors, this system computes maze curves showing input image details.

Elber and Cohen presented a scheme to cover free-form surface with isoparametric curves [5], later improved with variable density [3] and real-time rendering [4]. Hertzmann and Zorin [6] mentioned that the computation of directions for hatching in Elber's work is not reliable, and proposed a more robust technique to find the directions of hatching curves, used for line-drawing renderings of geometric models. The disadvantage of this work is the rendering time, which can be minutes depending on the density and the complexity. The systems of Elber and Cohen and of Hertzmann and Zorin operate on 3D models, while our approach is intended to work with input images.

Li and Mould used particle tracing to draw tessellation boundaries [11]. The main idea of this method is to release particles with different positions and directions, and track the path of each particle until it reaches another curve or a region boundary. The spacing between particles and the length of paths are controlled so that no narrow or short strokes can be drawn. The approximately parallel lines in their results are not shaped by coordination between particles but by a careful choice of initial particle placement and direction and by removing curves that are poorly spaced.

Schlechtweg et al. proposed an approach to render stroke-based non-photorealistic images with a multi-agent particle system [17]. In their method, intelligent agents called RenderBots produce different effects, such as edging, hatching, and painting, simultaneously in one image. Among their different agents, the "HatchingBots" are similar to our particles; these bots draw almost parallel lines and stop when they are too close to other bots or edges. However, the HatchingBots do not change direction and the authors did not attempt to make naturalistic curved strokes using these agents.

While the techniques we discussed above have been largely successful at achieving their goals, none of them precisely addresses the problem we sought to solve. Many techniques involve significant user effort, which is an extra cost. Approaches designed for 3D models are not applicable when the input is an image. Few techniques attempted to fill the image plane with curves; Inglis et al.'s Op Art, which is an exception, pursues a significantly different style from ours. In the following sections, we describe the details of our algorithm and show some results.

3 Algorithm

Our objective is to cover the image plane with irregular but approximately locally parallel curves. As an aesthetic choice, prompted by Guptill [10], we decided not to allow curves to begin or end in empty space: a curve should end either at the image boundary or at an intersection with another curve. Curves should not change directions suddenly. The patterns created by the curves can be either abstract or representational; if representational, they can be guided by an input image, in which case we approximate the input tone by tracing more curves in the darker areas and fewer in lighter areas.

We refer to our solution as a *coordinated particle system*. Commonly, the particles in a particle system are noninteracting. In the coordinated system, they interact by informing their neighbors about their location and direction; in this way, the method is similar to flocking. Unlike flocking, but like other particle systems, particles can be added to or removed from the simulation during runtime, depending on the local particle density and collisions with other particle trajectories. The coordinated particle system can create natural-looking patterns and engraving-like images; coordinating the movement and density of particles while tracing particle trajectories, we can create smooth, near-parallel curved tracks.

Beginning with an input image, we divide the image plane into foreground and background regions, possibly by thresholding the intensity. The foreground area will be covered by two sets of curves, and the background only one. We assign particle groups to regions of the image; by default, there are only two groups – one for the entire image and another restricted to the foreground region – but in principle there can be any number. Each group of particles is organized into a chain, linearly ordered. The chains are placed at their initial locations, possibly along an image boundary, and each particle is assigned its initial values. Subsequently, the particles move across the image plane, seeking to maintain even spacing with their neighbors in the chain. Particles are created or added as needed to maintain an appropriate local particle density along the chain, guided by monitoring the input image intensity. The simulation ends and the image is complete when all remaining particles have left the image region.

In our implementation, particles are a key data structure, with each particle possessing many attributes: position $\vec{x}$, bearing b, and other properties including state flags (such as whether a particle is presently visible or invisible) and a base threshold value describing the local desired interparticle spacing. The particles themselves are stored in a linked list and each particle stores the IDs of the adjacent particles, previous and next in the list.

A particle will move alongside its two neighbors. During the simulation, some particles will be removed and some new ones will be generated and inserted into the list, but we maintain the ordered structure. Particles seek to equalize their distances with their neighbors, always moving closer to the more distant one; they also try to maintain the same directions as their neighbors. Together, these two effects produce nearly parallel trajectories. The parallelism is disturbed by deliberately added randomness and by the particle chains' responses to the input image, creating and removing particles to achieve a desired density.

New particles are born when the local particle density becomes too low, as measured by distance between adjacent particles. Particles die when the local particle density becomes too high or when a particle collides with a previously drawn trajectory. While living, the particles move, coordinating with their neighbors, according to rules which will be described next.

The coordinated particles follow sigmoidal paths, which ensures the smoothness of the particle traces. When particles are born or killed, following sigmoidal paths also allows them to turn smoothly. Our sigmoid is based on the tangent function; we provide details below. Each particle's position $\vec{x}$ is updated using Euler integration with unit velocity along the particle's direction $\vec{d}$.

We want particles to have stable directions, so that they will turn smoothly; we want particles to be evenly spaced out; and we want the traces to be parallel, meaning that neighbouring particles have the same bearings. Note the similarity to Reynolds's flocking algorithm [15], where boids should be evenly spaced and share the same orientation as their flockmates. These desires led us to an equation for the bearing that includes a term for each factor mentioned. A new bearing b_{new} is computed from the old one b_{old} as follows:

$$b_{new} = (1-\alpha)b_{old} + \alpha(b_{aver} + b_{sig}), \tag{1}$$

where b_{aver} is the average of the two neighbors' bearings and b_{sig} is a bearing computed to produce sigmoidal movement towards the midpoint of the two neighbours. The parameter α is a small value governing the rate of change of bearing. The larger α is, the faster a particle's bearing will be changed, and the sharper the particle will turn.

To compute b_{sig}, we use the tangent function to approximate a sigmoid. By following a sigmoidal curve, particles will smoothly change directions.

For example, as shown in Figure 4, a particle, say C, moves in cooperation with its two neighbours A and B. Assume C is closer to A: we obtain an angle θ for the particle, interpolating between $-\pi/2$ and $\pi/2$ depending on the particle's position between A and the AB midpoint. The particle's bearing is then $\tan(\theta)$. If the particle is closer to B, we do the same, but reversing the direction of the interpolation so that θ is $-\pi/2$ near B, $\pi/2$ near the midpoint. More formally,

$$\theta = \begin{cases} \pi \times S_{AC}/S_{CB} - \pi/2, & AC < CB \\ \pi/2 - \pi \times S_{CB}/S_{AC} & \text{otherwise,} \end{cases} \tag{2}$$

where S_{AC} is the distance from A to C and similarly S_{CB} is the distance between C and B.

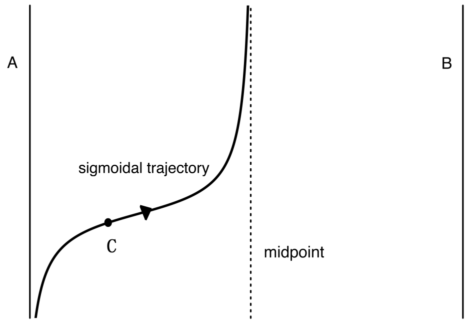

Figure 4: The path of a sigmoidal curve which leads a particle C towards the middle of its neighbours A and B.

3.1 Particle Birth and Death

Tone matching between the input image and the output curves is accomplished by varying the particle count locally. We do not attempt to strictly match tone, but to loosely approximate the input tone by placing more curves in dark areas and fewer curves in lighter areas. Our method manages the number of particles with continuous particle birth and death – when two neighbouring 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.

Each pair of neighbouring particles constantly monitors their separation distance. If the distance exceeds a threshold T_b, particle birth is triggered; if the distance is smaller than a threshold T_d, one of the particles is terminated. The thresholds T_b and T_d are computed based on a deviation from a base distance that is linearly interpolated from the local image intensity. Details are given later in this subsection.

Figure 5 shows the procedure of particle birth. The distance S_{AB} of neighbouring particles A and B is checked. When $S_{AB} > T_b$, a new particle C is created from the path of B with a direction inherited from B's direction and rotated slightly towards A. Once the particle enters the simulation, the sigmoidal component of its bearing updates will cause it to move towards A; at the same time, particle B will follow a sigmoidal path carrying it away.

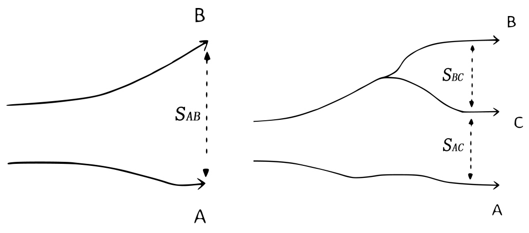

Figure 5: Left: Particles A and B move too far from each other; right: a new particle C spawns from B adding one more line to the image.

When particles enter a lighter area of the image, requiring a larger spacing to match the tone, it is difficult or impossible for them to push their neighbours away quickly enough to match the desired spacing. The solution is to remove some particles; particle death is triggered when the interparticle spacing becomes sufficiently smaller than the desired spacing. Figure 6 depicts a typical case. Particle B observes a small distance S_{BC} with neighbour C: formally, $S_{BC} < T_d$ where T_d is the death threshold. B and C are then marked as dying, and gradually merge their paths. Merging is accomplished by pushing each particle towards the midpoint of the outside neighbours, using the sigmoidal trajectory illustrated in Figure 4; in Figure 6, the relevant neighbours are particles A and D. Once the paths of B and C converge, one of the particles is killed and the other one continues the coordinated movement with its neighbors. After some time, the three surviving particles reach the desired spacing and have approximately the same bearings.

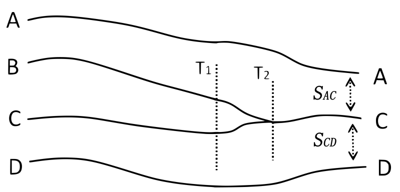

Figure 6: At time T_1, particles B and C are too close to each other and they are forced to merge; at T_2, the paths of B and C overlap. B is killed.

The birth and death thresholds are computed with respect to base threshold values stored on a per-particle basis. For each particle pair, their base thresholds are averaged, producing a base threshold for the gap, say T_g; the birth threshold T_b is $1.5 \times T_g$ and the death threshold T_d is $0.5 \times T_g$. The constants 0.5 and 1.5 were chosen somewhat arbitrarily, but empirically seem effective. This choice of constants ensures that the particles do not get closer than half the desired distance without triggering a termination, and that a gap that exceeds the desired spacing by 50% triggers a new particle. Note that the death threshold could be larger or the birth threshold smaller, making the particle set more stable, but the tradeoff is a greater variation from the desired separation distance. Conversely, making a smaller death threshold or larger birth threshold incurs the tradeoff in the opposite direction: the separation distance is in principle better matched, but there will be more birth and death events. These values can be changed for a particular input image based on the perceived aesthetics of the birth and death events and on the necessity of matching tone.

The base thresholds T_g are stored on a per-particle basis because they depend on the set of tones the particle has passed over. At every timestep, a target spacing is computed from the image tone at each particle's location, linearly interpolating between a user-specified minimum and maximum particle spacing. Particles update their current base threshold in the direction of the local target spacing, incrementing or decrementing by a parameter β depending on whether the current threshold is below or above the target. We used $\beta = 0.001$ in this paper. The use of dynamic, per-particle thresholds makes the spacing stable, smoothing out the particle trajectories; tiny features of the image do not cause sudden changes in the particle behaviour. In effect, the dynamic per-particle thresholds are equivalent to smoothing the image along the particle trajectory.

The base threshold stored in each particle is updated based on image intensity. The input image is converted to grayscale and a target spacing computed for each intensity level by linearly interpolating between a user-specified minimum and maximum spacing. There is no practical upper bound to the maximum spacing, although spacings that approach the image size are unlikely to be useful. However, by default, our implementation uses the original pixel resolution to perform calculations such as collision detection with previously drawn curves; accordingly, the minimum spacing should not be set to a value smaller than 2 pixels. With a minimum spacing set smaller than this, particles may die due to accidental collisions with others, creating a lot of unappealing short lines. In practice, we try to avoid settings smaller than about 8 pixels. If more closely packed lines are desired, we can upsample the grid before processing and downsample afterward; the downsampling is optionally facilitated by using vector instead of raster output.

In addition to particle spacing, we use multiple particle groups for cross-hatching to indicate tone and to show the structure of the input image. A segmentation of the input image divides the image plane into regions; we can assign different particle groups to different regions, each with a different direction, to indicate different image elements. Alternatively, we can divide the image into two regions, darkening one by drawing over it a second time with a second particle group with a different direction than the first. In principle, we could add further layers with yet more directions, but our initial experiments in this regard were unpromising and we did not pursue this idea: two groups are sufficient to distinguish between darker and lighter areas, and intermediate areas can be further distinguished by varying particle spacing. In almost all examples in this paper, we obtained a binary map by thresholding the intensity: dark areas will be crosshatched and light areas will be covered only with one set of lines. We refer to the map obtained by thresholding as a *foreground map* even though it does not always specifically separate foreground and background, and indeed is not guaranteed to have any semantic information when produced by thresholding.

3.2 Additional Details

We would like to apply some random variation to the particle directions to increase the variety and naturalness of the resulting images. However, simply applying a random perturbation to each particle's bearing each timestep does not produce a suitable result; under this regime, the particles simply acquire a high-frequency shivering motion but do not change their overall trajectories very much, since the long-term mean perturbation is zero.

Instead, we suggest a technique to increase the stability of the randomness. Each particle has a flag saying in which direction it should turn. At each timestep, a random value is added to the particle's bearing, causing it to turn slightly in the desired direction. Each particle also has a timer, counting down; when the timer reaches zero, it is reset to a random value and the turn flag is flipped to the other side. The magnitude of the random increment is between 0 and a parameter r that governs the size of the fluctuations. The second-derivative discontinuities from the timer resets are sometimes visible, and there is still room to improve this technique. Figure 7 shows some examples.

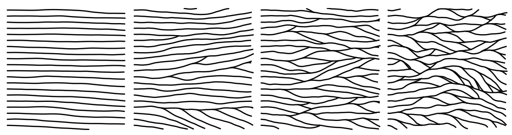

Figure 7: Different levels of randomness. From left to right: $r = 0.01$; $r = 0.03$; $r = 0.05$; $r = 0.09$.

The most interesting aspect of this scheme is the interaction between the randomness and the coordinated movement of multiple neighbouring particles. Because the particles' timers are not synchronized, particles will sometimes share a turn direction with one or both neighbours, and sometimes attempt to turn in opposite directions. However, particles seeking to turn towards each other are prevented by the coordination, where the particle directions are averaged. Conversely, when adjacent particles have the same turn flag, several particles turn simultaneously in the same direction. This behaviour is most apparent in flat-colored regions of the image plane, and we exploit the behaviour to create strong illusions of surface shape in abstract images, seen in the next section.

Our bearing calculations (equations 1 and 2) assume that all particles are approximately arranged in a line and their motion is roughly perpendicular to the line's direction, locally aligned with the vectors to the neighbours. However, because of randomness, variation in distance traveled owing to changing separation distance, or other factors, some particles may fall behind or move ahead compared to the local average. This may mean that the bearing calculations previously described no longer make sense; we need to take some steps to forestall the problem and ensure that the particles do not depart much from the linear structure. Our approach is to decide, for each particle pair, which one is ahead and which one is behind; then, the leading particle is dragged back and the lagging particle pushed ahead. Particles that are close together in progress will tend to alternate being pushed and dragged; although the resulting stuttering movement does not look very appealing in animation, the trajectories are not much affected in this case, especially since the magnitude of pushing or dragging will be near zero.

Consider a pair of particles A and B. Let ω be the projection of A's direction vector $A.\vec{d}$ on the displacement $\vec{s}_{AB}$. If $\omega > 0$, A is lagging; if $\omega < 0$, A is leading. We adjust the positions of A and B proportional to ω: $A.\vec{x}$ is incremented by $\Phi \times \omega \times A.\vec{d}$ and $B.\vec{x}$ is decremented by the same amount. Here Φ is a parameter controlling the magnitude of pushing or dragging; we used $\Phi = 0.005$. This calculation and correction is done for every pair of neighbouring particles at every timestep.

A compact summary of our method appears in the following pseudocode. The particle group in this code can travel over the entire image, i.e., not restricted by the foreground map.

Algorithm 1 Pseudocode for main simulation

```
    Each particle $P_i$ has properties: pointers $prev$ and $next$, a position vector $\vec{x}$, a bearing $b$, and a spacing threshold $T$. Designate the distance between $P_i$ and $P_{i-1}$ as $S_{P_iP_{i-1}}$.
Input: Spacing map $A$; array $P$ consists of particles.
Output: the array $P$ of particles updated.
 1: for each particle $P_i$ do
 2:     Take the spacing value at $P_i.\vec{x}$ from map $A$, say $A(P_i.\vec{x})$;
 3:     if $P_i.T > A(P_i.\vec{x})$ then                ▷ Update $P_i$'s spacing threshold
 4:         $P_i.T \leftarrow P_i.T - \beta$;
 5:     else if $P_i.T < A(P_i.\vec{x})$ then
 6:         $P_i.T \leftarrow P_i.T + \beta$;
 7:     Designate the base threshold as $T_g$;
 8:     $T_g \leftarrow (P_i.T + P_i.next.T)/2$;          ▷ Update base threshold
 9:     $T_b \leftarrow T_g \times 1.5$; $T_d \leftarrow T_g \times 0.5$;  ▷ Birth and death thresholds
10:     if $S_{P_iP_i.next} > T_b$ then                      ▷ Particle birth
11:         Create a new particle $P_m$;                  ▷ See Section 3.1
12:         $P_m.\vec{x} \leftarrow P_i.\vec{x}$;
13:     else if $S_{P_iP_i.next} < T_d$ then                 ▷ Particle death
14:         Start killing $P_i.next$;                     ▷ See Section 3.1
15: for each particle $P_i$ do                             ▷ Collision detection
16:     Get a list $L_i$ of all colliding particles;
17:     Kill all the particles in $L_i$;
18: for each particle $P_i$ do                   ▷ Position and bearing update
19:     Update $P_i.b$ and $P_i.\vec{x}$;                    ▷ See Algorithm 2
```

4 Discussion and Evaluation

In this section, we present some results from our coordinated particle system. We first illustrate the overall process, then show the results of applying the process to a few example images. Then, we show some results of purely abstract images with no guidance from any input image. Lastly, we compare our results with those of previous algorithms. Original images for the results shown are found in Figure 12.

4.1 Complete Procedure of Rendering

We begin with an input image, and compute a foreground map, possibly by thresholding. We then release two particle groups: the first group covers the whole canvas, while the second group only draws

Algorithm 2 Pseudocode for bearing and position update

```
    Designate $\Delta t$ as the timestep, $\vec{d}$ as the direction vector, and $b_{new}$ as the new bearing.
Input: Particle $P_i$.
Output: $P_i$ with its bearing and position updated.
 1: $P_i.\vec{x} = P_i.\vec{x} + \Delta t \times P_i.\vec{d}$;
 2: if $P_i$ is ahead of $P_i.next$ then          ▷ Let particles travel in a line
 3:     Push $P_i.next$ forward and drag $P_i$ backward;
 4: else if $P_i$ is behind $P_i.next$ then
 5:     Push $P_i$ and drag $P_i.next$;                    ▷ See Section 3.2
 6: Add randomness to $P_i.b$;                            ▷ See Section 3.2
 7: Compute $b_{new}$;                                    ▷ See Equation 1
 8: $P_i.b \leftarrow b_{new}$;
```

the black areas in the foreground map. The combination of lines drawn by both groups is our final result.

An overview of the rendering process appears in Figure 8. The first group of particles begins on the left side of the canvas traveling right, and the second group of particles travels from bottom to top. The second group only draws the trajectories when the particles are within the foreground area, as marked in the foreground map.

The first group of particles is shown in the first two subimages. Particle density is adjusted depending on image intensity, showing the dark landscape and ruin. Note that the particles approximately maintain the same pace, having been pushed or dragged to maintain the desired line formation. The second particle group is released on the bottom edge of the canvas; it darkens the landscape and the ruin even further. Notice how the output is able to portray different gray levels: the mountain has an impression of distance, apparently standing behind the ruin owing to the difference in particle spacing.

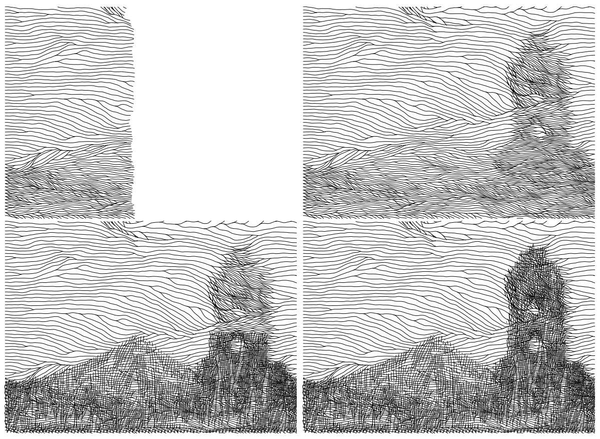
Figure 8: The procedure of creating a stylized image with our coordinated particle system.

4.2 Application to Images

In this section we show some results from applying the method to photographic images. As described previously, tones from the input image are approximately matched by varying particle spacing; also, we cross-hatch the darker portions of the image with a second set of particles. In principle, we could also vary line thickness depending on the underlying image tone; however, our vector rendering code does not presently support variable line width.

Overall, the large-scale structure of the input image is usually discernible in the output. For example, in Figure 9, the ship itself is nicely presented with the overlapping particle groups generating a darker tone, while the lighter area has only one group of particles and hence the output tone is also lighter. In this example, the particle traces shape a smooth, waving texture for the sky region, suggesting a complex surface despite no surface being actually modeled. The lighter clouds are somewhat suggested by the variation in line spacing; even in the relatively uniform region of the sky, interest has been increased by the fill of varied lines. Unfortunately, the sun is not shown: its tone is too low for our method to distinguish it easily from the surrounding clouds, and both tone values are outside the range separated by our thresholding. Since the water's surface is covered in dark waves, this area is populated by intermittent cross-hatching, suggesting the uneven surface and distinguishing it from the sky. In the line-drawing version of the image, the actual horizon is lost, but there is an apparent horizon that lines up with the bottom of the ship, so the main context of the image remains clear. We encourage the reader to view the digital version of the image and to zoom in so as to best appreciate the structure of the lines. Additional examples are given in the appendix.

Our system is most effective when the input image has high contrast or is otherwise easily able to be divided into foreground and background. Consider Figure 10. The wings and body of the butterfly are much darker than the background, allowing them to be emphasized with cross-hatching while the nearby regions are covered only by a single layer of lines. The flower is made indistinct, and hence deemphasized. The background, mostly quite unremarkable in the original image, has been replaced by a smooth and largely abstract texture. We consider this outcome to be a success.

One limitation of our system is that it is easy to miss some fine details when tracing out features. For example, the smallest branches in Figure 11 are not well portrayed, even though the branch edges are present in the foreground map. The process has succeeded to a large extent, and the bird itself and the thickest branches are portrayed nicely, but the numerous short strokes do not entirely resolve into a coherent depiction of the branches. The image also contains a suggestion as to how the situation can be improved: when the traces follow the branch direction, as is the case in the upper middle, the line is quite clear. The contrast in direction is sufficient to show the feature, even leaving aside the change in tone. It might be possible to improve on our results by having particles follow the structures in the input image, perhaps by exploiting the edge tangent field.

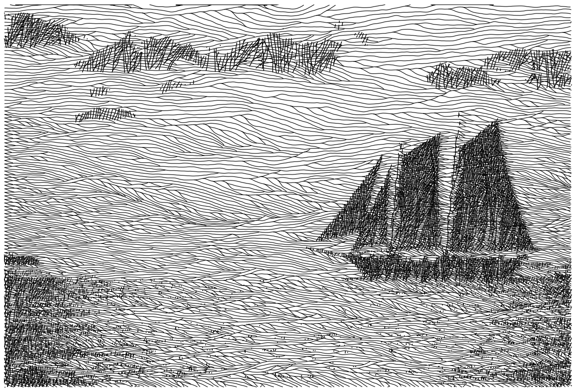
Figure 9: Stylized image: sailboat with sunset.

4.3 Abstract Patterns

In the previous subsection, we demonstrated that our system is capable of reproducing images with a natural smooth line-drawing style. It can also create compelling abstract images, such as the examples in Figure 13 where particles in multiple groups are used. We first draw a single curve on the canvas, and then release a particle

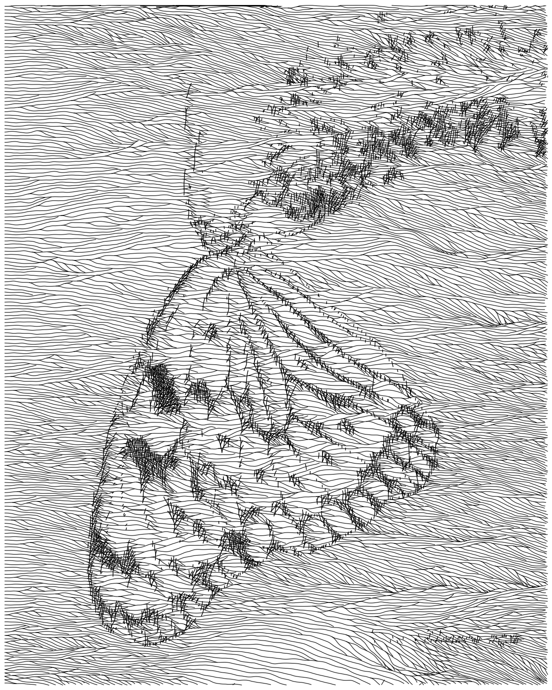
Figure 10: Stylized image: butterfly.

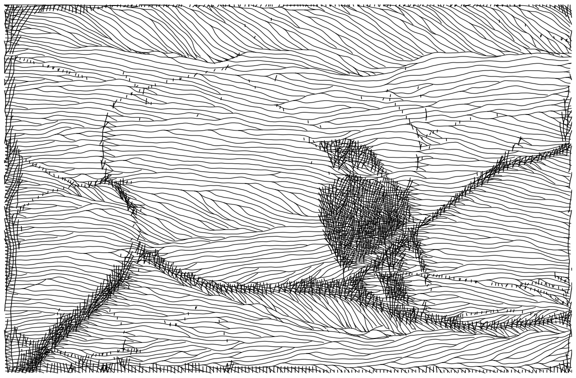
Figure 11: Stylized image: bird.

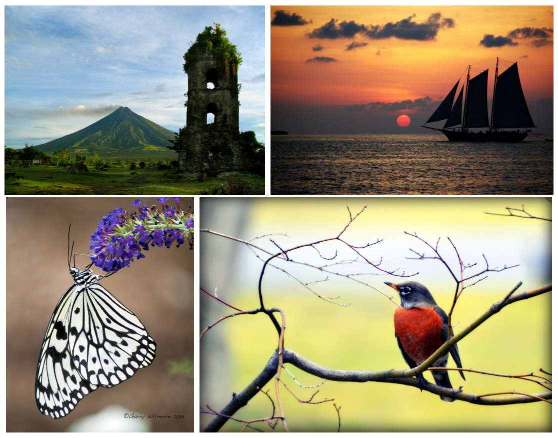
Figure 12: Original images used in this section.

group perpendicular to the curve. Each particle has an initial direction randomly changed from the perpendicular direction. The group undergoes the usual simulation process, with coordinated movement and particle birth and death, until the particles have either left the image or have collided with previously drawn trajectories. The curves traced by the head and tail of the group are tracked for later use: particles distributed along these curves are released once the main group has finished. In turn, the head and tail trajectories of later particle groups themselves form the basis of new groups. The process halts after nine groups have been processed; the number nine is quite arbitrary but generally fills most of the image plane, and the balance between filled and unfilled regions of the image is itself a source of interest in the final results.

Various abstract patterns are shown in Figures 13 and 14. By changing the randomness values that modify particles' directions, we can obtain curving lines which give an illusion of complex structure. For example, the patterns in the upper left of Figure 13 resemble braids. The upper right image seems to show a surface, such as desert dunes or water waves. The lower left image is a tangled structure reminiscent of the hairstyle drawings in Figure 2. On the lower right is a complex abstraction with many different elements. The larger example in Figure 14 also has many complexities; the wavering parallel lines occasionally give the impression of cloth or some other flexible material. There are small gaps where the particle systems have not filled the canvas; we found these interesting and did not seek to eliminate them, but if desired, they could be covered by additional groups of particles.

The overall appearance of the abstract images are controlled by the placement and shape of the initial thread line, the particle spacing (here constant over the entire canvas), the randomness parameter, and the number of groups to be released. We tried to choose examples that showed some variety, but due to the heavy use of random factors, all abstract images have some unexpected aspects.

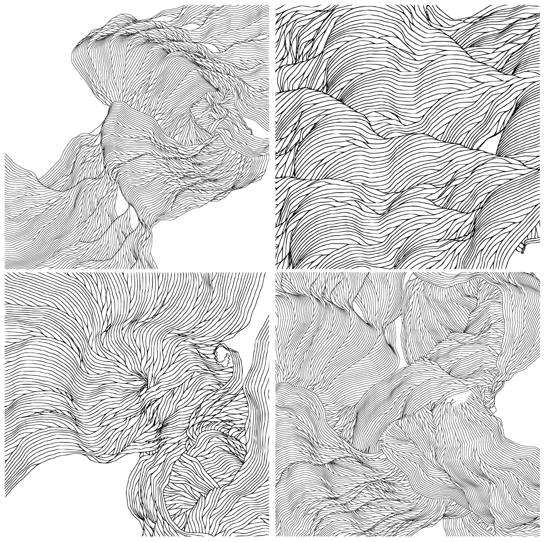
Figure 13: Some random abstract patterns.

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.

Li and Mould's method [11] to tessellate images using a particle system is algorithmically the most similar past work to ours. Figure 15 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 straighter lines, which allows the viewer to distinguish the background and the flower. Our result focuses both on the texture and the tone. The tone is conveyed both by the crosshatching in the darker areas and by the spacing. The structure of the lotus petals can be seen in our result because of the occasional crosshatched areas marking the shadows; contrast with the dark areas behind the flower is produced by small line spacing. As with other examples, a texture is introduced into the uniform area by the collection of lines. Our texture is smoother and less distracting than the chaotic flow seen in the artistic tessellation image.

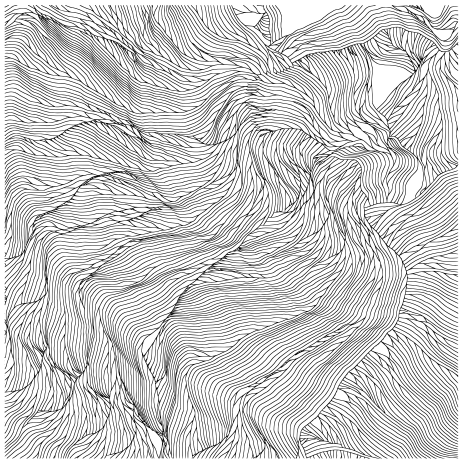
Figure 14: An abstract image.

Figure 16 is another example. We reproduced the lion in our black-and-white line-drawing style. The light and shadow are well managed, especially around the eye and mouth. Recall that we do not use the original image color to fill in the regions, as was done with artistic tessellation: the dark areas, such as the left-hand side, are made dark by cross-hatching with small spacing. The style is particularly good for this kind of image, where the flow of lines can suggest the texture of the lion's fur; additional suggestions of detail are given by the small foreground areas in the lion's mane, which are crosshatched with only a few short strokes.

Like the Op Art method proposed by Inglis et al. [7], our system is also able to convey different color sections with parallel lines moving in different directions. A generalization of the foreground map can indicate different segments to be drawn with different particle groups; each group is initialized with its own direction and only drawn within its designated segment.

As shown in Figure 17, the bottom left image of our result has a smooth appearance somewhat 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 Op Art, with its structured linewidth and directions, produces the characteristic shimmering perceptual effect; our does not, or at least not to as great an extent. However, the shape of the heart is as or more apparent in our result.

There are numerous small differences between the results of Op Art and our method. 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. Our results contain lines with variable, smoothly changing directions. In the Op Art method, the lines are continuous across segment boundaries, with lines carrying on from one section to another. Moreover, instead of using strictly parallel lines, our system reproduced the heart in the input image

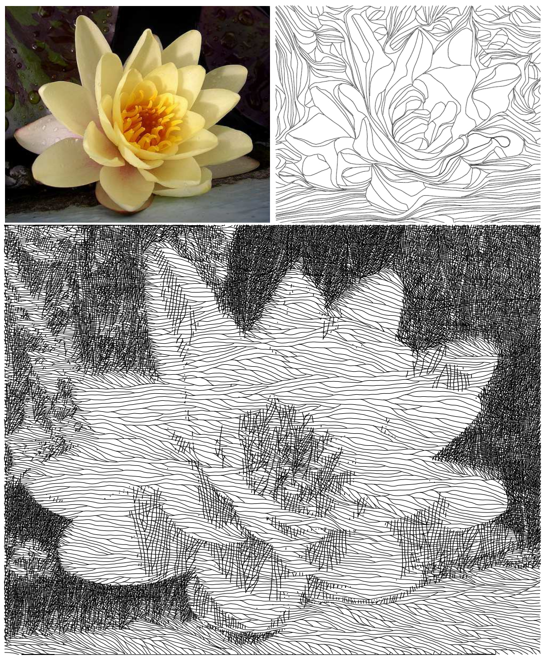

Figure 15: Top left: original image – lotus; top right: reproduced with tessellations [11]; bottom: our result.

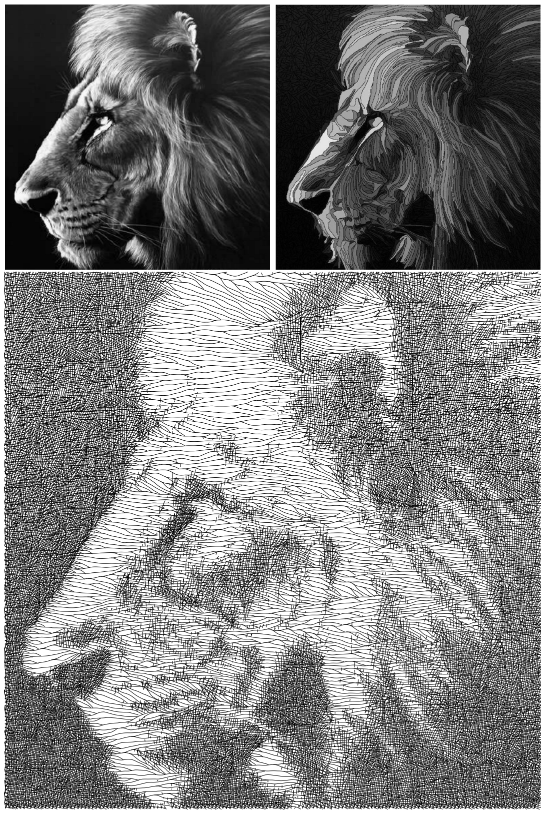

Figure 16: Top left: original image – lion; top right: reproduced with tessellations [11]; bottom: our result.

with some irregularities. We are also able to control the brightness by changing the line spacing – more lines appear in darker areas and less are shown in light areas. The Op Art method did not seek to reproduce tone at all.

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 a high level of randomness, the heart looks "furry" and lively, compared to the one in the bottom left example.

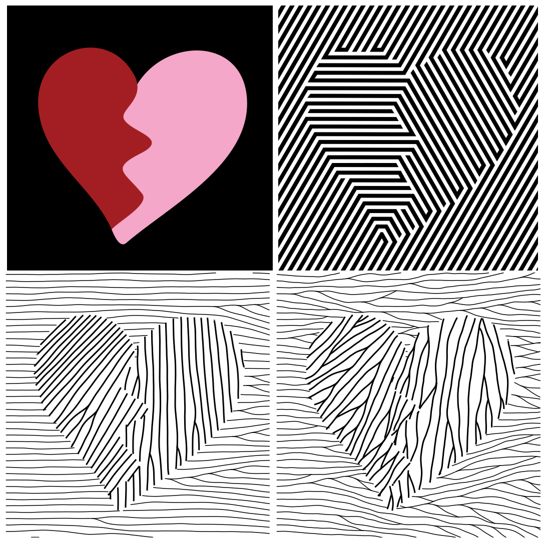

Figure 17: Top left: input image; top right: created with Op Art [7]; bottom left: our result with low randomness; bottom right: our result with high randomness.

6 Conclusion and Future Work

We presented a coordinated particle system which is capable of tracing out images with parallel lines of varied directions. To match tones and generate parallel lines, we add new particles to areas that have too few, and terminate particles when collisions occur or when particles are packed too closely together. Natural yet irregular patterns are achieved by randomly perturbing particle directions.

One of the disadvantages of our system is that the quality of results depends on the original image. Sometimes fine details cannot be portrayed because of the limitation of particles' spacings; our reliance on tone is also problematic, and using direction to convey features would also be useful. One possible approach 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 that conform to image content.

As we described before, particles in a group all start with the 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. Furthermore, with the ability to classify the texture orientations, we can have a new way to create foreground maps.

We would like to explore more in the area of abstract patterns. If we can arrange the threads properly, the particles traces will imitate a recognizable pattern such as waves or smoke. However, we also want to create abstract patterns without manual control, placing threads automatically.

Acknowledgements

Thanks to Hua Li and other GIGL members for their advice. Thanks to the anonymous reviewers for their helpful comments.

Many source images were taken from Flickr and are used under a Creative Commons license; thanks to photographers Garry Wilmore, Storm Crypt, Barbara Miers, Richard Skoonberg, Per Ola Wiberg, Andrew Krizhanovsky, Tambako the Jaguar, and Zuhair Ahmad for images of the butterfly, volcano, bird, sailboat, lotus, winter bridge, cat, and girl respectively. Financial support for this work was provided by the GRAND NCE.

References

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

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

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

[5] G. Elber and E. 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] A. Hertzmann and D. 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.

[7] T. Inglis, S. Inglis, and C. S. Kaplan. Op art rendering with lines and curves. *Computers and Graphics*, 36(6):607–621, 2012.

[8] T. C. Inglis and C. 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.

[9] H. Kang, S. Lee, and C. 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.

[10] A. L. Guptill. *Rendering in Pen and Ink: The Classic Book On Pen and Ink Techniques for Artists, Illustrators, Architects, and Designers*. Watson-Guptill, 1997.

[11] H. Li and D. 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.

[12] V. 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.

[13] H. Pedersen and K. Singh. Organic labyrinths and mazes. In *Proceedings of the 4th International Symposium on Non-photorealistic Animation and Rendering*, NPAR '06, pages 79–86, New York, NY, USA, 2006. ACM.

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

[15] C. W. Reynolds. Flocks, herds and schools: A distributed behavioral model. *SIGGRAPH Comput. Graph.*, 21(4):25–34, Aug. 1987.

[16] M. P. Salisbury, S. E. Anderson, R. Barzel, and D. 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] S. Schlechtweg, T. Germer, and T. Strothotte. Renderbots – multi agent systems for direct image generation. *Computer Graphics Forum*, 24(2):283–290, 2005.

[18] G. Winkenbach and D. 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.

[19] G. Winkenbach and D. 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.

[20] J. Xu and C. S. Kaplan. Image-guided maze construction. *ACM Trans. Graph.*, 26(3), July 2007.

Supplemental Images