Enhanced Lane: Interactive Image segmentation by Incremental Path Map Construction

Hyung W. Kang and Sung Yong Shin

Korea Advanced Institute of Science and Technology
373-1 Kusung-dong, Yusung-gu, Taejon 305-701, South Korea
e-mail:{kang,syshin}@jupiter.kaist.ac.kr

Abstract*—Live-wire type techniques for interactive image segmentation are of practical use for various applications such as medical image analysis, digital image composition, etc.** *Intelligent scissors* **[5] and** *live wire* **[8] are the representative techniques of this type, which are based on a graph search over an entire image. Another technique called** *live lane* **[8] is also based on a graph search but localizes the search domain to give an interactive feedback. Compared to the live wire, the live lane trades off the repeatability of segmentation for its time efficiency. In this paper, we present a novel image segmentation technique called** *enhanced lane, a modified version of the live lane that ensures both efficiency and repeatability. By restricting the search domain and updating the path map incrementally, the enhanced lane can extract objects from an image interactively with its efficiency comparable to that of the live lane while also keeping its repeatability comparable to that of the live wire. Based on the live lane paradigm, our technique also differs from the time-efficient version of live wire called** *live wire on the fly* **[9]: the enhanced lane always guarantees strictly bounded response time regardless of the image size and follows the target boundary with little digression which leads to better repeatability.**

***Keywords*—Interactive image segmentation, Livewire, Dynamic programming, Graph search**

I. Introduction

A. Motivation

Interactive image segmentation techniques are of practical use for various applications including image analysis, image composition, key extraction, etc. Compared to fully automated image segmentation, interactive segmentation exploits user's knowledge on the target object for tracing its boundary. In particular, live-wire type techniques provide the users with tight control in segmentation while yielding preciseness [5], [6], [8], [9].

The representative live-wire type techniques such as *intelligent scissors* [5] and *live wire* [8], are based on a graph search over an entire image. Another technique called *live lane* [8] is also based on a graph search but localizes the search domain to give an interactive feedback. These two complementary techniques, the live wire and the live lane trade-off between their time efficiencies and repeatabilities, as pointed out in [8]. That is, the live wire preserves the repeatability at the cost of its speed, while the live lane sacrifices its repeatability to gain better time efficiency.

In this paper, we propose a novel live-wire type technique called *enhanced lane*, that ensures both repeatability and time efficiency. By restricting the search domain to the user-controlled local window and updating the path map incrementally, the enhanced lane can extract object boundaries from an image interactively with its efficiency comparable to that of the live lane while also keeping its repeatability comparable to that of the live wire. Based on the live lane paradigm, our technique also differs from the time-efficient version of live wire called *live wire on the fly* [9]: the enhanced lane always guarantees strictly bounded response time regardless of the image size, and follows the target boundary with little digression which leads to better repeatability.

B. Related Work

Image segmentation is a process of partitioning an image into a set of disjoint regions with similar characteristics such as intensity, color, texture, etc. Many researchers in artificial intelligence or computer vision, have concentrated on achieving fully automated image segmentation [1], [2], [18], [19], [20]. However, fully automated segmentation often fails to detect the target boundary, due to the lack of the global knowledge on the objects. Moreover, when the image itself is noisy or contains many objects of complex shape, the exact target boundaries are hard to identify automatically. Therefore, user-guided semi-automatic image segmentation techniques have been proposed in this context [5], [8], [6], [7], [9], [10], [11], [13], [14], [15].

Interactive segmentation techniques can be either region-based or boundary-based. The typical example of region-based techniques is the so-called magic wand [16], which is available in many commercial painting systems. The magic wand enables a user to interactively select a seed point to grow a region by adding adjacent neighboring pixels which satisfy some similarity criteria on pixel attributes such as intensity color value. The region growing is automatically terminated when no more adjacent pixels satisfying the criteria are left available. The resulting region boundary is usually postprocessed since the region growing does not provide interactive visual feedback as pointed out in [5].

Among the boundary-based interactive segmentation techniques, active contour (also called snake) is one of the most popular and well-studied [13]. Snake requires a user to provide an initial curve that approximates the target boundary.

With an initial snake curve placed near a boundary, the curve automatically locks on to the boundary by minimizing an energy functional. The energy at a point on a curve is a combination of internal force such as the curvature at the point and external forces such as its image gradient. The snake facilitates automated boundary tracing, and thus the final shape of the boundary is hard to control interactively. If the resulting boundary is not acceptable, boundary tracing must be repeated with a new initial curve, or the boundary must be postprocessed.

Gleicher proposed an image snapping technique which is essentially a snake applied to individual points [15], for automatically attracting current mouse cursor position to nearby features such as edges. Image snapping is guided only by the external force of snakes, which is the gradient on the low-pass filtered feature map of an image. By connecting adjacent snapped points in sequence, the target boundary can be interactively traced. However, the line segment between two consecutive snapped points is not guaranteed to lie on the target boundary, which could affect the accuracy of segmentation and smoothness of the resulting boundary.

Another well-known boundary-based techniques are those of the live-wire type [5], [6], [7], [8], [9], [10], [11], [14]. Mortensen et al. proposed intelligent scissors based on global graph search [5], [6], and Falcão et al. presented a slightly different version called live wire [8]. With a seed point initially planted, a path map is constructed to provide the minimum-cost path from the seed to every point in the image. By interactively moving a cursor near the boundary of an object, the live wire is extended according to the path map to form a boundary segment. A sequence of seed points can possibly be chosen when the image is noisy or contains complex objects. Whenever a new seed point is selected, the path map starting from this point is computed for the entire image to replace the previous map. While providing highly interactive visual feedback, this type of tools is time-consuming to reconstruct the path map when the image size is large. Some interactive segmentation tools based on live-wire techniques have recently been available in commercial imaging software, such as Adobe Photoshop$^{\mathrm{TM}}$ [16] or ProntoMask$^{\mathrm{TM}}$ [17].

Originated from the live wire, another technique called live lane [8] is proposed to increase the efficiency of boundary construction. The live lane restricts the search domain to construct a path map within a local window centered at the current seed point. This map records the minimum-cost path from the seed to every point in the window. As a cursor moves in this window, the corresponding boundary segment is interactively displayed according to the path map. When the cursor crosses the window, the boundary segment from the seed point to the crossing point is automatically frozen. The crossing point becomes a new seed point, and a new path map is constructed within the window centered at this point. That is, a new seed point is added whenever the cursor crosses the current window. Hence, the live lane requires more seed points than the live wire, especially when the window size is small. Moreover, these seed points may not lie exactly on the target boundary, which may degrade the repeatability of the live lane.

Falcão et al. further extended the original idea of the live wire and proposed a new scheme called live wire on the fly [9]. The main goal of this tool is to improve the time efficiency of the live wire, while retaining its basic philosophy of global graph search. Exploiting the basic properties of Dijkstra's shortest path algorithm, the live wire on the fly incrementally expands the shortest path map only up to the cumulative cost of the current cursor position, thus avoiding the unnecessary computation for the paths with bigger cumulative costs. This idea has resulted in much faster segmentation for large images even on low-powered computers. However, the response time tends to get longer for a large image as the cursor moves far away from the seed point since the overall path map gets bigger. Also, the live wire on the fly traces a globally optimal path between two points, which may not necessarily correspond to the target boundary and thus require more seed points to plant. Restricting the search domain under user guidance, the live lane has some inherent merits such as locality and controllability over the live wire on the fly, which have not fully explored yet. Exploiting those merits, we develop a new segmentation technique based on the live lane.

C. Overview

We present a novel live-wire type segmentation technique called *enhanced lane* that ensures both time efficiency and repeatability. Adopting the philosophy of local search from the live lane, our segmentation technique computes a path map only within a local window centered at the seed point to provide time efficiency regardless of image size and seed point location. Also, the enhanced lane incrementally extends the path from the current seed point to every pixel in each successive window containing the cursor, to form a cumulative path map. Since no boundary segment is fixed when the cursor crosses the current window boundary, there is no need to plant a new seed at the crossing point. Like the live wire, the enhanced lane selects a new seed only when the boundary segment from the seed point to the cursor digresses from the target boundary. We also show that the enhanced lane always finds a locally optimal path in the cumulative region formed by the sequence of windows.

The remainder of this paper is organized as follows. Section II describes both the live wire and the live lane, the two complementary segmentation tools based on graph search, in detail. In Section III, we present the basic idea and algorithm of our tool together with detailed analysis and discussion, to compare it with other tools including the live wire, the live lane, and also the live wire on the fly. Sections IV and V deal with some implementation issues and experimental results with various test images, respectively. We conclude this paper in Section VI.

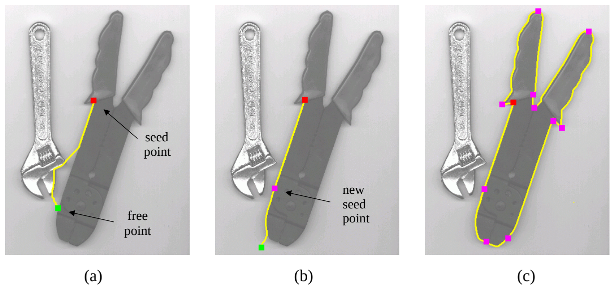

Fig. 1. *Live Wire: (a) a digression occurs (b) a new seed point is planted (c) the complete boundary is identified*

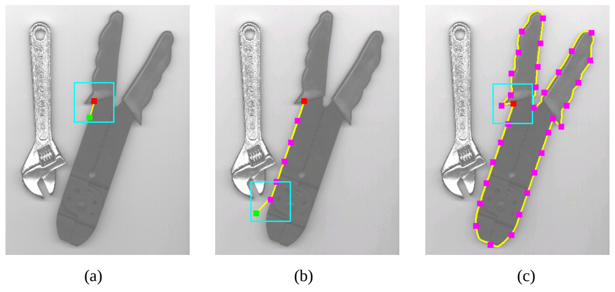

Fig. 2. *Live Lane: (a) the search region is restricted by a local window (b) new seed points created at the crossing points (c) the complete boundary is identified*

II. Live-wire type techniques: Live wire and Live lane

In [8], Falcão et al. proposed two techniques for extracting the boundary of an object in an image. Like intelligent scissors, both of them utilize path maps represented by weighted graphs. An image (or a portion of an image) is considered as a directed graph in which pixel corners and oriented pixel edges represent the vertices of the graph and its arcs, respectively. To each oriented pixel edge, a set of features is assigned to give its cost (or weight). Then, the problem of constructing the best boundary segment between any two points specified on the boundary is reduced to that of finding the minimum-cost path between the two vertices in the graph. By using oriented pixel edges, it is possible to trace the boundary of an object without distractions from closely running boundary segments even with one pixel apart. Moreover, the directed nature of the graph distinguishes boundary segments that have opposite orientations but otherwise have very similar properties.

The authors incorporate features such as intensities on each side of an edge and its orientation-sensitive gradient magnitude. They provide various functions to transform those features to their corresponding costs for the edge. The overall cost of an edge is given as a linear combination of those costs that can effectively discriminate, from the others, the pixel edges belonging to the boundary. They also developed a training technique to find an optimal set of features and transform functions for an application at hand. While their training scheme requires a separate training phase, the authors of [5], [6] proposed a more practical idea for training which can be done on the fly.

The difference between the two techniques lies in their graph search strategies and user interaction schemes. In both methods, the user is required to place a starting seed point on the target boundary. The live wire, which is based on a global graph search, constructs a path map containing the minimum-cost paths from the seed point to all the vertices in the image. Based on the global path map thus constructed, the user selects a desired boundary segment by interactively moving the other end of the path called free point (which is also controlled by the cursor). As the free point moves in proximity to the target boundary, the live wire displays the optimal path from the seed point to the free point, which gives the impression that the live-wire automatically snaps at the target boundary (See Figure 1)[1]. When the optimal path from the free point digresses from the desired object boundary, a new seed point is placed interactively or automatically to re-initiate the new path map construction, as shown in Figure 1(b)[2]. This causes a new boundary segment to be extended from this seed point while fixing the boundary segment computed up to the seed point. A complete boundary is identified via a set of consecutive boundary segments each detected in this fashion.

In the live lane, a path map is initially constructed only in the square window containing the seed point planted at its center (see Figure 2). As the free point starts from the seed point and moves inside the window, the boundary segment linking the two points is interactively displayed. If the free point crosses the window boundary, the crossing point automatically becomes a new seed point, fixing the current boundary segment in the window. A new path map is then constructed in a window centered at the new seed point, from which the boundary segment is extended. Unlike the live wire, the target boundary is detected only if the user correctly steers the cursor in the vicinity of the boundary within a lane of certain width. The lane width is either fixed or varied adaptively along the boundary, making it larger for strong boundary segments and smaller for uncertain ones. Since the users usually steer the cursor faster near a strong boundary segment, the width can be controlled in terms of the speed and acceleration of the cursor motion.

In evaluating the performance of boundary construction methods, the authors considered three factors to be of prime importance: speed, repeatability, and accuracy. Accuracy means the degree of agreement of the extracted boundary with the target boundary. In general, the use of better cost functions in constructing the path map could lead to better accuracy in tracking strong edge features. Unfortunately, it is hard to define the target boundary itself since it is not always composed of the edges with the strongest features. Thus, a manually traced boundary is often considered to be a good approximation as the target boundary. However, the credibility of manual tracing is still questionable since its

[1] As a cost function for Figure 1 and Figure 2, we employed the gradient magnitude with the inverted linear transformation function [8].

[2] In this simple input image, it may be possible to reduce some digressions by applying training schemes [5], [8].

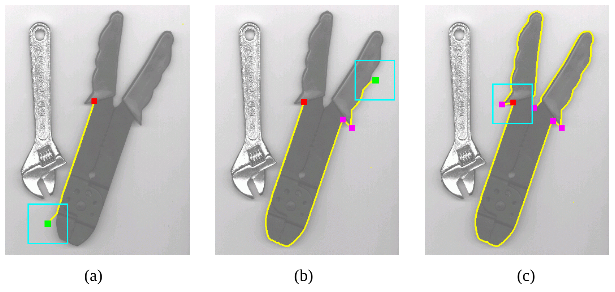
Fig. 3. *Enhanced Lane: (a) The digression is avoided by restricting the search region (b) A new seed point is created where the digression is inevitable (c) The complete boundary is identified*

repeatability is worse than that of the live wire or live lane. This results from the fact that machines are usually better than human beings in preciseness. Hence the authors evaluated the performance of segmentation methods as a function of only speed and repeatablility. Speed is given by the inverse of the amount of time required to extract an object boundary. Repeatability is inversely proportional to the variation over the boundaries extracted from repetitive experiments. As evaluated in [8], the live lane is better than the live wire in speed by localizing the search domain, especially when the image size is large. The live wire, on the contrary, has better repeatability than the live lane since the live lane usually requires more seed points which may not lie on the target boundary.

III. Enhanced lane

In this section, we explain the mechanism of enhanced lane and discuss how it preserves both speed and repeatability.

A. Basic Idea

Inspired by the paradigm of the live lane, the enhanced lane also localizes the search domain while extending path map incrementally. With a seed point planted, the initial path map is constructed only inside a local window centered at the point. Thus, the optimal path from the seed point to every point in the window is computed and recorded in the path map. Starting from the seed point, a free point is then moved along the boundary of an object within the window and the boundary segment between two points is interactively displayed using the path map. Unlike the live lane, the free point of the enhanced lane serves as the center point of the window, so that the window goes along with it. As the free point moves, we fill in the portion of the new window not overlapping the previous one to incrementally extend the path map. This incremental extension of the path map enables the path to grow from one window to another without reinitializing the map (see Figure 3).

B. Boundary construction

As given in [8], an image is represented as a graph, where vertices and arcs correspond to pixel corners and pixel edges, respectively. Also, each pixel edge is assigned a local cost in a similar way to that for the live wire or the live lane, considering various edge features (gradient magnitude, intensity on each side of the pixel edge, etc.) and cost assignment functions (inverted linear, inverted Gaussian, etc.). The optimal boundary segment in this graph search formulation is defined as the minimum cumulative cost path from a start vertex to a goal vertex where the cumulative cost of a path is the sum of the local pixel-edge costs on the path.

Let $T = (t_0, t_1, ..., t_n)$ be a sequence of time instances where t_i represents the ith time instance at which the position of the cursor is sampled. For any t_i, let $v(t_i)$ and $w(t_i)$ denote, respectively, the vertex pointed to by the cursor and the window to which the path map construction is performed. The path map construction is based on a dynamic programming technique that constructs a shortest-path tree on a graph by successively choosing a vertex to assign its optimal cost and path information [4], [5], [8]. Upon selection of the seed point $v(t_0)$, the initial path map is constructed inside $w(t_0)$ that is centered at $v(t_0)$. Initially, both the seed $v(t_0)$ and the free point $v(t_i)$ of the boundary segment are located at $v(t_0)$. As the cursor moves from $v(t_0)$, we extend the path map to include the successive windows $w(t_1)$, $w(t_2)$, ..., and so on. Accordingly the boundary segment is extended. At each time instance t_i, the boundary segment connecting $v(t_0)$ and $v(t_i)$ is displayed guided by the extended path map, where $v(t_i)$ denotes the vertex located at the center of $w(t_i)$.

Given the path map constructed up to t_i, we extend the path map to include $w(t_{i+1})$. The domain of path map expansion, denoted $\mathcal{D}(t_{i+1})$, certainly contains the vertices in the non-overlapping region of $w(t_{i+1})$ with the previous windows since they have not been explored yet (Figure 4(a)). $\mathcal{D}(t_{i+1})$ may also contain some vertices in the overlapping region. As shown in Figure 4(b), the paths to the latter vertices can move into $w(t_{i+1})$ through some vertices on the border of $w(t_i)$ contained in $w(t_{i+1})$ and get back to them later to lower their costs. Clearly, their current costs should not be lower than that of the minimum-cost vertex, denoted $v_s(t_{i+1})$, on the border of $w(t_i)$ contained in $w(t_{i+1})$. Here, we put into $\mathcal{D}(t_{i+1})$ the vertices in the previous windows contained in $w(t_{i+1})$ whose costs are higher than that of $v_s(t_{i+1})$. This is equivalent conceptually to backtracking the path map construction to the point where $v_s(t_{i+1})$ has been about to be expanded. Once the domain $\mathcal{D}(t_{i+1})$ is defined in this way, the path map is constructed to interactively display the boundary segment to the free point $v(t_{i+1})$.

If the path is observed to digress from the target boundary, a new seed point, again denoted $v(t_0)$, is planted where the digression begins. The new seed point freezes the previous boundary segment and starts finding the next boundary segment. This process repeats until the complete target boundary is identified by a sequence of boundary segments (See Algorithm 1).

Since the update domain is restricted to the inside of the

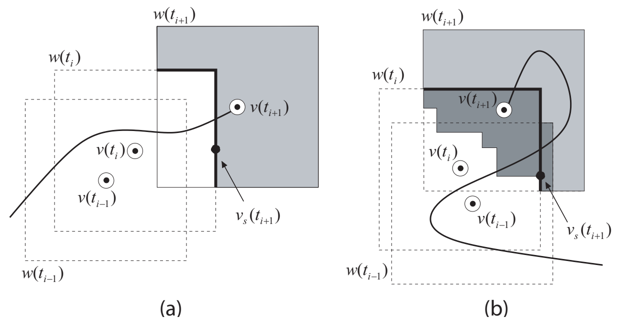

Fig. 4. *Enhanced lane: (a) The boundary segment enters a non-overlapping region in $w(t_{i+1})$. (b) The boundary segment is allowed to pass through one of the previous windows in $w(t_{i+1})$. Lightly painted is a non-overlapping region, and a darkly painted region contains the vertices with higher costs than that of $v_s(t_{i+1})$.*

Algorithm 1 Enhanced Lane

```
INPUT : An initial vertex $v_s$ and a goal vertex $v_g$
OUTPUT : A set of boundary segments forming an optimal path from $v_s$ to $v_g$
1:  $v(t_0) := v_s$
2:  $k := 0$
3:  while $v(t_k) \neq v_g$ do
4:      if $k = 0$ then
5:          $v_s(t_k) := v(t_k)$
6:          $\mathcal{D}(t_k) := w(t_k)$
7:      else
8:          Get $v(t_k)$
9:          $v_s(t_k) :=$ the minimum-cost vertex in the border of $w(t_{k-1})$ in
            $w(t_k)$
10:         $\mathcal{D}(t_k) :=$ the non-overlapping region in $w(t_k) \cup$ the region con-
            taining the vertices of the previous windows in $w(t_k)$ whose costs
            are higher than that of $v_s(t_k)$
11:     end if
12:     Construct a path map in $\mathcal{D}(t_k)$ starting from $v_s(t_k)$
13:     Display a boundary segment from $v(t_0)$ to $v(t_k)$ according to the
        path map
14:     if $v(t_k)$ is a new seed then
15:         freeze the previous boundary segment from $v(t_0)$ to $v(t_k)$
16:         $v(t_0) := v(t_k)$
17:         $k := 0$
18:     else
19:         $k := k + 1$
20:     end if
21: end while
```

current local window, the time for the path map construction at each time instance is bounded by the local window size. We assume that the window sequence does not miss the target boundary, that is, for any time instance t_i, the vertex on the border of $w(t_i)$ over which the target boundary crosses should be inside of $w(t_{i+1})$. This assumption, however, does not necessarily mean that a new seed point should be planted whenever a window misses the target boundary. Rather, a simple cursor movement would suffice to get it back on track as will be discussed in section 4.

C. Analysis

We claim that our algorithm always produces the same result as that of the live wire assuming that the window sequence completely contains the target boundary in the right order. This can be formally stated as follows: The enhanced lane can find a globally optimal path P_G if a window sequence $W = (w(t_0), ..., w(t_n))$ completely contains P_G in the right order, that is, for any time instance t_i, the vertex on the border of $w(t_i)$ over which P_G crosses is inside of $w(t_{i+1})$.

Our proof will be an induction on time t_i, $0 \leq i \leq n$, to show the following property holds true for each t_i: In the the path map constructed up to t_i by the enhanced lane, the boundary segment ending at the vertex on the boundary of $w(t_i)$, denoted $v(P_G, t_i)$, through which P_G passes from $w(t_i)$ to $w(t_{i+1})$, is the same as the initial segment of the globally optimal path P_G ending at $v(P_G, t_i)$.

For the first window $w(t_0)$, the boundary segment starting from the seed point $v(t_0)$ and ending at $v(P_G, t_0)$ is defined by the initial path map construction. This segment should be the same as the target path segment between $v(t_0)$ and $v(P_G, t_0)$ since otherwise there would be a better path connecting the seed point and $v(P_G, t_0)$, which contradicts the assumption that the target path is globally optimal. Note that any segment of a globally optimal path connecting the seed point and an intermediate vertex is itself a globally optimal path, according to Bellman's principle of optimality [3].

Now suppose that the property holds for all i such that $0 \leq i \leq k < n - 1$. That is, the boundary segment ending at $v(P_G, t_i)$ which is constructed up to t_i by the enhanced lane, is the same as the target path segment connecting the seed point and $v(P_G, t_i)$, for $0 \leq i \leq k < n - 1$. Now, we will show that this also holds true for $i = k + 1$.

We first observe the case where the target path does not meet any previous window in $w(t_{k+1})$ after it passes through $v(P_G, t_k)$ (Figure 5(a)). In this case, the boundary segment to extend at t_{k+1} is entirely contained in the non-overlapping region of $w(t_{k+1})$ with any previous windows. Since this region has not been explored, our algorithm takes the vertices of this region into the domain $\mathcal{D}(t_{k+1})$. Therefore, the boundary segment that has been constructed up to $v(P_G, t_k)$ is extended to reach $v(P_G, t_{k+1})$ and contains the target path segment between $v(P_G, t_k)$ and $v(P_G, t_{k+1})$. This path segment lies in $w(t_{i+1})$ by assumption. If a different boundary segment were obtained, the live wire also would take that segment, contradicting the assumption that P_G is globally optimal.

The second case is that the boundary segment does meet one or more previous windows overlapping $w(t_{k+1})$ after it passes through $v(P_G, t_k)$ (Figure 5(b)). In this case, some vertices in the previous windows are to be updated to have lower costs. Note that any vertices in the previous windows which have been assigned higher costs than $v(P_G, t_k)$ will be updated. All of these vertices are included in $\mathcal{D}(t_{k+1})$ by the algorithm since their costs will be higher than that of $v_s(t_{k+1})$, which should be less than or equal to that of $v(P_G, t_k)$. By a similar argument to the previous case, the new boundary segment in $w(t_{k+1})$ that leads to $v(P_G, t_{k+1})$ is the same as the target path segment between $v(P_G, t_k)$

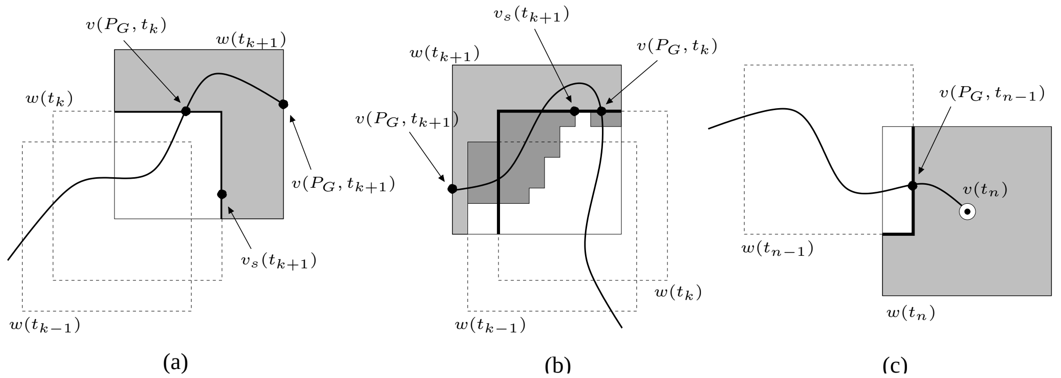

Fig. 5. *Enhanced lane: (a) The 1st case: the* $(k+1)$*th boundary segment is entirely contained in a non-overlapping region in* $w(t_{k+1})$*. (b) The 2nd case: The* $(k+1)$*th boundary segment passes through to one or more previous windows in* $w(t_{k+1})$*. (c) The process is terminated at* t_n*. Lightly painted is a non-overlapping region, and a darkly painted region contains the vertices with higher costs than that of* $v_s(t_{k+1})$*.*

and $v(P_G, t_{k+1})$. Hence, we can conclude that the enhanced lane constructs the boundary segment leading to $v(P_G, t_i)$ that is the same as the target path segment connecting the seed point $v(t_0)$ and $v(P_G, t_i)$ for all i such that $0 \leq i \leq n-1$.

Finally, the boundary segment from $v(P_G, t_{n-1})$ to the goal vertex $v(t_n)$ is obtained at t_n, and it is also the same as the target path segment between $v(P_G, t_{n-1})$ and $v(t_n)$. Note that all the vertices on this final segment are contained in $\mathcal{D}(t_n)$ since P_G is globally optimal and hence these vertices must have been assigned higher cost than that of $v(P_G, t_{n-1})$ (Figure 5(c)). This completes the proof of our assertion that the boundary segment between the seed and cursor (free) vertices generated by the enhanced lane is the same as that of the globally optimal path P_G as long as the window sequence completely contains P_G in the right order.[3]

The time complexity of the enhanced lane is dependent upon the size and the number of local windows used. Mortensen et al. showed that the path map construction time for a given image can be reduced to $O(N)$ where N is the number of image pixels for which optimal paths have been computed from the pixel to a seed point [5], [6]. In our algorithm, the number of vertices in the domain $\mathcal{D}(t_i)$ at each time instance t_i, is always bounded by the size of the corresponding local window $w(t_i)$. Hence, the time complexity for the path map construction at t_i by our algorithm is $O(M)$, where M is the number of vertices in $w(t_i)$. If we let L be the total number of windows used, then the overall time complexity for constructing a boundary segment will be $O(ML)$, assuming that the local window size is fixed. Since M (and also ML) is usually much smaller than N (the total number of vertices in the entire image), the overall computation time is much less than the case of global graph search.

D. Comparison with live wire on the fly

In this section, we make detailed comparison between the enhanced lane and the live wire on the fly, the time-efficient version of the live wire. Both techniques update their path maps incrementally as the cursor moves. However, their ways of achieving incremental update are quite different. The live wire on the fly confines path map construction up to the path length (cost) of the current cursor position to avoid unnecessary computation. Meanwhile, the enhanced lane accelerates the construction by restricting the search domain within a local window.

Suppose that the target path is globally optimal. Then the live wire on the fly can trace this path, adopting the search paradigm from the live wire. The enhanced lane can also trace it, as long as the window sequence completely contains the path in the right order. If the target path is not globally optimal, the live wire on the fly may require extra seed points to prevent the boundary segment from digressing from the target path. However, the enhanced lane does find the desired path without digression if the target path is *locally optimal* within the union of the local windows used.[4] Thus, the enhanced lane can trace the target path between the start and goal points, by interactively steering the local window. This effectively keeps the boundary segment from snapping at nearby features not on the target path, to reduce the number of seed points. Therefore, the enhanced lane can also achieve better repeatability of segmentation since the seed points are inherently not guaranteed to lie on the target path.

Another issue is the response time. For the live wire on the fly, the response time depends on the number of nodes in

[3] We note that our proof is based on the assumption that the globally optimal path is unique. If there are multiple globally optimal paths between seed and cursor (free) vertices, that is, the paths with the same least cost but different vertex sequences, our algorithm always chooses one of them that satisfies our assumption.

[4] The locally optimal path means the minimum-cost path in a given local region or in the union of those regions. Given a window sequence $W = (w(t_0), ..., w(t_n))$, the enhanced lane finds a path P_L if W completely contains P_L in the right order and P_L is locally optimal in the union of all the windows in W (This can be proved in a similar way as in Section C).

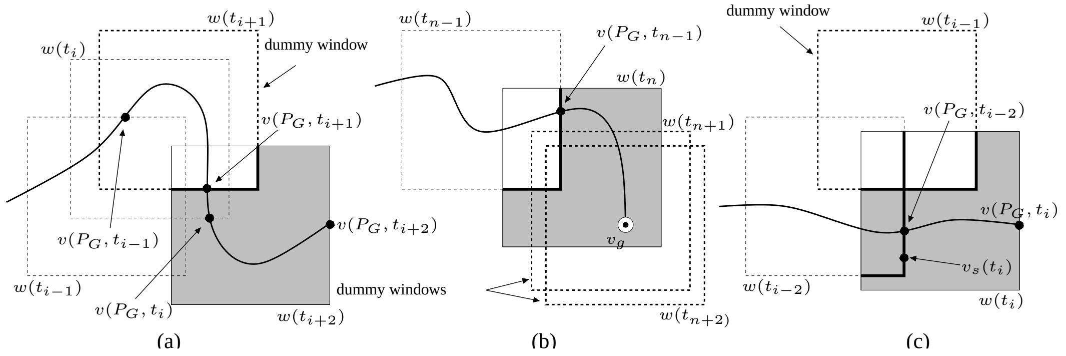

Fig. 6. *Dummy windows: (a) The same result is obtained without a dummy window (b) Terminating situation (c) A window is completely out of the target path*

the current shortest path tree. The number of nodes being expanded at each cursor movement is small at first, but increases as the size of the tree gets bigger. That is, the response time gets longer as the cursor moves farther away from the seed point. This will be more noticeable for a large image as the size of the shortest path tree tends to increase accordingly.[5] In order to maintain the real-time response in this case, a new seed point needs to be planted even without digression, which could degrade the repeatability of segmentation. For the enhanced lane, however, the response time is always bounded by the window size, thus guarantees interactive speed regardless of the image size. Moreover, whenever a new seed point is planted by a mouse click, the live wire on the fly needs to take some time for clearing all the path tree nodes still remaining in the circular queue. The number of those nodes is also proportional to the size of the path tree. The enhanced lane, on the contrary, does not cause such delay since it handles only the nodes lying in the current window.

IV. Implementation details

As mentioned in the previous section, we assume that for any time instance t_i, the vertex on the border of $w(t_i)$ over which the target boundary crosses lies inside of $w(t_{i+1})$. In practice, this rule may be somewhat restrictive for users since whenever the rule is broken, the path map needs to be reinitialized with a new seed point. With our algorithm, this rule can be applied rather flexibly by allowing some *dummy windows* to be created during interactive segmentation. A window in a sequence of windows is called dummy if the same boundary segment is achieved regardless of it. Figure 6 shows several cases where dummy windows appear. In each case, consider a globally optimal path P_G. In Figure 6(a), after P_G passes through $v(P_G, t_{i-1})$, the boundary segment between $v(P_G, t_{i-1})$ and $v(P_G, t_i)$ is constructed in $w(t_i)$. However, as shown in the figure, P_G then directly goes into $w(t_{i+2})$ without passing through $w(t_{i+1})$. This implies $w(t_{i+1})$ does not contribute to the construction of P_G, and hence it is a dummy window in this case. That is, our assumption will hold true even if we remove $w(t_{i+1})$ from the window sequence, and thus the boundary segment up to $v(P_G, t_{i+2})$ will be constructed correctly. Note, however, our algorithm can successfully find P_G even with the existence of $w(t_{i+1})$, since $w(t_{i+1})$ does no harm but just causes some backtracking at t_{i+2} (to go back to $v(P_G, t_{i+1})$ which is ahead of $v(P_G, t_i)$).

[5] In the worst case, the size of the path tree can match the size of the input image.

In our proof based on a mathematical induction, after constructing the boundary segment from v_s up to $v(P_G, t_{n-1})$, we find the final boundary segment reaching v_g at t_n. In fact, while the final boundary segment is constructed at t_n in the path map, it may not yet be completely displayed unless the user places the cursor exactly on the goal vertex v_g (Figure 6(b)). That is, it may take a few more steps of cursor adjustment by the user to place v_g exactly at the center of the window. These additional steps are just needed for display and do not affect the boundary construction algorithm since the optimal boundary has already been defined at t_n in the path map. Thus, as shown in Figure 6(b), windows $w(t_{i+1})$ and $w(t_{i+2})$ are dummy.

Figure 6(c) shows a case where P_G does not even touch a window $w(t_{i-1})$, and our algorithm cannot find P_G in this case since $\mathcal{D}_{i-1}$ does not contain $v(P_G, t_{i-2})$. However, even this kind of user's mistake can be permitted if we slightly modify the domain selecting strategy of the algorithm. That is, we can select the least cost vertex $v_s(t_i)$ that defines $\mathcal{D}(t_i)$, from the boundaries of *all* the previous windows lying in $w(t_i)$ (denoted as thicker lines in the figure), rather than just from the boundary of $w(t_{i-1})$ in $w(t_i)$. This is conceptually equivalent to backtracking the path map construction to a specific time instance when a vertex with the lowest cost ($v_s(t_i)$ in the figure) in all the previous window boundaries lying in $w(t_i)$ was about to be expanded. Thus, any wrong cursor movement thereafter can be repaired. With this strategy, we do not require any new seed point to be planted and just a simple window movement toward the right path will suffice to get back on track. In an extreme case, we can backtrack to reach the seed point to resume the target boundary tracking when the window sequence is completely out of the target path and the boundary segment has been chasing a wrong path for a while.

When the image is noisy, low contrasted, or contains many complex objects, the possibility of path digression goes high with the live wire and live wire on the fly, and excessive seed points may be required for the entire segmentation. The enhanced lane, however, allows an interactive control of window size to minimize the path digression and hence reduce the number of seed points needed. In an extreme case, the number of seed points can be reduced to just 'one' (which is the starting point) provided that the window size is effectively controlled. This enables the enhanced lane to have better repeatability than the live wire, the live wire on the fly, and the live lane. However, if the window size gets too small, the efficiency of user's interactability can be affected, degrading the overall performance of the path construction. To avoid this, we provide a separate window which shows the magnified view of the local window. Moreover, when the digression occurs despite the window size control, the image snapping technique [15] is employed to put the seed point as close as possible to the target boundary. These additional features help improve the repeatability of the enhanced lane even more.

V. Results

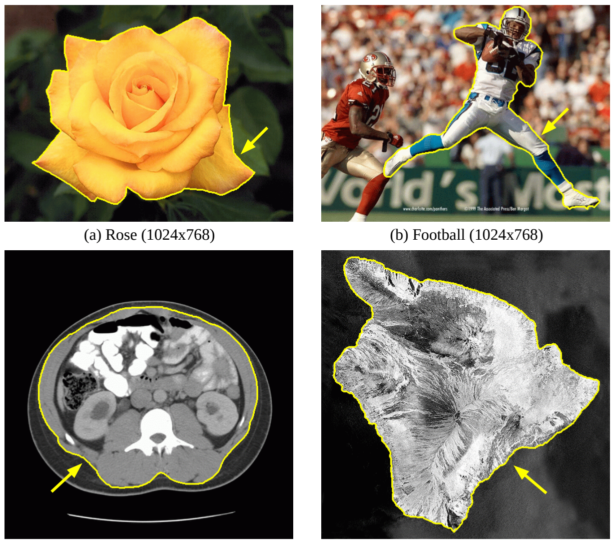
(a) Rose (1024x768) (b) Football (1024x768)
(c) CT image (1024x1024) (d) Satellite image (2048x2048)

Fig. 7. *Various test images*

We have tested our segmentation technique on more than 200 images of various sizes and complexities. Figure 7 shows some of the test images used to assess the efficiency, robustness and generality of each method of interest (Live Wire, Live Lane, and Live Wire On The Fly and Enhanced Lane)[6]. A wide variety of images are included, from gray-scale to color images, some of which are blurry, noisy, low-contrasted and contain objects with complex boundaries. The target object in Figure 7(a) has rather a simple shape with a monotone background. Figure 7(b) contains an object with highly complex boundary surrounded by a complicated background of arbitrary colors. Figure 7(c) is a CT scan image of a human chest, in which the boundaries between organs are rather blurry. Figure 7(d) is a gray-scale satellite image of an island. Note that sufficiently large images (with more than 700,000 pixels) are chosen for the experiments to show the relative merits of our method over others. All the experiments have been conducted on Intel $\mathrm{Pentium}^{\mathrm{R}}$ PC (P4 2.20 GHz processor with 512 MB memory).

image	size	measure	time (sec.)	# seeds	repeatability
Rose	1024 × 768	LW	44.91	9.1	0.956
		LL	22.16	47.6	0.867
		LWOF	16.77	8.7	0.962
		EL	9.73	8.5	0.969
Football	1024 × 768	LW	167.03	25.3	0.871
		LL	25.68	60.1	0.805
		LWOF	29.84	27.2	0.861
		EL	21.78	24.8	0.884
CT	1024 × 1024	LW	115.67	19.1	0.804
		LL	19.49	43.2	0.751
		LWOF	24.65	19.5	0.821
		EL	15.69	18.7	0.836
Satellite	2048 × 2048	LW	156.10	7.1	0.979
		LL	30.47	107.3	0.835
		LWOF	51.27	6.8	0.977
		EL	21.87	6.1	0.986

TABLE I
TEST RESULTS ON VARIOUS IMAGES

Table I demonstrates the test results. Three different users participated in the test, and each of them made three separate trials of segmentation for each image. In the table, LW, LL, LWOF, and EL stand for the four segmentation techniques as mentioned above, respectively. *Segmentation time* shows the average computation time taken for constructing the target boundary. For each image, the time is measured by the sum of the response times for all cursor movements and mouse clicks. *# of seeds* means the average number of the total seed points planted during segmentation. *Repeatability* is measured in the same way as defined in [8] for every pair of three trials of segmentation for each user. Table II shows the average repeatability obtained for Figure 7(b). U1, U2, U3 represent three different users, and T1, T2, T3 denote their three separate trials, respectively. For the live lane and the enhanced lane, the window of a fixed size (90×90) is

measure	user	T1,T2	T2,T3	T1,T3	avg.
LW	U1	0.861	0.872	0.874	0.871
	U2	0.903	0.893	0.899	
	U3	0.853	0.839	0.845	
LL	U1	0.811	0.822	0.827	0.805
	U2	0.842	0.823	0.831	
	U3	0.750	0.747	0.794	
LWOF	U1	0.854	0.856	0.839	0.861
	U2	0.901	0.889	0.887	
	U3	0.840	0.849	0.834	
EL	U1	0.893	0.868	0.890	0.884
	U2	0.913	0.904	0.911	
	U3	0.843	0.875	0.865	

TABLE II
REPEATABILITIES FOR THE 'FOOTBALL' IMAGE

[6] The arrows indicate the target boundaries.

used, and the image snapping technique is not employed for planting seed point. As the cost function, we employed the gradient magnitude with the inverted linear transformation function for all the images and methods [8].[7]

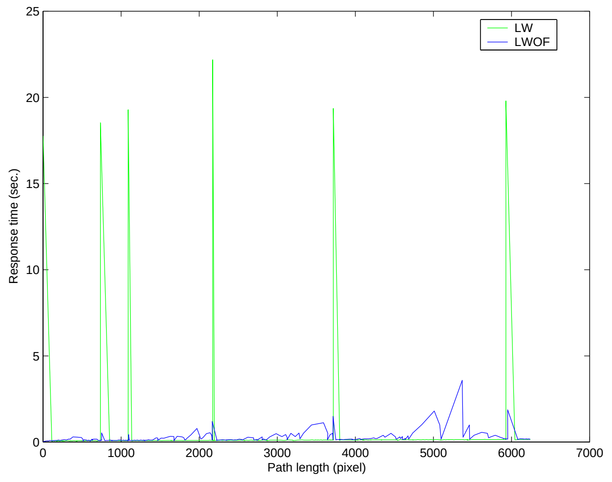

Fig. 8. *Comparison of the response time between LW and LWOF*

As shown in Table I, the enhanced lane takes the least segmentation time. The live wire takes considerable response time at each mouse click (seed point planting), to construct a new path map over the entire image. The live lane, like the enhanced lane, bounds each of its response time by the local window size, but the sum of all response times is generally longer than the enhanced lane, since the live lane moves slowly near the border of every window to generate a large number of mouse events. In the case of the live wire on the fly, the response time is rather small at first but increases as the cursor moves away from the seed point. Figures 8 and 9 show the patterns of response time for different techniques,

[7] The use of different cost functions may produce different time, number of seed points, and repeatability for each segmentation method, but the relative superiority between methods remained the same.

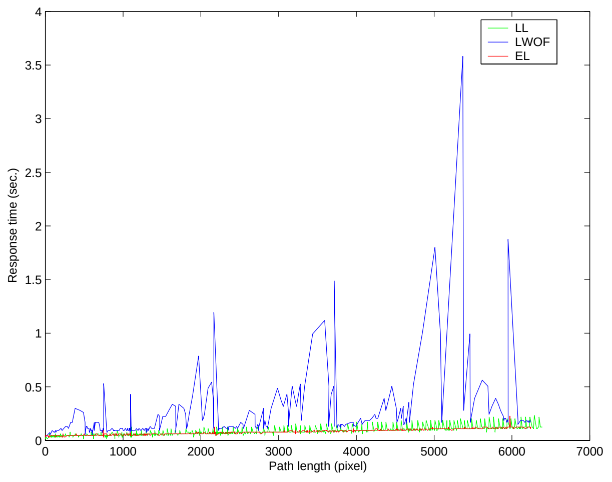

Fig. 9. *Comparison of the response time between EL, LL and LWOF*

size	measure	time (sec.)	# of seeds	repeatability
256 × 256	LW	5.51	14.8	0.826
	LL	2.75	16.7	0.782
	LWOF	3.08	14.5	0.838
	EL	4.02	14.6	0.831
512 × 512	LW	26.33	18.2	0.814
	LL	10.35	24.5	0.782
	LWOF	13.62	18.0	0.805
	EL	9.46	17.5	0.830
1024 × 1024	LW	115.67	19.1	0.804
	LL	19.49	43.2	0.751
	LWOF	24.65	19.5	0.821
	EL	15.69	18.7	0.836
2048 × 2048	LW	396.35	18.4	0.851
	LL	29.51	74.5	0.694
	LWOF	43.34	18.1	0.863
	EL	23.93	16.5	0.895

TABLE III
TEST RESULTS ON THE 'CT IMAGE' FOR VARIOUS SIZES

computed along the boundary path length (measured in pixels) from the starting point.[8] As exhibited in Figure 8, the live wire yields a peak time at each seed point. The live wire on the fly takes much less response time at the seed point than the live wire as expected. However, its response time at each mouse movement tends to increase as the cursor moves farther from the seed point. As shown in Figure 9, while the response time for the live wire on the fly is proportional to the distance from seed point, the response times for both the enhanced lane and the live lane are very small and remain almost constant regardless of the distance.

As for the other statistics on the table, the live lane consumes the largest number of seed points since a new seed point is created on the border of each window. The enhanced lane creates less seed points than the live wire and the live wire on the fly since the localization of search domain reduces the digression possibility. As the same cost function is used for all the methods, the level of repeatability depends mainly on the number of seed points used. That is, the repeatability is inversely proportional to the number of seed points, and thus the enhanced lane shows the highest repeatability among four test methods.[9] Although we used a fixed window size for all images, the enhanced lane can possibly achieve even higher repeatability by dynamically controlling the window size.

Tables III and IV respectively exhibit the test results on the Figures 7(c) and 7(d) with varying the image size (from 256×256 to 2048×2048). As shown in these tables, the differences between segmentation methods become more obvious as the image gets larger. For a large image of size over 1000×1000, the segmentation time for the live wire is $4.8 \sim 16.6$ times longer than the enhanced lane. The live lane requires $2.3 \sim 17.5$ times more seed points than the enhanced lane, resulting in $0.77 \sim 0.89$ times lower repeata-

[8] This experiment is conducted on Figure 7(d).
[9] The repeatabilities obtained from our tests are a bit lower than those reported in [8]. The repeatability (and also the time efficiency) can be affected by various factors such as the usage of different data sets, the choice of cost functions, testee's skill, etc.

size	measure	time (sec.)	# of seeds	repeatability
256×256	LW	3.41	4.5	0.988
	LL	2.95	15.5	0.945
	LWOF	2.85	4.3	0.987
	EL	3.37	4.5	0.981
512×512	LW	11.33	5.7	0.978
	LL	8.60	30.5	0.933
	LWOF	9.25	5.2	0.975
	EL	4.66	5.3	0.981
1024×1024	LW	46.80	6.5	0.957
	LL	15.96	61.5	0.892
	LWOF	18.95	6.7	0.966
	EL	9.84	6.4	0.972
2048×2048	LW	156.10	7.1	0.979
	LL	30.47	107.3	0.835
	LWOF	51.27	6.8	0.977
	EL	21.87	6.1	0.986

TABLE IV
TEST RESULTS ON THE 'SATELLITE IMAGE' FOR VARIOUS SIZES

bility.[10] For the live wire on the fly, the segmentation time for a large image is much faster than that of the live wire,[11] and the repeatability is much better than that of the live lane by producing almost the same number of seed points as the live wire. Compared to the live wire on the fly, the enhanced lane spends less segmentation time (especially for a large image), due to the strictly bounded response time regardless of the image size ($1.5 \sim 2.3$ times faster). Also, the enhanced lane consumes a smaller number of seed points by its path map localization, which leads to slightly better repeatability of segmentation ($1.01 \sim 1.03$ times better in repeatability).

VI. Conclusion

We have presented a novel image segmentation tool called enhanced lane. Based on the live lane paradigm, the enhanced lane constructs a path map in a local window centered at a seed point that is initially planted on the target boundary. As a cursor moves close to the target boundary, the minimum-cost path from the seed point to the cursor point is interactively displayed, which gives an impression that the path snaps to and wraps around the target boundary. The local window moves along with the cursor point, and the path map is incrementally updated to extend the minimum-cost path until the path digresses from the target boundary. As a new seed point is planted where the digression occurred, the new path map is created, and the former path segment is fixed as an identified target boundary segment. The complete boundary is obtained when a sequence of these path segments forms a closed path.

[10] The repeatability of the live lane can be improved by employing the seed point snapping technique whenever the cursor crosses the window border [15], but the resulting boundary still does not guarantees the optimal target path.

[11] The gap of segmentation time between LW and LWOF computed in our experiments is much smaller than that reported in [9]. This gap can vary with different input images, and especially depends heavily on the number of seed points used. The more seed points we use, the bigger becomes the gap.

We have proven that if the target path is globally optimal, the enhanced lane can always capture it as long as the local window sequence contains it. We compare our method with others such as the live wire, the live lane, and the live wire on the fly [8], [9] in terms of time efficiency and repeatability. While the live wire preserves the repeatability at the cost of its speed (especially for a large image), the live lane sacrifices its repeatability to gain better time efficiency. The live wire on the fly and the enhanced lane both improve their efficiencies by incrementally updating the path maps. However, while the live wire on the fly confines path map construction up to the current path length (cost), the enhanced lane accelerates the construction by restricting the search domain within a local window. Based on live lane paradigm, the enhanced lane always guarantees strictly bounded response time regardless of the image size unlike the live wire on the fly. Also, our method reduces the number of digressions by its path map localization, which leads to better repeatability.

Our boundary construction algorithm inherently provides considerable freedom in cursor movement by allowing dummy windows created during the interactive process. To avoid user's fatigue and low efficiency caused by an extremely small window, the feature of automatic window zooming is provided when the window size gets smaller than some threshold. In conclusion, the enhanced lane is capable of segmenting complex foreground objects from an arbitrary background of a noisy, low-contrasted image with interactive speed regardless of its size.

References

[1] Dana H. Ballard and Christopher M. Brown. *Computer Vision*. Prentice-Hall Inc., 1982.

[2] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Addison-Wesley Publishing Company, Inc., 1992

[3] R. Bellman and S. Dreyfus, *Applied Dynamic Programming*. Princeton, NJ: Princeton University Press, 1962.

[4] E. W. Dijkstra. A Note on Two Problems in Connexion with Graphs. In *Numerical Mathematik*, vol.1, pp.269-270, 1959.

[5] Eric N. Mortensen and William A. Barrett. Intelligent Scissors for Image Composition. In *Computer Graphics (SIGGRAPH '95 Proceedings)*, pages 191-198, 1995.

[6] Eric N. Mortensen and William A. Barrett. Interactive Segmentation with Intelligent Scissors. *Graphical Models and Image Processing*, No. 60, pp.349-384, 1998.

[7] E. N. Mortensen and W. A. Barrett. Toboggan-Based Intelligent Scissors with a Four Parameter Edge Model. In *Proc. IEEE Computer Vision and Pattern Recognition (CVPR '99)*, Vol. II, pp. 452-458, June 1999.

[8] Alexandre X. Falcão, Jayaram K. Udupa, Supun Samarasekera, Shoba Sharma, Bruce Elliot Hirsch,

and Roberto de A. Lotufo. User-Steered Image Segmentation Paradigms: Live Wire and Live Lane. *Graphical Models and Image Processing*, No. 60, pp.233-260, 1998.

[9] A. X. Falcão, J. K. Udupa, and F. K. Miyazawa. An Ultra-Fast User-Steered Image Segmentation Paradigm: Live Wire on the Fly. *IEEE Transactions on Medical Imaging*, Vol. 19, No. 1, pp.55-62, 2000.

[10] A. X. Falcão and J. K. Udupa. A 3D Generalization of User-Steered Live-Wire Segmentation. *Medical Image Analysis*, Vol. 4, pp.389-402, 2000.

[11] K. C. -H. Wong, P. -A. Heng, and T. -T. Wong. Accelerating 'Intelligent Scissors' Using Slimmed Graphs. *Journal of Graphics Tools*, Vol. 5, No. 2, pp.1-13, 2000.

[12] G. Ramalingam and T. Reps. An Incremental Algorithm for a Generalization of the Shortest-Path Problem. *Journal of Algorithms*, Vol. 21, No. 2, pp.257-305, 1996.

[13] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active Contour Models. *International Journal of Computer Vision*, Vol. 1, No. 4, pp. 321-331, 1988.

[14] J. Liang, T. McInerney, D. Terzopoulos. United Snakes. In *Proc. IEEE International Conference on Computer Vision (ICCV '99)*, pp. 933-940, Sept. 1999.

[15] Michael Gleicher. Image Snapping. In *Computer Graphics (SIGGRAPH '95 Proceedings)*, pages 183-190, 1995.

[16] Linnea Dayton and Jack Davis. *The Photoshop 6 Wow! Book*. Peachpit press, 2001.

[17] SDC Information Systems. *ProntoMask: User Manual (version 1.0)*. May, 2000.

[18] F. R. Hansen and H. Elliott. Image segmentation using simple Markov Random Field models. *Computer Graphics and Image Processing*, Vol. 20, pages 101-132, 1982.

[19] Robert M. Haralick and Linda G. Shapiro. Image Segmentation Techniques. *Computer Vision, Graphics, and Image Processing*, Vol. 29, pp. 100-132, 1985.

[20] I. Y. Kim. An Integrated Approach for Image Understanding Based on MRF Model. Ph.D. thesis, Korea Advanced Institute of Science and Technology, 1994.