The Visual Computer manuscript No.
(will be inserted by the editor)

Abstracting Images into Continuous Line Artistic Styles

Fernando J. Wong · Shigeo Takahashi

Abstract This paper focuses on the problem of designing and generating illustrations that portray a given scene with a single non-intersecting line. In this approach, users partition an image into regions, assigning a type or style to each of them. Next, a grid is generated over the drawing space, based on the parameters specified for each region. The illustration is then obtained in the form of a path that covers most areas of the grid. Contrary to previous works, our approach allows users to control the overall flow of the line throughout any given region, by providing the means to define tensor fields per region which directly influence the line orientation. We also extend this work for generating continuous line paintings, a similar style consisting of a single line that varies in color and thickness while covering the entire drawing space. This is achieved by transforming drawings obtained with the above mentioned approach through a Voronoi-based strategy.

Keywords Continuous Line Art · Line Drawing · Image Abstraction · Non-Photorealistic Rendering

1 Introduction

Continuous line illustration (CLI) [29] is a drawing style consisting of portraying a scene with a single line. Such drawings are frequently seen in the media, in the form of advertisements, logos, etc. Despite their restricted nature, these illustrations surprisingly manage to convey the most important details of a scene with only one line. The works of contemporary artist J. Eric Morales [18], for example, portray a single line that varies in density as necessary, portraying the features and overall shading of the scene in question. Apart from purely artistic purposes, possible applications of CLIs include the generation of labyrinth patterns [22], digital watermarking and steganography [8], among others.

Previous works addressed the generation of non-intersecting CLIs from a purely automatic perspective, by solving an instance of the traveling salesman problem (TSP), which is known to be NP-hard [2], on a set of stippled points [4,15]. While their results are visually appealing, they often require a significant amount of points to properly portray the shading of the image, and may not succeed for images with low contrast. Moreover, their computation is expensive due to the complexity of the TSP, even with state-of-the-art heuristic TSP solvers [3]. The line orientation is also random in these approaches, as the TSP computes a tour of minimum length, disregarding any orientation information. Thus, it is necessary to develop methods that provide more precise control over the CLI, while reducing the complexity of the problem.

This work presents a region-based framework for aiding users in the creation of non-intersecting CLIs. Users define regions in a given image (Figure 1(a)) and assign a type to each of them (Figure 1(b)). An orientation field is computed for each region based on its type, and then a grid oriented according to the field is generated (Figure 1(c)). All resulting grids are then combined into a single one that spans across the drawing space. A CLI is modeled as a path of grid cells covering most of this grid (Figures 1(d) and 1(e)), while smoothing it prior to its rendering (Figure 1(f)). By employing this type of region-based approach, we can better control the orientation, spacing and smoothing of the line illustration in any given region, which is difficult to achieve through purely automatic methods.

Fernando J. Wong and Shigeo Takahashi
The University of Tokyo
E-mail: {fjwong,shigeo}@visual.k.u-tokyo.ac.jp

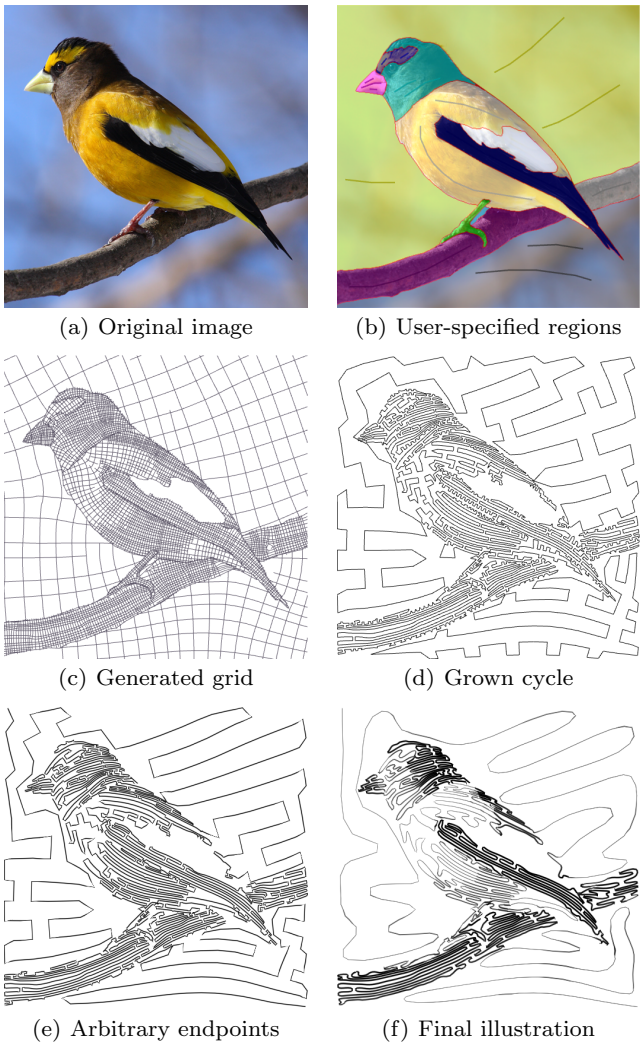
CLIs have also evolved into other visual art styles, such as continuous line paintings (CLP). Apart from having only one line to depict a scene, the line placement should be chosen carefully so as to match the borders and colors of the objects represented in these paintings. In particular, the style of Geoff Slater [26] seems difficult to reproduce. His paintings consist in covering the canvas with a line that changes in color and thickness, while still conveying the borders, colors, and other features of the scene. Our approach can be extended to produce CLPs by obtaining a set of walls that define the line in the painting based on a CLI, and diffusing colors along both sides of each wall.

This paper is organized as follows: Related work relevant to our research is presented in Section 2. Details of our CLI design approach are provided in Section 3. Our strategy for converting CLIs into CLPs is described in Section 4. Results of our approach are discussed in Section 5, while Section 6 presents concluding remarks and possible extensions of our work.

2 Related Work

CLIs are fairly new to computer graphics, with the earliest approaches being less than a decade old. The first of these works addressed CLI generation as an instance of the TSP over a set of points derived from the image intensity as described earlier [4]. A later extension used modern stippling techniques for improving the point distribution [15]. These works produced lines that approximated the shading of the image, in a style similar to that of Morales [18]. Due to the nature of the TSP, the line orientation throughout the drawing space was entirely random in these methods, an issue that is addressed in this paper. A recent work describes the generation of CLIs that focus on portraying the image contours instead of its shading [29]. This approach models a CLI as an Eulerian path in a graph derived from image edges, and produces lines that contain self-intersections, while portraying the object contours. However, this approach does not apply to our case, as we attempt to produce CLIs that are free of intersections.

CLIs and CLPs also share a relationship with mazes. For example, methods for evolving sets of curves into maze-like structures through the iterative use of forces can bear similar results to CLIs [22]. Also, these drawings could be considered as mazes with no branches. Maze enthusiast Walter Pullen [23] describes a method for creating such mazes on square grids, by creating a maze on the grid and splitting each passage through the middle. The final maze has twice the dimensions of the grid, and contains a non-branching path with endpoints adjacent to each other. A similar approach was employed recently for the creation of mazes whose solution path resembles a picture [20]. Although effective, these methods are difficult to apply in our case, due to the irregular nature of grids created in our approach. Another work by Wong and Takahashi [28] proposes a different strategy for generating image-based maze solutions that can be applied to such grids. This approach creates a solution path by growing and merging cycles across grid areas corresponding to image features, and also allows to specify arbitrary endpoints for it. This paper employs these concepts for CLI generation, as shown in Figure 10, where we have applied them for automatically generating CLIs.

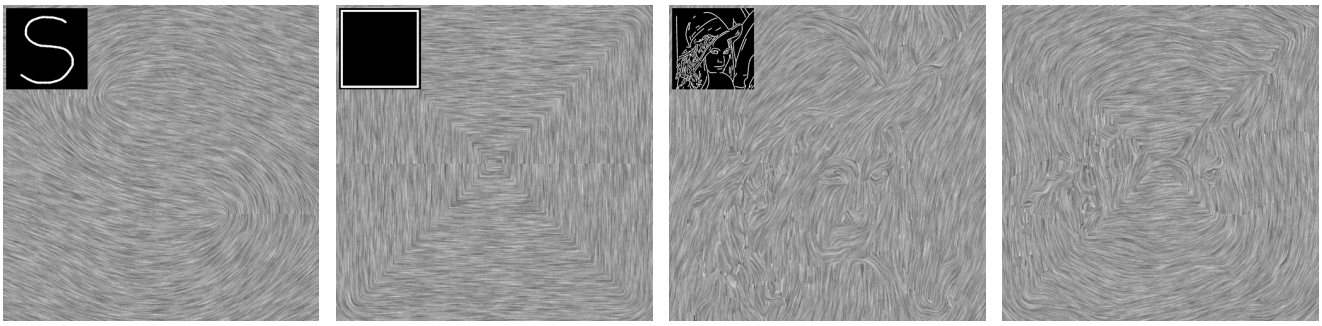


(a) Original image (b) User-specified regions

(c) Generated grid (d) Grown cycle

(e) Arbitrary endpoints (f) Final illustration

Fig. 1 Overview of our approach: (a) An input image is (b) segmented into regions by the users. The strokes seen on some of these regions define the orientation of the directional field for the region. (c) A grid is then created based on the orientation field of each region. (d) A cycle covering a majority of the cells in the grid is generated and (e) optionally converted into a path with arbitrary endpoints. (f) The CLI is finished after smoothing the resulting path.



(a) Directional (b) Boundary (c) Gradient (d) Combined

Fig. 2 Types of orientation fields supported by our system: (a) A directional field derived from user-specified strokes forming an S-shape. (b) Boundary field inferred from the contours of a square region. (c) Gradient field obtained from the features of an image. (d) The result of combining all three fields together with $w_{R_d} = w_{R_b} = w_{R_g} = 1.0$.

3 Continuous Line Illustration Design

Our framework takes as input an arbitrary image and a set of user-specified regions over the image domain. Users define these regions by clicking and dragging the mouse over the image, marking the pixels located at region boundaries, and then specifying a region as an area enclosed within the marked pixels.

We then compute an orientation field for each region in order to guide the CLI direction. As it is difficult to automatically infer fields that capture the region properties, we provide a variety of basic patterns or types which can be applied by the users to each region. The region types allowed by our system are:

- Directional: The field is generated according to the orientation of user-specified strokes (Figure 2(a)).
- Boundary: The field is oriented according to the region boundaries (Figure 2(b)).
- Gradient: The field is created based on image features in the region (Figure 2(c)).
- Blank: No processing is applied on these regions.

A similar variety of regions was used recently for the interactive design of painterly animations [19]. This approach makes use of different techniques for creating fields for each of these region types, while in our case, we use a simple unified strategy for computing tensor fields based on sets of line segments for all region types.

3.1 Orientation Tensor Fields from Line Sets

We compute an orientation field based on a set of segments S. A tensor with orientation Θ is defined as [31]

$$O(\Theta) = \begin{pmatrix} cos(2\Theta) & sin(2\Theta) \\ sin(2\Theta) & -cos(2\Theta) \end{pmatrix}. \quad (1)$$

The field defined by S at point p is then given by

$$T(p, S) = \sum_{s \in S} \frac{l_s}{1 + r(p, s)^2} \times O(\Theta_s), \quad (2)$$

where l_s and Θ_s are the length and orientation of s, respectively, and $r(p, s)$ is the point-line distance between p and s. Thus, longer segments have more strength over the field, while influence decays as the distance increases.

3.2 Directional, Boundary, and Gradient Fields

For directional regions, we obtain a field from user-specified strokes in a region R by first subdividing each stroke into a set of straight line segments S_{R_d} [24,9]. The directional tensor field T_{R_d} is then defined as

$$T_{R_d}(p) = T(p, S_{R_d}). \quad (3)$$

This results in smooth tensor fields oriented in the directions specified by the user (Figure 2(a)).

Fields for boundary type regions are defined similarly by computing tensor field

$$T_{R_b}(p) = T(p, S_{R_b}), \quad (4)$$

where S_{R_b} is the set of segments that constitute the boundary of R (Figure 2(b)).

For gradient type fields, we first detect edges in the image through a flow-based Difference of Gaussians (FDoG) filter [14]. The FDoG outputs a binarized image in which black regions correspond to edges detected in the image, which are then processed through a line-thinning algorithm [12] in order to convert them into a set of segments S_G. Let $S_{R_g} \subseteq S_G$ denote the subset of edges detected within region R, then the gradient-type tensor field T_{R_g} is given by

$$T_{R_g}(p) = T(p, S_{R_g}). \quad (5)$$

This results in a field whose orientation is dictated by the most important features in the region, while ignoring its minor ones (Figure 2(c)). While the FDoG filter employed earlier requires a field based on image gradients, which could be employed instead of this formulation, this field often captures the orientation of a great number of minor image details, which we would like to avoid for the purpose of image abstraction.

3.3 Combined Fields

We allow users to perform simple combinations of region types, in order to provide a level of customization, while keeping users from dealing with excessive degrees of freedom in their design (Figure 2(d)). For example, we can generate fields that are based on image features, while having a slight bias towards a particular direction. A combined field for a given region R is defined by:

$$T_{R_c}(p) = w_{R_d}\frac{T_{R_d}(p)}{\lambda_{R_d}(p)} + w_{R_b}\frac{T_{R_b}(p)}{\lambda_{R_b}(p)} + w_{R_g}\frac{T_{R_g}(p)}{\lambda_{R_g}(p)}, \quad (6)$$

where w_{R_t} is a user-specified weight in the range $[0, 1]$ that controls the strength of field type $t \in \{d, b, g\}$ in R, and $\lambda_{R_t}(p)$ is the major eigenvalue of tensor $T_{R_t}(p)$. Dividing $T_{R_t}(p)$ by $\lambda_{R_t}(p)$ results in an orientation tensor of the same form as Equation (1), which allows us to linearly combine tensors of different types.

3.4 Grid Generation

We generate grids for each region by making use of a conventional technique for quad-dominant remeshing [1], which has been used with success in maze generation [30, 28]. Streamlines are traced parallel and perpendicular to the orientation field in each region [13]. In order to approximate the shading of the image with the CLI, during the streamline placement phase, we set d, the distance between streamlines, as

$$d(p) = (d_{\max} - d_{\min})I(p) + d_{\min}, \quad (7)$$

where $d_{\min}$ and $d_{\max}$ are the minimum and maximum distance between streamlines, respectively, and $I(p)$ is the normalized image intensity at point p. We usually set $d_{\min} = 3$ pixels and $d_{\max} = 15$ pixels in our examples, although these values can be modified per region by the users. Also, we perform histogram equalization on the image intensity prior to this step, in order to have a higher contrast, resulting in a greater variation in the distance between streamlines.

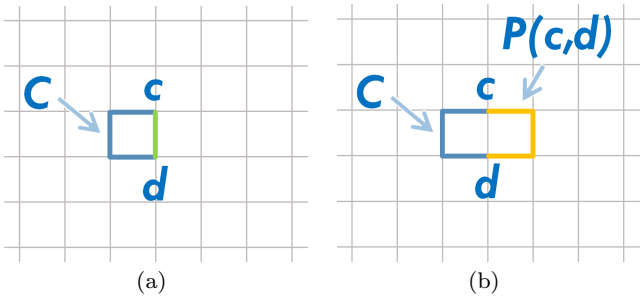
A grid of the entire image is then created by combining all generated grids (Figure 1(c)). During the grid creation process, we keep track of which segments correspond to streamlines traced parallel and perpendicular to the field. This information is used in the next subsection in order to obtain a path in the grid.

3.5 A Path that Covers the Grid

The next step is to find a path of cells spanning all over the grid (Figure 1(e)). Finding such a path is not trivial, as this is equivalent to computing a Hamiltonian path on the grid, a known NP-complete problem [10]. As a workaround, we adapt an algorithm for the generation of maze solution paths based on the growth and merging of cycles in a grid [28].

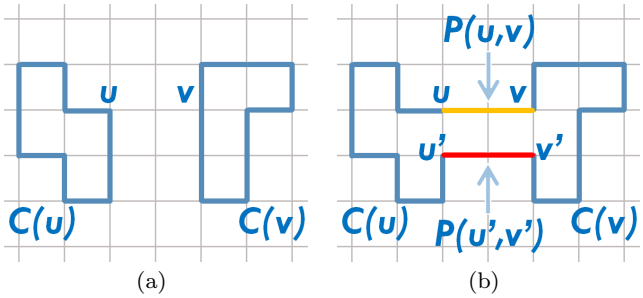
3.5.1 Cycle Growth and Merging

The algorithm first computes the dual graph $G = (V, E)$ of the grid, for which each $v \in V$ corresponds to a grid cell and each $e \in E$ to a grid wall. E is then divided into sets $E^{\parallel}$ and $E^{\perp}$ of edges parallel and perpendicular to the field, respectively. As G is the dual of the grid, edges in $E^{\parallel}$ correspond to grid walls perpendicular to the field, while those in $E^{\perp}$ to walls parallel to it. The process then splits in two phases: a cycle growth phase and a cycle merging phase.



(a) (b)

Fig. 3 Cycle growth: (a) Cycle C is grown from edge (c, d) by (b) removing (c, d) and appending path $P(c, d)$ to C.



(a) (b)

Fig. 4 Cycle merging: (a) Cycles $C(u)$ and $C(v)$ can be merged from vertices u and v by (b) bridging them with paths $P(u, v)$ and $P(u', v')$.

During the growing phase, a cycle C is iteratively grown from an edge $(c,d) \in C$ (Figure 3(a)) by replacing it with a shortest path $P(c,d)$ with edges in $E \setminus C$ (Figure 3(b)). By applying this process, C grows to cover as many vertices of G as possible. The algorithm can be controlled so that cycles tend to grow more in a particular direction. This is done by assigning a weight w in range $[w^{\parallel}_{\min}, w^{\parallel}_{\max}]$ to each edge $(c,d) \in C \cap E^{\parallel}$ and in $[w^{\perp}_{\min}, w^{\perp}_{\max}]$ to each $(c,d) \in C \cap E^{\perp}$. C is then grown at each step from the edge with lowest w. For the purposes of CLI generation, we typically set $w^{\perp}_{\min} = 0$, $w^{\perp}_{\max} = w^{\parallel}_{\min} = 1$ and $w^{\parallel}_{\max} = 2$, in order to grow cycles mostly parallel to the field.

If C fails to cover different areas of G, several cycles can be grown over G and merged into one. Two cycles $C(u)$ and $C(v)$ (Figure 4(a)), for which $C(u) \neq C(v)$, can be merged by first finding a shortest path $P(u,v)$ between two vertices u and v belonging to cycles $C(u)$ and $C(v)$, respectively. A second path $P(u',v')$ between vertices u' and v', for which $C(u) = C(u')$ and $C(v) = C(v')$, is then computed. Cycles $C(u)$ and $C(v)$ are merged together by opening and bridging them through $P(u,v)$ and $P(u',v')$ accordingly (Figure 4(b)).

3.5.2 Growth and Merging Across Regions

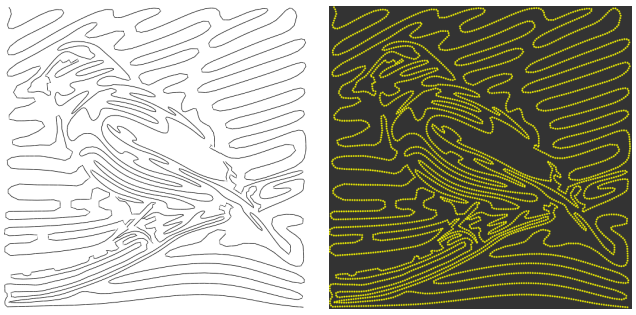
We would like to avoid the cycle from unnecessarily crossing over the region boundaries, as this lowers the overall perception of region borders in the CLI. This is done by restricting the growing phase to vertices within a region R_i, obtaining a set of disjoint cycles in it. Cycles are then merged within each R_i separately, resulting in one cycle per R_i. Finally, the cycles of all regions are merged into a cycle C_G spanning most of G.

We can also convert C_G into a path with arbitrary endpoints p and q, by splitting C_G in two paths at p and q, discarding the shorter one, and ending up with a path $P_G(p,q)$. Cycles can be grown in the now empty areas and merged with $P_G(p,q)$. This is also done per R_i, by first growing cycles from vertices $V \setminus P_G(p,q)$ within R_i and merging them with sections of $P_G(p,q)$ crossing over R_i. $P_G(p,q)$ thus becomes a path with endpoints p and q that covers most of G.

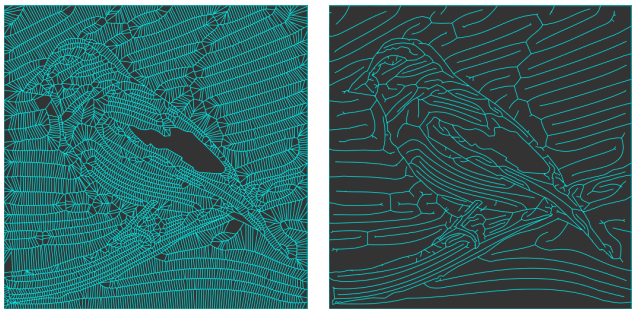
Since the process is designed to include as many vertices as possible, $P_G(p,q)$ often presents sections resembling square wave patterns, which disturb the line flow as in the cycle in Figure 1(d). We remove these artifacts in a post-processing phase. In some cases, the algorithm fails to cover some areas due to the irregularity of the grid. For those cases, we allow users to modify the path by moving, inserting or removing vertices. Finally, the path is smoothed through a cubic b-spline least squares fitting algorithm [11]. In general, we smooth the path more strongly across gradient and boundary type regions, and also vary the thickness and opacity of the line according to the image intensity. These properties, however, can be adjusted per region. Readers can refer to Figure 10 for examples of region-based CLIs generated using our approach.

4 From CLIs to CLPs

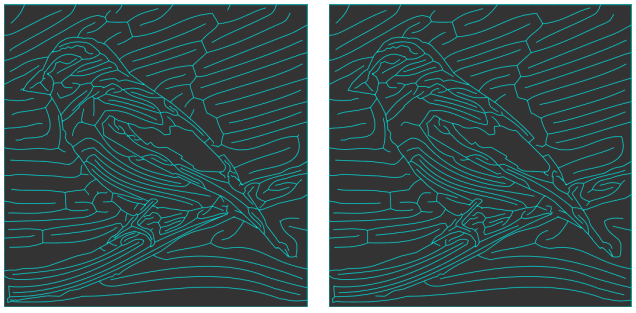
Our strategy for generating continuous line paintings does not create the line itself, but rather computes the "walls" defining the line in the CLP. This is a key difference from CLI approaches, as we cannot generate



(a) Generated CLI (b) CLI Samples



(c) Voronoi diagram (d) Edge removal step 1



(e) Edge removal step 2 (f) Edge removal step 3

Fig. 5 Creating walls for a CLP: (b) Samples are taken at fixed intervals along (a) a CLI. (c) A Voronoi diagram for each region is computed from the samples (blank-type regions are marked in white). (d) Edges between faces of subsequent samples are removed. (e) Edges whose length is below the predefined threshold are removed. (f) Edges whose orientation greatly differs from that of the CLI are removed as well.

aesthetic CLP styles by simply thickening a CLI, which could incur in self-overlappings along the line.

A naive approach for obtaining these walls would be to create a maze grid [30,27] of the drawing space, and remove grid walls accordingly, until all cells are merged into a non-branching path covering the grid. However, finding such a path is NP-complete as stated earlier. Instead, we devise a strategy for converting CLIs into CLPs through a Voronoi-based approach.

4.1 Voronoi Diagrams

The first step towards obtaining a set of walls for our CLPs involves the computation of a Voronoi diagram for each region R_i. This is done by taking a set of samples P (Figure 5(b)) at small intervals h of a CLI (further explanation is provided in Section 4.2). A diagram is computed for each R_i, by using the set $P \cap R_i$ as its seeds. Voronoi faces are then clipped against the boundaries of their respective region. After creating diagrams for each R_i, we combine them all into one diagram of the entire drawing space, while respecting the region boundaries. This new Voronoi diagram $V_D = (V, E, F)$ is a planar graph defined by a set of vertices V, a set of edges E and a set of faces F (Figure 5(c)).

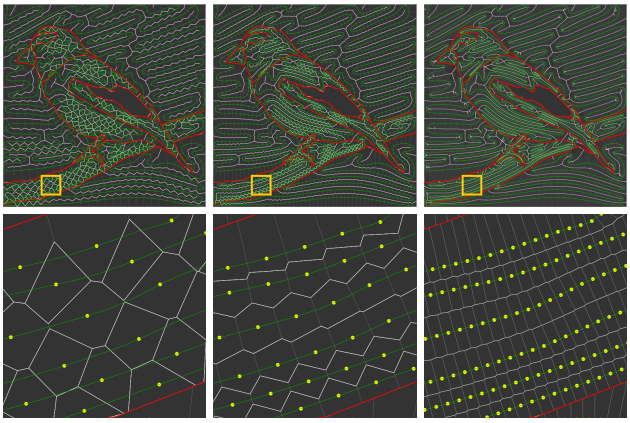
Although we could theoretically use any of our CLIs for computing V_D, in practice, we raise the values of $d_{\min}$ and $d_{\max}$ in Equation (7) when generating CLIs that are meant for creating CLPs, in order to avoid faces that are too small in size.

4.2 Voronoi Edge Removal

Our goal now is to remove a set of edges in V_D in order to give the appearance of a single line. Edge removal is performed in three steps.

On the first step, we remove edges separating faces whose seeds correspond to subsequent sample points (Figure 5(d)). Let $f_i \in F$ denote the face defined by $p_i \in P$, the i-th CLI sample taken from the CLI, and let $(f_i, f_j) \in E$ denote the edge between faces f_i and f_j. Then, an edge (f_i, f_j) is removed from V_D if $|i - j| = 1$.

The above process should merge all faces in V_D if performed correctly, except for those corresponding to blank-type regions. In order to achieve the desired effect, the sampling distance h (Section 4.1) should be chosen so that $h < d_{\min}$ (Equation 7). Otherwise, the process could fail, as two faces $f_i, f_{i+1} \in F$ might be located too far for an edge $(f_i, f_{i+1}) \in E$ to exist between them (Figure 6(a)). In addition, as h becomes smaller, the remaining edges attain a smoother appearance as shown in Figures 6(b) and 6(c). This is a desirable trait, as these edges will become the walls that define the CLP. After testing with different values, we have found that setting $h = 3$ pixels and $d_{\min} = 5$ pixels results in smooth curves while avoiding the presence of unmerged faces.



(a) $h = 20$ pixels (b) $h = 10$ pixels (c) $h = 3$ pixels

Fig. 6 Effects of the CLI sampling distance h on CLPs: The figure visualizes results of the first edge removal process on Voronoi diagrams generated with different h values, by showing the remaining edges (white), those removed (gray), the CLI path (green), region boundaries (red), and CLI samples (yellow). (a) If h is large, the process might fail to merge all faces. (b) Reducing h allows all faces to be merged, but a certain jaggedness is displayed in the remaining edges if h is higher than the line spacing in the region. (c) These edges present an increased smooth behavior as h becomes smaller.

At this point, the face enclosed within the remaining edges spans the entire space, but it still looks far from being a line. In particular, there is a high amount of short branches at points where the underlying CLI bends. We remove most of these edges as follows: Let $\deg(v)$ denote the degree of vertex $v \in V$, then all edges $(u, v) \in E$ for which $\deg(u) = 1$ or $\deg(v) = 1$, and whose length is shorter than a given threshold (10 pixels in our implementation) are removed (Figure 5(e)).

The final removal step involves the computation of a tensor field based on the CLI orientation. Let S_{CLI} denote the segments composing the CLI, then a tensor field T_{CLI} can be obtained by using Equation (2) as

$$T_{CLI}(p) = T(p, S_{CLI}). \tag{8}$$

Let $\overrightarrow{e_{CLI}}(p)$ denote the major eigenvector of $T_{CLI}(p)$, and $\overrightarrow{o}(u, v)$ denote the orientation of edge $(u, v) \in E$. We remove (u, v) if $|\overrightarrow{e_{CLI}}(p) \cdot \overrightarrow{o}(u, v)| < \cos(\Theta_{\max})$, and $\deg(u) = 1$ or $\deg(v) = 1$ (Figure 5(f)). Here, $\Theta_{\max}$ is the maximum angular difference between the orientations of the field and the edge in question ($\Theta_{\max} = \frac{\pi}{6}$ in our implementation).

After these operations, the set E contains the edges that will become walls for the CLP. Let $P(u, v)$ denote

a path between vertices $u, v \in V$, then the set of walls is defined as

$$W = \{P(u, v) \mid \deg(u) \neq 2 \text{ and } \deg(v) \neq 2\}. \quad (9)$$

That is, walls are defined as paths between two intersections ($\deg(v) > 2$) or end vertices ($\deg(v) = 1$).

4.3 Coloring the Line

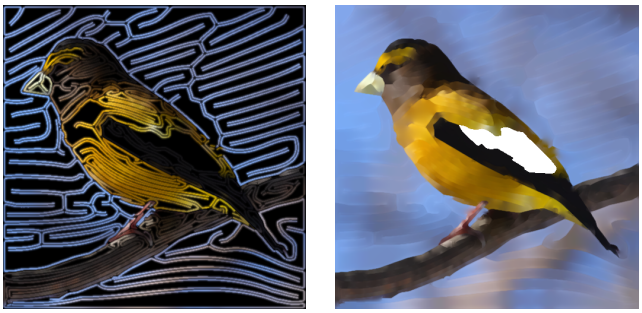
After creating the walls, we apply color to the CLP itself, i.e. the face within the walls. This is done by computing left and right colors for each wall $w \in W$ (Figure 7(a)), and treating them as diffusion curves [21].

During the creation of the Voronoi diagram (Section 4.1), we assign a color to each face $f_i \in F$ as $c(f_i) = I(p_i)$. That is, the color assigned to a face is the same as the pixel located at the coordinates of the sample $p_i \in P$ defining face f_i. The left color at a vertex $v \in w$ is then computed as

$$C_l(v) = \frac{\sum_{f \in F_l(v)} \text{area}(f) \times c(f)}{\sum_{f \in F_l(v)} \text{area}(f)}, \quad (10)$$

where $F_l(v) \subseteq F$ is the set of Voronoi faces located at the left of w at vertex v and area(f) is the area of face $f \in F$. The same formulation is employed for obtaining the color $C_r(v)$ at the right side of v.

After specifying colors for each wall, we render them as diffusion curves by employing the GPU-based implementation described in [21] (Figure 7(b)).



(a) Wall colors

(b) Diffusion curves

(c) Walls traced on top

(d) Final painting

Fig. 7 Rendering a CLP: (a) Left and right colors are defined for each computed wall, and (b) rendered as diffusion curves. (c) Wall segments are then traced over the diffused colors. (d) Users can control the color of the CLP per region, change the wall thickness, and smooth wall intersections. A blank type region was specified for the white areas of the bird's wing in this example.

4.4 Adding the Final Details

The painting is finished by tracing the walls on top of the results obtained in the previous section (Figure 7(c)). Our system varies the thickness of these walls so that they are thinner at their endpoints. Also, curved concave polygons are rendered at wall intersections in order to remove sharp corners in the CLP. Blank-type regions are rendered in white, as is the case for the clouds in Figure 10(c). Optionally, users can adjust the color and intensity of the image as a whole, or on a per region basis (Figure 7(d)). Examples of CLPs created through this approach can be appreciated in Figure 10.

5 Results and Discussion

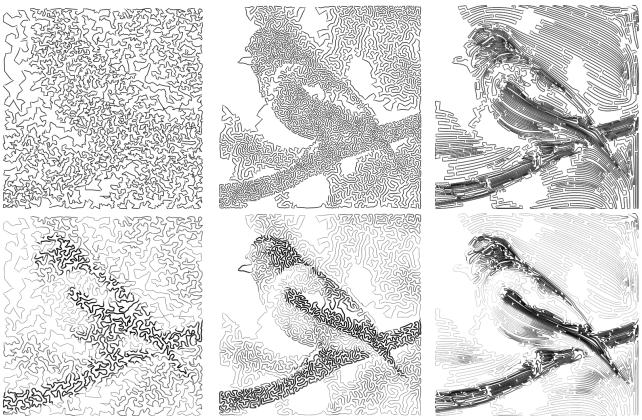
Our prototype system was implemented in C++ on an Intel Core 2 Duo E6550 2.33 Ghz CPU with 2 GB of RAM. The CGAL library was used for the exact computation of grids and Voronoi diagrams [5]. Generating a CLI from a 1024×768 image took an average time of 45 seconds on this setup.

Several continuous line art results can be appreciated in Figure 10, while their original images and user-specified regions are shown in Figures 11 and 12, respectively. All results were created to have endpoints at the top-left and bottom-right corners. The house walls and roof in Figures 10(b) and 10(c) are directional regions, while the grass and sky are combined directional + gradient regions, giving them a mostly horizontal orientation but without having a strictly directional pattern. Boundary regions were assigned to the dish in Figures 10(e) and 10(f), and the sunflower in Figures 10(h) and 10(i). Gradient regions were used in parts where a more organic-like texture was required, such as in the trees of Figure 10(b) and the background of the sunflower in Figure 10(h). Blank regions can be appreciated in the clouds of Figure 10(c), and in the triangular-shaped structure at the center of the dish in Figure 10(f). The colors of CLPs in Figure 10 were adjusted in order to give them a more artistic look.

The proposed method does not assume any restrictions on the input image, while the quality of the final result depends on the segmentation conducted by the users and on the objects portrayed. If a user defines a region that is too small or too narrow to grow a cycle, it

might be excluded from the path, disconnecting it from the rest of the line. In particular, objects such as trees, which are difficult to partition into regions, are prone to such problems. This imposes additional work on the users, which would have to adjust the path in order to cover such areas.

CLIs can also be generated automatically by computing the path over a grid derived from edge detection results. This is equivalent to assigning the entire canvas as a gradient type region. In these cases, we modify the cycle growth algorithm to avoid areas with a normalized intensity over 0.9, in order to better approximate the image intensity. Figures 10(a), 10(d), 10(g) and 10(j) show CLI examples created through this approach. Although object borders are often difficult to visualize, these results are comparable to those of Kaplan and Bosch [15], as shown in Figure 8. Figure 8(a) consists of a polyline composed of 8403 points and took 19 seconds to generate, while Figure 8(b) contains 10000 points and took 75 seconds. In contrast, the line in Figure 8(c) has a total of 10056 points and took 13 seconds to create. This approach, however, can fail for images containing many small details in areas of high intensity as shown in Figure 9, resulting in CLIs with reduced aesthetic appeal that are also difficult to recognize. This is due to the irregularity of the generated grid in such areas, coupled with the increased line spacing assigned to brighter sections of the image. However, this could be addressed by making use of the region-based CLI design approach described earlier.



(a) Bosch-Herman (b) Kaplan-Bosch (c) Ours

Fig. 8 Method comparison. Top row: Results of Bosch and Herman [4], Kaplan and Bosch [15] and our approach. Bottom row: The same CLIs after smoothing, modulating opacity and varying the line thickness.

Fig. 9 CLI generation can fail with images containing too many small details. The problem is worsened in this case as many details are contained in areas of high intensity, for which the line spacing is higher, and resulting in CLIs that are difficult to recognize.

6 Conclusions

We have introduced a user-guided approach for creating CLIs. Users segment an image into regions, and a grid is generated for each of them based on orientation tensor fields derived from user-specified parameters. A CLI is derived from a cycle obtained through the iterative growth and merging of cycles in the grid. The process is biased to encourage cycle growth in directions parallel to the field, and the cycle can be transformed into a path with arbitrary endpoints as well. Our approach generates CLIs whose orientation is determined by user-specified region types, as opposed to [4] and [15], where the line orientation was random.

We have also extended this approach for creating CLPs, by computing an appropriate set of walls that define the line painting, based on a previously obtained CLI. For this purpose, a Voronoi diagram is computed according to the CLI, and the walls are obtained by removing Voronoi edges accordingly. Coloring of the line takes place by computing and diffusing colors from both sides of each wall.

As future work, we would like to explore applications of our techniques for addressing similarly constrained artistic styles, for example, by placing text along the CLI [17], mapping panoramas or image collections to the line, or creating optical illusions by placing repeated asymmetric patterns along the curve [6]. The user interface of our system is rather primitive, requiring a lot of effort from the users for specifying regions in the image. We plan to address this by implementing more sophisticated interactive segmentation techniques based on graph cuts [16,25]. We also seek to improve our CLPs by developing better methods for wall removal, as well as implementing additional features such as automatic color correction and harmonization for each region [7].

Acknowledgements We thank all the participants in our preliminary user study for helping us in this algorithm formulation. This research has been partially supported by Japan Society for the Promotion of Science under Grants-in-Aid for Scientific Research (B) No. 21300033, and Challenging Exploratory Research No. 23650042.

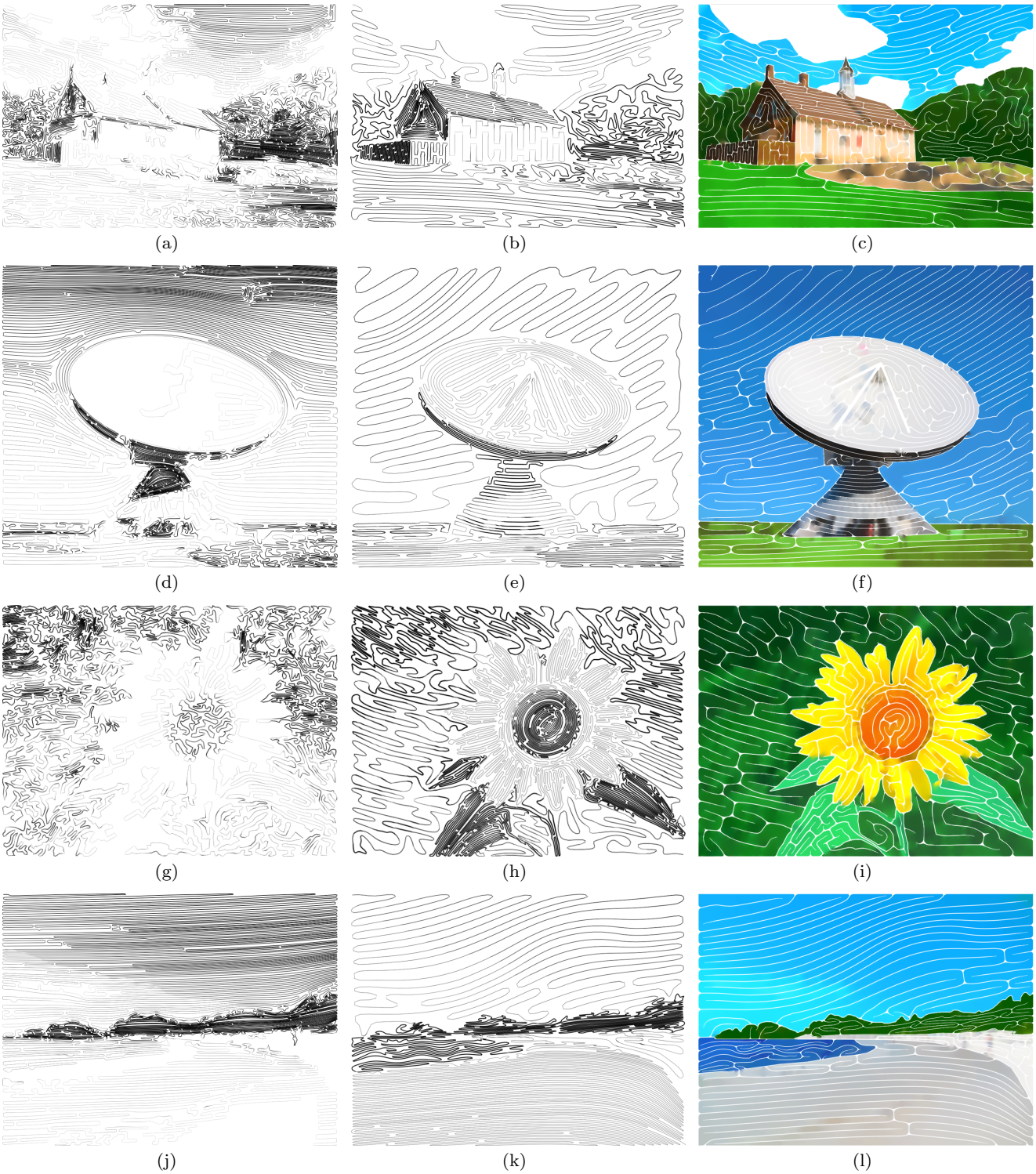


Fig. 10 Continuous line art results. Left: Automatically generated CLIs. Center: Region-based CLIs. Right: CLP results.

References

1. Alliez, P., Cohen-Steiner, D., Devillers, O., Lévy, B., Desbrun, M.: Anisotropic polygonal remeshing. ACM Transactions on Graphics **22**(3), 485–493 (2003)
2. Applegate, D.L., Bixby, R.E., Chvatal, V., Cook, W.J.: The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics). Princeton University Press, Princeton, NJ, USA (2007)
3. Applegate, D.L., Bixby, R.E., Chvatal, V., Cook, W.J.: Concorde TSP solver (2011).

Fig. 11 Original images employed for generating our results.

Fig. 12 User-specified regions for the region-based CLIs and CLPs shown in Figure 10.

http://www.tsp.gatech.edu/concorde/index.html
4. Bosch, R., Herman, A.: Continuous line drawings via the traveling salesman problem. Operations Research Letters **32**(4), 302–303 (2004)
5. CGAL: CGAL, Computational Geometry Algorithms Library (2012). Http://www.cgal.org
6. Chi, M.T., Lee, T.Y., Qu, Y., Wong, T.T.: Self-animating images: illusory motion using repeated asymmetric patterns. ACM Transactions on Graphics **27**(3), 62:1–62:8 (2008)
7. Cohen-Or, D., Sorkine, O., Gal, R., Leyvand, T., Xu, Y.Q.: Color harmonization. ACM Transactions on Graphics **25**(3), 624–630 (2006)
8. Cox, I., Miller, M., Bloom, J., Fridrich, J., Kalker, T.: Digital Watermarking and Steganography, 2 edn. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2008)
9. Douglas, D.H., Peucker, T.K.: Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. Cartographica: The International Journal for Geographic Information and Geovisualization **10**(2), 112–122 (1973)
10. Garey, M.R., Johnson, D.S.: Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA (1990)
11. Granger, G.: EOL BSpline Library (2009). URL http://www.eol.ucar.edu/homes/granger/bspline/doc/
12. Huang, L., Wan, G., Liu, C.: An improved parallel thinning algorithm. In: ICDAR '03: Proceedings of the Seventh International Conference on Document Analysis and Recognition, p. 780. IEEE Computer Society, Washington, DC, USA (2003)
13. Jobard, B., Lefer, W.: Creating Evenly-Spaced Streamlines of Arbitrary Density. In: Proceedings of the 8th Eurographics Workshop on Visualization in Scientific Computing, pp. 45–55 (1997)
14. Kang, H., Lee, S., Chui, C.K.: Flow-based image abstraction. IEEE Transactions on Visualization and Computer Graphics **15**(1), 62–76 (2009)
15. Kaplan, C.S., Bosch, R.: TSP Art. In: Proceedings of Bridges 2005, Mathematical Connections in Art, Music and Science, pp. 301–308 (2005)
16. Li, Y., Sun, J., Tang, C.K., Shum, H.Y.: Lazy snapping. ACM Transactions on Graphics **23**(3), 303–308 (2004)
17. Maharik, R., Bessmeltsev, M., Sheffer, A., Shamir, A., Carr, N.: Digital micrography. ACM Transactions on Graphics **30**(4), 100:1–100:12 (2011)
18. Morales, J.E.: Virtual Mo (2005). http://www.virtualmo.com/
19. O'Donovan, P., Hertzmann, A.: Anipaint: Interactive painterly animation from video. IEEE Transactions on Visualization and Computer Graphics **18**(3), 475–487 (2012)
20. Okamoto, Y., Uehara, R.: How to make a picturesque maze. In: CCCG, pp. 137–140 (2009)
21. Orzan, A., Bousseau, A., Winnemöller, H., Barla, P., Thollot, J., Salesin, D.: Diffusion curves: a vector representation for smooth-shaded images. ACM Transactions on Graphics **27**(3), 92:1–92:8 (2008)
22. Pedersen, H., Singh, K.: Organic labyrinths and mazes. In: NPAR '06: Proceedings of the 4th international symposium on Non-photorealistic animation and rendering, pp. 79–86. ACM, New York, NY, USA (2006)
23. Pullen, W.D.: Think labyrinth (2008). http://www.astrolog.org/labyrnth.htm
24. Ramer, U.: An iterative procedure for the polygonal approximation of plane curves. Computer Graphics and Image Processing **1**(3), 244–256 (1972)
25. Rother, C., Kolmogorov, V., Blake, A.: "GrabCut": Interactive foreground extraction using iterated graph cuts. ACM Transactions on Graphics **23**(3), 309–314 (2004)
26. Slater, G.: Geoff Slater - the kilted, single line artist (2001). Http://www.geoffslater.com/
27. Wan, L., Liu, X., Wong, T.T., Leung, C.S.: Evolving mazes from images. IEEE Transactions on Visualization and Computer Graphics **16**(2), 287–297 (2010)
28. Wong, F.J., Takahashi, S.: Flow-based automatic generation of hybrid picture mazes. Computer Graphics Forum **28**(7), 1975–1984 (2009)
29. Wong, F.J., Takahashi, S.: A graph-based approach to continuous line illustrations with variable levels of detail. Computer Graphics Forum **30**(7), 1931–1939 (2011)
30. Xu, J., Kaplan, C.S.: Image-guided maze construction. ACM Transactions on Graphics **26**(3), 29 (2007)
31. Zhang, E., Hays, J., Turk, G.: Interactive tensor field design and visualization on surfaces. IEEE Transactions on Visualization and Computer Graphics **13**(1), 94–107 (2007)